

协议特点

FastBFT是一个分布式共识机制：

1. 借助TEE(Trusted Execution Environment)可将网络的节点数量由 $3f + 1$ 降为 $2f + 1$ ，不存在拜占庭节点
2. 网络节点按照平衡树结构组成拓扑，根节点为 $primary$
3. 通过轻量 Secret Sharing 协议和树结构，可将通信复杂度由 $O(n^2)$ 降为 $O(n)$ ， $n = 2f + 1$

假设区块大小为1MB，每笔交易平均长度为250Byte，FastBFT可实现大于100,000笔交易每秒的吞吐量。

基础概念

SMR(State Machine Replication)

SMR模型，即每个节点都是其它节点的状态副本，通过共识算法保证所有正常节点完成相同的状态切换。每个节点都有可能会宕机，或者作恶(即拜占庭节点)。宕机导致无响应，拜占庭节点则会在一轮共识过程中，向不同节点发送不同甚至相互矛盾的信息，试图引起共识失败，称此行为叫**混淆(Equivocation)**。

Fault-Tolerant SMR必须满足两个性质

1. Liveness，一定达成共识
2. Safety，共识一定正确

FLP论文已经证明，在异步通信模型中，即不保证消息在确定时间范围内一定到达，没有算法可以保证实现Fault-Tolerant SMR。pBFT通过弱同步假设，即假设消息在一个时间限制之内一定到达，绕过了FLP不可能难题。

两种分类

为了在pBFT的基础之上进一步提高网络的吞吐量和网络规模，科学家设计一系列BFT算法，可将它们分作两类

1. Speculative，副节点没有共识过程，直接执行主节点广播的用户请求操作，所有副节点直接返回结果给用户。如果主节点故意混淆，则用户会收到不一致的结果，诚实的用户会向副节点反馈存在状态不一致的问题？帮助网络恢复到一致的状态。
2. Optimistic，在所有的副节点中只有一部分会执行共识算法，在完成用户的请求之后，由主节点向用户返回结果，并通知其余副节点同步状态。

FastBFT属于Optimistic类型

TEE安全机制

TEE是一种硬件安全机制，提供了受保护和隔离的执行环境，无论是操作系统还是其它软件都无法控制或者观察到数据如何在TEE中被存储或者被处理。同时，TEE允许其它设备上的TEE远程核查其状态和行为，这意味着TEE只有可能宕机崩溃，而无法成为拜占庭节点。

通过在本地的TEE中维护一个连续增长的整数计数器，要求节点必须将每条消息与唯一的计数器数值绑定。因为无法实现混淆，即无法将同一个数值绑定到不同的消息中，可禁止拜占庭节点的出现，实现原理见后。如此便将节点数量由 $3f + 1$ 减至 $2f + 1$ ， f 是最大错误节点数量。

聚合消息

在pBFT中，副本节点会向所有节点广播消息，因此通信复杂度为 $O(n^2)$ 。通过消息聚合机制，每个副本节点仅需接受和发送少量消息，Fast通过Secret Sharing和树拓扑实现集合消息，可将通信复杂度降为 $O(n)$ 。

- Secret Sharing：秘密分享值得是将一条完整的信息，分成若干部分，并将每一部分安全保密地交给不同的对象，每部分的持有者无法仅从自己掌握的部分信息还原成完整的信息，只有所有持有者或者不少于某个数值的持有者人数，彼此公布秘密信息，才能完整回复秘密。两类常见的秘密分享算法分别基于多项式运算和XOR运算，后者较为简单，为此论文所采用。
- 树拓扑：为了提高通信效率和减少通信冗余，执行共识算法的节点按照平衡树的结构组成网络拓扑，树的根节点为主节点，树上的节点接收来自子节点的信息，向父节点或者根节点发送消息，例如满二叉树

FastBFT Setup

FastBFT基于弱同步通信模型，每个加入网络的节点需要维护一个TEE，用来与其它节点建立通信。论文假设，节点可能作恶，但是TEE只能崩溃。

假设网络的节点数量为 $2f + 1$ ， $f + 1$ 个节点组成的活跃节点集合 $\{S_i\}$ 执行共识过程，剩下 f 个节点组成的非活跃节点集合 $\{S'_i\}$ 接收共识结果。另外存在一个主节点 S_p 负责预处理，接收、广播用户请求，向用户和非活跃节点返回结果。

