**摘要**

### **建模版本8：**

### 1、将广播功能F\_bc嵌入理想功能具体流程中，并且对同步切换视图进行了修正，实现了所有节点同步进入下一视图。

**证明版本4：**

1. 新增分场景终止性证明章节，将敌手攻击细化为两类延迟：（σadv≤Δ与 σadv>Δ），并进一步按敌手控制的角色（领导者/普通副本）拆分子场景，每个场景包含详细的现实世界执行流程、理想世界模拟及不可区分性分析。
2. 细化时间计算，补充分阶段时间拆解（如Prepare/PreCommit/Commit阶段耗时）。

**目录**

**[摘要 1](#_Toc26939)**

**[Hotstuff协议UC建模 3](#_Toc5692)**

[一、 整体框架图： 3](#_Toc28112)

[二、 功能描述 3](#_Toc23406)

[三、 理想功能 6](#_Toc25453)

[四、 协议描述 12](#_Toc29316)

**[HotStuff协议的UC终止性证明 17](#_Toc21620)**

[定理声明（UC终止性） 17](#_Toc1793)

[一、敌手模型与行为定义 17](#_Toc30478)

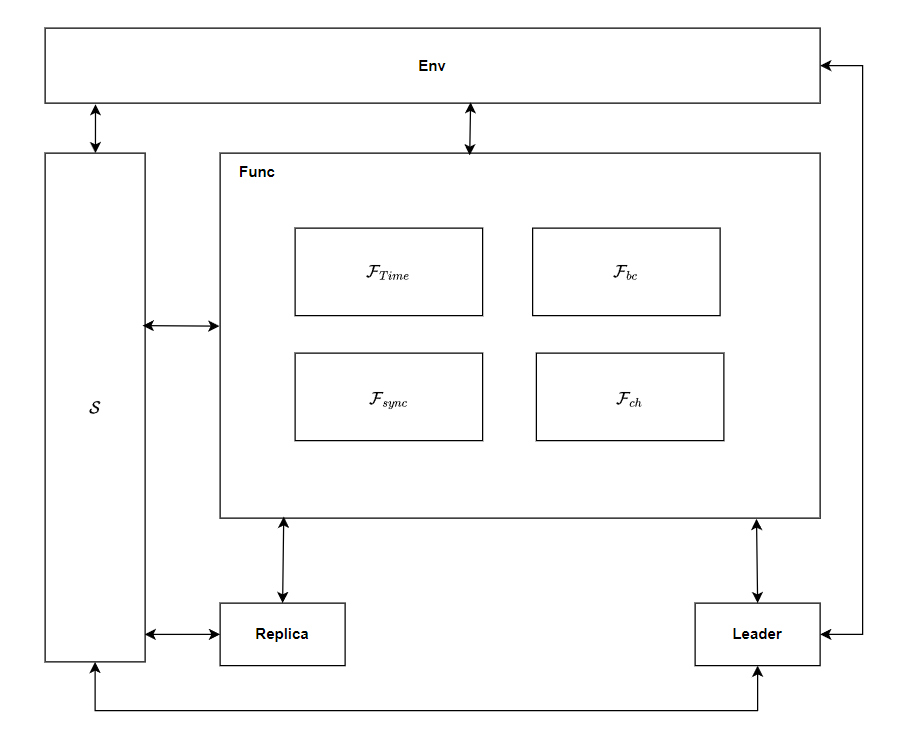
[二、分场景终止性证明 17](#_Toc15731)

[三、全局时间分析 24](#_Toc27250)

[四、不可区分性证明 25](#_Toc24654)

# Hotstuff协议UC建模

## 整体框架图：



## 功能描述

### F\_{Proposal}

初始化：设置。

–当收到消息时，

* 选取这些消息中最高的prepareQC最为highQC：
* 在highQC的节点的叶子上写入客户指令，提出新的提案B：

-将提案B、highQC封装在MSG中广播给replica：

### F\_{Vote} 初始化：设置。

–当收到来自的消息时，

* 先检查m是否与自己状态匹配：
* 检查叶子节点是否是本地lockedQC对应节点后继以及QC是否比本地lockedQC对应节点的视图更高：
* 在highQC的节点的叶子上写入客户指令，提出新的提案B：

