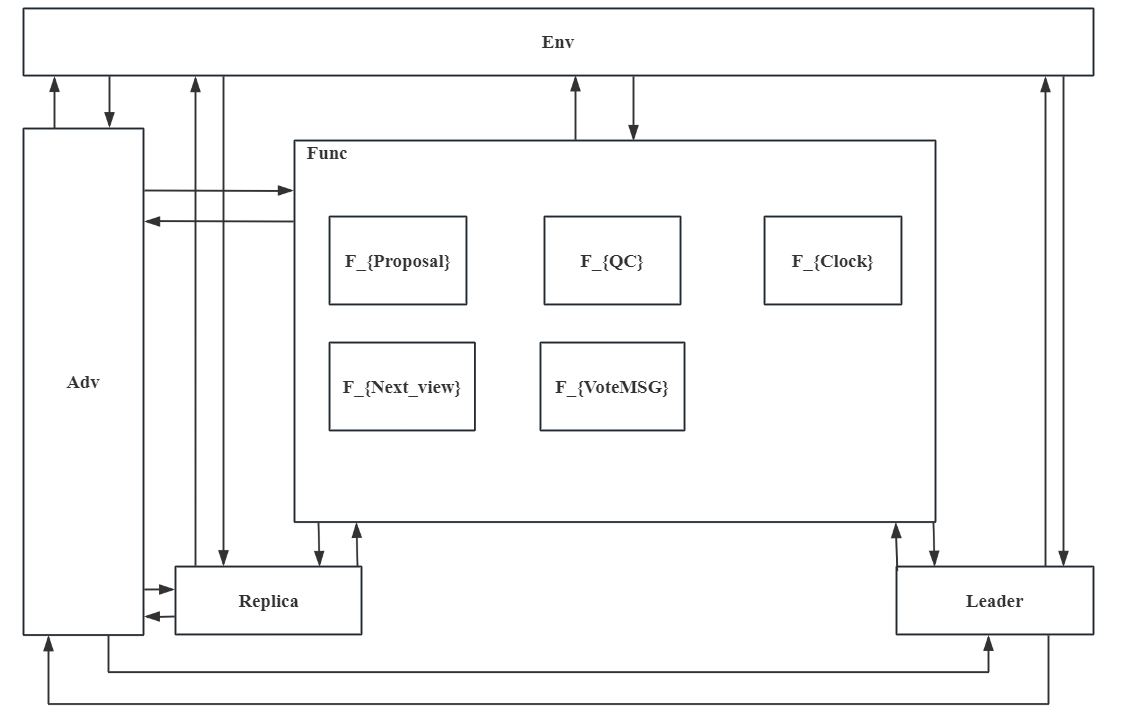
**Hotstuff建模进度**

**摘要：**

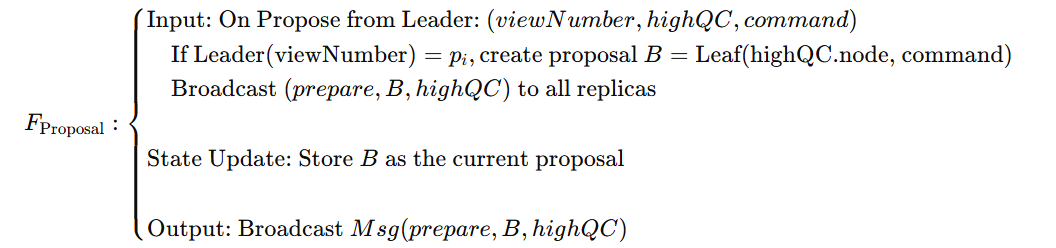
这篇文档是关于Hotstuff共识协议的11.13建模进度报告。针对上次形式化程度不足口语化较多的问题，对理想功能进行了更详细描述，网络延迟或视图切换可能导致副本在不同阶段失去同步。这篇文档主要是借鉴之前读过的两篇关于实现UC同步的论文，为了增强 HotStuff 协议中副本的同步性，我们引入 Fclock作为同步原语。，确保所有副本能协调一致地进入新一轮或阶段。

