MODULE *NominationPlusCal*

This is a high-level specification of *SCP* focusing on the nomination protocol.

Currently, as implemented, before voting for a txset hash, nodes wait to obtain its preimage. Delaying the point at which we wait for the pre-image would leave more room for disseminating the txset in parallel to nomination. However this has to be done carefully to maintain the main property of nomination: assuming that there is a nomination round with a good leader and during which the network is fast engough, at least a Tier-1 quorum must eventually enter *balloting*.

In the version specified in this document, we do not wait on the pre-image to vote for a txset hash, but we do wait for the pre-image before accepting it.

In the previous version of this document, we even accepted without a pre-image. There is a problem with this: it could create a situation in which not enough nodes can start *balloting* (*i.e.* not a full quorum) and the whole system is stuck.

The problem stems from the fact that, in the nomination protocol, nodes that confirm a candidate then stop voting for new values (otherwise nomination is not guaranteed to converge). So if a blocking set B confirms a candidate but somehow other nodes cannot get the pre-images they need to do so, more nomination rounds will not help because the members of B have stopped voting, which blocks the progress of any new candidate. Depending on how pre-images are disseminated, this can potentially be exploited by an attacker to halt the system.

So accepting without a pre-image is only workable if there is some way to guarantee that, once a Tier-1 blocking set has a pre-image, then everybody in Tier-1 eventually gets it.

Another problem is that we want it to be likely that a quorum starts *balloting* already in agreement and roughly at the same time. If we delay checking pre-images to the confirm stage, an attacker could first send the pre-image to a set A of nodes, which then enter *balloting* at time T_A, but not send the pre-image to another set B of nodes, which then enter *balloting* at time $T\_B \rangle T\_A$ because they need to get the pre-image from A before starting *balloting*. For example, if it takes $500ms$ for members of B to get the pre-image from members of A, then $T\_B = T\_A + 500ms$. This can cause the first ballot to end without a decision. Members of B could also start a new nomination round before T_B and then enter *balloting* not only late but also with a different value than members of A.

EXTENDS *Naturals*, *FiniteSets*

CONSTANTS

- V, validators
- $TxSet$, blocks
- Bot, default value
- $Quorum(\_)$, $Quorum(v)$ is the set of quorums of validator v
- $Blocking(\_)$, $Blocking(v)$ is the set of blocking sets of validator v
- $Combine(\_)$, the functions that combines candidates to produce a txset for *balloting*
- H, domain of hashes
- $Hash(\_)$ hash function

--algorithm *SCP* **{**

variables global variables (*e.g.* representing messages or cross-component variables like $ballotingTxSet$):

$ballotingTxSet = [v \in V \mapsto Bot]$ **;** for each validator, the nominated txset for *balloting*

$decision = [v \in V \mapsto Bot]$ **;** for each validator, the *balloting* decision

$voted = [v \in V \mapsto \{\}]$ **;** X in the whitepaper (nomination Section)

```
        accepted = [v ∈ V ↦ {}] ;   Y in the whitepaper (nomination Section)
    process ( nomination ∈ V )
        variables   local variables:
            round = 0 ;   nothing happens in round 0; the protocol start at round 1
            candidates = {} ;   Z in the whitepaper (nomination Section)
            preImage = [h ∈ H ↦ Bot] ;   the pre-images the validator knows about
            leader = Bot ;   leader for the current round
    {
ln1:    while ( TRUE )
        either {   timeout and go to next round (this also starts round 1)
            round := round + 1 ;
            with ( l ∈ V ) {   pick a leader
                leader := l ;
                if ( l = self )   if the leader is the current node, pick a txset and vote for it
                with ( txs ∈ TxSet ) {
                    preImage[Hash(txs)] := txs ;
                    voted[self] := voted[self] ∪ {Hash(txs)}
                }
            }
        }
        or if ( candidates = {} ) {   vote for what the leader voted for, unless we have a candidate already
            when leader ≠ Bot ;
            with ( hs = voted[leader] ) {
                await hs ≠ {} ;   wait to hear from the leader
                voted[self] := voted[self] ∪ hs   vote for what the leader has voted for
                  in the whitepaper version, we would only vote for the hashes for which we have a pre-image:
                  with (hsWithPreimage = {h ∈ hs : preImage[h] ≠ Bot})
                  voted[self] := voted[self] ∪ hsWithPreimage
            }
        }
        or with ( Q ∈ Quorum(self), h ∈ H ) {   accept when voted or accepted by a quorum and we have the pre-im
            when preImage[h] ≠ Bot ;   we must have received the block
            when ∀ w ∈ Q : h ∈ voted[w] ∨ h ∈ accepted[w] ;   a quorum has voted or accepted h:
            accepted[self] := accepted[self] ∪ {h} ;   accept h
        }
        or with ( Bl ∈ Blocking(self), h ∈ H ) {   accept when accepted by a blocking set and we have the pre-image
            when preImage[h] ≠ Bot ;   we must have received the block
            when ∀ w ∈ Bl : h ∈ accepted[w] ;
            accepted[self] := accepted[self] ∪ {h} ;   accept h
        }
        or with ( txs ∈ TxSet ) {   receive a txset
            preImage[Hash(txs)] := txs ;
        }
        or with ( Q ∈ Quorum(self), h ∈ H ) {   confirm b as candidate
            when preImage[h] ≠ Bot ;   we must have received the block
```

```
            when ∀ w ∈ Q : h ∈ accepted[w] ;   a quorum has accepted h:
            candidates := candidates ∪ {preImage[h]} ;   add h to the confirmed candidates
              update the block used in balloting:
            ballotingTxSet[self] := Combine(candidates) ;   this starts the balloting protocol (see below)
        }
    }
    as a first approximation, balloting just decides on one of the balloting blocks:
    note we cannot reuse the process ID identifiers used in nomination, so we add the "balloting" tag
  process ( balloting ∈ {⟨v, "balloting"⟩ : v ∈ V} ) {
lb1:  await ballotingTxSet[self[1]] ≠ Bot ;   wait for a confirmed candidate from nomination
lb2:  with ( b ∈ {ballotingTxSet[v] : v ∈ V} \ {Bot} ) {
          when ∀ w ∈ V : decision[w] ≠ Bot ⇒ b = decision[w] ;
          decision[self[1]] := b ;
        }
    }
}
```