-将投票信息结点m、自己的部分签名封装在VOTEMSG中发送给leader。

-当收到来自的消息时，

* 先检查QC是否与自己状态匹配：
* 如果决定投票且type是PREPARE阶段，更新本地状态：
* 如果决定投票且type是PRE-COMMIT阶段，更新本地状态：

-将投票信息m.justify.node、自己的部分签名封装在VOTEMSG中发送给leader：

### F\_{QC}

初始化：设置。

-当收到2f+1条投票消息时：

* 先检查m是否与自己状态匹配：
* 收集replica的投票，把部分签名组合：

-将QC封装在MSG中广播给replica：

### F\_{TIME}

初始化：设置，。

–当从任意replica 接收到请求时，将更新为 ← ，向replica 返回一个消息, 然后开始倒计时。

–当从某一个时，它会向对应的replica 发送一个消息。

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

初始化：设置，。

–当从任意replica m收到 next\_view请求MSG(⊥，m,prepareQC)时，将更新为，将prepareQC更新为。

–将、封装在NEW-VIEW message中发送给下一视图的leader。

1. **F\_{Ch}**

我们定义一组参与方，其中 和 表示该集合中的两个参与方，分别是消息 的发送方和接收方。

是基于功能的参数进行定义的。消息标识符 由功能新选定。

在接收到来自 的输入 时，输出 给 。

在接收到来自 的 后，向发送 。

根据以下参数化函数设置 ：

– 对于 设置 , 。在接收到来自 的 后，向 发送 。

– 对于 设置 。

– 对于 设置 。

– 对于 设置 。

– 对于 设置 。在接收到来自 的 后，向 发送 。

– 对于 设置 。

在接收到来自 的 后，向 发送 。

在接收到来自 的 后，向 发送 。

在接收到来自 的 后，输出 给 。

在接收到来自 的 后，向 发送 。

1. **F\_{bc}**

广播功能 由集合 参数化，具体流程如下：

在接收到来自某方 的 后，向集合 中的所有实体以及 发送 。

## 理想功能

**Functionality**

**Network Delay Attack**

**Parameters**:

* : Replica Set
* ：the Maximum timeout duration
* : the Maximum timeout duration of Replica within current view.
* : Ideal functionality for timing.
* : Ideal functionality for broadcast.
* : Ideal functionality for synchronization.

**Symbol Explanation:**

#### **Upon receiving message** from :

1. .
2. If is corrupted:

* Send to .

1. Send to wait for a response of the form :
   * Set which .

#### **Upon receiving message** from :

1. Send to and wait for a response of the form :
   * If :
     + set
     + send to and .
     + Send to .
     + Send to ,receive its response .
       - If ,set .
       - else re-execute this step.
   * else,Send to .
2. If :
   * Set .
   * If ,Send to and wait for a response of the form :
     + Create.
     + If no is received from :
       - send to .
       - Set .
     + Else:
       - set .
       - send to and .
       - Send to .
       - Send to ,receive its response .
         * If ,set .
         * else re-execute this step.
3. Else，ignore this message.

#### **Upon receiving message** from :

1. Send to and wait for a response of the form :
   * If :
     + set
     + send to and .
     + Send to .
     + Send to ,receive its response .
       - If ,set .
       - else re-execute this step.
   * else,Send to .
2. If :
   * If the is higher than , update
   * If and no has been received from :
     + Send to .
     + Send to and wait for a response of the form :
       - Set .
   * else:
     + set .
     + send to and .
     + Send to .
     + Send to ,receive its response .
       - If ,set .
       - else re-execute this step.
3. Else，ignore this message.

#### **Upon receiving message** from :

1. Send to and wait for a response of the form :
   * If :
     + set
     + send to and .
     + Send to .
     + Send to ,receive its response .
       - If ,set .
       - else re-execute this step.
   * else,Send to .
     + If :
     + Set
     + If :
     + Create.
     + If no is received from :
       - send to .
       - Send to and wait for a response of the form :
         * Set .
     + Else:
       - set .
       - send to and .
       - Send to .
       - Send to ,receive its response .
         * If ,set .
         * else re-execute this step.
2. Else，ignore this message.

#### **Upon receiving message** from :

1. Send to and wait for a response of the form :
   * If :
     + set .
     + send to and .
   * else,Send to .
2. If :
   * If no has been received from :
     + Set .
     + Send to .
     + Send to and wait for a response of the form :
       - Set .
   * else:
     + set .
     + send to and .
     + Send to .
     + Send to ,receive its response .
       - If ,set .
       - else re-execute this step.
3. Else，ignore this message.