系统模型

1. **pre-processing**： S_p 生成由若干条随机秘密组成的集合 $\{s_c\}$ ，并向所有节点公布每条秘密的哈希值 $\{h_c\}$ 。然后， S_p 将每条完整的秘密分成 $f + 1$ 份，并安全保密地发送给 $f + 1$ 个活跃节点。每条秘密仅仅使用一次。
2. **prepare**： S_p 接收一条来自用户 C 的请求，并将其和一条从没使用过的秘密绑定，并向 $\{S_i\}$ 广播
3. **commit**：每个活跃节点 S_i 通过聚合消息向 S_p 公布自己掌握的秘密，作为其对用户请求的Commitment。一旦 S_p 收集到了足够数量来自活跃节点的秘密，便能够回复出完整秘密 s_c ，并向 $\{S_i\}$ 公开， $\{S_i\}$ 通过验证 $s_c = h_c$ 来判断共识是否达成。
4. **reply**：一旦 $\{S_i\}$ 验证 $s_c = h_c$ 通过，便执行用户请求，将结果返回给 S_p ，同时再次通过集合消息向 S_p 公布下一条秘密。如果 S_p 收集到足够数量的下一条秘密信息，便确认了共识已经完成，将结果返回给用户，并通知所有非活跃节点 $\{S'_i\}$ 同步结果。

很明显，此模型中 S_p 的角色极为关键，因为其实现掌握了所有的秘密信息，也知道 $\{S_i\}$ 掌握的所有秘密，因此它可以伪装成任何活跃节点。这个问题将被TEE解决。

FastBFT要求每个节点必须在本地TEE中维护一个单调连续增长的整数计数器，同时主节点的TEE还负责生成随机的完整的秘密信息，向其它节点的TEE安全保密地分发部分的秘密信息。

因为所有的秘密都是由 S_p 在**pre-processing**阶段生成，因此 S_p 可以为不同的用户请求使用相同的秘密信息，这样存在安全风险(阅读后文可以理解，这样使得恶意节点能够伪造其它节点的共识响应，因为秘密在**commit**阶段已公开，全体可知)。为了禁止 S_p 这么做，在**pre-processing**阶段，每条秘密将与TEE中计数器的一个唯一数值绑定，在**prepare**阶段，用

这么做的结果就是每条密码将于一个用户请求绑定，全体副本节点需要追踪主节点计数器的最新值 C_{latest} ，每次完成用户请求之后需要更新最新值。这里的关键要求就是秘密信息与计数值必须保持一一对应的关系

通信拓扑

尽管借助秘密分享机制已经实现了通信复杂度降为 $O(n)$ ，但主节点面临的通信负载是最大的，FastBFT还可以进一步突破此瓶颈，将通信负载均匀地分摊到所有活跃节点。

将 S_p 和 $\{S_i\}$ 组成成平衡树网络拓扑，根节点为 S_p ，按照自底向上的方式分担通信负载：叶节点向父节点公布自己的秘密，中间节点收集来自子节点的聚合信息，再补充自己上的秘密信息，然后向父节点发送，最终 S_p 仅需要接收一个少量的集合消息即可。

错误检测

FastBFT还提供了一种检测机制用来检测 $non-primary$ 错误，即某活跃节点 S_i 发生宕机：父节点可以且尽可以报告其子节点可能出现问题，向上发送一个 $SUSPECT$ 消息。一旦收错误报告， S_p 将会从非活跃节点集合 $\{S'_i\}$ 中挑选一个节点替换掉被报告的问题节点，同时并将报告节点置于叶节点，以避免其继续报告。

FastBFT算法细节

一些注解

符号	解释
C	用户
S	节点
n	节点总数量
f	最大容错数量
p	主节点编号
v	视图编号
c	虚拟计数器数值
C	硬件计数器数值
$H()$	哈希函数
h	哈希值
$E()/D()$	对称加密/解密函数
k	用于对称加解密的密钥
ϱ	对称加密之后的密文
$Enc()/Dec()$	非对称加密/解密函数
ω	公钥之后加密的密文
$Sign()/Vfy()$	签名与验证函数
$\langle x \rangle_{\sigma_i}$	S_i 对 x 的数字签名结果

