TBFT 建模进度

摘要

版本 4：本文档对 TBFT 共识机制的建模进行了调整。与之前的建模版本相比，本文完善了$\mathcal{F}_{STATE}$的功能，对$\mathcal{F}_{TIME}$进行了精简，对$\mathcal{F}_{TBFT}$的描述进行了更新，设定了参数的初始值，并修改了轮间同步机制的相关内容。

一、 初步框架

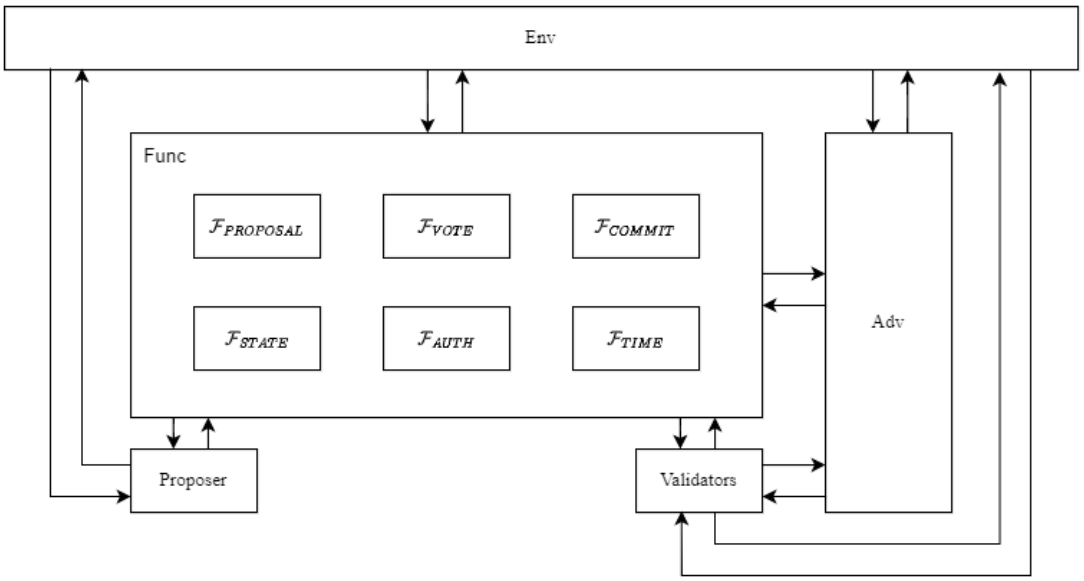

图 1 TBFT 协议初步框架

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 := ⊥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：

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

    - 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$v_i \in V$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$v_i \in V$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$\mathcal{F}_{STATE}$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$c_i := 0$，$c_i \in C$。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$c_i := 1$，设置 commitTime 为当前时间，向$\mathcal{F}_{STATE}$发送(newHeight)。
    - 否则，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 $t_i := \perp$，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$t_{sid} \leftarrow \delta$，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。

三、 理想功能$\mathcal{F}_{\mathrm{TBFT}}$

功能 $\mathcal{F}_{tbft}^{V,\Delta,\Sigma}$ [$\mathcal{F}_{TIME}, \mathcal{F}_{BB}, \mathcal{F}_{SYNC}$]

参数：

- V: 验证者集合.
- Δ : 每个阶段的最大执行时间.
- Σ : 延迟攻击中的最大延迟.
- $\mathcal{F}_{TIME}$: 时间的理想功能.
- $\mathcal{F}_{BB}$: 广播的理想功能.
- $\mathcal{F}_{SYNC}$: 同步的理想功能.

符号说明：

δ : 实际执行时间，由 S 初始化，默认值为 Δ

σ : 实际延迟，由 S 初始化，默认值为 0

h_p : 当前高度，或当前正在执行的共识实例，初始化为 0

$round_p$: 当前轮次编号，初始化为 0

$phase_p$ ∈ {propose, prevote, precommit, commit} : 标记当前轮次内的共识阶段，初始化为 propose.