#### **Upon receiving message** from :

1. Send to and wait for a response of the form :
   * If :
     + set
     + send to and .
     + Send to .
     + Send to ,receive its response .
       - If ,set .
       - else re-execute this step.
   * else,Send to .
     + If :
     + Set
     + If :
     + Create.
     + If no is received from :
       - send to .
       - Send to and wait for a response of the form :
         * Set .
     + Else:
       - set .
       - send to and .
       - Send to .
       - Send to ,receive its response .
         * If ,set .
         * else re-execute this step.
2. Else，ignore this message.

#### **Upon receiving message** from :

1. Send to and wait for a response of the form :
   * If :
     + set .
     + send to and .
   * else,Send to .
2. If :
   * If no has been received from :
     + Set .
     + Send to .
     + Send to and wait for a response of the form :
       - Set .
   * else:
     + set .
     + send to and .
     + Send to .
     + Send to ,receive its response .
       - If ,set .
       - else re-execute this step.
3. Else，ignore this message.

#### **Upon receiving message** from :

1. Send to and wait for a response of the form :
   * If :
     + set
     + send to and .
     + Send to .
     + Send to ,receive its response .
       - If ,set .
       - else re-execute this step.
   * else,Send to .
     + If :
     + Set
     + If :
     + Create.
     + If no is received from :
       - send to .
       - send to and .
       - Send to .
       - Send to ,receive its response .
         * If ,set .
         * else re-execute this step.
     + Else:
       - set .
       - send to and .
       - Send to .
       - Send to ,receive its response .
         * If ,set . set .
         * else re-execute this step.
2. Else，ignore this message.

#### **Upon receiving message** from :

1. Send to and wait for a response of the form :
   * If :
     + set .
     + send to and .
   * else,Send to .
2. If :
   * If no has been received from :
     + Execute new commands through
     + Send to .
     + Send to ,receive its response .
       - If ,set ,set .
       - else re-execute this step.
   * else:
     + set .
     + send to and .
     + Send to .
     + Send to ,receive its response .
       - If ,set .
       - else re-execute this step.
3. Else，ignore this message.

## 协议描述

**The protocol**

|  | Leader |  | Replica |  |
| --- | --- | --- | --- | --- |
|  | 1: Send to |  |  |  |
|  | 2: Send to ,receive |  |  |  |
|  | 3: Send to |  |  |  |
|  | 4: If a message is received from ,Send to |  |  |  |
|  | 5: Else,broadcast to Replicas |  |  |  |
|  | 6: Send to |  |  |  |
|  |  |  | 7: Send to |  |
|  |  |  | :If a message is received from ,Send to |  |
|  |  |  | 8: Else,if ,call |  |
|  | 9: Send to |  |  |  |
|  | 10: Call |  |  |  |
|  |  |  | 11: Send to |  |
|  |  |  | 12: Set |  |
|  |  |  | 13: Call |  |
|  | 14: Send to |  |  |  |
|  | 15: Call |  |  |  |
|  |  |  | 16:Send to |  |
|  |  |  | 17: Set |  |
|  |  |  | 18: Call |  |
|  | 19: Send to |  |  |  |
|  | 20: Call |  |  |  |
|  | 21: Send to |  |  |  |
|  |  |  | 22: Send to |  |
|  |  |  | 23: If a message is received from ,Send to |  |
|  |  |  | 24: Else,execute new commands through |  |
|  |  |  | 25: Send to |  |
|  |  |  |  |  |

### **The Functionality**

| REPLICA |  |  |
| --- | --- | --- |
|  |  | Initialize: |
| 1: Send to |  |  |
|  |  | 2: set ,return |
|  |  | 3: When |
|  |  | 4: Send |

**The Functionality**

|  | Replica |  |  |
| --- | --- | --- | --- |
| Prepare | 1: Send to |  |  |
|  |  |  | 2 : Send to |
|  |  |  | 3 : If no is received from : send to Leader |
|  |  |  |  |
| Precommit | 1: Send to |  |  |
|  |  |  | 2 : Send to |
|  |  |  | 3 : If no is received from : send to Leader |
| Commit | 1: Send to |  |  |
|  |  |  | 2 : Send to |
|  |  |  | 3 : If no is received from : send to Leader |