TEE函数

persistent variables:

maintained by all replicas

(c_{latest}, v)

maintained only by primary

$\{S_i, k_i\}$

maintained only by active replica S_i

k_i

function *be_primary*($\{S'_i\}, T'$) \triangleright called by S_p

$\{S_i\} := \{S'_i\} \quad T := T' \quad v := v + 1 \quad c := 0$

foreach S_i in $\{S_i\}$

$k_i \leftarrow \{0, 1\}^l \quad \triangleright$ key for encrypting secrets

$\omega_i \leftarrow Enc(k_i) \quad \triangleright$ encrypt the key by public key of S_i

return $\{\omega_i\}$

function *update_view*($\langle x, (c, v) \rangle_{\sigma_p}, \omega_i$) \triangleright called by S_i

if $Vfy(\langle x, (c, v) \rangle_{\sigma_p}) = 0$ **return** invalid signature

elseif $c \neq c_{latest} + 1$ **return** invalid counter

else $c_{latest} := 0 \quad v := v + 1$

if S_i in active

$k_i \leftarrow Dec(\omega_i)$

```

function preprocessing( $m$ )  $\triangleright$  called by  $S_p$  for  $m$  requests
  for  $1 \leq a \leq m$ 
     $c := c_{latest} + a$   $s_c \leftarrow \{0, 1\}^l$   $h_c \leftarrow H(\langle s_c, (c, v) \rangle)$ 
     $s_c^1 \oplus \dots \oplus s_c^{f+1} \leftarrow s_c$ 
    foreach active replica  $S_i$ 
      foreach direct children  $S_j$  of  $S_i$ 
         $\hat{h}_c^j := H(s_c^j \oplus_{k \in \phi_j} s_c^k)$   $\triangleright \phi_j$  are descendants(subtree) of  $S_j$ , aggregation of secret information
         $\varrho_c^i \leftarrow E(k_i, \langle s_c^i, (c, v), \{\hat{h}_c^j\}, h_c \rangle)$   $\triangleright$  cardinality of  $\{\hat{h}_c^j\}$  is the total direct children of  $S_i$ 
         $\langle h_c, (c, v) \rangle_{\sigma_p} \leftarrow \text{Sign}(\langle h_c, (c, v) \rangle)$ 
      return  $\{\langle h_c, (c, v) \rangle_{\sigma_p}, \{\varrho_c^i\}_c$   $\triangleright$  cardinality of  $\{\}_c$  is  $m$ , cardinality of  $\{\}_i$  is  $f + 1$ 

function request_counter( $x$ )  $\triangleright$  called by  $S_p$ , counter for REQUEST messages from  $C$ 
   $c_{latest} := c_{latest} + 1$ 
   $\langle x, (c_{latest}, v) \rangle_{\sigma} \leftarrow \text{Sign}(\langle x, (c_{latest}, v) \rangle)$ 
  return  $\langle x, (c_{latest}, v) \rangle_{\sigma}$ 

function verify_counter( $\langle x, (c', v') \rangle_{\sigma_p}, \varrho_c^i$ )  $\triangleright$  called by active  $S_i$ 
  if  $Vfy(\langle x, (c', v') \rangle_{\sigma_p}) = 0$  return invalid signature
  elseif  $\langle s_c^i, (c'', v''), \{\hat{h}_c^j\}, h_c \rangle \leftarrow D(\varrho_c^i)$  failed return invalid encryption
  elseif  $(c', v') \neq (c'', v'')$  return invalid counter value
  elseif  $c' \neq c_{latest} + 1$  return invalid counter value
  else  $c_{latest} := c_{latest} + 1$  return  $\langle s_c^j, \{\hat{h}_c^j\}, h_c \rangle$ 

function update_counter( $s_c, \langle h_c, (c, v) \rangle_{\sigma_p}$ )  $\triangleright$  called by passive  $S_i$ 
  if  $Vfy(\langle h_c, (c, v) \rangle_{\sigma_p}) = 0$  return invalid signature
  elseif  $c \neq c_{latest} + 1$  return invalid counter
  elseif  $H(\langle s_c, (c, v) \rangle) \neq h_c$  return invalid secret
  else  $c_{latest} := c_{latest} + 1$ 

function reset_counter( $\{L_i, \langle H(L_i), (c', v') \rangle_{\sigma_p}\}$ )
  if  $\geq f + 1$  consistent  $L_i, (c', v')$  choose  $f + 1$  new active replicas  $S_i$  and construct a tree  $T'$ 
     $c_{latest} := c'$   $v := v'$ 