$count_{phase_p}$: 记录每个阶段的投票次数，初始化为 0

$decision_p[\ ]$: 记录各节点在不同高度下达成的最终共识值，初始化为 nil.

$isVote_{COMMIT}$: 标记提交阶段本身是否已完成投票，初始化为 false.

$*$: 空参数.

在收到消息 ⟨NEWROUND, h_p, round_p,$*$⟩ **来自** $\mathcal{E}$，且 $\mathrm{phase}_p = \mathrm{propose}$ 时：

1. 向 $\mathcal{S}$ 发送 ⟨Sleep,sid, phase_p⟩，等待形如 ⟨Wake,sid, $\mathrm{phase}_p, \delta, \sigma$⟩ 的响应。
2. 如果（$\delta + \sigma > \Delta$）∨ $\sigma > \Sigma$：
   - 返回步骤 1。
3. 否则：
   - 向 $\mathcal{F}_{TIME}$ 发送 ⟨timeStart,sid, phase_p, σ⟩，并暂停执行。
   - 收到来自 $\mathcal{F}_{TIME}$ 的 ⟨timeOver,sid, phase_p, σ⟩ 后恢复执行。
   - 向 $\mathcal{F}_{TIME}$ 发送 ⟨timeStart,sid, phase_p, δ⟩。
   - 向 $\mathcal{S}$ 发送 ⟨CreateProposal,sid, phase_p⟩，等待形如 ⟨StartProposal,sid, phase_p⟩ 的响应。

- 向 Proposer(h_p, round_p) 发送 ⟨StartProposal,sid, phase_p⟩，等待形如 ⟨PROPOSAL,sid, phase_p, v⟩ 的响应。
- 如果 Proposer(h_p, round_p) 被破坏，
  - 向 $\mathcal{S}$ 发送 ⟨Input,sid, h_p, round_p, v⟩。
- 如果 valid(v) 且尚未收到来自 $\mathcal{F}_{TIME}$ 的 ⟨timeOver,sid, phase_p, δ⟩：
  - 广播 ⟨PROPOSAL, h_p, round_p, v⟩。
- 否则：
  - 返回步骤 1。
- 更新 $\text{phase}_p \leftarrow$ prevote。

在收到消息 ⟨PROPOSAL, h_p, round_p, v⟩ 来自 Proposer(h_p, round_p)，且 $\text{phase}_p =$ prevote 时：

1. 向 $\mathcal{S}$ 发送 ⟨Sleep,sid, phase_p⟩，等待形如 ⟨Wake,sid, $\text{phase}_p, \delta, \sigma$⟩ 的响应。
2. 如果 $(\delta + \sigma > \Delta) \vee \sigma > \Sigma$：
   - 广播 ⟨PREVOTE, h_p, round_p,nil⟩。
3. 否则：
   - 向 $\mathcal{F}_{TIME}$ 发送 ⟨timeStart,sid, phase_p, σ⟩，并暂停执行。
   - 收到来自 $\mathcal{F}_{TIME}$ 的 ⟨timeOver,sid, phase_p, σ⟩ 后恢复执行。
   - 向 $\mathcal{F}_{TIME}$ 发送 ⟨timeStart,sid, phase_p, δ⟩。
   - 如果 valid(v) 且尚未收到 ⟨timeOver,sid, phase_p, δ⟩：
     - 广播 ⟨PREVOTE, h_p, round_p,id(v)⟩。
   - 否则：
     - 广播 ⟨PREVOTE, h_p, round_p,nil⟩。
4. 向 $\mathcal{F}_{SYNC}$ 发送 ⟨RoundOK⟩。
5. 更新 $\text{phase}_p \leftarrow$ precommit。

在收到消息 ⟨PREVOTE, h_p, round_p,id(v)⟩ 来自 Validator(h_p, round_p)，且如果 $\text{phase}_p =$ precommit：

