TBFT 建模进度

摘要

版本 7：本文完善了 TBFT 共识机制的建模框架。1、完善了广播理想功能 $\mathcal{F}_{\mathrm{BC}}$，确保了信息在指定集合中的可靠传递以及对手的可见性。2、完善$\mathcal{F}_{TBFT}$功能描述，加入了随机交易剔除功能，使得系统在处理交易时更加灵活和安全。3、细化协议描述π_{TBFT}，加入了对各子功能的描述，为协议的实现提供了更加明确的指导。4、协议实现的 UC 建模完成度比例为 80%左右。

一、 整体框架

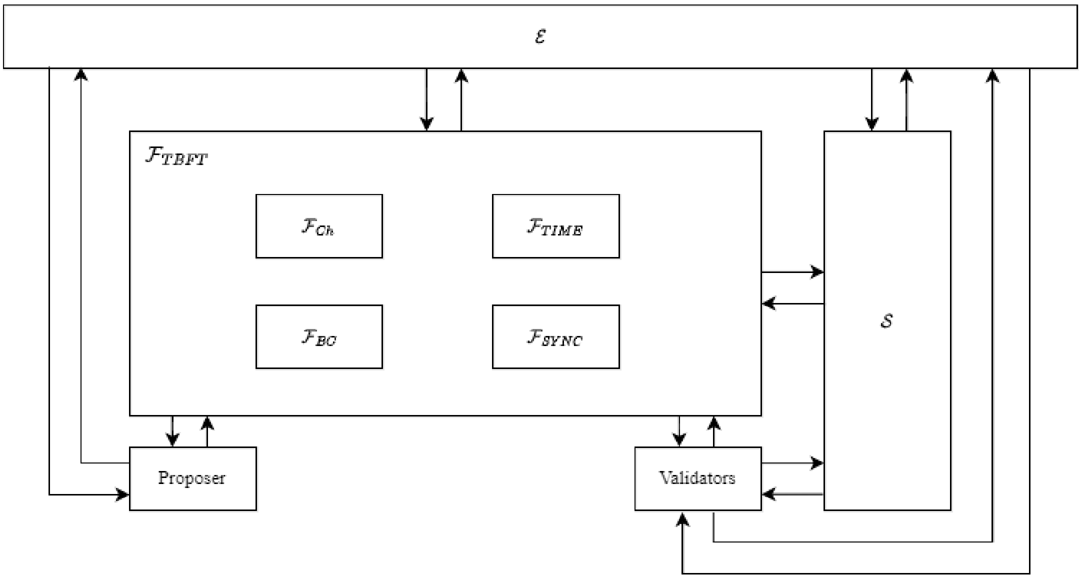

图 1 TBFT 协议整体框架

二、功能描述

（一）功能$\mathcal{F}_{Ch}$

初始化：定义一组参与方，其中 $\mathcal{S}$ 和 $\mathcal{R}$ 分别表示该组中的两个参与方，作为消息 m 的发送方和接收方。

Δ 根据功能参数定义如下。消息标识符 mid 由功能随机选择。

1. 当从 $\mathcal{S}$ 接收到输入（Send,sid,$\mathcal{R},m$）时，向 $\mathcal{A}$ 输出（Send,sid,Δ,mid）。
2. 当从 $\mathcal{A}$ 接收到（Ok,sid,mid）时，向 $\mathcal{R}$ 发送（Received,sid,$\mathcal{S},m$）。

根据以下参数化函数设置 Δ：

– 对于 $\mathcal{F}_{\mathrm{Ch}}^{\mathrm{ac}}$，设置 $\Delta=(\mathcal{S},\ \mathcal{R},m)$。当从 $\mathcal{A}$ 接收到（Ok.Snd,sid,mid）时，向 $\mathcal{S}^{a}$ 发送(Continue,sid)。

– 对于 $\mathcal{F}_{\mathrm{Ch}}^{\mathrm{sra}}$，设置 $\Delta=(\mathcal{S},|m|)$。

– 对于 $\mathcal{F}_{\mathrm{Ch}}^{\mathrm{ssa}}$，设置 $\Delta=(\mathcal{R},|m|)$。

– 对于 $\mathcal{F}_{\mathrm{Ch}}^{\mathrm{fa}}$，设置 $\Delta=|m|$。

– 对于 $\mathcal{F}_{\mathrm{Ch}}^{\mathrm{sc}}$，设置 $\Delta=(\mathcal{S},\mathcal{R},|m|)$。当从 $\mathcal{A}$ 接收到（Ok.Snd,sid,mid）时，向 $\mathcal{S}$ 发送(Continue,sid)。