```

- *be_primary*：进入下一个视图，并选择一个编号为 $p = v \bmod n$ 的节点成为主节点，重置计数值，并且为每一个 S_i 生成一个会话私钥 k_i 。
- *update_view*：所有节点切换视图，活跃节点获得会话私钥。
- *pre_processing*：预处理阶段为每一个计数值 c ，生成一条秘密 s_c ，并计算哈希值 h_c ，同时将秘密 s_c 分成 $f + 1$ 份，并为每个活跃节点 S_i 计算其所有子节点的集合秘密信息的哈希值 $\{\hat{h}_c^j\}$ ，用来查验所有后代节点的秘密是否准确，每个节点的聚合秘密信息是由自己的秘密和所有的后代节点的秘密经过 \oplus 运算获得，以上信息通过会话密钥 k_i 加密后发送给 S_i 。
- *request_counter*：提高计数值 c_{latest} ，并且将其和视图 v 还有输入信息 x 绑定
- *verify_counter*：首先验证 S_p 的签名，然后确认 S_p 发送的秘密信息的完整性，接着验证秘密信息携带的计数值是否等于本地计数值+1，然后提高本地计数值用以选择下一条秘密(保证使用相同的秘密进行共识确认)
- *update_counter*：非活跃节点用来同步计数值

以上函数中，*be_primary* 和 *update_view* 主要用于视图切换，*pre_processing* 为共识生成了一系列的带编号的秘密用于在共识过程中公开 Commitment，*_counter 系列函数用于共识过程中各个节点同步使用相同的秘密。

共识过程(Normal-case operation)

```

upon invoke PREPROCESSING at  $S_p$  do
   $\{\langle h_c, (c, v) \rangle_{\sigma_p}, \{\varrho_c^i\}_c \leftarrow TEE.preprocessing(m)$ 
  foreach active  $S_i$ , send  $\{\varrho_c^i\}_c$  to  $S_i$   $\triangleright$  cardinality of  $\{\}_c$  is  $m$ , send all secrets to  $S_i$ 

```

upon reception of $M = \langle REQUEST, op \rangle_{\sigma_c}$ at S_p **do**
 $\langle H(M), (c, v) \rangle_{\sigma_p} \leftarrow TEE.request_counter(H(M))$
multicast $\langle PREPARE, M, \langle H(M), (c, v) \rangle_{\sigma_p} \rangle$ to all active S_i

upon receipt $\langle PREPARE, M, \langle H(M), (c, v) \rangle_{\sigma_p} \rangle$ at S_i **do**
 $\langle s_c^i, \{\hat{h}_c^j\}, h_c \rangle \leftarrow TEE.verify_counter(\langle H(M), (c, v) \rangle_{\sigma_p}, \varrho_c^i)$
 $\hat{s}_c^i := s_c^i$
if S_i is a leaf node, send \hat{s}_c^i to its parent
else set timers for its direct children

upon timeout of share from S_j at S_i **do**
send $\langle SUSPECT, S_j \rangle$ to its parent \triangleright to parent of S_i

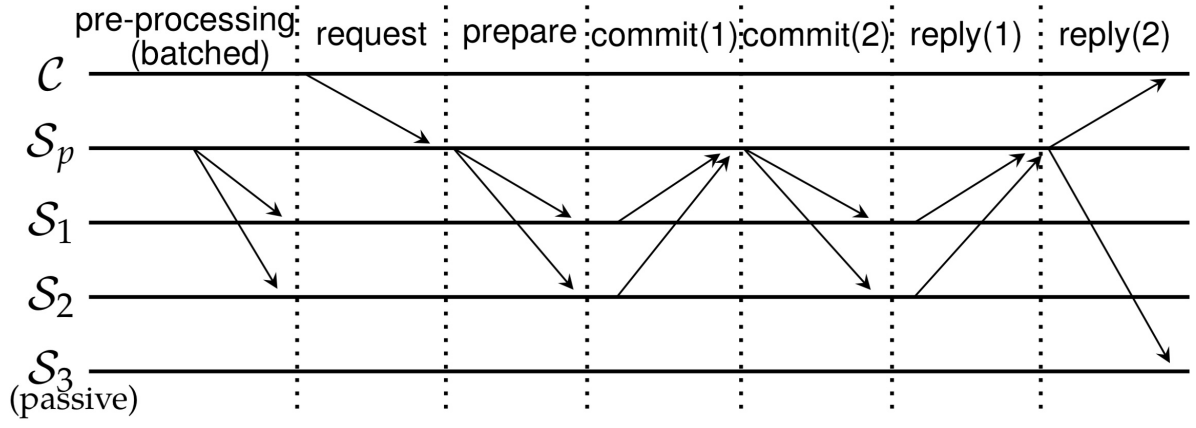
upon receipt \hat{s}_c^j at S_i or S_p **do**
if $H(\hat{s}_c^j) = \hat{h}_c^j$
 $\hat{s}_c^i := \hat{s}_c^i \oplus \hat{s}_c^j$
else send $\langle SUSPECT, S_j \rangle$ to its parent
if S_i has received all valid $\{\hat{s}_c^j\}_j$
send \hat{s}_c^i to its parent
if S_p has received all valid $\{\hat{s}_c^j\}_j$
if s_c is for the commit phase
 $res \leftarrow exe(op)$
 $x \leftarrow H(M || res)$
send $\langle COMMIT, s_c, res, \langle x, (c+1, v) \rangle_{\sigma_p} \rangle$ to all active S_i
elseif s_c is for the reply phase
send
 $\langle REPLY, M, res, s_{c-1}, s_c, \langle h_{c-1}, (c-1, v) \rangle_{\sigma_p}, \langle h_c, (c, v) \rangle_{\sigma_p}, \langle H(M), (c-1, v) \rangle_{\sigma_p}, \langle H(M || res), (c, v) \rangle_{\sigma_p} \rangle$
to C and all passive replicas

