TBFT Modeling Progress

Abstract

Version 7: The modeling framework for the TBFT consensus mechanism has been refined.

1.The broadcast ideal functionality $\mathcal{F}_{BC}$ has been enhanced to ensure reliable transmission of information within the designated set and visibility to adversaries.

2.The functionality description of $\mathcal{F}_{TBFT}$ has been improved by incorporating a random transaction-removal feature, making the system more flexible and secure when processing transactions.

3.The protocol description π_{TBFT} has been detailed, with added descriptions of sub-functionalities, providing clearer guidance for protocol implementation.

4. The completion rate of UC modeling achieved by the protocol is about 80%.

1. Overall Framework

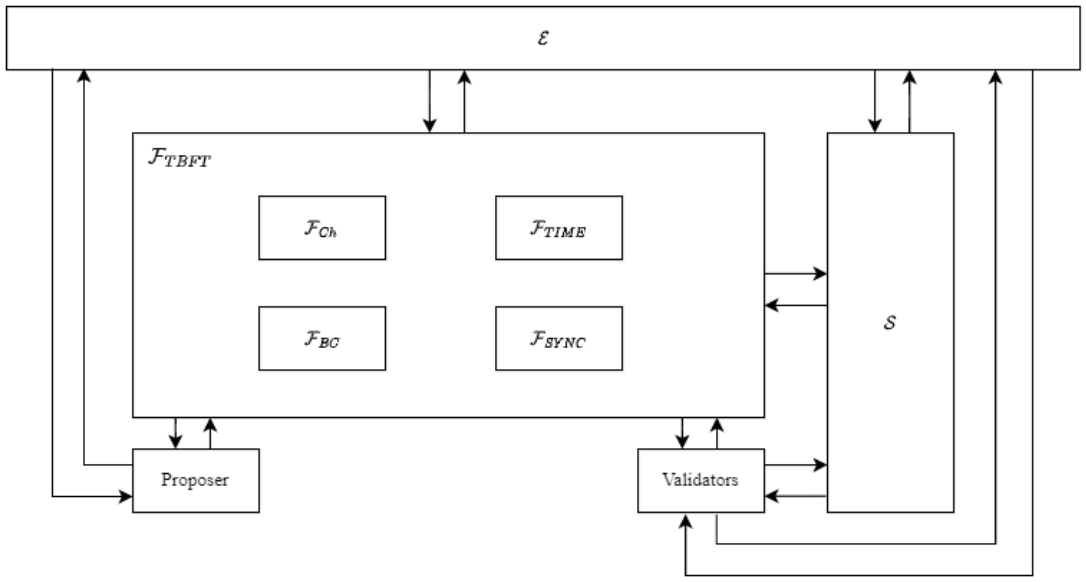

Figure 1: Overall Framework of the TBFT Protocol

2. Function Description

a). Functionality $\mathcal{F}_{\mathrm{Ch}}$

Initialization: Let define a set of parties where $\mathcal{S}$ and $\mathcal{R}$ denote two parties of the set as the sender and receiver of a message m respectively.

Δ is defined as follows based on parameters of functionality. Message identifier mid is selected freshly by the functionality.

1. Upon input $(\mathrm{Send,sid},\mathcal{R},m)$ from $\mathcal{S}$, output $(\mathrm{Send,sid},\Delta,\mathrm{mid})$ to $\mathcal{A}$.
2. Upon receiving $(\mathrm{Ok,sid,mid})$ from $\mathcal{A}$, send $(\mathrm{Received,sid},\mathcal{S},m)$ to $\mathcal{R}$.

Set Δ based on the following parameterized functions:

– for $\mathcal{F}_{\mathrm{Ch}}^{\mathrm{ac}}$ set $\Delta=(\mathcal{S},\ \mathcal{R},m)$. Upon receiving $(\mathrm{Ok.Snd,sid,mid})$ from $\mathcal{A}$, send $(\mathrm{Continue,sid})$

to $\mathcal{S}^a$.

– for $\mathcal{F}_{\text{Ch}}^{\text{sra}}$ set $\Delta = (\mathcal{S}, |m|)$.

– for $\mathcal{F}_{\text{Ch}}^{\text{ssa}}$ set $\Delta = (\mathcal{R}, |m|)$.

– for $\mathcal{F}_{\text{Ch}}^{\text{fa}}$ set $\Delta = |m|$.

– for $\mathcal{F}_{\text{Ch}}^{\text{sc}}$ set $\Delta = (\mathcal{S}, \mathcal{R}, |m|)$. Upon receiving (Ok.Snd,sid,mid) from $\mathcal{A}$, send (Continue,sid)to $\mathcal{S}$.

– for $\mathcal{F}_{\text{Ch}}^{\text{sa}}$ set $\Delta = (\mathcal{R}, m)$.