1. 设置 $\text{count}_{\text{prevote}} \leftarrow \text{count}_{\text{prevote}} + 1$
2. 向 $\mathcal{S}$ 发送 ⟨Sleep,sid, phase_p⟩ 并等待形式为 ⟨Wake,sid, $\text{phase}_p, \delta, \sigma$⟩ 的响应。
3. 如果 $(\delta + \sigma > \Delta) \vee \sigma > \Sigma$：
   - 广播 ⟨PRECOMMIT, h_p, round_p,nil⟩。
4. 否则：
   - 向 $\mathcal{F}_{TIME}$ 发送 ⟨timeStart,sid, phase_p, σ⟩ 并暂停执行。
   - 在收到来自 $\mathcal{F}_{TIME}$ 的 ⟨timeOver,sid, phase_p, σ⟩ 消息后，恢复执行。
   - 向 $\mathcal{F}_{TIME}$ 发送 ⟨timeStart,sid, phase_p, δ⟩。
   - 如果 valid$(v) \wedge (\text{count}_{\text{prevote}} > 2f + 1)$ 且尚未从 $\mathcal{F}_{TIME}$ 收到 ⟨timeOver,sid, phase_p, δ⟩：
     - 广播 ⟨PRECOMMIT, h_p, round_p,id(v)⟩。
   - 否则：
     - 广播 ⟨PRECOMMIT, h_p, round_p,nil⟩。
5. 更新 $\text{phase}_p \leftarrow$ commit。

在收到消息 ⟨PRECOMMIT, h_p, round_p,id(v)⟩ 来自 Validator(h_p, round_p)，且如果 $\text{phase}_p =$ commit：

1. 设置 $\text{count}_{\text{precommit}} \leftarrow \text{count}_{\text{precommit}} + 1$
2. 向 $\mathcal{S}$ 发送 ⟨Sleep,sid, phase_p⟩ 并等待形式为 ⟨Wake,sid, $\text{phase}_p, \delta, \sigma$⟩ 的响应。
3. 如果 $(\delta + \sigma > \Delta) \vee \sigma > \Sigma$：
   - 广播 ⟨COMMIT, h_p, round_p,nil⟩，并设置 $\text{isVote}_{COMMIT} \leftarrow$ false。
4. 否则：

- 向 $\mathcal{F}_{TIME}$ 发送 ⟨timeStart,sid, phase_p, σ⟩ 并暂停执行。
- 在收到来自 $\mathcal{F}_{TIME}$ 的 ⟨timeOver,sid, phase_p, σ⟩ 消息后，恢复执行。
- 向 $\mathcal{F}_{TIME}$ 发送 ⟨timeStart,sid, phase_p, δ⟩。
- 如果 $\text{valid}(v) \wedge \left(\text{count}_{\text{precommit}} > 2f + 1\right)$ 且尚未从 $\mathcal{F}_{TIME}$ 收到 ⟨timeOver,sid, phase_p, δ⟩：
  - 广播 ⟨COMMIT, h_p, round_p,id(v)⟩，并设置 $\text{isVote}_{COMMIT} \leftarrow \text{true}$。
- 否则：
  - 更新 $\text{phase}_p \leftarrow \text{propose}$ 且 $\text{round}_p \leftarrow \text{round}_p + 1$。

在收到消息 ⟨COMMIT, h_p, round_p,id(v)⟩ 来自 Validator$\left(h_p, \text{round}_p\right)$，且如果 $\text{phase}_p =$ commit：

1. 设置 $\text{count}_{\text{commit}} \leftarrow \text{count}_{\text{commit}} + 1$
2. 如果 $\text{valid}(v) \wedge (\text{count}_{\text{commit}} > 2f + 1) \wedge \text{isVote}_{COMMIT}$：
   - 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. 向 $\mathcal{F}_{SYNC}$ 发送 ⟨RoundOK⟩。
4. 向 $\mathcal{F}_{SYNC}$ 发送 ⟨RequestRound⟩，接收响应 d_i，
   - 如果 $d_i = 0$，更新 $\text{phase}_p \leftarrow \text{propose}$ 并重置 $\text{round}_p, \text{count}_{phase_p}$。
   - 否则重新执行此步骤。

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。