Modified Tendermint With Granular Timing Model

Peifeng He
Duke University
ph162@duke.edu

***Abstract*—Current Byzantine fault-tolerant (BFT) consensus protocols typically rely on either fully synchronous networks or partially synchronous networks, which divide time frames into two phases. These protocols generally assume uniform timing properties between parties. However, in practice, the timing model is often more complex, forming a granular structure among parties. In this setting, asynchronous, synchronous, and partially synchronous links can coexist during the operation of a consensus protocol. Tendermint, a variation of BFT consensus, could potentially be modified to accommodate this granular network graph structure with both safety and liveness satisfied. We proved that a modified Tendermint with $n \geq 2f + 1$ is solvable if and only if any set of $n - 2f$ honest nodes can communicate synchronously with a set of $f + 1$ honest nodes.**

I. Introduction

Previous studies on BFT consensus have typically focused on a singular timing model. However, a granular graph structure more accurately reflects real-world conditions, where synchrony, asynchrony, and partial synchrony coexist. Prior research has implemented granular partial synchrony (where synchrony and partial synchrony coexist) and granular synchrony (where synchrony, partial synchrony, and asynchrony coexist) in BFT/CFT protocols [1].

Tendermint [2], a variation of the BFT protocol, serves as the foundation for our work. Our objective is to extend granular synchrony to protocols beyond the traditional BFT framework. Specifically, we aim to explore whether other consensus protocols can be adapted to incorporate a graph structure under granular synchrony. Our primary goal is to enhance Tendermint's resilience against potential Byzantine activities within a granular synchrony setting. Additionally, we seek to determine whether Tendermint has the capacity to tolerate a greater number of Byzantine faults and simplify protocol complexity under these conditions.

Tendermint has multiple versions, each with slight differences compared to the version described in the Tendermint white paper. The specific version we reference is included in the appendix. There are notable differences between our version of Tendermint and the PBFT protocol. In our version, the leader is not required to receive $n - t$ status messages S before proposing a block, which differentiates Tendermint from PBFT. Instead, the leader only sends the updated highest lock for the Status set to all parties, rather than the entire set of Status.

We modified the Tendermint protocol. We applied granular partial synchrony to Tendermint, arguing that, for any honest set A with at least $n-2f$ parties, there exist a potentially larger honest set B with at least $f + 1$ parties such that for a random node b in B, we can find a node a in A such that a and b can communicate synchronously.

II. Related Work

a) Tendermint: Tendermint operates by electing a proposer (leader) in a round-robin fashion to propose a block, followed by voting phases where validators signal their agreement or disagreement. Each round of the protocol consists of three main phases: proposal, pre-vote, and pre-commit. This structure makes Tendermint resilient to $t < \frac{n}{3}$ Byzantine parties, capable of continuing operation even when some nodes behave maliciously or go offline [2].

b) Granular Synchrony: Granular synchrony represents a network as a combination of synchronous, partially synchronous, and asynchronous communication links. It acts as a comprehensive framework where existing mainstream models can be viewed as specific instances of it.[1].

III. Models and Definitions

a) Consensus Protocol: In a consensus protocol, each node starts with an initial input value and must reach a decision on a value that fulfills the following criteria:

- **Agreement:** No two correct nodes decide different values.
- **Termination:** Every correct node eventually decides.
- **Validity:** If all nodes have the same input value, then that is the decision value. Here we use External Validity [3].

b) Graph Structure: For graph $G = (V, E)$, each vertex V represents a party, and each edge E represents the communication link between parties V_i and V_j. All the E are undirected, indicating that for the two nodes of this E, the property of E is the same for both of them. Parties don't know any information about the timing model. The link can be either synchronous or partial synchronous in *granular partial synchrony* and can be synchronous, partial synchronous, or asynchronous in *granular asynchrony*.

c) Synchronous Path : Same as in [1]. Added to the Appendix.