3. Upon receiving (Ok,sid,mid) from $\mathcal{A}$, send (Received,sid,m,mid) to $\mathcal{R}$.
   Upon receiving (Ok.Snd,sid,mid) from $\mathcal{A}$, send (Continue,sid) to $\mathcal{S}$.
4. Upon receiving (Send,sid,mid,m') from $\mathcal{R}$, output (Send,sid,$\mathcal{R}$,m',mid) to $\mathcal{A}$.
   Upon receiving (Ok.End,sid,mid) from $\mathcal{A}$, send (Received,sid,$\mathcal{R}$,m')to $\mathcal{S}$.

b). Functionality $\mathcal{F}_{\text{BC}}$

Initialization: Broadcast functionality $\mathcal{F}_{\text{BC}}$ parameterized by the set $\mathbb{M} = \{\text{M}_1, \dots, \text{M}_D\}$ proceeds as follows:

Upon receiving (Broadcast,sid,m) from a party P, send (Broadcasted,sid,P,m) to all entities in the set $\mathbb{M}$ and to $\mathcal{A}$.

c). Functionality $\mathcal{F}_{\text{PROPOSAL}}$

Initialization: Set Proposal := $\perp$and Round := 0。

- Upon receiving the message (startProposal)
  - Select the proposer Proposer $\in$ H through the Round-robin rule, where H is the set of honest validators in V.
    - Initialize the voting power of the validator to its staked funds:
      $$votingPower_i = stake_i, \quad \forall\, i \in \{1, \dots, N\}$$
    - Elect the Proposer in turn according to the Round-robin rule, and update Round := Round + 1.
  - Update votingPower:
    - For validators not selected:
      $$\text{votingPower}_i \leftarrow votingPower_i + \text{stake}_i$$
    - For the selected validator:
      $$\text{votingPower}_i \leftarrow votingPower_i - \sum_{j \neq i} stake_j$$
- (Timeout handling): Upon receiving the message (timeout, T) from the adversary A, if T is valid, set Round = Round + 1 and select a new proposer.

d). Functionality $\mathcal{F}_{\text{VOTE}}$

Initialization: Send the (timeStart, δ) command to $\mathcal{F}_{TIME}$. If a (timeOver) message is received from $\mathcal{F}_{TIME}$ at any stage, vote for the nil block directly.

- Upon receiving the message (Prevote, Proposal) from validator $\text{v}_\text{i} \in \text{V}$,
  - If a Proposal is received, send (v_i, queryState) to $\mathcal{F}_{STATE}$ to obtain the PoLC.
    - Query the PoLC. If v_i is locked on the Proposal from the previous round, sign and broadcast the previous round's block (v_i, $prevote$, Vote(B')).
    - Otherwise, sign and broadcast the current round's block(v_i, $prevote$, Vote(B)).
  - Otherwise, sign and broadcast (v_i, $prevote$, $Vote(nil)$).

- Upon receiving the message (Precommit, Proposal) from validator $v_i \in V$,
  - If more than 2f+1 prevote votes are received,
    - Sign and broadcast $(v_i, precommit, Vote(B))$, send ($v_i$, unlock, B') to $\mathcal{F}_{STATE}$ to unlock the previous round's block, then send (v_i, lock, B) to $\mathcal{F}_{STATE}$ to lock the current block.
  - If more than 2f+1 null prevote votes are received,
    - Sign and broadcast $(v_i, precommit, Vote(nil))$, send ($v_i$, unlock, ALL) to $\mathcal{F}_{STATE}$ to release all locked blocks.
  - Otherwise, do not lock any blocks.

f). Functionality $\mathcal{F}_{\text{COMMIT}}$

Initialization: For $v_i \in V$, set $c_i := 0$, $c_i \in C$, indicating whether the Proposal has been committed. Send the (timeStart,δ) command to $\mathcal{F}_{TIME}$. If a (timeOver) message is received from $\mathcal{F}_{TIME}$ at any stage, send (newRound) to $\mathcal{F}_{TIME}$.

- Upon receiving the message (Commit, Proposal) from validator $v_i \in V$,
  - If more than 2f+1 precommit votes are received,
    - Sign and broadcast $(v_i, commit, Vote(B))$, and collect commit votes from the entire network.
    - If v_i has already broadcast a commit vote for block B and has collected more than 2f+1 commit votes, set $c_i := 1$, send (allowCommit, Proposal) to the validator, and send (newHeight) to $\mathcal{F}_{STATE}$.
    - Otherwise, send (denyCommit, Proposal) to the validator and send (newRound) to $\mathcal{F}_{STATE}$.
  - Otherwise, send (newRound) to $\mathcal{F}_{STATE}$ to start the next round.
- Upon receiving the message (request_status) from any party v_k:
  - Return the set C and indicate whether block B has been completed.

g). Functionality $\mathcal{F}_{\text{STATE}}$

Initialization: Set Height := 0, Round := 0, and PoLC := $\perp$.