upon receipt $\langle SUSPECT, S_k \rangle$ from S_j at S_i **do**
if $i = p$
generate new tree T' replacing S_k with a passive replica and placing S_j at a leaf
 $\langle H(T || T'), (c, v) \rangle_{\sigma_p} \leftarrow TEE.request_counter(H(T || T'))$
broadcast $\langle NEW - TREE, T, T', \langle H(T || T'), (c, v) \rangle_{\sigma_p} \rangle$
else cancel timer for S_j and forward the $SUSPECT$ message up

upon receipt $\langle COMMIT, s_c, res, \langle H(M || res), (c+1, v) \rangle_{\sigma_p} \rangle$ at S_i **do**
if $H(s_c) \neq h_c$ or $exe(op) \neq res$
broadcast $\langle REQ - VIEW - CHANGE, v, v' \rangle$
 $\langle s_{c+1}^i, \{\hat{h}_{c+1}^j\}, h_{c+1} \rangle \leftarrow TEE.verify_counter(\langle H(M || res), (c+1, v) \rangle_{\sigma_p}, \varrho_c^i)$
if S_i is leaf
send s_{c+1}^i to its parent
else $\hat{s}_{c+1}^i := s_{c+1}^i$, set timers for its direct children

upon receipt
 $\langle REPLY, M, res, s_c, s_{c+1}, \langle h_c, (c, v) \rangle_{\sigma_p}, \langle h_{c+1}, (c+1, v) \rangle_{\sigma_p}, \langle H(M), (c, v) \rangle_{\sigma_p}, \langle H(M || res), (c+1, v) \rangle_{\sigma_p} \rangle$
at S_i **do**
if $H(s_c) \neq h_c$ or $H(s_{c+1}) \neq h_{c+1}$
multicast $\langle REQ - VIEW - CHANGE, v, v' \rangle$
else \triangleright update state based on res
 $TEE.update_counter(s_c, \langle h_c, (c, v) \rangle_{\sigma_p})$
 $TEE.update_counter(s_{c+1}, \langle h_{c+1}, (c+1, v) \rangle_{\sigma_p})$

以上算法描述了一次正确的共识过程中，各个节点如何同步状态切换的。



- **Preprocessing** : S_p 为可能出现的 m 条用户请求生成了 m 条秘密，并赋予了在当前视图下的唯一编号，并将每条秘密分成 $f + 1$ 份发送给活跃节点 S_i ，在活跃节点公布自己的秘密之前，其它活跃节点无从得知其内容。
- **Request** : 用户 C 发送请求信息 $M = \langle REQUEST, op \rangle_{\sigma_C}$ 给 S_p ，并计时等待其回复，如果超时未应答，则向所有节点广播 M 。
- **Prepare** : 当 S_p 收到 M ，便调用 $request_counter(H(M))$ ，为 M 赋予一个计数值 (c, v) ，由于每个 (c, v) 均与一条唯一的秘密 s_c 绑定，此操作即将用户请求与秘密绑定。 S_p 向所有活跃节点 $\{S_i\}$ 广播 $\langle PREPARE, M, \langle H(M), (c, v) \rangle_{\sigma_p} \rangle$ ，此时称作 M is prepared。
- **Commit** : 当 S_i 收到 $PREPARE$ 消息，便调用 $verify_counter(\langle H(M), (c, v) \rangle_{\sigma_p}, \rho_c^i)$ ，验证与 M 绑定的计数值是否与解密 ρ_c 之后获得的计数值相同，如果相同则意味着 $M, (c, v), s_c$ 三者绑定正确， S_i 准备公开其掌握的部分秘密，以及集合信息的哈希和完整的秘密哈希 $\langle s_c^i, \{\hat{h}_c^j\}, h_c \rangle$ 。
如果 S_i 是叶子节点，直接发送 s_c^i 给其父节点；否则 S_i 等待接收每一个子节点 S_j 发来的聚合信息 \hat{s}_c^j ，并验证 $H(\hat{s}_c^j) = \hat{h}_c^j$ ，如果所有子节点均验证通过，则计算 $\hat{s}_c^i = s_c^i \oplus_{j \in \phi_i} \hat{s}_c^j$ ， ϕ_i 表示 S_i 所有子节点的集合，并向父节点发送 \hat{s}_c^i ，直到 S_p 接收到所有子节点的集合信息便能恢复 s_c 。
恢复 s_c 之后， S_p 执行 op 并将结果 res 返回给所有活跃节点 $\{S_i\}$ ，即广播 $\langle COMMIT, s_c, res, \langle H(M || res), (c + 1, v) \rangle_{\sigma_p} \rangle$ ，此时称作 M is committed。