IV. Tendermint Consensus in Granular Partial Synchrony

a) Theorem 1: Under granular partial synchrony, Tendermint with $n \geq 2f + 1$ is solvable if and only if, for a graph $G = (V, E)$, any set F with maximum of 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 $A \to B$.

b) Proof of Necessity: We prove the "only if" part of Theorem 1 first. Here we want to prove that Tendermint protocol that solves the consensus problem must have a graph structure like what has been described in the Theorem 1. The proof is exactly the same as the proof for necessity in granular partial synchrony BFT protocol in [1]. The reason is because the consensus problem(to achieve Agreement, Termination, and Validity) and the graph structure are both the same.

c) Protocol: Based on the graph structure in Theorem 1, we provide a Tendermint protocol that achieves *external validity*, agreement, and termination. Again, leaders in Tendermint are not required to receive a set of $n - f$ $Status$ messages to propose in a specific view.

The $lock$ is defined as a set of $n - f$ signed vote messages $(Vote - 1, view_{index}, value)$. Δ is the synchronous parameter. Every node starts with $lock \leftarrow \perp$. The rank of the lock is determined by the view number.

Following is the protocol, which is also illustrated by Algorithm 1:

Status step: Every node sends a $Status$ message to the leader. And then they set up their view timer to $(6 + \delta)\Delta$.

Leader proposal broadcast step: A leader of a view v wait for Δ time. Then the leader collects a set S of all the $Status$ message it received from all the nodes. If S is not empty, the leader should find the highest-ranked lock $lock_h$ among all these locks. If all the locks are still in the default state, the leader put its own $input$ value as the value for this proposal. If not, the leader should put val from the highest lock. Then, the leader sends $(Propose, v, val, lock_h)$ to all the parties.

Equivocation check step: When a node received this $Propose$ message $(Propose, v, val, lock_h)$, it first checks whether this val is equal to the val in the $lock_h$. Besides that, the node also ensures that $lock_h$ ranks equal to or higher than the $lock$ of its own. If so, this node will send $(Propose, v, val, lock_h)$ to all the nodes. At the same time, this node starts a timer $vote_timer$ for $d\Delta$ to ensure that no equivocal $Propose$ message exists (message with $val' \neq val$). If the node does receive conflicting $Propose$ message, the node forwards the two conflicted messages to all other nodes. Since the leader is corrupted, the node will echo $(ViewChange, v)$ for the current view as well. If after the timer expired, the node did not receive any conflicting information, then the node vote for this proposal, sending $(Vote - 1, v, val)$ to all.

Locking step: If a node received $n - f$ $(Vote - 1, v, val)$, then the node creates a lock certificate L for these Vote-1 messages. Then the node updates its $lock$ to L, and sends $(Vote - 2, v, val)$ to all parties.

Commit step: If a node received a set C contains $n - f$ $(Vote - 2, v, val)$, the node sends commit message $(Commit, C)$ to all parties. The node commits if it received C or a $(Commit, C)$ then terminates.

View update step: A node also sends $ViewChange$ message to all if the view timer expired. Upon receiving a set VC with $f + 1$ $(ViewChange, v)$ messages, with v larger than or equal to v_i (the view index for party i), a node will stop sending $Vote$ messages for views up to v. Then, it sends VC and the current lock $(Locked, lock)$ to all. The node waits for 2Δ, then enters the next view $v + 1$.

At any time: When a node received a lock message $(Locked, lock')$ during the protocol, the node with update its own $lock$ if $lock'$ ranked higher. Then the node sends $(Locked, lock')$ to all.

d) Analysis: Our protocol ensures external validity, which stands if all correct nodes only vote for validated proposed value. Now, we want to show the Agreement and Termination.

Lemma 1: ***In a view*** v***, if two lock certificates*** L_1, L_2 ***exist, then*** val_{L1} **=** val_{L2}**.**

Proof: Let's assume that there exist L_1 with $n - f$ $(Vote - 1, v, val_{L1})$ and L_2 with $n - f$ $(Vote - 1, v, val_{L2})$, and $val_{L1} \neq val_{L2}$.

