

TBFT 建模进度

摘要

版本 5: 本文档调整了 TBFT 共识机制的建模, 具体包括调整框架图, 完善 \mathcal{F}_{COMMIT} 功能, 完善 \mathcal{F}_{BC} 功能, 修改 \mathcal{F}_{TBFT} 功能, 细化提案处理与状态更新流程, 引入锁定机制, 优化拉回落后节点的机制。

一、初步框架

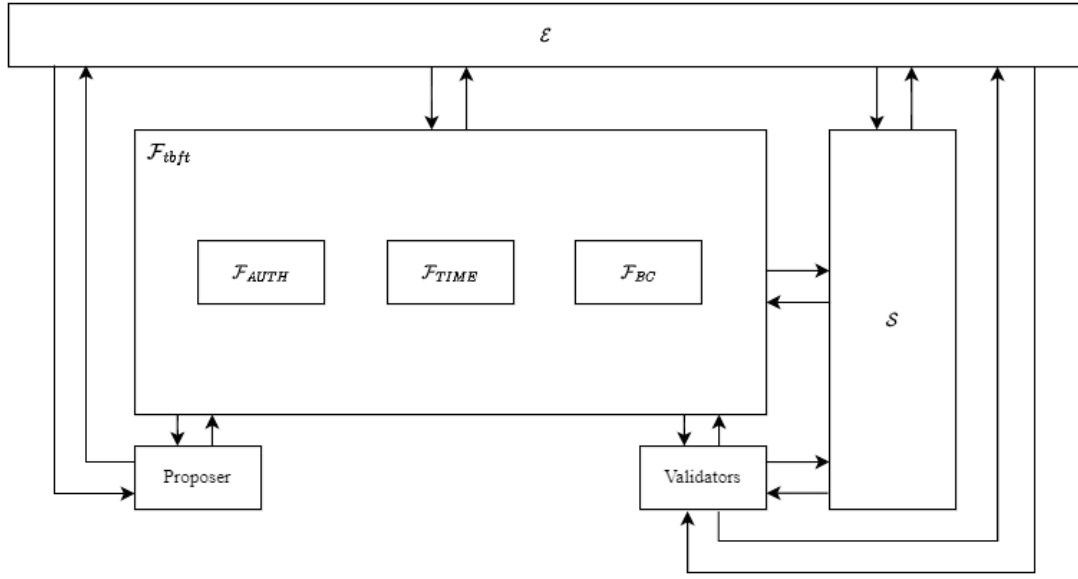


图 1 TBFT 协议初步框架

二、功能描述

（一）功能 \mathcal{F}_{AUTH}

- 当从参与方 A 收到(Send,sid, B, m)时,
将(Sent, sid, A, B, m)发送给 A。
- 当从 A 收到(Send, sid, B', m')时, 执行以下操作:
如果 A 是被破坏的, 则向参与方 B'输出(Sent, sid, A, m')。
否则, 向参与方 B 输出(Sent, sid, A, m)。终止操作。

（二）功能 $\mathcal{F}_{PROPOSAL}$

初始化: 设置 Proposal := \perp 和 Round := 0。

- 当收到消息(startProposal)时,
 - 通过 Round-robin 规则选定提议者 Proposer $\in H$, H 为 V 中诚实者的集合,
 - 初始化 Validator 的 votingPower 为其质押资金:
$$\text{votingPower}_i = \text{stake}_i, \quad \forall i \in \{1, \dots, N\}$$
 - 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:
$$\text{votingPower}_i \leftarrow \text{votingPower}_i + \text{stake}_i$$
 - 被选中的 Validator 更新为:

$$\text{votingPower}_i \leftarrow \text{votingPower}_i - \sum_{j \neq i} \text{stake}_j$$

- (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 $\mathcal{F}_{\text{VOTE}}$