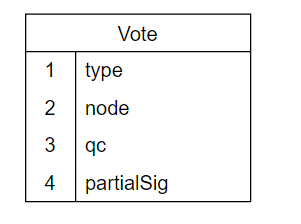
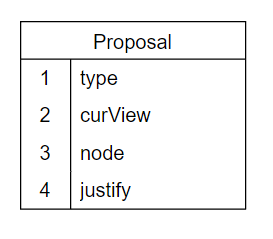
### **The Functionality**

|  | Leader |  |  |
| --- | --- | --- | --- |
| **PreCommit** | 1: Send to |  |  |
|  |  |  | 2: Send to |
|  |  |  | 3: If no is received from and more than prepare votes are received: broadcast |
| **Commit** | 1: Send to |  |  |
|  |  |  | 2: Send to |
|  |  |  | 3: If no is received from and more than precommit votes are received: broadcast |
| **Decide** | 1: Send to |  |  |
|  |  |  | 2: Send to |
|  |  |  | 3: If no is received from and more than commit votes are received: broadcast |

### **The Functionality**

| Leader/REPLICA |  |  |
| --- | --- | --- |
| 1: Send to |  |  |
|  |  | 2: Return n-f NEW-VIEW messages which |
| 1. Send to |  |  |
|  |  | 2: Set ,return |

相关数据结构：



# HotStuff协议的UC终止性证明

## 定理声明（UC终止性）

在HotStuff协议中，假设敌手最多控制个节点（包括可能的领导者），且消息延迟有界（），则存在模拟器，使得对于任意环境，以下成立：

1. **终止性**：所有正确副本在时间内达成共识并输出相同的值。
2. **不可区分**：环境无法区分真实协议与理想功能的执行和输出结果。(详细证明在最后一节）

## 一、敌手模型与行为定义

### 敌手能力

1. **控制节点**：
   * 敌手控制个节点，可包括**领导者（Leader）**或**普通副本（Replica）**。
   * 被控制的节点称为**腐败节点**，其余为**诚实节点**。
2. **攻击行为**：
   * **延迟消息**：腐败节点可延迟其发送或转发的消息，最大延迟为。
   * **偏离协议**：腐败节点可发送无效提案（Proposal）、拒绝投票（Vote）或生成非法QC。
3. **限制**：
   * 敌手无法永久阻止消息传递（消息最终必达）。
   * 敌手无法伪造合法签名（假设存在不可伪造的签名机制）。

## 二、分场景终止性证明

## 1. 敌手延迟（未触发超时）

### 1.1 Leader节点被Corrupted（敌手控制领导者）

#### 1.1.1 现实世界执行

* **敌手行为**：
  + 控制当前视图的Leader节点，通过延迟发送消息，但保证。
  + 发送无效提案（如不包含前序QC哈希），但被诚实副本过滤。
* **协议流程**：
  1. **Prepare阶段**：腐败Leader在调用广播提案，诚实副本在内接收。
  2. **PreCommit阶段**：诚实副本在内生成个，调用形成。
  3. **Commit阶段**：诚实副本在内生成个，调用形成。

#### 1.1.2 理想世界模拟

* **模拟器操作**：
  1. **消息截获**：当腐败Leader调用广播无效提案时，调用FILTER(msg)丢弃该提案。
  2. **超时同步**：若诚实副本未收到合法提案，通过触发TIMEOUT事件，递增视图并更新。
  3. **QC生成**：对合法提案，调用GENERATE-QC(Prepare)生成。

#### 1.1.3 不可区分性分析

* **现实世界**：
  + **协议执行过程**：
    - 腐败Leader在发送提案（），诚实副本在前接收。
    - 诚实副本在内生成，并在内生成。
    - 所有消息在时间内交付，未触发超时。
  + **环境视角**：
    - 观察到提案在前被接收。
    - 各阶段QC（PrepareQC、PreCommitQC）在前生成。
    - 最终决策在提交。
* **理想世界**：
  + **协议执行过程**：
    - 模拟器过滤腐败Leader的无效提案，通过同步超时检测。
    - 合法提案的QC生成严格遵循投票规则。
    - 各阶段消息在时间内交付。
  + **环境视角**：
    - 观察到与真实世界相同的提案接收时间和QC生成时序。
    - 腐败Leader的无效提案被静默过滤，无额外可观测痕迹。
* **现实世界与理想世界的对比**：
  + **时间一致性**：
    - 提案交付、QC生成和决策提交时间在两种世界中均为。
    - 消息延迟被时间窗口完全覆盖。
  + **输出一致性**：
    - 环境观测到的QC哈希链、投票计数及最终决策完全一致。
    - 腐败Leader的无效操作在理想功能中被完美模拟为网络延迟。