For the $n - f$ nodes that send the $(Vote - 1, v, val_{L1})$ messages, at least $n-2f$, denoted as set S, of them are honest. According to theorem 1, we know that S is able to contact synchronously with a set H of $f+1$ honest nodes ($S \to H$). From $n >= 2f+1$, we know that there is at least one node p in both L_2 and H. Suppose s is from S. Since the graph is undirected, if $s \to p$, then $p \to s$.

Suppose that p starts the vote timer at time t. At the same time, p forwards the $(Propose, v, val_{L2}, lock_h)$ message to all. Since $p \to s$, s will receive this propose message at time $t + d\Delta$. Remember that s has voted for val_{L1}. Therefore, s already forwarded the proposal $(Propose, v, val_{L1}, lock_h)$ before time $t + d\Delta$. Because we have $s \to p$, p will receive the proposal for val before its own vote timer expires. Since p is honest, p will receive two conflicting proposals, so p will not vote, a contradiction with our assumption that p votes for val_{L2}.

Algorithm 1 Tendermint in granular partial synchrony for node i

Initialize view counter and lock: $v_i \leftarrow 0$, $lock \leftarrow \perp$
enter view 1

upon entering view v:
update $v_i \leftarrow v$
start the view timer for view changing: $view\_timer \leftarrow timer((5+d)\Delta)$
send $(\text{Status}, v, lock)$ to L_v

if node i = L_v:
wait Δ
$S \leftarrow$ all $Status$ messages L_v has received
if $S \neq \emptyset$ **then**
    $val \leftarrow$ value in the highest locked value in S, or $input_i$ if all locks in S are $\perp$
    send $(\text{Propose}, v_i, val, lock_h)$ to all
end if

upon receiving $(\text{Propose}, v_i, val, lock_h)$ from L_v:
if val matches the val' in $lock_h$ or $lock_h = \perp$ **AND** $lock_h$ is ranked equal to or higher than $lock$ **then**
    echo $(\text{Propose}, v_i, val, lock_h)$ to all
    start $vote\_timer \leftarrow timer(d\Delta)$ (for equivocation detection)
end if

upon receiving $(\text{Propose}, v_i, val, lock_h)$ and $(\text{Propose}, v_i, val', lock_h)$ where $val' \neq val$:
echo $(\text{Propose}, v_i, val, lock_h)$ and $(\text{Propose}, v_i, val', lock_h)$ to all
send $(\text{ViewChange}, v_i)$ to all

upon $vote\_timer$ expiring and no equivocation detected:
send $(\text{Vote-1}, v_i, val)$ to all

upon receiving $L \leftarrow n - f$ $(\text{Vote-1}, v_i, val)$:
$lock \leftarrow L$
send $(\text{Vote-2}, v_i, val)$ to all

upon receiving $C \leftarrow n - f$ $(\text{Vote-2}, v_i, val)$ or one (Commit, C):
send (Commit, C) to all
commit val and terminate

upon $view\_timer$ expiring:
send $(\text{ViewChange}, v_i)$ to all

upon receiving $VC \leftarrow f + 1$ $(\text{ViewChange}, v)$ where $v >= v_i$:
stop sending $Vote$ messages for views up to v
echo VC to all
echo $(\text{Locked}, lock)$ to all
wait $2d\Delta$
enter view $v + 1$

upon receiving $(\text{Locked}, lock')$ do:
$lock \leftarrow$ higher lock between $lock$ and $lock'$
echo $(\text{Locked}, lock')$ to all

Lemma 2: ***If some nodes commit*** val ***in view*** v***, then any set of lock certificate*** $n-f$ $(Vote-1, v', val')$ ***in view*** $v' \geq v$ ***must have*** $val' = val$***.***

Proof: We prove this using Induction on v.