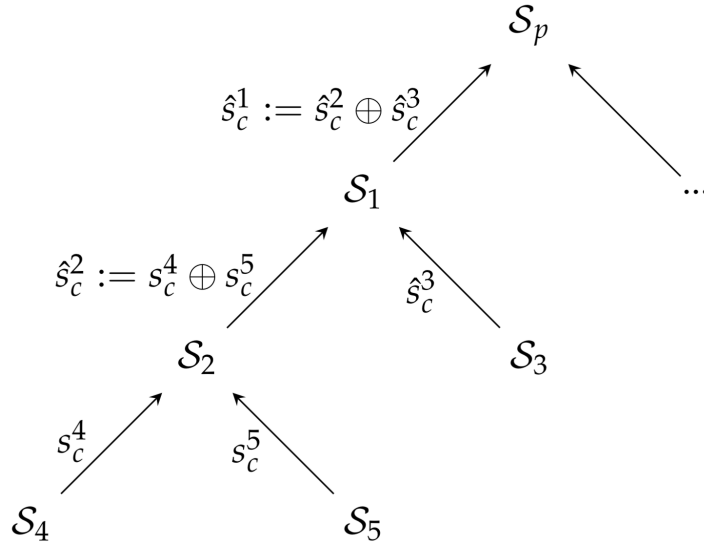


Fig. 5: Communication structure for the commit/reply phase.

- **Reply** : 当 S_i 收到 $COMMIT$ 消息，便会验证 $H(s_c) = h_c$ ，验证通过则执行用户请求，并验证 $exe(op) = res$ ，如果再次通过，则后用下一条秘密 s_{c+1} ，并通过类似 **Commit** 阶段的流程，向 S_p 答复集合消息。当 S_p 再次收集到足够的集合信息，并恢复出 s_{c+1} ，便向用户和所有非活跃节点 $\{S'_i\}$ 发送 $\langle REPLY, M, res, s_c, s_{c+1}, \langle h_c, (c, v) \rangle_{\sigma_p}, \langle h_{c+1}, (c + 1, v) \rangle_{\sigma_p}, \langle H(M), (c, v) \rangle_{\sigma_p}, \langle H(M || res), (c + 1, v) \rangle_{\sigma_p} \rangle$ 。此时称作 M is replied。用户将验证

- 一个有效的 $\langle h_c, (c, v) \rangle_{\sigma_p}$ 表示 (c, v) 和 s_c 绑定
- 一个有效的 $\langle H(M), (c, v) \rangle_{\sigma_p}$ 表示 (c, v) 和 M 绑定
- 因此 M 和 s_c 绑定
- 一个有效的 $s_c : H(s_c, (c, v)) = h_c$ 表示所有活跃节点均同意将在计数值为 c 的条件下执行操作 op
- 一个有效的 $s_{c+1} : H(s_{c+1}, (c+1, v)) = h_{c+1}$ 表示所有活跃节点执行完成 op 之后的结果均为 res