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

- 当从验证者 $v_i \in V$ 传入(Prevote, Proposal)消息时，
 - 若收到 Proposal，则向 $\mathcal{F}_{\text{STATE}}$ 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 $v_i \in V$ 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 $\mathcal{F}_{\text{STATE}}$ 发送(v_i , unlock, B')解锁上一轮区块，然后向 $\mathcal{F}_{\text{STATE}}$ 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 $\mathcal{F}_{\text{STATE}}$ 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 $\mathcal{F}_{\text{COMMIT}}$

初始化：对于 $v_i \in V$ ，设置 $c_i := 0$ ， $c_i \in C$ 。表示 Proposal 是否已 Commit。向 $\mathcal{F}_{\text{TIME}}$ 发送(timeStart, δ)命令。若在任何阶段从 $\mathcal{F}_{\text{TIME}}$ 收到(timeOver)消息，向 $\mathcal{F}_{\text{STATE}}$ 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 $c_i := 1$ ，向验证者发送(allowCommit, Proposal)消息，向 $\mathcal{F}_{\text{STATE}}$ 发送(newHeight)。
 - 否则，向验证者发送(denyCommit, Proposal)消息，向 $\mathcal{F}_{\text{STATE}}$ 发送(newRound)。
 - 否则，向 $\mathcal{F}_{\text{STATE}}$ 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 $\mathcal{F}_{\text{STATE}}$

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

- 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 $(v_i, \text{unlock}, \text{ALL})$ 消息时, 设置 $\text{PoLC} := \perp$ 。
- 当从 v_i 接收到 $(v_i, \text{queryState})$ 消息时, 返回 PoLC。

(六) 功能 $\mathcal{F}_{\text{TIME}}$

- 初始化: 设置 $t_i \in T$, $t_i := \perp$ 。
- 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 $t_i := \perp$, 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 $t_{\text{sid}} \leftarrow \delta$, 向理想功能 $\mathcal{F}_{\text{tbft}}$ 返回一个 (timeOK) 消息, 然后开始倒计时。
 - 当从 $t_{\text{sid}} \in T$, $t_{\text{sid}} = 0$ 时,
 - 会向对应的调用者发送一个 (timeOver, sid, phase_p , δ) 消息。

三、理想功能 $\mathcal{F}_{\text{TBFT}}$

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

参数:

- V : 验证节点集合.
- Δ : 每个阶段的最大执行时间.
- Σ : 延迟攻击的最大延迟.
- $\mathcal{F}_{\text{TIME}}$: 时间理想功能.
- \mathcal{F}_{BC} : 广播理想功能.
- $\mathcal{F}_{\text{SYNC}}$: 同步理想功能.

符号说明:

- δ : 实际执行时间, 初始由 S 设置, 默认值为 Δ .
- σ : 实际延迟, 初始由 S 设置, 默认值为 0.
- h_p : 当前高度, 或者当前正在执行的共识实例, 初始化为 0.
- round_p : 当前轮次, 初始化为 0.
- $\text{phase}_p \in \{\text{propose}, \text{prevote}, \text{precommit}, \text{commit}\}$: 标记当前轮次中的共识阶段, 初始化为 propose.
- $\text{count}_{\text{phase}_p}$: 记录每个阶段投票的数量, 初始化为 0.
- $\text{decision}_p[\]$: 记录各节点在不同高度上达成的最终共识值, 初始化为 nil.
- lockedValue_p : 锁定的值, 表示当前锁定的提案, 初始化为 nil.
- lockedRound_p : 锁定的轮次, 表示锁定值的轮次, 初始化为 -1.
- validValue_p : 有效的值, 表示当前有效的提案, 初始化为 nil.
- validRound_p : 有效的轮次, 表示有效值的轮次, 初始化为 -1.
- $\text{isVote}_{\text{COMMIT}}$: 标记提交阶段是否已经完成投票, 初始化为 false.
- $*$: 空参数.