Base case: this is when $v' = v$, which is already proved in Lemma 1.

Induction Step: Suppose that Lemma 2 holds until view $v'-1$, indicating that all lock certificates are for val. And we want to show Lemma 2 holds for view v'.

We prove by contradiction. We assume that in this view v', $n-f \geq f+1$ (at least one honest) $(Vote-1, v', val')$ has been sent. Then, among the message senders, a set H with $n-2f$ of correct nodes are included. According to theorem 1, we have $H \rightarrow P$, where P is a set of $f+1$ honest parties.

We know that some nodes committed val in view v, indicating that they have received $n-f$ ($n-2f$ are honest) $(Vote-2, v, val)$. We call these $n-2f$ nodes in a set R. And before that, R must have locked on a certificate with val in view v (so they can send $Vote-2$ message). Noticed that there is at least one honest node in both set R and P, and we call this node p. From the previous paragraph, we know that $H \rightarrow P$, and because the graph is undirected, so there exist a node h in set H such that $p \rightarrow h$.

From the hypothesis, we know that all the certificates are only locked on val until view $v'-1$. A node only updates its lock according to the progress of the view. Therefore, we can deduce that p has locked on val with a view number that is at least v ($\geq v$).

Considering a time point t_h which is the time this node h forwards VC for view $v'-1$, By time $t_h + d\Delta$, because of the synchronous path, node p will receive this VC message. Upon receiving the VC, node p will forward its own $lock$ to all the nodes, which will be received by node h after $d\Delta$ of time. If h's lock is ranked lower than v, then h will update its own lock. Otherwise, h should already locked on val according to the induction hypothesis. Now, remember that we have assumed that the h votes for val' in v', which indicates that h has received a proposal. However, according to the hypothesis, the $lock$ that has val' in this proposal must rank lower than what h has, and then it will not pass the rank check when h, an honest party, has received the proposal. Hence, there is a contradiction.

Then, we demonstrate that the protocol satisfies the consensus requirement:

Proof of External Validity As mentioned before, External Validity is easy to guarantee if honest nodes validate the val in the proposal before decide to vote for the value.

Proof of Agreement *If two honest nodes commit* v *and* v'*, then* $v = v'$.

Proof: The agreement could be derived from the Lemma we just proved. If some nodes commit val in view 1, then from Lemma 2, no other val exists in any other certificate starts from view 1. Therefore, no honest party will receive $n-f$ $Vote-2$ for any other val and commit a different value.

Proof of Liveness *All honest nodes eventually decide and terminate*

Proof: At some point after GST $+2d\Delta$, there should be a leader that is honest. Suppose that this leader is leading view v. Now we want to prove that the protocol terminates in this view.

Let this t be the time an honest node h enters this view v. h forwards VC for view $v-1$ to all nodes at time $t-2\Delta$. Since it is post GST, all correct nodes will receive this VC at time $t-2d\Delta+\Delta$, then wait for $2d\Delta$, and enter view v at time $t+\Delta$.

Upon entering this new view, honest parties will send $(Status, v, val)$ to l_v. The leader will wait Δ and receive $(Status, v, val)$ by $t+2\Delta$. The leader will forward the propose message at this point, which will be received by all nodes at $t+3\Delta$. Here, honest nodes will start the $view\ timer$. Since we have an honest leader, they will start to vote $Vote-1$ at time $t+(3+d)\Delta$. Then, they will all receive enough $Vote-1$ at $t+(4+d)\Delta$ and send out $Vote-2$. These $vote-2$ will be received at $t+(5+d)\Delta$. They will all commit at this time, and they will not heading to the next view.

Noted that the termination may be negatively influenced if Byzantine Party could perform an attack similar to Racing Condition attack. Specifically, Byzantine nodes could still control the arrival time within Δ after GST, so normally they could ensure that a small group of honest nodes lock on the new certificate before receiving the proposal. Therefore, these group of nodes will not approve the proposal because of the rank check. However, the attack will not happen here because every party is asked to wait for $d\Delta$ time in order to check the equivocation. This waiting time is longer than the maximum arrival time (Δ) of a proposal. This waiting period ensures that every honest node must have a valid proposal in hand before any further actions.