The type-safety invariant:

$$
\begin{aligned}
&\mathit{TypeOkay} \triangleq \\
&\quad \land\ \mathit{ballotingTxSet} \in [V \to \mathit{TxSet} \cup \{\mathit{Bot}\}] \\
&\quad \land\ \mathit{decision} \in [V \to \mathit{TxSet} \cup \{\mathit{Bot}\}] \\
&\quad \land\ \mathit{voted} \in [V \to \text{SUBSET } H] \\
&\quad \land\ \mathit{accepted} \in [V \to \text{SUBSET } H] \\
&\quad \land\ \mathit{round} \in [V \to \mathit{Nat}] \\
&\quad \land\ \mathit{candidates} \in [V \to \text{SUBSET } \mathit{TxSet}] \\
&\quad \land\ \mathit{preImage} \in [V \to [H \to \mathit{TxSet} \cup \{\mathit{Bot}\}]] \\
&\quad \land\ \mathit{leader} \in [V \to V \cup \{\mathit{Bot}\}]
\end{aligned}
$$

Next we specify a liveness property that we can easily check with the *TLC* model-checker.

This property is that, if a validator v enters *balloting*, then eventually all validators enter *balloting*. This will hold in simple configurations where the whole network is top tier.

For the property to hold, we also need to add fairness assumptions (*e.g.* if a node can vote for a value, it will eventually do so). Unfortunately the TLA+ code generated from the *PlusCal* specification is in a form that makes stating fairness assumptions hard. It seems that we would need to rewrite this in pure TLA+ to tackle liveness.

$$
\begin{aligned}
&\mathit{NominationLiveness} \triangleq \\
&\quad \forall\, v,\, w \in V : \Box(\mathit{ballotingTxSet}[v] \neq \mathit{Bot} \Rightarrow \Diamond(\mathit{ballotingTxSet}[w] \neq \mathit{Bot}))
\end{aligned}
$$

Definition for model-checking:

Concrete hashing for the model-checker:

$$
\begin{aligned}
&\mathit{TestH} \triangleq 1 \,..\, \mathit{Cardinality}(\mathit{TxSet}) \\
&\mathit{TestHash}(b) \triangleq \\
&\quad \text{LET } f \triangleq \text{CHOOSE } f \in [\mathit{TxSet} \to H] : \forall\, \mathit{txs}1,\, \mathit{txs}2 \in \mathit{TxSet} : \mathit{txs}1 \neq \mathit{txs}2 \Rightarrow f[\mathit{txs}1] \neq f[\mathit{txs}2]
\end{aligned}
$$

IN $f[b]$

Debugging canaries:

$Canary1 \triangleq \forall\, v \in V : decision[v] = Bot$
$Canary2 \triangleq \forall\, v \in V : Cardinality(candidates[v]) \leq 1$
$Canary3 \triangleq \forall\, v \in V : ballotingTxSet[v] = Bot$

$TestQuorums \triangleq \{Q \in \text{SUBSET } V : 2 * Cardinality(Q) > Cardinality(V)\}$
$TestBlocking \triangleq \{Bl \in \text{SUBSET } V : Cardinality(Bl) > 1\}$

\ * Modification History
\ * Last modified *Thu Mar* 30 20:16:04 *PDT* 2023 by *nano*
\ * Created *Thu Mar* 30 20:15:37 *PDT* 2023 by *nano*