– 对于 $\mathcal{F}_{\mathrm{Ch}}^{\mathrm{sa}}$，设置 $\Delta=(\mathcal{R},m)$。

1. 当从 $\mathcal{A}$ 接收到 (Ok,sid,mid) 时，向 $\mathcal{R}$ 发送 (Received,sid,m,mid)。当从 $\mathcal{A}$ 接收到 (Ok.Snd,sid,mid) 时，向 $\mathcal{S}$ 发送 (Continue,sid)。
2. 当从 $\mathcal{R}$ 接收到 (Send,sid,mid,m') 时，向 $\mathcal{A}$ 输出 (Send,sid,$\mathcal{R}$,m',mid)。当从 $\mathcal{A}$ 接收到 (Ok.End,sid,mid) 时，向 $\mathcal{S}$ 发送 (Received,sid,$\mathcal{R}$,m')。

[a] 这赋予了对手 $\mathcal{A}$ 更多的权力，因为 UC 模型中需要顺序发送消息，$\mathcal{A}$ 决定发送方何时可以继续。

（二）功能$\mathcal{F}_{PROPOSAL}$

初始化：设置 Proposal := ⊥和 Round := 0。

－当收到消息(startProposal)时，

- 通过 Round-robin 规则选定提议者 Proposer ∈ H，H 为 V 中诚实者的集合，
  - 初始化 Validator 的 votingPower 为其质押资金：

$$\text{votingPower}_i = \text{stake}_i, \quad \forall\, i \in \{1, \ldots, \text{N}\}$$

  - 按 Round-robin 规则依次选举 Proposer，更新 Round := Round+1。
- 更新 votingPower：
  - 未被选中的 Validator 更新为：

$$\text{votingPower}_i \leftarrow v\text{otingPower}_i + \text{stake}_i$$

  - 被选中的 Validator 更新为：

$$\text{votingPower}_i \leftarrow v\text{otingPower}_i - \sum_{j\neq i} stake_j$$

－（超时处理）：当从敌手 A 接收到(timeout, T)消息时，如果 T 有效，设置 Round = Round + 1，并选择新的提议者。

（三）功能$\mathcal{F}_{VOTE}$

初始化：向$\mathcal{F}_{TIME}$发送(timeStart,δ)命令。若在任何阶段从$\mathcal{F}_{TIME}$收到(timeOver)消息，直接投票给 nil 块。

－当从验证者v_i ∈ V传入(Prevote, Proposal)消息时，

- 若收到 Proposal，则向$\mathcal{F}_{STATE}$发送(v_i, queryState)，获取 PoLC。
  - 查询 PoLC，若v_i锁定在上一轮 Proposal，则签名并广播上一轮区块(v_i, $prevote$, Vote(B'))。
  - 否则，签名并广播当前轮区块(v_i, $prevote$, Vote(B))。
- 否则，则签名并广播(v_i, $prevote$, $Vote(nil)$)。

－当从验证者v_i ∈ V传入(Precommit, Proposal)消息时，

- 若收到超过 2f+1 的 prevote 投票，
  - 签名并广播 (v_i, $precommit$, $Vote(B)$)，向$\mathcal{F}_{STATE}$发送(v_i, unlock, B')解锁上一轮区块，然后向$\mathcal{F}_{STATE}$发送(v_i, lock, B)锁定当前区块。
- 若收到超过 2f+1 的空 prevote 投票，
  - 签名并广播 (v_i, $precommit$, $Vote(nil)$)，向$\mathcal{F}_{STATE}$发送(v_i, unlock, ALL)释放所有锁定的区块。
- 否则，不锁定任何区块。

（四）功能$\mathcal{F}_{COMMIT}$

初始化：对于$v_i \in V$，设置$c_i \coloneqq 0$，$c_i \in C$。表示 Proposal 是否已 Commit。向$\mathcal{F}_{TIME}$发送(timeStart,δ)命令。若在任何阶段从$\mathcal{F}_{TIME}$收到(timeOver)消息，向$\mathcal{F}_{STATE}$发送(newRound)。