V. Discussions and Future Work

The Byzantine Fault-Tolerant (BFT) protocol has several variations. Our goal is to extend the concept of granular synchrony to a broader range of BFT consensus protocols, aiming to identify common properties when different types of communication links coexist. In this paper, we discovered that granular pattern turns out to fit Tendermint well. We assume that similar graph structure could be applied to other BFT variations. For the next step, specific graph structures, we are particularly interested in evaluating the performance of various BFT protocols to determine if the coexistence of asynchronous and synchronous links can enable certain protocols to outperform others.

Link properties could also be further extended. We ensure that messages arrive with in Δ under synchrony, but in the real world network, nodes could

be "Sluggish" [4]. Sluggish model allows the message to be delayed within honest nodes. The messages that sent by or forwarded to the sluggish nodes is delayed until the node is prompt again. Here we attempt to design a model that has $f+1$ prompt nodes, with $n \geq 2f+1$ (total of f Byzantine nodes and sluggish nodes). Noted that sluggish nodes are adaptive or mobile, which means the group of sluggish nodes is changing.

We will discuss more in the appendix.

REFERENCES

[1] N. Giridharan, I. Abraham, N. Crooks, K. Nayak, and L. Ren, "Granular synchrony," *arXiv*, vol. arXiv:2408.12853, Aug. 2024, Accessed: Oct. 24, 2024. [Online]. Available: http://arxiv.org/abs/2408.12853.

[2] J. Kwon and E. Buchman, "Tendermint: Byzantine fault tolerance in blockchain," *Technical report*, 2016.

[3] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography," *Extended Abstract*, 2000.

[4] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin, "Sync hotstuff: Simple and practical synchronous state machine replication," in *2020 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA: IEEE, May 2020, pp. 106–118. DOI: 10.1109/SP40000.2020.00044.

[5] Y.-C. Guo, R. Pass, and E. Shi, "Synchronous, with a chance of partition tolerance," *arXiv preprint arXiv:1905.08574*, 2019, Preprint available on arXiv. [Online]. Available: https://arxiv.org/abs/1905.08574.

VI. APPENDIX

Synchronous path. Node a has a synchronous path to node b, written as $a \rightarrow b$, if there exists 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., $a \rightarrow a$, $\forall a \in V$. We generalize the notion of synchronous paths from two nodes to two sets of nodes A and B.

We also define: $A \rightarrow B$ if $\exists b \in B, \exists a \in A$ such that $a \rightarrow b$.

Sluggish Granular Partial Synchrony Tendermint We provide some tentative modifications for this model.

We could add a quorum check before starting a view timer. A quorum check of receiving $f+1$ $(Propose, v, val, loch_h)$ could be added to make sure there the node at least received one proposal from an honest prompt party. Then, the node i ensures that at least one prompt party has the $(Propose, v, val, loch_h)$ in hand. By starting the timer at that time, the node could ensure no equivocation exists among prompt party if further actions (vote, lock...) exist. Here, this indicates that $n-f$ parties will attempt to work on voting val, keeping the Lemma 1 true since they would not vote for or lock on val'.

We believe that Lemma 2 is stilling working since even if a node is sluggish, the node will receive all messages as soon as it is prompt again. Therefore, even if a sluggish node does not receive the update of a higher ranked $lock$ after Δ after VC, you still have to drag it back to the prompt area to make mistakes (voting for $val' \neq val$).

Since [5] has proved that the maximum of Byzantine nodes t and sluggish nodes b should not exceed f for $n \geq 2f+1$ in a solvable consensus problem, the next step should be trying to see if sluggish nodes could make more damages to the protocol other than breaking the Lemma 1 and 2.