#### 1.1.4 时间计算

1. **Prepare阶段**：
   * 敌手Leader在发送提案（）。
   * 诚实副本在前接收提案。
   * **时间消耗**：（最大等待时间）。
2. **PreCommit阶段**：
   * 诚实副本在内生成。
   * **时间消耗**：。
3. **Commit阶段**：
   * 诚实副本在内生成。
   * **时间消耗**：。

**总时间**：（与无攻击一致）。

### 1.2 普通副本被Corrupted（敌手控制个副本）

#### 1.2.1 现实世界执行

* **敌手行为**：
  + 控制个副本，在Prepare/PreCommit阶段延迟发送，但保证。
  + 发送无效投票（如签名错误），但被过滤。
* **协议流程**：
  1. **Prepare阶段**：诚实Leader广播合法提案，诚实副本在内响应，所有节点应该在内接收。诚实节点在时间内调用，腐败节点将在时间内调用，收集票后进入Precommit。
  2. **PreCommit阶段**：诚实节点在时间内调用 ，腐败节点将在时间内调用 收集票后进入Commit。
  3. **Commit阶段**：诚实节点在时间内调用 ，腐败节点将在时间内调用 收集票后进入Decide。

#### 1.2.2 理想世界模拟

* **模拟器操作**：
  1. **投票过滤**：通过自动丢弃腐败副本的无效投票。
  2. **投票调度**：对控制的节点，调用设置延迟。
  3. **QC生成**：当合法投票数达时，调用GENERATE-QC(phase)生成QC。

#### 1.2.3 不可区分性分析

* **现实世界**：
  + **协议执行过程**：
    - 诚实Leader在广播提案，腐败副本在内延迟发送投票。
    - 诚实副本收集合法投票生成QC，超时未被触发。
    - 所有阶段在内完成。
  + **环境视角**：
    - 观察到腐败副本的投票延迟在窗口内完成。
    - 各阶段QC按预期生成，无超时事件。
* **理想世界**：
  + **协议执行过程**：
    - 模拟器通过过滤无效投票，并调度腐败副本的延迟投票。
    - QC生成条件与真实协议一致（合法签名）。
    - 消息交付时间通过严格同步。
  + **环境视角**：
    - 投票计数和时序与真实世界无差异。
    - 腐败副本的延迟行为被映射为理想功能内的合法操作。
* **现实世界与理想世界的对比**：
  + **时间一致性**：
    - Prepare/PreCommit/Commit阶段分别耗时，总时间。
    - 腐败副本的延迟被窗口吸收。
  + **输出一致性**：
    - 环境观测到的QC内容、视图编号及阶段切换事件完全一致。
    - 无效投票的过滤逻辑在两种世界中不可区分。

#### 1.2.4 时间计算

1. **Prepare阶段**：
   * 诚实Leader在发送提案，所有副本在前接收。
   * **时间消耗**：。
2. **PreCommit阶段**：
   * 腐败副本在发送投票（）。
   * 投票收集完成时间仍为。
   * **时间消耗**：。
3. **Commit阶段**：
   * 同理，投票收集在完成。
   * **时间消耗**：。

**总时间**：（延迟被Δ覆盖）。

## 2. 敌手延迟（触发超时）

### 2.1 Leader节点被Corrupted

#### 2.1.1 现实世界执行

* **敌手行为**：
  + 控制Leader节点，在Prepare阶段延迟提案至，触发超时。
  + 强制副本调用NEW-VIEW进入下一视图。
* **协议流程**：
  1. **视图切换**：副本检测超时后递增，更新。
  2. **新视图共识**：新Leader（可能诚实）在超时窗口内完成共识。

#### 2.1.2 理想世界模拟

* **模拟器操作**：
  1. **超时触发**：当检测到时，调用TIMEOUT(curView)。
  2. **视图递增**：递增并广播NEW-VIEW消息，通过同步所有副本。
  3. **参数更新**：，与真实协议严格一致。