- Upon receiving the message (newHeight) from any validator $v_i \in V$
  Update Height := Height + 1 and reset Round to 0.
- Upon receiving the message (newRound) from any validator $v_i \in V$,
  Update Round := Round + 1.
- Upon receiving the message (getProposal, sid, $phase_p$, *) from the proposer,
  Retrieve the Proposals from the configuration file and return them to the caller.
- Upon receiving the message (updateProposal, sid, $phase_p$, Proposals),
  Update the Proposals in the configuration file.
- Upon receiving the message (v_i, lock, B) from v_i,
  Add v_i to the ValidatorSet corresponding to (Height, Round, B) in the PoLC.
- Upon receiving the message (v_i, unlock, B) from v_i,
  Remove v_i from the ValidatorSet corresponding to (Height, Round, B) in the PoLC.
- Upon receiving the message (v_i, unlock, ALL) from v_i,
  Set PoLC := $\perp$.
- Upon receiving the message (v_i, queryState) from v_i,
  Return the PoLC.

h). Functionality $\mathcal{F}_{\text{TIME}}$

Initialization: Set $t_i \in T$, $t_i := \perp$.

- Upon a (GetTime) request is received,

  Return the current t_i to the requester.

- Upon a (ResetTime) request is received,

  Reset t_i to $t_i := \perp$, and return a (timeOK) message to the caller.

- Upon a (timeStart, sid, $phase_p$, δ) request is received,

  Update t_{sid} to $t_{sid} \leftarrow \delta$, return a (timeOK) message to the ideal functionality $\mathcal{F}_{tbft}$, and then start the countdown.

- Upon $t_{sid} \in T, t_{sid} = 0$,

  Send a (timeOver, sid, $phase_p$, δ) message to the corresponding caller.

3. Ideal Functionality $\mathcal{F}_{\text{TBFT}}$

Functionality $\mathcal{F}_{TBFT}^{V,\Delta,\Sigma}[\mathcal{F}_{TIME}, \mathcal{F}_{BC}, \mathcal{F}_{SYNC}]$

Parameters:

- V: Validator Set.
- Δ : Maximum network delay.
- Σ : Maximum delay in delay attack.
- $\mathcal{F}_{TIME}$: Ideal functionality for timing.
- $\mathcal{F}_{BC}$: Ideal functionality for broadcast.
- $\mathcal{F}_{SYNC}$: Ideal functionality for synchronization.

Symbol Explanation:

- δ : actual execution time, initialized by S, default value is Δ.
- σ : actual delay, initialized by S, default value is 0.
- h_p : current height, or consensus instance we are currently executing, initialized to 0.
- $round_p$: current round number, initialized to 0.
- $phase_p \in$ {propose, prevote, precommit, commit} : marks consensus phase within the current round, initialized to propose
- $count_{phase_p}$: record the number of votes cast at each phase, initialized to 0.
- $decision_p[\ ]$: record the final consensus value reached by each node at various heights, initialized to nil.
- $lockedValue_p$: locked value, indicating the currently locked proposal, initialized to nil.
- $lockedRound_p$: locked round, indicating the round of locked value, initialized to -1.
- $validValue_p$: valid value, indicating the currently valid proposal, initialized to nil.
- $validRound_p$: valid round, indicating the round of valid value, initialized to -1.
- $isVote_{COMMIT}$: marks whether the commit phase itself has completed voting, initialized to false.
- $*$: empty parameter.
- B : Threshold for consecutive block production (blocksPerProposer).
- preProposer : Index of the proposer in the previous round, initialized to 0.
- $|V|$: Total number of nodes in the validator set.
- $\text{txID}_{\text{random}}$: Set of transaction IDs determined to be random.
- $\text{count}_{\text{random}}(\text{txID})$: Exclusion vote count for transaction txID.

Upon receiving message $\langle \text{NEWROUND}, h_p, \text{round}_p, * \rangle$ from $\mathcal{S}$, while $\text{phase}_p = \text{propose}$:

1. Send $\langle \text{Sleep,sid,phase}_p \rangle$ to $\mathcal{S}$ and wait for a response of the form $\langle \text{Wake,sid,phase}_p, \delta, \sigma \rangle$.
2. If $(\delta + \sigma > 2\Delta) \vee \sigma > \Sigma$:
   - Return to step 1.

3. Otherwise:
    - Send $\langle \text{timeStart,sid, phase}_p, \sigma \rangle$ to $\mathcal{F}_{TIME}$, and suspend execution.
    - Upon receiving $\langle \text{timeOver,sid, phase}_p, \sigma \rangle$ from $\mathcal{F}_{TIME}$, resume execution.
    - Send $\langle \text{timeStart,sid, phase}_p, \delta \rangle$ to $\mathcal{F}_{TIME}$.
    - Send $\langle \text{CreateProposal,sid, phase}_p \rangle$ to $\mathcal{S}$ and wait for a response of the form $\langle \text{StartProposal,sid, phase}_p \rangle$.
    - Send $\langle \text{StartProposal,sid, phase}_p \rangle$ to $\text{Proposer}(h_p, \text{round}_p)$ and wait for a response of the form $\langle \text{PROPOSAL,sid, phase}_p, v \rangle$.
    - If $\text{Proposer}(h_p, \text{round}_p)$ is corrupted,
        - Send $\langle \text{Input,sid}, h_p, \text{round}_p, v \rangle$ to $\mathcal{S}$.
    - If $\text{valid}(v)$ and no $\langle \text{timeOver,sid, phase}_p, \delta \rangle$ has been received from $\mathcal{F}_{TIME}$:
        - Broadcast $\langle \text{PROPOSAL}, h_p, \text{round}_p, v \rangle$.
    - Otherwise:
        - Return to step 1.
    - Update $\text{phase}_p \leftarrow$ prevote.