接收到来自 S 的消息 $(\text{NEWROUND}, h_p, \text{round}_p, *)$, 当 $\text{phase}_p = \text{propose}$ 时:

1. 向 S 发送 $(\text{Sleep}, \text{sid}, \text{phase}_p)$, 并等待返回形式为 $(\text{Wake}, \text{sid}, \text{phase}_p, \delta, \sigma)$ 的响应。
2. 如果 $(\delta + \sigma > \Delta) \vee \sigma > \Sigma$:
 - 返回第 1 步。
3. 否则:
 - 向 $\mathcal{F}_{\text{TIME}}$ 发送 $(\text{timeStart}, \text{sid}, \text{phase}_p, \sigma)$, 并挂起执行。

- 收到 \mathcal{F}_{TIME} 返回的 $\langle \text{timeOver}, \text{sid}, \text{phase}_p, \sigma \rangle$ 后恢复执行。
- 向 \mathcal{F}_{TIME} 发送 $\langle \text{timeStart}, \text{sid}, \text{phase}_p, \delta \rangle$ 。
- 向 \mathcal{S} 发送 $\langle \text{CreateProposal}, \text{sid}, \text{phase}_p \rangle$ ，并等待返回形式为 $\langle \text{StartProposal}, \text{sid}, \text{phase}_p \rangle$ 的响应。
- 向 $\text{Proposer}(h_p, \text{round}_p)$ 发送 $\langle \text{StartProposal}, \text{sid}, \text{phase}_p \rangle$ ，并等待返回形式为 $\langle \text{PROPOSAL}, \text{sid}, \text{phase}_p, v \rangle$ 的响应。
- 如果 $\text{Proposer}(h_p, \text{round}_p)$ 被篡改，
 - 向 \mathcal{S} 发送 $\langle \text{Input}, \text{sid}, h_p, \text{round}_p, v \rangle$ 。
- 如果 $\text{valid}(v)$ 并且没有收到来自 \mathcal{F}_{TIME} 的 $\langle \text{timeOver}, \text{sid}, \text{phase}_p, \delta \rangle$ ：
 - 广播 $\langle \text{PROPOSAL}, h_p, \text{round}_p, v \rangle$ 。
- 否则：
 - 返回第 1 步。
- 更新 $\text{phase}_p \leftarrow \text{prevote}$ 。

接收到来自 $\text{Proposer}(h_p, \text{round}_p)$ 的消息 $\langle \text{PROPOSAL}, h_p, \text{round}_p, v \rangle$ ，当 $\text{phase}_p = \text{prevote}$ 时：

1. 向 \mathcal{S} 发送 $\langle \text{Sleep}, \text{sid}, \text{phase}_p \rangle$ ，并等待返回形式为 $\langle \text{Wake}, \text{sid}, \text{phase}_p, \delta, \sigma \rangle$ 的响应。
2. 如果 $(\delta + \sigma > \Delta) \vee \sigma > \Sigma$ ：
 - 广播 $\langle \text{PREVOTE}, h_p, \text{round}_p, \text{nil} \rangle$ 。
3. 否则：
 - 向 \mathcal{F}_{TIME} 发送 $\langle \text{timeStart}, \text{sid}, \text{phase}_p, \sigma \rangle$ ，并挂起执行。
 - 收到 \mathcal{F}_{TIME} 返回的 $\langle \text{timeOver}, \text{sid}, \text{phase}_p, \sigma \rangle$ 后恢复执行。
 - 向 \mathcal{F}_{TIME} 发送 $\langle \text{timeStart}, \text{sid}, \text{phase}_p, \delta \rangle$ 。
 - 如果 $\text{valid}(v) \wedge (\text{lockedRound}_p = -1 \vee \text{lockedValue}_p = v)$ 且没有收到来自 \mathcal{F}_{TIME} 的 $\langle \text{timeOver}, \text{sid}, \text{phase}_p, \delta \rangle$ ：
 - 广播 $\langle \text{PREVOTE}, h_p, \text{round}_p, \text{id}(v) \rangle$ 。
 - 否则：
 - 广播 $\langle \text{PREVOTE}, h_p, \text{round}_p, \text{nil} \rangle$ 。
4. 向 \mathcal{F}_{SYNC} 发送 $\langle \text{RoundOK} \rangle$ 。
5. 更新 $\text{phase}_p \leftarrow \text{precommit}$ 。