于是，共识完成。

错误检测

由于活跃节点存在宕机的可能性，引起共识失败，网络需要检测到问题的出现，并提换掉该错误节点。FastBFT通过计时器来检测可能宕机的节点，每个节点 S_i 等待来自子节点 S_j 的聚合信息时，会针对其启动一个计时器，如果超时未获得响应，或者聚合信息未能通过验证 $H(\hat{s}_c^j) = \hat{h}_c^j$ ，便向 S_i 的父节点报告 $\langle SUSPECT, S_j \rangle$ ，如果父节点从子节点处接收到一条报告，便会取消针对该子节点的计时，并继续向上转发错误报告。

切换视图

upon receipt $f+1 \langle REQ - VIEW - CHANGE, L, \langle H(L), (c, v) \rangle_{\sigma_i} \rangle$ messages at new primary $S_{p'}$
do

build execution history O based on message logs $\{L\}$
choose $f+1$ new active replicas $\{S'_i\}$ and construct a tree T'
 $\langle H(O||T'), (c+1, v) \rangle_{\sigma_{p'}} \leftarrow TEE.request_counter(H(O||T'))$
 $\{\omega_i\} \leftarrow TEE.be_primary(\{S'_i\}, T')$
broadcast $\langle NEW - VIEW, O, T', \langle H(O||T'), (c+1, v) \rangle_{\sigma_{p'}}, \{\omega_i\} \rangle$

upon receipt $\langle NEW - VIEW, O, T', \langle H(O||T'), (c+1, v) \rangle_{\sigma_{p'}}, \{\omega_i\} \rangle$ at S_i **do**

if O is valid
 $\langle H(O||T'), (c+1, v) \rangle_{\sigma_i} \leftarrow TEE.request_counter(H(O||T'))$
broadcast $\langle VIEW - CHANGE, \langle H(O||T'), (c+1, v) \rangle_{\sigma_i} \rangle$

upon receipt $f \langle VIEW - CHANGE, \langle H(O||T'), (c+1, v) \rangle_{\sigma_i} \rangle$ at S_i **do**

execute the requests in O that have not been executed
extract and store information from T'
 $TEE.update_view(\langle H(O||T'), (c+1, v) \rangle_{\sigma_{p'}}, \omega_i)$

当用户超时依旧未能获得来自主节点的回复，便向所有节点广播请求。网络中的其它节点知晓请求，并且超时未能获得来自主节点的 $PREPARE/COMMIT/REPLY$ 消息，便启动视图切换，即广播 $\langle REQ - VIEW - CHANGE, L, \langle H(L), (c, v) \rangle_{\sigma_i} \rangle$ 给所有节点，其中 L 是自上一个检查点以来的所有收到或者回复的消息的日志集合。注意：当非活跃节点判断主节点超时无响应之后，也能够广播 $REQ - VIEW - CHANGE$ 消息。

1. 当新视图下的主节点 $S_{p'}$ 收到 $f+1$ 份 $REQ - VIEW - CHANGE$ 之后，正式启动视图切换请求。
 - 它从消息日志集合 $\{L\}$ 中恢复自上一个检查点(Checkpoint)以来的执行历史记录 O
 - $S_{p'}$ 挑选新的 $f+1$ 个活跃节点组合一个平衡树 T' ，并且为每个活跃节点生成会话密钥
 - 广播 $NEW - VIEW$ 消息
2. 当节点 S_i 收到 $NEW - VIEW$ 消息后，首先检查 O 的有效性(HOW)。验证通过以后广播 $VIEW - CHANGE$ 消息。
3. 当节点 S_i 接收到 f 份 $VIEW - CHANGE$ 消息后，按顺序执行 O 中还未执行的操作请求，在本地恢复平衡树拓扑 T' ，然后更新视图。

视图切换成功。

安全性

引理1

在视图 v 中，如果某个正确节点以计数值 (c, v) 执行了用户操作请求 op ，就不会有其它正确节点再以相同计数值 (c, v) 去执行一个不同的用户操作请求 (c, v) 。