1. **整体框架图：**



1. **功能描述**

### F\_{Proposal}



HotStuff的提案功能在prepare阶段主要实现了提案生成与广播。在这一阶段，作为领导者的副本r执行如下操作：

**视图同步**：在开始新的视图 curView前，领导 r需要等待至少 n−f 个来自上一个视图的“new\_view”消息，以确认多数副本已经准备好进入新的视图。此步骤通过F{New\_view}功能实现，确保领导者具备最新的链状态。

**提案生成**：在获得上一个视图的多数证明后，领导者从这些new\_view消息中提取最高的法定证明（highest quorum certificate, highQC），用于生成当前视图的提案。在伪代码中，这对应于 highQC ← arg max {m.justify.viewNumber}.justify，并用于调用createLeaf(highQC.node, client’s command) 创建新的区块提案B。

**提案广播**：生成提案后，领导者通过 broadcast Msg(prepare, B, highQC) 向所有副本广播新的prepare消息。

理想功能F\_{Proposal}需要保证只有视图中的领导者可以生成提案，提案必须包含上一个视图的最新法定证明以确保链的延续性。

### F\_{Vote}

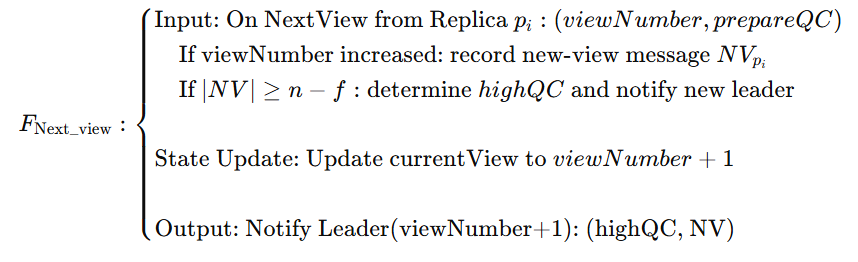
### F\_{Vote}功能用于管理参与者对区块提案的投票过程，确保每个参与者在一个特定轮次中只能投票一次。具体要求如下：

**Prepare阶段投票**：在收到领导者的prepare消息后，副本检查提案的合法性（即提案是否由 m.justify.node 派生并且 safeNode(m.node, m.justify) 为真）。如果提案合法，则副本会通过 send voteMsg(prepare, m.node, ⊥) 向领导者发送投票。

**Pre-commit和Commit阶段的投票**：在这些阶段，领导者会收集 n−fn - fn−f 个prepare和pre-commit投票并生成相应的法定证明（prepareQC和precommitQC），然后广播进入下一个阶段。副本则在收到领导者的pre-commit或commit消息后，对应地向领导者投票。

该功能确保了投票过程的完整性和准确性，防止恶意参与者在一个轮次中多次投票或提交无效投票。

1. **F\_{Next\_view}**

  
新视图功能在协议的视图更替阶段发挥作用。它帮助协议从一个失效的视图切换到下一个视图，确保系统可以平稳地继续共识。

**视图切换触发**：在任何阶段的等待过程中，如果副本发现领导者未能及时达成共识，它会触发视图切换（伪代码中的 nextView interrupt），调用 F\_{Next\_view}进入下一个视图。

**新视图消息**：每个副本会向下一视图的领导者发送一个包含其当前最高法定证明的new-view消息 send Msg(new-view, ⊥, prepareQC)。新领导者通过收集这些消息生成新的highQC，并生成新的提案。

理想功能 F\_{Next\_view}确保协议在领导者失效时，能顺利进入下一个视图，并保持链的延续性和一致性

### F\_{QC}

### F\_{QC}功能负责管理法定证书的生成和验证，确保只有在达成足够多的投票同意时才会形成证书。具体要求如下：

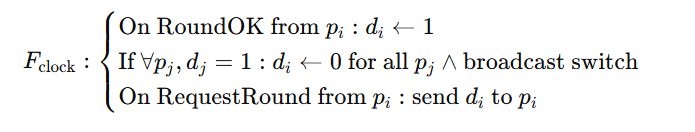
**收集投票**：F\_{QC}收集并验证参与者的投票，以确认其数量达到法定人数要求。