－当收到从验证者$v_i \in V$传入(Commit,Proposal)消息时，

- 若收到超过 2f+1 的 precommit 投票，
  - 签名并广播(v_i, $commit$, $Vote(B)$)，同时收集全网的 commit 投票。
  - 若v_i已为区块 B 广播 commit 投票且收集到超过 2f+1 的 commit 投票，则设置

$c_i \coloneqq 1$，向验证者发送(allowCommit,Proposal)消息，向$\mathcal{F}_{STATE}$发送(newHeight)。

■ 否则，向验证者发送(denyCommit,Proposal)消息，向$\mathcal{F}_{STATE}$发送(newRound)。

● 否则，向$\mathcal{F}_{STATE}$发送(newRound)，开启下一轮。

－收到来自任意方v_k的消息(request_status)时：

● 返回集合 C 并指示区块 B 是否已完成。

（五）功能$\mathcal{F}_{STATE}$

初始化：设置 Height := 0，Round := 0 和 PoLC := ⊥。

－当从任意验证者$v_i \in V$接收到(newHeight)消息时，

更新 Height := Height+1 并将 Round 重置为 0。

－当从任意验证者$v_i \in V$接收到(newRound)消息时，

更新 Round := Round +1。

－当从出块人 Proposer 接收到(getProposal, sid, $phase_p$，$*$)消息时，

从配置文件中获取 Proposals，然后将其返回给调用者。

－当从接收到 (updateProposal, sid, $phase_p$, Proposals)消息时，

将 Proposals 更新到配置文件中。

－当从v_i接收到(v_i,lock,B)消息时，

将v_i加入到 PoLC 中 (Height,Round,B)对应的 ValidatorSet 中。

－当从v_i接收到(v_i,unlock,B)消息时，

将v_i在对应的 PoLC 中 (Height,Round,B)的 ValidatorSet 中删除。

－当从v_i接收到(v_i,unlock,ALL)消息时，设置 PoLC := ⊥。

－当从v_i接收到(v_i,queryState)消息时，返回 PoLC。

（六）功能$\mathcal{F}_{TIME}$

初始化：设置$t_i \in T$，t_i := ⊥。

－当收到(GetTime)请求时，将当前的t_i返回给请求方。

－当收到(ResetTime)请求时，

将t_i重置为 $t_i \coloneqq \perp$，向调用者返回一个(timeOK)消息。

－当收到(timeStart, sid, $phase_p$，δ)请求时，

将t_{sid}更新为$t_{sid} \leftarrow$ δ，向理想功能$\mathcal{F}_{tbft}$返回一个(timeOK)消息，然后开始倒计时。

－当从$t_{sid} \in T, t_{sid} = 0$时，

会向对应的调用者发送一个(timeOver, sid, $phase_p$，δ)消息。

（七）功能 $\mathcal{F}_{\mathrm{BC}}$

初始化：由集合 $\mathbb{M} = \{\mathrm{M}_1, \ldots, \mathrm{M}_D\}$ 参数化，其执行过程如下：

－当从参与方 P 接收到 (Broadcast,sid,m) 时，向集合 $\mathbb{M}$ 中的所有实体以及 $\mathcal{A}$ 发送(Broadcasted,sid,P,m)。