#### 2.1.3 不可区分性分析

* **现实世界**：
  + **协议执行过程**：
    - 腐败Leader延迟提案至，触发超时。
    - 副本通过次视图切换（总时间）进入诚实视图。
    - 最终视图在时间内完成共识。
  + **环境视角**：
    - 观察到超时事件及视图切换请求按指数退避规则递增。
    - 最终决策在时提交。
* **理想世界**：
  + **协议执行过程**：
    - 模拟器通过检测超时，严格递增视图和。
    - 新视图的Prepare/PreCommit/Commit阶段超时设为。
    - QC生成和消息广播通过同步。
  + **环境视角**：
    - 视图切换次数、超时参数更新与真实世界完全一致。
    - 最终决策时间与公式严格匹配。
* **现实世界与理想世界的对比**：
  + **时间一致性**：
    - 每次视图切换的超时时间和总时间完全同步。
    - 最终诚实视图的三阶段耗时在两种世界中相同。
  + **输出一致性**：
    - 环境无法通过视图切换日志或QC链检测差异。
    - 敌手的超时触发行为被理想功能完美模拟为网络延迟。

#### 2.1.4 时间计算

1. **Prepare阶段超时**：
   * 敌手延迟提案至。
   * **首次超时时间**：。
2. **视图切换**：
   * 第次切换的超时时间：。
   * **总切换时间**：。
3. **最终诚实视图**：
   * Prepare/PreCommit/Commit阶段各耗时。
   * **总时间**：。

**全局时间上界**：

### 2.2 普通副本被Corrupted（敌手控制个副本）

#### 2.2.1 现实世界执行

* **敌手行为**：
  + 控制个副本，在Prepare/PreCommit阶段延迟投票至，触发超时。
  + 发送无效NEW-VIEW消息干扰视图同步。
* **协议流程**：
  1. **超时触发**：诚实副本因未收到足够投票触发TIMEOUT。
  2. **视图切换**：需至少个NEW-VIEW消息（含个伪造消息+1个诚实节点超时）。

#### 2.2.2 理想世界模拟

* **模拟器操作**：
  1. **伪造消息处理**：验证NEW-VIEW消息的合法性，仅同步包含诚实节点的请求。
  2. **QC同步**：当进入新视图时，调用ADVANCE-PHASE()重置阶段，确保与真实协议一致。

#### 2.2.3 不可区分性分析

* **现实世界**：
  + **协议执行过程**：
    - 腐败副本延迟投票至，触发次视图切换。
    - 最终诚实Leader在超时窗口内完成各阶段QC生成。
  + **环境视角**：
    - 观察到超时事件由腐败副本的投票缺失引起。
    - 视图切换后的QC链包含前序哈希，确保状态一致性。
* **理想世界**：
  + **协议执行过程**：
    - 模拟器丢弃无效的NEW-VIEW消息，仅同步合法请求。
    - 通过强制重置阶段变量，确保与真实协议状态同步。
  + **环境视角**：
    - 视图切换请求的过滤逻辑与真实网络行为不可区分。
    - 最终QC的生成时间和内容与真实世界一致。
* **现实世界与理想世界的对比**：
  + **时间一致性**：
    - 每次视图切换的耗时和总时间严格匹配。
    - 腐败副本的延迟攻击被限制在指数退避框架内。
  + **输出一致性**：
    - 环境观测到的视图切换次数、QC哈希链及提交时间完全一致。
    - 理想功能的状态重置操作与现实协议的本地变量更新无差异。

#### 2.2.4 时间计算

1. **Prepare阶段超时**：
   * 腐败副本延迟投票至。
   * **首次超时时间**：（等待提案+投票超时）。
2. **视图切换**：
   * 第次切换的超时时间：。
   * **总切换时间**：。
3. **最终诚实视图**：
   * Prepare/PreCommit/Commit阶段各耗时。
   * **总时间**：。

**全局时间上界**：

## 3. 敌手在最终诚实视图（）的攻击

### 3.1 现实世界执行

* **敌手限制**：
  + 无法控制该视图的Leader（诚实节点），仅能延迟消息至。
* **协议流程**：
  1. **Prepare阶段**：诚实Leader广播提案，所有副本在超时窗口内接收。
  2. **QC生成**：即使敌手延迟投票，诚实副本仍可收集票生成QC。
  3. **决策提交**：最终在时间内达成共识。