Upon receiving message $\langle \text{PROPOSAL}, h_p, \text{round}_p, v \rangle$ from $\text{Proposer}(h_p, \text{round}_p)$, while $\text{phase}_p = $ prevote:

1. Send $\langle \text{Sleep,sid, phase}_p \rangle$ to $\mathcal{S}$ and wait for a response of the form $\langle \text{Wake,sid, phase}_p, \delta, \sigma \rangle$.
2. If $(\delta + \sigma > 2\Delta) \vee \sigma > \Sigma$:
    - Broadcast $\langle \text{PREVOTE}, h_p, \text{round}_p, \text{nil} \rangle$.
3. Otherwise:
    - Send $\langle \text{timeStart,sid, phase}_p, \sigma \rangle$ to $\mathcal{F}_{TIME}$, and suspend execution.
    - Upon receiving $\langle \text{timeOver,sid, phase}_p, \sigma \rangle$ from $\mathcal{F}_{TIME}$, resume execution.
    - Send $\langle \text{timeStart,sid, phase}_p, \delta \rangle$ to $\mathcal{F}_{TIME}$.
    - If $\text{valid}(v) \wedge (\text{lockedRound}_p = -1 \vee \text{lockedValue}_p = v)$ and no $\langle \text{timeOver,sid, phase}_p, \delta \rangle$ has been received from $\mathcal{F}_{TIME}$:
        - Broadcast $\langle \text{Execute}, v.\text{Transactions} \rangle$ and wait for a response $\langle \text{ReadWriteHash}, H_{\text{exec}} \rangle$.
        - If $H_{\text{exec}} \neq v.H_{\text{readWrite}}$:
            - Broadcast $\langle \text{IdentifyRandom}, v.\text{Transactions} \rangle$ and receive $\text{txID}_{\text{random}}$.
            - Broadcast $\langle \text{PREVOTE}, h_p, \text{round}_p, \text{nil}, \text{txID}_{\text{random}} \rangle$.
        - Otherwise:
            - If $\text{valid}(v) \wedge (\text{lockedRound}_p = -1 \vee \text{lockedValue}_p = v)$: Broadcast $\langle \text{PREVOTE}, h_p, \text{round}_p, \text{id}(v) \rangle$.
            - Otherwise: Broadcast $\langle \text{PREVOTE}, h_p, \text{round}_p, \text{nil} \rangle$.
    - Otherwise:
        - Broadcast $\langle \text{PREVOTE}, h_p, \text{round}_p, \text{nil} \rangle$.
4. Send $\langle \text{RoundOK} \rangle$ to $\mathcal{F}_{SYNC}$.
5. Update $\text{phase}_p \leftarrow$ precommit.

Upon receiving message $\langle \text{PREVOTE}, h_p, \text{round}_p, \text{nil}, \text{txID}_{\text{random}} \rangle$ from $\text{Validator}(h_p, \text{round}_p)$, while $\text{phase}_p = $ prevote:

1. For each $\text{txID} \in \text{txID}_{\text{random}}$:
    - Set $\text{count}_{\text{random}}(\text{txID}) \leftarrow \text{count}_{\text{random}}(\text{txID}) + 1$.
2. If there exists a txID such that $\text{count}_{\text{random}}(\text{txID}) \geq f + 1$:
    - Broadcast $\langle \text{RemoveTx,txID} \rangle$ t to remove the transaction from the transaction pool.

- Reset $\text{count}_{\text{random}}(\text{txID}) \leftarrow 0$.

Upon receiving message $\langle \text{PREVOTE}, h_p, \text{validRound}_p, \text{id}(v) \rangle$ from $\text{Validator}(h_p, \text{round}_p)$, while $\text{phase}_p = \text{prevote} \wedge (\text{validRound}_p \geq 0 \wedge \text{validRound}_p < \text{round}_p)$:

1. Set $\text{count}_{\text{prevote}} \leftarrow \text{count}_{\text{prevote}} + 1$.
2. If $\text{valid}(v) \wedge (\text{count}_{\text{prevote}} > 2f + 1) \wedge (\text{lockedRound}_p \leq \text{validRound}_p \vee \text{lockedValue}_p = v)$:
   - Broadcast $\langle \text{PREVOTE}, h_p, \text{round}_p, \text{id}(v) \rangle$.
3. Otherwise:
   - Broadcast $\langle \text{PREVOTE}, h_p, \text{round}_p, \text{nil} \rangle$.