三、 理想功能$\mathcal{F}_{\mathrm{TBFT}}$

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 propos
- $count_{phase_p}$: 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.
- txID$_{\text{random}}$: Set of transaction IDs determined to be random.
- count$_{\text{random}}$(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 $\langle \text{Sleep,sid}, \text{phase}_p \rangle$ to $\mathcal{S}$ and wait for a response of the form $\langle \text{Wake,sid}, \text{phase}_p, \delta, \sigma \rangle$.
2. If $(\delta + \sigma > 2\Delta) \vee \sigma > \Sigma$:
    - Return to step 1.
3. Otherwise:
    - Send $\langle \text{timeStart,sid}, \text{phase}_p, \sigma \rangle$ to $\mathcal{F}_{TIME}$, and suspend execution.
    - Upon receiving $\langle \text{timeOver,sid}, \text{phase}_p, \sigma \rangle$ from $\mathcal{F}_{TIME}$, resume execution.
    - Send $\langle \text{timeStart,sid}, \text{phase}_p, \delta \rangle$ to $\mathcal{F}_{TIME}$.
    - Send $\langle \text{CreateProposal,sid}, \text{phase}_p \rangle$ to $\mathcal{S}$ and wait for a response of the form $\langle \text{StartProposal,sid}, \text{phase}_p \rangle$.
    - Send $\langle \text{StartProposal,sid}, \text{phase}_p \rangle$ to $\text{Proposer}(h_p, \text{round}_p)$ and wait for a response of the form $\langle \text{PROPOSAL,sid}, \text{phase}_p, v \rangle$.
    - 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 $\langle \text{timeOver,sid}, \text{phase}_p, \delta \rangle$ has been received from $\mathcal{F}_{TIME}$:
        - Broadcast $\langle \text{PROPOSAL}, h_p, \text{round}_p, v \rangle$.
    - Otherwise:
        - Return to step 1.
    - Update $\text{phase}_p \leftarrow \text{prevote}$.

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

1. Send $\langle \text{Sleep,sid}, \text{phase}_p \rangle$ to $\mathcal{S}$ and wait for a response of the form $\langle \text{Wake,sid}, \text{phase}_p, \delta, \sigma \rangle$.
2. If $(\delta + \sigma > 2\Delta) \vee \sigma > \Sigma$:
    - Broadcast $\langle \text{PREVOTE}, h_p, \text{round}_p, \text{nil} \rangle$.
3. Otherwise:
    - Send $\langle \text{timeStart,sid}, \text{phase}_p, \sigma \rangle$ to $\mathcal{F}_{TIME}$, and suspend execution.
    - Upon receiving $\langle \text{timeOver,sid}, \text{phase}_p, \sigma \rangle$ from $\mathcal{F}_{TIME}$, resume execution.

- Send $\langle \text{timeStart}, \text{sid}, \text{phase}_p, \delta \rangle$ to $\mathcal{F}_{TIME}$.
- If $\text{valid}(v) \wedge (\text{lockedRound}_p = -1 \vee \text{lockedValue}_p = v)$ and no $\langle \text{timeOver}, \text{sid}, \text{phase}_p, \delta \rangle$ 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 $\langle \textbf{PREVOTE}, h_p, \textbf{round}_p, \textbf{nil}, \textbf{txID}_{\textbf{random}} \rangle$ from $\textbf{Validator}(h_p, \textbf{round}_p)$, while $\textbf{phase}_p = \textbf{prevote}$:

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 $\langle \text{RemoveTx}, \text{txID} \rangle$ t to remove the transaction from the transaction pool.
   - Reset $\text{count}_{\text{random}}(\text{txID}) \leftarrow 0$.

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

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

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 $\text{validValue}_p \leftarrow v$, $\text{validRound}_p \leftarrow \text{round}_p$.
        o Otherwise:
            ▪ Broadcast $\langle \text{PRECOMMIT}, h_p, \text{round}_p, \text{nil} \rangle$.
5. Update $\text{phase}_p \leftarrow \text{commit}$.
```

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

```
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 $(\delta + \sigma > 2\Delta) \vee \sigma > \Sigma$:
    o Broadcast $\langle \text{COMMIT}, h_p, \text{round}_p, \text{nil} \rangle$, and set $\text{isVote}_{COMMIT} \leftarrow \text{false}$.
4. Otherwise:
    o Send $\langle \text{timeStart}, \text{sid}, \text{phase}_p, \sigma \rangle$ to $\mathcal{F}_{TIME}$, and suspend execution.
    o Upon receiving $\langle \text{timeOver}, \text{sid}, \text{phase}_p, \sigma \rangle$ from $\mathcal{F}_{TIME}$, resume execution.
    o Send $\langle \text{timeStart}, \text{sid}, \text{phase}_p, \delta \rangle$ to $\mathcal{F}_{TIME}$.
    o If $\text{valid}(v) \wedge \left(\text{count}_{\text{precommit}} > 2f + 1\right)$ 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 $\text{isVote}_{COMMIT} \leftarrow \text{true}$.
    o Otherwise:
        ▪ Update $\text{phase}_p \leftarrow \text{propose}$ and $\text{round}_p \leftarrow \text{round}_p + 1$.
```

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

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

Upon receiving message $\langle *, h_p, \textbf{round}, *, * \rangle$:

```
1. Set $\text{count}_{\text{nextround}} \leftarrow \text{count}_{\text{nextround}} + 1$.
2. If $(\text{count}_{\text{nextround}} > f + 1) \wedge \text{round} > \text{round}_p$:
    o Send $\langle \text{NEWROUND}, h_p, \text{round}, * \rangle$ to $\mathcal{S}$.
```

四、协议描述

Tendermint-BFT 协议通过轮次机制和投票阶段确保多个验证者之间就区块达成一致，并最终提交区块。该协议支持容忍少量恶意节点，依赖于消息广播、延迟处理和投票收集来实现共识。

－Party Z:

StartProposal：开始共识，调用$\mathcal{F}_{PROPOSAL}$，选择并激活一个提议者 Proposer。

－Party Proposer:

Initialize: 向$\mathcal{F}_{TIME}$发送(timeStart,δ)命令。若从$\mathcal{F}_{TIME}$收到(timeOver)消息，则直接跳转执

行 RoundOK 部分。

Input: 从功能$\mathcal{F}_{STATE}$中接收并选择一个提案，确定其区块B有效后将其作为提议区块。

Propose: 将提议信息 L(|Proposal|)发送给敌手 A，然后签名并广播(Proposal)给验证者。

RoundOK: 调用$\mathcal{F}_{PROPOSAL}$更新轮次，重新选择提议者，开始新的轮次。

- Party Validator:

Initialize: 向$\mathcal{F}_{STATE}$发送自己的提案。

Input: 在收到来自 Proposer 的 Proposal 后，验证 Proposal 的完整性和有效性。

Prevote: 根据收到 Proposal 的，调用$\mathcal{F}_{VOTE}(Prevote,\ Proposal)$。

Precommit: 根据收到的 Proposal，调用$\mathcal{F}_{VOTE}(Precommit,\ Proposal)$。若共识失败跳转执行 RoundOK 部分。

Commit: 根据收到的 Proposal，调用$\mathcal{F}_{COMMIT}(Commit,\ Proposal)$。若共识失败跳转执行 RoundOK 部分。

RoundOK: 调用$\mathcal{F}_{PROPOSAL}$更新轮次，重新选择提议者，开始新的轮次。

The Protocol π_{TBFT}

$\mathcal{Z}$	Proposer	$\mathcal{F}_{AUTH}$	Validator	$\mathcal{A}$
→	1: Send $\langle$timeStart,sid, $\text{phase}_p, \delta\rangle$ to $\mathcal{F}_{TIME}$	→ $\mathcal{F}_{TIME}$		
	2: Send $\langle$getProposal,sid, phase_p,$*\rangle$ to $\mathcal{F}_{STATE}$	→ $\mathcal{F}_{STATE}$		
	3 : Get $\langle$proposalReceived,sid, phase_p,Prop from $\mathcal{F}_{STATE}$	← $\mathcal{F}_{STATE}$		
	4: Select a Proposal value v from the Proposals.			
	5: Send $\langle$Input,sid, h_p, $\text{round}_p, v\rangle$ to $\mathcal{A}$			→
	6: If valid(v) and get $\langle$timeOver,sid, $\text{phase}_p, \delta\rangle$ from $\mathcal{F}_{TIME}$, then call $\mathcal{F}_{PROPOSAL}\langle$NewRound, h_p, round_p 1$\rangle$	← $\mathcal{F}_{TIME}$ → $\mathcal{F}_{PROPOSA}$		
	7: Otherwise: broadcast $\langle$PROPOSAL, h_p, $\text{round}_p, v\rangle$	→ Broadcas		
		$\mathcal{F}_{TIME}$ ←	8: Send $\langle$timeStart,sid, $\text{phase}_p, \delta\rangle$ to $\mathcal{F}_{TIME}$	
		Broadcast ←	9: If valid(v), then call $\mathcal{F}_{VOTE}\langle$Prevote,PROPOSAL	↔
		→ Broadcas	10: Send $\langle$timeStart,sid, $\text{phase}_p, \delta\rangle$ to $\mathcal{F}_{TIME}$	
		Broadcast ←	11: If valid(v), then call $\mathcal{F}_{VOTE}\langle$Precommit,PROPOS	↔

$\mathcal{Z}$	Proposer	$\mathcal{F}_{AUTH}$	Validator	$\mathcal{A}$
		→ 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, ⊥⟩ to $\mathcal{Z}$	$\mathcal{F}_{PROPOSAL}$ ←	15 : Otherwise: Call $\mathcal{F}_{PROPOSAL}$⟨NewRound, h_p, $\text{round}_p + 1$⟩	