证明如下：

假设存在2个正确的节点 S_i 和 S_j ，它们均在计数值为 (c, v) 时执行了不同的用户操作请求 op_i 和 op_j ，则有如下5种情况：

1. S_i, S_j 均在正常的共识过程(Normal-case)中执行 S_i 和 S_j 。那么，它们必然都收到了有效的 $COMMIT$ 消息，即数字签名 $\langle H(M_i || res_i), (c, v) \rangle_{\sigma_p}$ 和 $\langle H(M_j || res_j), (c, v) \rangle_{\sigma_p}$ 均判定有效。而这是不可能，因为 S_p 的TEE不会以相同的计数值为不同的操作请求进行签名。
2. S_i 在正常的共识过程中执行 op_i ，而 S_j 在视图切换中执行 op_j 。当 S_i 收到 $COMMIT$ 消息时，意味着 S_p 已经收集了足够的集合消息，即 $f + 1$ 个活跃节点 $Q = \{S_i\}$ 公布了手中的 s_{c-1}^i ，并恢复了秘密 s_{c-1} 。在视图切换过程中， S_j 将执行由 $f + 1$ 个新的活跃节点 $Q' = \{S'_i\}$ 提供的用户请求历史记录 O 。又因为任意多数集合必然包含至少一个相同节点 S_k ，即 $S_k \in Q \cap Q'$ ， S_k 将提供与 S_i 接收到的相同的 $PREPARE(c - 1, v)$ 和 $COMMIT(c, v)$ 消息(情况1)，所以 S_j 也必将以 (c, v) 执行用户操作请求 op_i 。
3. S_i, S_j 均在视图切换中执行 op_i 和 op_j 。在视图切换中，所有节点均获得 $f + 1$ 份相同的历史消息记录。因为 S_p 的TEE不可能以相同的计数值 (c, v) 为不同的 $PREPARE$ 消息签名，所以 op_i 和 op_j 的计数值必然不同。

引理2

如果一个正确节点在视图 v 中执行了用户操作请求 op ，则不会有其它的正确节点在没有执行 op 的情况下切换到下一个视图。

假设存在2个正确的节点 S_i 和 S_j ，前者在视图 v 中执行了 op ，后者准备切换到下一视图。分以下两种情况讨论

1. S_i 在共识过程中执行了 op ，则其相应的 $PREAPRE$ 消息必然包含在执行历史记录 O 中。因此，一个正确的 S_j 必然在切换视图之前会执行 op 。
2. S_i 也是在切换视图中执行了 op ，则又可继续分两种情况
 - 在 S_j 切换视图之前， S_i 在切换视图中执行了 op 。因此，至少有 $f + 1$ 节点的消息历史记录中包含了请求 op 。由于 S_j 同样收到了至少 $f + 1$ 个 $REQ - VIEW - CHANGE$ 消息，因此至少存在一个节点 S_k 在其 $REQ - VIEW - CHANGE$ 消息的日志 L 中包含了 op ，正确节点 S_j 将会在视图切换中执行 op 。
 - S_j 切换视图已经完成， S_i 在切换视图中执行了 op 。基于和上一情况相同的原因，可以推出 S_j 必然已经执行了 op 。

定理1

假设一个正确节点 S_i 一次执行了用户操作请求序列 $seq = \langle op_1, \dots, op_m \rangle$ ，则其它所有正确节点必然会执行相同的序列，或者其前缀序列，即 $seq' = \langle op_1, \dots, op_{k \leq m} \rangle$ 。

假设 S_j 执行的序列并未 seq 的前缀，则至少存在一个操作请求 op'_k 不同于 $op_k \in seq$ 。不妨假定， S_j 在视图 v' 中执行了 op'_k ，而 S_i 在视图 v 中执行了 op_k 。如果 $v' = v$ ，则与引理1矛盾；如果 $v' \neq v$ ，则与引理2矛盾。安全性得证。

活性

当主节点是正确的时候，称该视图为稳定的视图。

引理3

在一个稳定视图中，用户的操作请求必将被完成。

因为主节点不作恶，所以它会广播一条有效的 $PREAPRE$ 消息。如果所有活跃节点均行为正确，用户的请求将被执行完成。如果某个活跃节点 S_j 宕机或者出错，它将被其父节点检测到行为异常，从而一个非活跃节点提换掉。只要出错的节点数量不超过 f ，共识总能达成。

引理4

只要 $f + 1$ 个正确节点发起请求，总能切换到一个稳定视图。

假设 $f + 1$ 个正确节点集合 Q 发起了视图切换，则有以下三种情况

1. 新的主节点 S_p 是正确的，而且全体 Q 成员均收到了有效的 $NEW - VIEW$ 消息，则他们成功切换到下一个稳定视图
2. 所有正确节点均没有收到有效的 $NEW - VIEW$ 消息，则将继续发起视图切换
3. 只有少于 $f + 1$ 个正确节点集合 q 收到了有效的 $NEW - VIEW$ 消息，则每个节点无法收集满 f 份 $VIEW - CHANGE$ 消息，于是继续发起视图切换。

基于弱同步通信模型的假设，情况2和3将最终收敛到情况1，即视图切换成功。

定理2

用户的操作情况最终会被完成。

由引理3和4，得到。于是，活性性质得证。