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成的信息，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$2f+1$，$f+1$个节点组成的主动节点集合 $\{S_i\}$执行共识过程，剩下f个节点组成的被动节点集合 $\{S_i'\}$ 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，向用户和被动节点返回结果。

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$f+1$个主动节点。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**：每个主动节点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主动节点的秘密，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，并通知所有被动节点$\{S_p'\}$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主动节点。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主动节点。

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，即某主动节点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将会从被动节点集合$\{S_p'\}$中挑选一个节点提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 $be\_primary(\{S_i'\}, T')$ ▷ called by S_p
$\{S_i\} := \{S_i'\}$ $T := T'$ $v := v + 1$ $c := 0$
foreach S_i in $\{S_i\}$
$k_i \leftarrow \{0,1\}^l$ ▷ key for encrypting secrets
$\omega_i \leftarrow Enc(k_i)$ ▷ encrypt the key by public key of S_i
return $\{\omega_i\}$

function $update\_view(\langle x, (c, v)\rangle_{\sigma_{p'}}, \omega_i)$ ▷ 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 $c_{latest} := 0$ $v := v + 1$
if S_i in active
$k_i \leftarrow Dec(\omega_i)$

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

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

function $verify\_counter(\langle x, (c', v')\rangle_{\sigma_p}, \varrho_c^i)$ ▷ 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 encryptiton
  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 $update\_counter(s_c, \langle h_c, (c,v)\rangle_{\sigma_p})$ ▷ 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 $\ge f+1$ consistent $L_i, (c', v' choose f+1 new active replicas Si\prime and contruct a tree T\prime)$
    $c_{latest} := c' \quad v := v'$

- $be\_primary$：进入下一个视图，并选择一个编号为 $p = v \mod n$ 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，主动节点获得会话私钥。
- $pre\_processing$：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，并为每个主动节点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)
- $update\_counter$：被动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

---

upon invoke PREPROCESSING at S_p **do**
  $\{\langle h_c, (c,v)\rangle_{\sigma_p}, \{\varrho_c^i\}_i\}_c \leftarrow TEE.preprocessing(m)$
  foreach active S_i, send $\{\varrho_c^i\}_c$ to S_i ▷ 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 acitve S_i

upon recept $\langle PREPARE, M, \langle H(M), (c, v) \rangle_{\sigma_p} \rangle$ 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

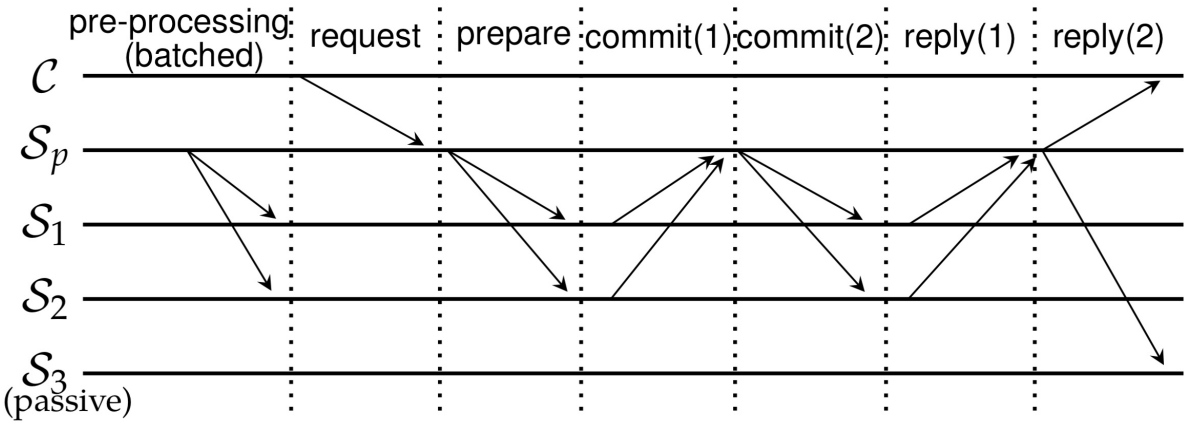
upon recept $\hat{s}_c^j$ at S_i or S_p **do**
  if $H(\hat{s}_c^j) = \hat{h}_c^j$
    $\hat{s}_c^i := \hat{s}_c^i \oplus \hat{s}_c^j$
  else send $\langle SUSPECT, S_j \rangle$ to its parent
  if S_i has received all valid $\{\hat{s}_c^j\}_j$
    send $\hat{s}_c^i$ to its parent
  if S_p has received 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 acitve 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 recept $\langle SUSPECT, S_k \rangle$ 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 cancle timer for S_j and forward the $SUSPECT$ message up