Upon receiving message $\langle \text{PREVOTE}, h_p, \text{round}_p, \text{id}(v) \rangle$ from $\text{Validator}(h_p, \text{round}_p)$, while $\text{phase}_p = \text{precommit}$:

1. Set $\text{count}_{\text{prevote}} \leftarrow \text{count}_{\text{prevote}} + 1$.
2. Send $\langle \text{Sleep}, \text{sid}, \text{phase}_p \rangle$ to $\mathcal{S}$ and wait for a response of the form $\langle \text{Wake}, \text{sid}, \text{phase}_p, \delta, \sigma \rangle$.
3. If $(\delta + \sigma > 2\Delta) \vee \sigma > \Sigma$:
   - Broadcast $\langle \text{PRECOMMIT}, h_p, \text{round}_p, \text{nil} \rangle$.
4. Otherwise:
   - Send $\langle \text{timeStart}, \text{sid}, \text{phase}_p, \sigma \rangle$ to $\mathcal{F}_{TIME}$, and suspend execution.
   - Upon receiving $\langle \text{timeOver}, \text{sid}, \text{phase}_p, \sigma \rangle$ from $\mathcal{F}_{TIME}$, resume execution.
   - Send $\langle \text{timeStart}, \text{sid}, \text{phase}_p, \delta \rangle$ to $\mathcal{F}_{TIME}$.
   - If $\text{valid}(v) \wedge (\text{count}_{\text{prevote}} > 2f + 1)$ and no $\langle \text{timeOver}, \text{sid}, \text{phase}_p, \delta \rangle$ has been received from $\mathcal{F}_{TIME}$:
     - Set $\text{lockedValue}_p \leftarrow v$, $\text{lockedRound}_p \leftarrow \text{round}_p$.
     - Broadcast $\langle \text{PRECOMMIT}, h_p, \text{round}_p, \text{id}(v) \rangle$.
     - Set $\text{validValue}_p \leftarrow v$, $\text{validRound}_p \leftarrow \text{round}_p$.
   - Otherwise:
     - Broadcast $\langle \text{PRECOMMIT}, h_p, \text{round}_p, \text{nil} \rangle$.
5. Update $\text{phase}_p \leftarrow \text{commit}$.

Upon receiving message $\langle \text{PRECOMMIT}, h_p, \text{round}_p, \text{id}(v) \rangle$ from $\text{Validator}(h_p, \text{round}_p)$, while $\text{phase}_p = \text{commit}$:

1. Set $\text{count}_{\text{precommit}} \leftarrow \text{count}_{\text{precommit}} + 1$.
2. Send $\langle \text{Sleep}, \text{sid}, \text{phase}_p \rangle$ to $\mathcal{S}$ and wait for a response of the form $\langle \text{Wake}, \text{sid}, \text{phase}_p, \delta, \sigma \rangle$.
3. If $(\delta + \sigma > 2\Delta) \vee \sigma > \Sigma$:
   - Broadcast $\langle \text{COMMIT}, h_p, \text{round}_p, \text{nil} \rangle$, and set $\text{isVote}_{COMMIT} \leftarrow \text{false}$.
4. Otherwise:
   - Send $\langle \text{timeStart}, \text{sid}, \text{phase}_p, \sigma \rangle$ to $\mathcal{F}_{TIME}$, and suspend execution.
   - Upon receiving $\langle \text{timeOver}, \text{sid}, \text{phase}_p, \sigma \rangle$ from $\mathcal{F}_{TIME}$, resume execution.
   - Send $\langle \text{timeStart}, \text{sid}, \text{phase}_p, \delta \rangle$ to $\mathcal{F}_{TIME}$.
   - If $\text{valid}(v) \wedge (\text{count}_{\text{precommit}} > 2f + 1)$ and no $\langle \text{timeOver}, \text{sid}, \text{phase}_p, \delta \rangle$ has been received from $\mathcal{F}_{TIME}$:
     - Broadcast $\langle \text{COMMIT}, h_p, \text{round}_p, \text{id}(v) \rangle$, and set $\text{isVote}_{COMMIT} \leftarrow \text{true}$.
   - Otherwise:
     - Update $\text{phase}_p \leftarrow \text{propose}$ and $\text{round}_p \leftarrow \text{round}_p + 1$.

Upon receiving message $\langle \text{COMMIT}, h_p, \text{round}_p, \text{id}(v) \rangle$ from

Validator(h_p, round_p), while $\text{phase}_p = \text{commit}$:

1. Set $\text{count}_{\text{commit}} \leftarrow \text{count}_{\text{commit}} + 1$.
2. If $\text{valid}(v) \wedge (\text{count}_{\text{commit}} > 2f + 1) \wedge \text{isVote}_{COMMIT}$:
    - Set $\text{decision}_p[h_p] = v$, and update $\text{isVote}_{\text{COMMIT}} \leftarrow \text{false}$, $h_p \leftarrow h_p + 1$.
3. Send ⟨RoundOK⟩ to $\mathcal{F}_{SYNC}$.
4. Send ⟨RequestRound⟩ to $\mathcal{F}_{SYNC}$, receive its response d_i:
    - If $d_i = 0$:
        - Update $\text{phase}_p \leftarrow$ propose and reset $\text{round}_p, \text{count}_{\text{phase}_p}, \text{lockedRound}_p, \text{lockedValue}_p, \text{validRound}_p, \text{validValue}_p$.
        - Send ⟨NEWROUND, h_p, round_p,∗⟩ to $\mathcal{S}$.
    - Otherwise re-execute this step.