接收到来自 $\text{Validator}(h_p, \text{round}_p)$ 的消息 $\langle \text{PREVOTE}, h_p, \text{validRound}_p, \text{id}(v) \rangle$ ，当 $\text{phase}_p = \text{prevote} \wedge (\text{validRound}_p \geq 0 \wedge \text{validRound}_p < \text{round}_p)$ 时：

1. 设置 $\text{count}_{\text{propose}} \leftarrow \text{count}_{\text{propose}} + 1$ 。
2. 如果 $\text{valid}(v) \wedge (\text{count}_{\text{propose}} > 2f + 1) \wedge (\text{lockedRound}_p \leq \text{validRound}_p \vee \text{lockedValue}_p = v)$ ：
 - 广播 $\langle \text{PREVOTE}, h_p, \text{round}_p, \text{id}(v) \rangle$ 。
3. 否则：
 - 广播 $\langle \text{PREVOTE}, h_p, \text{round}_p, \text{nil} \rangle$ 。

接收到来自 $\text{Validator}(h_p, \text{round}_p)$ 的消息 $\langle \text{PREVOTE}, h_p, \text{round}_p, \text{id}(v) \rangle$ ，当 $\text{phase}_p = \text{precommit}$ 时：

1. 设置 $\text{count}_{\text{prevote}} \leftarrow \text{count}_{\text{prevote}} + 1$ 。
2. 发送 $\langle \text{Sleep}, \text{sid}, \text{phase}_p \rangle$ 到 \mathcal{S} ，并等待返回 $\langle \text{Wake}, \text{sid}, \text{phase}_p, \delta, \sigma \rangle$ 格式的响应。
3. 如果 $(\delta + \sigma > \Delta) \vee \sigma > \Sigma$ ：
 - 广播 $\langle \text{PRECOMMIT}, h_p, \text{round}_p, \text{nil} \rangle$ 。
4. 否则：
 - 发送 $\langle \text{timeStart}, \text{sid}, \text{phase}_p, \sigma \rangle$ 到 \mathcal{F}_{TIME} ，并挂起执行。
 - 接收到来自 \mathcal{F}_{TIME} 的消息 $\langle \text{timeOver}, \text{sid}, \text{phase}_p, \sigma \rangle$ 后，恢复执行。

- 发送 $\langle \text{timeStart}, \text{sid}, \text{phase}_p, \delta \rangle$ 到 \mathcal{F}_{TIME} 。
 - 如果 $\text{valid}(v) \wedge (\text{count}_{\text{prevote}} > 2f + 1)$ 且没有收到来自 \mathcal{F}_{TIME} 的 $\langle \text{timeOver}, \text{sid}, \text{phase}_p, \delta \rangle$ 消息：
 - 设置 $\text{lockedValue}_p \leftarrow v$, $\text{lockedRound}_p \leftarrow \text{round}_p$ 。
 - 广播 $\langle \text{PRECOMMIT}, h_p, \text{round}_p, \text{id}(v) \rangle$ 。
 - 设置 $\text{validValue}_p \leftarrow v$, $\text{validRound}_p \leftarrow \text{round}_p$ 。
 - 否则：
 - 广播 $\langle \text{PRECOMMIT}, h_p, \text{round}_p, \text{nil} \rangle$ 。
5. 更新 $\text{phase}_p \leftarrow \text{commit}$ 。

接收到来自 $\text{Validator}(h_p, \text{round}_p)$ 的消息 $\langle \text{PRECOMMIT}, h_p, \text{round}_p, \text{id}(v) \rangle$ ，当 $\text{phase}_p = \text{commit}$ 时：