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^*: $votingPower_{v^*} \leftarrow votingPower_{v^*} - \sum_{i=1}^{N} s\ take_i$
		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
		Broadcast ←	5 : If v_i is locked on Proposal from the previous round: Sign and broadcast ⟨v_i,prevote,Vote($v_i{'}$)⟩
		Broadcast ←	6 : Otherwise: Sign and broadcast ⟨v_i,prevote,Vote(v_i)⟩
		Broadcast ←	7 : If no valid(v) : Sign and broadcast ⟨v_i,prevote,Vote(nil)⟩
Precommit	1: Send ⟨Precommit,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,precommit,Vote(nil)⟩
		Broadcast ←	4 : Otherwise, If valid(v) : Upon receiving more than $2f+1$ prevote votes: Sign and broadcast ⟨v_i,precommit,Vote(v_i)⟩
		$\mathcal{F}_{STATE}$ ←	5 : Send ⟨v_i,unlock, $v_i{'}$⟩ to $\mathcal{F}_{STATE}$

	Validator	$\mathcal{F}_{AUTH}$	$\mathcal{F}_{VOTE}$
		$\mathcal{F}_{STATE}$ ←	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 $\langle$Commit,PROPOSAL$\rangle$ to $\mathcal{F}_{COMMIT}$	→	
		$\mathcal{F}_{TIME}$ ←	2 : Send $\langle$timeStart,sid, phase$_p, \delta\rangle$ to $\mathcal{F}_{TIME}$
		$\mathcal{F}_{PROPOSAL}$ ←	3 : If no $\langle$timeOver,sid, phase$_p, \delta\rangle$ is received from $\mathcal{F}_{TIME}$: Send $\langle$NewRound, h_p, round$_p + 1\rangle$ to $\mathcal{F}_{PROPOSAL}$
		Broadcast ←	4 : Otherwise, If valid(v) : If more than $2f+1$ precommit votes are received: Sign and broadcast $\langle v_i$,commit,Vote$(v_i)\rangle$
		→	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 $\langle$allowCommit,PROPOSAL$\rangle$ to Validator
		$\mathcal{F}_{STATE}$ ←	7 : Send $\langle$newHeight$\rangle$ to $\mathcal{F}_{STATE}$
		←	8 : Otherwise: Send $\langle$denyCommit,PROPOSAL$\rangle$ to Validator
		$\mathcal{F}_{PROPOSAL}$ ←	9 : Send $\langle$NewRound, h_p, round$_p + 1\rangle$ 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 $\mathcal{F}_{STATE}$	→	
			2: Update Proposals to Configuration File
	$\mathcal{F}_{VOTE}$		
GetPoLC	1 : Send ⟨v_i,queryState⟩ to $\mathcal{F}_{STATE}$	→	
			2 : Return PoLC to the caller
	$\mathcal{F}_{VOTE}$		
UnLock	1 : Send ⟨v_i,unlock, $v_i{}'$⟩ to $\mathcal{F}_{STATE}$	→	
			2 : Delete v_i from the ValidatorSet corresponding to ⟨h_p,round, v_i⟩ in PoLC
	1 : Send ⟨v_i,unlock,ALL⟩ to $\mathcal{F}_{STATE}$	→	
			2 : Set PoLC := ⊥
Lock	1 : Send ⟨v_i,lock, v⟩ to $\mathcal{F}_{STATE}$	→	
			2 : Add v_i to the ValidatorSet corresponding to ⟨h_p,round, v_i⟩ in PoLC.
	$\mathcal{F}_{COMMIT}$		
NewHeight	1 : Send ⟨newHeight⟩ to $\mathcal{F}_{STATE}$	→	
			2 : Set $h_p := h_p + 1$, $round_p := 0$
	$\mathcal{F}_{PROPOSAL}$		
NewRound	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$_p$, δ⟩ to $\mathcal{F}_{TIME}$	→	

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