Upon receiving message ⟨∗, h_p,round,∗,∗⟩:

1. Set $\text{count}_{\text{nextround}} \leftarrow \text{count}_{\text{nextround}} + 1$.
2. If $(\text{count}_{\text{nextround}} > f + 1) \wedge \text{round} > \text{round}_p$:
    - Send ⟨NEWROUND, h_p,round,∗⟩ to $\mathcal{S}$.

4. Protocol Description

The Tendermint-BFT protocol ensures consensus among multiple validators on a block and eventually commits the block through a round-based mechanism and voting phases. The protocol supports tolerance of a small number of malicious nodes and relies on message broadcasting, delay handling, and vote collection to achieve consensus.

– **Party Z**:

StartProposal: Initiates consensus by calling $\mathcal{F}_{PROPOSAL}$, selecting and activating a proposer (Proposer).

– **Party Proposer**:

Initialize: Sends the $(timeStart, \delta)$ command to $\mathcal{F}_{TIME}$. Upon receiving the $(timeOver)$ message from $\mathcal{F}_{TIME}$, directly proceeds to execute the RoundOK phase.

Input: Receives and selects a proposal from the $\mathcal{F}_{STATE}$ function. After validating the proposal's block B as valid, uses it as the proposed block.

Propose: Sends proposal information $L(|Proposal|)$ to adversary A, then signs and broadcasts $Proposal$ to the validators.

RoundOK: Calls $\mathcal{F}_{PROPOSAL}$ to update the round, reselect the proposer, and start a new round.

– **Party Validator**:

Initialize: Sends their own proposal to $\mathcal{F}_{STATE}$.

Input: Upon receiving the proposal from the Proposer, validates the proposal's integrity and validity.

Prevote: Based on the received proposal, calls $\mathcal{F}_{VOTE}(Prevote, Proposal)$.

Precommit: Based on the received proposal, calls $\mathcal{F}_{VOTE}(Precommit, Proposal)$. If consensus fails, proceeds to the RoundOK phase.

Commit: Based on the received proposal, calls $\mathcal{F}_{COMMIT}(Commit, Proposal)$. If consensus fails, proceeds to the RoundOK phase.

RoundOK: Calls $\mathcal{F}_{PROPOSAL}$ to update the round, reselect the proposer, and start a new round.

The Protocol π_{TBFT}

$\mathcal{Z}$	Proposer	$\mathcal{F}_{AUTH}$	Validator	$\mathcal{A}$
→	1: Send ⟨timeStart,sid, phase$_p$, δ⟩ to $\mathcal{F}_{TIME}$	→ $\mathcal{F}_{TIME}$		
	2: Send ⟨getProposal,sid, phase$_p$,$*$⟩ to $\mathcal{F}_{STATE}$	→ $\mathcal{F}_{STATE}$		
	3 : Get ⟨proposalReceived,sid, phase$_p$,Prop from $\mathcal{F}_{STATE}$	← $\mathcal{F}_{STATE}$		
	4: Select a Proposal value v from the Proposals.			
	5: Send ⟨Input,sid, h_p, round$_p$, v⟩ to $\mathcal{A}$			→
	6: If valid(v) and get ⟨timeOver,sid, phase$_p$, δ⟩ from $\mathcal{F}_{TIME}$, then call $\mathcal{F}_{PROPOSAL}$⟨NewRound, h_p, round$_p$ 1⟩	← $\mathcal{F}_{TIME}$ → $\mathcal{F}_{PROPOSA}$		
	7: Otherwise: broadcast ⟨PROPOSAL, h_p, round$_p$, v⟩	→ Broadcas		
		$\mathcal{F}_{TIME}$ ←	8: Send ⟨timeStart,sid, phase$_p$, δ⟩ to $\mathcal{F}_{TIME}$	
		Broadcast ←	9: If valid(v), then call $\mathcal{F}_{VOTE}$⟨Prevote,PROPOSAL	↔
		→ Broadcas	10: Send ⟨timeStart,sid, phase$_p$, δ⟩ to $\mathcal{F}_{TIME}$	
		Broadcast ←	11: If valid(v), then call $\mathcal{F}_{VOTE}$⟨Precommit,PROPOS	↔
		→ Broadcas	12: Send ⟨timeStart,sid, phase$_p$, δ⟩ to $\mathcal{F}_{TIME}$	
		Broadcast ←	13: If valid(v), then call $\mathcal{F}_{COMMIT}$⟨Commit,PROPOS	
←	Output ⟨Success,sid, $id(v)$⟩ to $\mathcal{Z}$	$\mathcal{F}_{PROPOSAL}$ ←	14 : If get ⟨allowCommit,PROPOSAL from $\mathcal{F}_{COMMIT}$ Call $\mathcal{F}_{PROPOSAL}$⟨NewRound, $h_p + 1$,round=0⟩	
←	Output ⟨Failure,sid, $\perp$⟩ to $\mathcal{Z}$	$\mathcal{F}_{PROPOSAL}$ ←	15 : Otherwise: Call $\mathcal{F}_{PROPOSAL}$⟨NewRound, h_p, round$_p + 1$⟩	