updon recept $\langle COMMIT, s_c, res \langle H(M||res), (c+1, v) \rangle_{\sigma_p} \rangle$ 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 recept
$\langle REPLY, M, res, s_c, s_{c+1}, \langle h_c, (c, v) \rangle_{\sigma_p}, \langle h_{c+1}, (c+1, v) \rangle_{\sigma_p}, \langle H(M), (c, v) \rangle_{\sigma_p}, \langle H(M||res), (c+1, v) \rangle_{\sigma_p} \rangle$
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$f+1$份发送给主动节点S_i，在主动节点公布自己的秘密之前，其它主动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向所有主动节点$\{S_i\}$广播$\langle PREPARE, M, \langle H(M),(c,v)\rangle_{\sigma_p}$，此时称作 $M\ is\ prepared$。
- **Commit**：当S_i收到$PREPARE$消息，便调用$verify_counter(\langle H(M),(c,v)\rangle_{\sigma_p}, \varrho_c^i)$，验证与$M$绑定的计数值是否与解密$\varrho_c$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$\hat{s}_c^i = s_c^i \oplus_{j\in\phi_i} \hat{s}_c^j$，$\phi_i$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返回给所有主动节点$\{S_i\}$，即广播$\langle COMMIT, s_c, res, \langle H(M||res),(c+1,v)\rangle_{\sigma_p}\rangle$，此时称作$M\ is\ commited$。

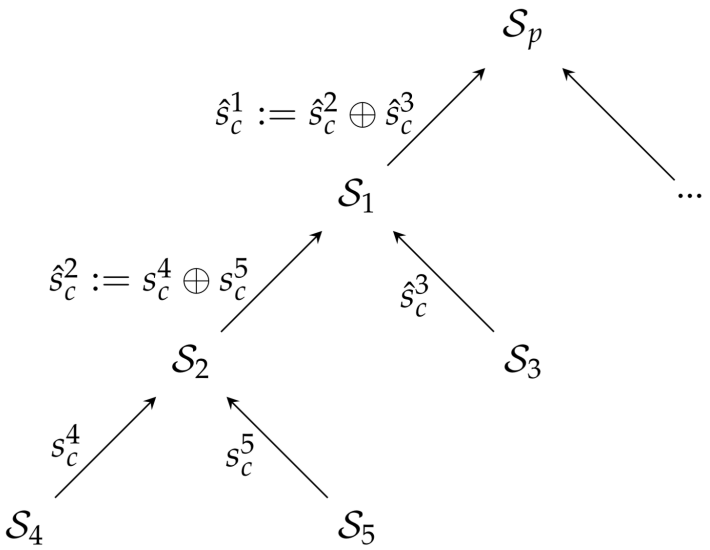


Fig. 5: Communication structure for the commit/reply phase.

- **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，则启用下一条秘密s_{c+1}^i，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}，便向用户和所有被动节点$\{S_i'\}$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：$H(s_c,(c,v))=h_c$表示所有主动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}：$H(s_{c+1},(c+1,v))=h_{c+1}$表示所有主动节点执行完成$op$之后的结果均为$res$

于是，共识完成。

错误检测

由于主动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$\langle SUSPECT,S_j\rangle$，父节点继续向上推送错误报告。

切换视图

---

upon recept $f+1\langle REQ-VIEW-CHANGE,L,\langle H(L),(c,v)\rangle_{\sigma_i}\rangle$ 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 contruct 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 recept $\langle NEW-VIEW,O,T',\langle H(O||T'),(c+1,v)\rangle_{\sigma_{p'}},\{\omega_i\}\rangle$ 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 recept 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$\langle REQ-VIEW-CHANGE,L,\langle H(L),(c,v)\rangle_{\sigma_i}\rangle$，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。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。

论文中提到的视图切换流程过于简陋，还有很多细节需要推敲。