1. 设置 $\text{count}_{\text{precommit}} \leftarrow \text{count}_{\text{precommit}} + 1$ 。
2. 发送 $\langle \text{Sleep}, \text{sid}, \text{phase}_p \rangle$ 到 \mathcal{S} ，并等待返回 $\langle \text{Wake}, \text{sid}, \text{phase}_p, \delta, \sigma \rangle$ 格式的响应。
3. 如果 $(\delta + \sigma > \Delta) \vee \sigma > \Sigma$ ：
 - 广播 $\langle \text{COMMIT}, h_p, \text{round}_p, \text{nil} \rangle$ ，并设置 $\text{isVote}_{\text{COMMIT}} \leftarrow \text{false}$ 。
4. 否则：
 - 发送 $\langle \text{timeStart}, \text{sid}, \text{phase}_p, \sigma \rangle$ 到 \mathcal{F}_{TIME} ，并挂起执行。
 - 接收到来自 \mathcal{F}_{TIME} 的消息 $\langle \text{timeOver}, \text{sid}, \text{phase}_p, \sigma \rangle$ 后，恢复执行。
 - 发送 $\langle \text{timeStart}, \text{sid}, \text{phase}_p, \delta \rangle$ 到 \mathcal{F}_{TIME} 。
 - 如果 $\text{valid}(v) \wedge (\text{count}_{\text{precommit}} > 2f + 1)$ 且没有收到来自 \mathcal{F}_{TIME} 的 $\langle \text{timeOver}, \text{sid}, \text{phase}_p, \delta \rangle$ 消息：
 - 广播 $\langle \text{COMMIT}, h_p, \text{round}_p, \text{id}(v) \rangle$ ，并设置 $\text{isVote}_{\text{COMMIT}} \leftarrow \text{true}$ 。
 - 否则：
 - 更新 $\text{phase}_p \leftarrow \text{propose}$ 和 $\text{round}_p \leftarrow \text{round}_p + 1$ 。

接收到来自 $\text{Validator}(h_p, \text{round}_p)$ 的消息 $\langle \text{COMMIT}, h_p, \text{round}_p, \text{id}(v) \rangle$ ，当 $\text{phase}_p = \text{commit}$ 时：

1. 设置 $\text{count}_{\text{commit}} \leftarrow \text{count}_{\text{commit}} + 1$ 。
2. 如果 $\text{valid}(v) \wedge (\text{count}_{\text{commit}} > 2f + 1) \wedge \text{isVote}_{\text{COMMIT}}$ ：
 - 设置 $\text{decision}_p[h_p] = v$ ，并更新 $\text{isVote}_{\text{COMMIT}} \leftarrow \text{false}$, $h_p \leftarrow h_p + 1$ 。
3. 发送 $\langle \text{RoundOK} \rangle$ 到 \mathcal{F}_{SYNC} 。
4. 发送 $\langle \text{RequestRound} \rangle$ 到 \mathcal{F}_{SYNC} ，接收其响应 d_i ，
 - 如果 $d_i = 0$ ：
 - 更新 $\text{phase}_p \leftarrow \text{propose}$ ，并重置 $\text{round}_p, \text{count}_{\text{phase}_p}, \text{lockedRound}_p, \text{lockedValue}_p, \text{validRound}_p, \text{validValue}_p$ 。
 - 发送 $\langle \text{NEWROUND}, h_p, \text{round}_p, * \rangle$ 到 \mathcal{S} 。
 - 否则重新执行此步骤。

接收到消息 $\langle *, h_p, \text{round}, * \rangle$ 时：

1. 设置 $\text{count}_{\text{nextround}} \leftarrow \text{count}_{\text{nextround}} + 1$ 。
2. 如果 $(\text{count}_{\text{nextround}} > f + 1) \wedge \text{round} > \text{round}_p$ ：
 - 发送 $\langle \text{NEWROUND}, h_p, \text{round}, * \rangle$ 到 \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 E:

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。