The Functionality $\mathcal{F}_{PROPOSAL}$

$\mathcal{Z}$ or Validators	$\mathcal{F}_{AUTH}$	$\mathcal{F}_{PROPOSAL}$
1: Send ⟨NewRound, h_p,round⟩ to $\mathcal{F}_{PROPOSAL}$	→	
	$\mathcal{F}_{STATE}$ ←	2: Send ⟨getProposal,sid, phase$_p$,$*$⟩ to $\mathcal{F}_{STATE}$
	$\mathcal{F}_{STATE}$ →	3 : Get ⟨proposalReceived,sid, phase$_p$,Proposals⟩ from $\mathcal{F}_{STATE}$
		4 : Each validator i initializes their $votingPower$: $votingPower_i = stake_i \quad \forall i \in \{1, \dots, N\}$
		5 : The validator v^* is chosen as the proposer: $v^* = \text{argmax}_{v_i \in H} stake_i$
	$\mathcal{F}_{STATE}$ ←	6 : Send ⟨newRound⟩ to $\mathcal{F}_{STATE}$
		7 : For unselected validators $v_i \neq v^*$: $votingPower_i \leftarrow votingPower_i + stake_i$
		8 : For the selected proposer v^*: $votingPower_{v^*} \leftarrow votingPower_{v^*} - \sum_{i=1}^{N} stake_i$
		9 : Update Proposals based on $votingPower$
	$\mathcal{F}_{STATE}$ ←	10 : Send ⟨updateProposal,sid, phase$_p$,Proposals⟩ to $\mathcal{F}_{STATE}$

The Functionality $\mathcal{F}_{VOTE}$

	Validator	$\mathcal{F}_{AUTH}$	$\mathcal{F}_{VOTE}$
Prevote	1: Send ⟨Prevote,PROPOSAL⟩ to $\mathcal{F}_{VOTE}$	→	
		$\mathcal{F}_{TIME}$ ←	2 : Send ⟨timeStart,sid, phase$_p$, δ⟩ to $\mathcal{F}_{TIME}$
		Broadcast ←	3 : If no ⟨timeOver,sid, phase$_p$, δ⟩ is received from $\mathcal{F}_{TIME}$: Sign and broadcast ⟨v_i,prevote,Vote(nil)⟩
		$\mathcal{F}_{STATE}$ ←	4 : Otherwise, If valid(v) : Send ⟨v_i,queryState⟩ to $\mathcal{F}_{STATE}$ to get PoLC

	Validator	$\mathcal{F}_{AUTH}$	$\mathcal{F}_{VOTE}$
		Broadcast ←	5 : If v_i is locked on Proposal from the previous round: Sign and broadcast $\langle v_i$,prevote,Vote$(v_i')\rangle$
		Broadcast ←	6 : Otherwise: Sign and broadcast $\langle v_i$,prevote,Vote$(v_i)\rangle$
		Broadcast ←	7 : If no valid(v) : Sign and broadcast $\langle v_i$,prevote,Vote(nil)$\rangle$
Precommit	1: Send ⟨Precommit,PROPOSAL⟩ to $\mathcal{F}_{VOTE}$	→	
		$\mathcal{F}_{TIME}$ ←	2 : Send ⟨timeStart,sid, phase$_p$, $\delta\rangle$ to $\mathcal{F}_{TIME}$
		Broadcast ←	3 : If no ⟨timeOver,sid, phase$_p$, $\delta\rangle$ is received from $\mathcal{F}_{TIME}$: Sign and broadcast $\langle v_i$,precommit,Vote(nil)$\rangle$
		Broadcast ←	4 : Otherwise, If valid(v) : Upon receiving more than $2f+1$ prevote votes: Sign and broadcast $\langle v_i$,precommit,Vote$(v_i)\rangle$
		$\mathcal{F}_{STATE}$ ←	5 : Send $\langle v_i$,unlock, $v_i'\rangle$ to $\mathcal{F}_{STATE}$
		$\mathcal{F}_{STATE}$ ←	6 : Send $\langle v_i$,lock, $v_i\rangle$ to $\mathcal{F}_{STATE}$
		Broadcast ←	7 : Upon receiving more than $2f+1$ null prevote votes: Sign and broadcast $\langle v_i$,precommit,Vote(nil)$\rangle$
		$\mathcal{F}_{STATE}$ ←	8 : Send $\langle v_i$,unlock,ALL$\rangle$ to $\mathcal{F}_{STATE}$
			9 : If no valid(v) , Do not lock any blocks.