**生成法定证书**：一旦达成法定人数， F\_{QC}会生成一个合法的证书，并将其广播给所有参与者。

**验证证书的有效性**：每次新的区块提案都需要包含上一个区块的法定证书。F\_{QC}会检查证书是否有效，以确保新提案的合法性。

该功能确保了系统在每个区块提案中都包含足够的参与者支持，防止篡改和伪造证书的攻击，从而维持共识协议的安全性和正确性。

### F\_{clock}



**初始化：**每个副本 pi都维护一个本地同步位di，初始值0。

**消息处理：(RoundOK)：**

当pi认为当前回合完成时，向 Fclock发送 RoundOK消息。

Fclock将di设置为 1。如果所有诚实副本的di=1，则重置所di=0，并向所有副本发送 switch信号，表明进入下一个同步点。

**(RequestRound)：**

当副本 pi请求当前的同步状态时，Fclock返回其 di值。

**prepare阶段伪代码修改示例：**

1: for curView ← 1, 2, 3, . . . do

2: prepare phase

3: as a leader

4: wait for (n − f) new-view messages: M ← {m | matchingMsg(m, new-view, curView − 1)}

5: highQC ← max{m.justify.viewNumber | m ∈ M}.justify

6: curProposal ← createLeaf(highQC.node, client’s command)

7: broadcast Msg(prepare, curProposal, highQC)

8: as a replica

9: wait for message m : matchingMsg(m, prepare, curView) from leader(curView)

10: if m.node extends from m.justify.node ∧ safeNode(m.node, m.justify) then

11: send voteMsg(prepare, m.node, ⊥) to leader(curView)

12: invoke F\_{clock}(RoundOK) // 通知本地回合完成

13: synchronize replicas:

14: wait for F\_{clock} to return (switch) // 等待所有副本完成本阶段

1. **协议描述**

–Party Leader:

**New\_view**：新领导者开始时从(n - f)个副本中收集"new-view"消息。这些消息包含每个副本在上一轮的（prepareQC）。

**Proposal**: 领导者从New\_view这些消息中选择具有最高视图号的prepareQC，并基于此创建一个新的提案（Proposal）

**Broadcast MSG**: 领导者向所有副本广播这个提案，并附带其选择的最高prepareQC作为安全证明。

–Party Replica:

**safeNode**: 在收到来自Leader的提议消息m后，它检查提案消息m，m携带QC的正确性参数（justification）m.justify，检查后确定m.node是否可以安全接受。

**Prepare**: 根据收到的Proposal消息m，调用F\_{VOTEMSG}(PREPARE,m.node,⊥)。

**Pre-commit**: 根据收到的Proposal消息m，调用

F\_{VOTEMSG}(PRE-COMMIT,m.justify.node,⊥)。

**Commit**: 根据收到的Proposal消息m，调用F\_{VOTEMSG}(COMMIT,m.justify.node,⊥)。

**Next\_view**:

在所有阶段中，副本在视图viewNumber处等待消息的超时时间，超时时间由辅助的nextView（viewNumber）实用程序确定。如果nextView（viewNumber）中断等待，副本调用F\_{Next\_view},增加viewNumber并开始下一个视图。

**RoundOK**: 副本将等待轮次更新，开始新的轮次。

1. **问题**
2. 对时间的假设较为模糊，如“等待足够的消息”或“视图切换的超时机制”。可能可以通过明确超时条件，例如等待 Δ 时间未收到足够消息时触发视图切换。或者描述事件之间的时间关系，例如“某个阶段的结束信号必须在所有投票收到后触发”。

2.之前提到的终止性、有效性、不可篡改性、数据完整性在目前在建模中还未体现充分。