### 3.2 理想世界模拟

* **模拟器操作**：
  1. **超时窗口扩展**：设置，匹配敌手最大延迟。
  2. **延迟投票注入**：腐败副本的投票在内被调度，确保QC生成。
  3. **提交同步**：调用DECIDE接口提交决策，并广播。

### 3.3 不可区分性分析

* **超时覆盖性**：理想世界的超时窗口完全覆盖敌手延迟，与真实协议行为一致。
* **最终性保证**：两种世界均在时间内提交，且QC链式哈希不可逆。
* **结论**：环境无法通过最终视图的时序或QC内容区分世界。

## 三、全局时间分析

## 分项验证与阶段拆解

### 1. 恶意视图切换时间

#### 计算逻辑

* **视图切换次数**：敌手最多控制个节点，需次切换以确保进入诚实Leader视图。
* **指数退避规则**：每次切换后超时时间翻倍：
  + 视图的超时时间：。
* **总时间**：

#### 分阶段验证

| 视图编号 | 超时时间 | 累计时间（） |
| --- | --- | --- |
| 0 |  |  |
| 1 |  |  |
| 2 |  |  |
| ... | ... | ... |
|  |  |  |

### 2. 最终诚实视图时间

#### 计算逻辑

* **视图的超时参数**：。
* **协议阶段**：
  1. **Prepare阶段**：等待提案和投票，超时时间为。
  2. **PreCommit阶段**：等待QC生成，超时时间为。
  3. **Commit阶段**：等待最终提交，超时时间为。
* **总时间**：

#### 分阶段验证

| 阶段 | 超时时间 | 累计时间（） |
| --- | --- | --- |
| Prepare |  |  |
| PreCommit |  |  |
| Commit |  |  |

### 3. 全局时间上界

#### 综合计算

* **总时间**：
* **大O简化**：
  + 高阶项为，故。

## 四、不可区分性证明

### 1. 消息传递阶段（映射1）

#### 真实世界行为

* 腐败节点发送消息，可能延迟。
* 诚实节点按协议在时间内广播消息。

#### 理想世界模拟

* 调用BROADCAST(m)，将加入队列。
* 计算交付时间：

#### 不可区分性计算

* **消息时序一致性**：
  + 对任意消息，观测到的时间差为：
  + 若敌手尝试施加的延迟 **未超过时间上界**（即 ），则 ，因此 。这意味着真实世界与理想世界的消息延迟差值为零，环境 无法观察到任何差异。
    - 若敌手试图施加**超过上界的延迟**（即 ），则 。然而，这种情况被协议主动排除：超时机制会触发视图切换，腐败节点的消息将被标记为过期或无效，敌手的延迟攻击被中止。
* **通信内容一致性**：
  + 腐败节点的无效消息被过滤（通过签名验证），诚实消息内容完全一致。
  + 引入变量对延迟时间和超时时间进行记录，此修改存储其他信息，但不会改变通信，因此无法区分。

### 2. 超时检测与视图切换（映射2）

#### 真实世界行为

* 若副本在时间内未收到消息，触发超时并广播NEW-VIEW。
* 腐败节点可能伪造超时事件。

#### 理想世界模拟

* 通过检测到。
* 调用递增视图：curView := curView + 1。
* 更新超时参数：。
* 调用NEW-VIEW(curView)，广播视图切换消息，调用实现同步视图切换。

#### 不可区分性

* 视图切换次数、时序、消息内容在两种世界中完全一致
* 无法获取节点本地超时检测和达成视图同步的内部状态，仅能观测网络消息

### 3. QC生成与阶段推进（映射3）

#### 真实世界行为

* QC生成需满足个合法投票，且链式哈希链接。
* 腐败节点可能发送无效投票。

#### 理想世界模拟

* 监控中的投票，若合法数，调用GENERATE-QC。
* 腐败节点的无效投票被丢弃。

#### 不可区分性计算

* **QC生成条件**：
  + 真实世界合法投票数：
  + 理想世界合法投票数：
  + 由于，有，与真实世界一致。
  + 消息计数逻辑与真实协议一致，无法察觉监控行为
* **哈希链一致性**：
  + 真实世界的QC哈希链接由协议规则保证。
  + 理想世界中，强制所有QC必须包含前序哈希，否则拒绝生成。