The Functionality $\mathcal{F}_{COMMIT}$

	Validator	$\mathcal{F}_{AUTH}$	$\mathcal{F}_{COMMIT}$
Commit	1: Send ⟨Commit,PROPOSAL⟩ to $\mathcal{F}_{COMMIT}$	→	

Validator	$\mathcal{F}_{AUTH}$	$\mathcal{F}_{COMMIT}$
	$\mathcal{F}_{TIME}$ ←	2 : Send ⟨timeStart,sid, phase_p, δ⟩ to $\mathcal{F}_{TIME}$
	$\mathcal{F}_{PROPOSAL}$ ←	3 : If no ⟨timeOver,sid, phase_p, δ⟩ is received from $\mathcal{F}_{TIME}$: Send ⟨NewRound, h_p, $\text{round}_p + 1$⟩ to $\mathcal{F}_{PROPOSAL}$
	Broadcast ←	4 : Otherwise, If valid(v) : If more than $2f + 1$ precommit votes are received: Sign and broadcast ⟨v_i,commit,Vote(v_i)⟩
	→	5 : Collect commit votes from the network.
	←	6 : If node has already broadcast a commit vote for v_i and collected more than $2f + 1$ commit votes: Send ⟨allowCommit,PROPOSAL⟩ to Validator
	$\mathcal{F}_{STATE}$ ←	7 : Send ⟨newHeight⟩ to $\mathcal{F}_{STATE}$
	←	8 : Otherwise: Send ⟨denyCommit,PROPOSAL⟩ to Validator
	$\mathcal{F}_{PROPOSAL}$ ←	9 : Send ⟨NewRound, h_p, $\text{round}_p + 1$⟩ to $\mathcal{F}_{PROPOSAL}$

The Functionality $\mathcal{F}_{STATE}$

	Proposer	$\mathcal{F}_{AUTH}$	$\mathcal{F}_{STATE}$
Get Proposals	1: Send ⟨getProposal,sid, phase_p,∗⟩ to $\mathcal{F}_{STATE}$	→	
			2: Get Proposals from Configuration File
		←	3: Send ⟨proposalReceived,sid, phase_p,Prc to Proposer
	$\mathcal{F}_{PROPOSAL}$		
Set Proposals	1 : Send ⟨updateProposal,sid, phase_p,Pro to $\mathcal{F}_{STATE}$	→	

	Proposer	$\mathcal{F}_{AUTH}$	$\mathcal{F}_{STATE}$
			2: Update Proposals to Configuration File
GetPoLC	$\mathcal{F}_{VOTE}$ 1 : Send $\langle v_i$,queryState$\rangle$ to $\mathcal{F}_{STATE}$	→	
			2 : Return PoLC to the caller
UnLock	$\mathcal{F}_{VOTE}$ 1 : Send $\langle v_i$,unlock, $v_i'\rangle$ to $\mathcal{F}_{STATE}$	→	
			2 : Delete v_i from the ValidatorSet corresponding to $\langle h_p$,round, $v_i\rangle$ in PoLC
	1 : Send $\langle v_i$,unlock,ALL$\rangle$ to $\mathcal{F}_{STATE}$	→	
			2 : Set PoLC := $\perp$
Lock	1 : Send $\langle v_i$,lock, $v\rangle$ to $\mathcal{F}_{STATE}$	→	
			2 : Add v_i to the ValidatorSet corresponding to $\langle h_p$,round, $v_i\rangle$ in PoLC.
NewHeight	$\mathcal{F}_{COMMIT}$ 1 : Send ⟨newHeight⟩ to $\mathcal{F}_{STATE}$	→	
			2 : Set $h_p := h_p + 1$, $round_p := 0$
NewRound	$\mathcal{F}_{PROPOSAL}$ 1 : Send ⟨newRound⟩ to $\mathcal{F}_{STATE}$	→	
			2 : Set $round_p := round_p + 1$

The Functionality $\mathcal{F}_{TIME}$

	Proposer or Validator	$\mathcal{F}_{AUTH}$	$\mathcal{F}_{TIME}$
Countdown	1: Send ⟨timeStart,sid, phase_p, $\delta\rangle$ to $\mathcal{F}_{TIME}$	→	
		←	2: Set $t_{sid} \leftarrow \delta$, start the countdown. Return ⟨timeOK⟩
		←	3 : When $t_{sid} \in T$ and $t_{sid} = 0$: Send ⟨timeOver,sid, phase_p, $\delta\rangle$ to the caller
ResetTime	1: Send ⟨ResetTime,sid, phase_p, $\delta\rangle$ to $\mathcal{F}_{TIME}$	→	
		←	2 : Set $t_i \leftarrow \perp$, stop the countdown Return ⟨timeOK⟩

	Proposer or Validator	$\mathcal{F}_{AUTH}$	$\mathcal{F}_{TIME}$
GetTime	1: Send $\langle \text{getTime,sid,phase}_p, \delta \rangle$ to $\mathcal{F}_{TIME}$	→	
		←	2 : Return t_i to the caller