Checking Validator Set (CheckVS)

As part of the light client, the CheckVS procedure has to check, whether given two headers (whose heigts differ by more than 1), the LightClient can trust the newer header under the assumption it trusted the old one.

Definitions

- header fields
    - $height$
    - $bfttime$: the chain time when the header (block) was generated
    - V: validator set containing validators.
    - $nextV$: next validators
- tp: trusting period
- for a time t, the predicate $correct(v, t)$ is true if the validator v follows the protocol until time t (we will see about recovery later). Similarly we define $faulty(v, t)$.
- For each header h it has locally stored, the LightClient stores whether it trusts h. We write $trust(h) = true$, if this is the case.
- Validator fields. We will write a validator as a tuple (v, p) such that
    - v is the identifier (we assume identifiers are unique in each validator set)
    - p is its voting power

LightClient Trusting Spec

LightClient Invariant

For each LightClient l and each header h: if l has set $trust(h) = true$, then validators that are correct until time $h.bfttime + tp$ have more than two thirds of the voting power in $h.V$. (Or/and $h.nextV$)

Formally,

$$\sum_{\substack{(v,p)\in h.V\wedge \\ correct(v,h.bfttime+tp)}} p > 2/3 \sum_{(v,p)\in h.V} p$$

Equivalently,

$$\sum_{\substack{(v,p)\in h.V\wedge \\ faulty(v,h.bfttime+tp)}} p < 1/3 \sum_{(v,p)\in h.V} p$$

Question: What should be the precise assumption here. Is the invariant on $h.V$ or on $h.NextV$ or both?

Assumption: If a header is properly generated, then the above equations hold.

Liveness

Draft: If a header h as been properly generated by the blockchain (and its age is less than the trusting period), then a correct LightClient will eventually set $trust(h) = true$.

High Level Solution

Upon initialization, the LightClient is given a header *inithead* it trusts by social consensus. It is assumed that *inithead* satisfies the LightClient Invariant.

When a LightClients sees a new header it has to decide whether to trust the new header. Trust can be obtained by (possibly) the combination of two methods.

1. **an uninterrupted sequence of proof.** If a block is appended to the chain, where the last block is trusted (and properly committed by the old validator set in the next block), and the new block contains a new validator set, the new block is trusted if the LightClient knows all headers in the prefix. Intuitively, a trusted validator set is assumed to not chose a new validator set that violates the fault assumption.

2. **trusting period.** Based on a trusted block *h*, and the LightClient Invariant, which ensures the fault assumption during the trusting period, we can try to infer wether it is guaranteed that the validator set of the new block contains > 2/3 correct voting power. If such a validator set commits a block, we can trust it, as these processes have been continuously correct by the invariant.

Examples: for the “trusting period” method

- *oh*: the old trusted header
- *nh*: the new header that has to be checked

Let’s assume $oh.bfttime + tp > nh.bfttime$ and $oh.bfttime + tp > now$. In the following examples, the pairs (v, p) denote validators and their voting power.

Example: Identical VSets

$$oh.V = \{(1,1), \dots (4,1)\}$$
$$nh.V = \{(1,1), \dots (4,1)\}$$

As we trust oh.V (at oh.bfttime) and the trusting period is not over yet, we trust nh.V.

Example: Changed Voting powers

$$oh.V = \{(1,1), \dots (4,1)\}$$
$$nh.V = \{(1,1), \dots (4,2)\}$$

Validator 4 has more than a third voting power in nh.V. As trusting oh does not rule out that 4 is faulty, the fault assumption might be violated in $nh.V$. Thus, $nh.V$ cannot be trusted.

Example: Lucky case with

$$oh.V = \{(1,1), \dots (6,1)\}$$
$$nh.V = \{(1,1), \dots (7,1)\}$$

By the fault assumption ($n > 3t$), at most one validator in $oh.V$ is faulty. In addition, validator 7 may be faulty. As a result there are at most 2 faulty validators in $nh.V$. Because $7 > 3 \cdot 2$ we say that oh.V provides sufficient trust in order to trust $nh.V$.

Example: Swapping validators

$$oh.V = \{(1,1), \dots (4,1)\}$$
$$nh.V = \{(2,1), \dots (5,1)\}$$

Observe that validator 1 is not present in $nh.V$. Conservatively, we have to assume 1 is correct, and there may be a fault among 2,3,4. In addition, we don't know 5, so that conservatively, we have to assume 5 may be faulty. Thus among 2,3,4,5, there may be two faults which violates the faults assumption. Thus oh does **not** provide sufficient trust in order to trust nh.

Basics for the "trusting period" method

The function `CheckVS(oh, nh)` returns true, when *oh* provides sufficient trust to trust nh.

Assumptions

1. $tp < unbondingperiod$.
2. $nh.bfttime < now$
3. $nh.bfttime < oh.bfttime + tp$
4. $trust(oh) = true$

Some Maths

Observation 1. If $oh.bfttime + tp > now$, we trust the old validator set $oh.V$
.

In the following let's assume oh is trusted and sufficiently new.

Definition 1. Let $PA \subseteq oh.V$ be a *potential adversary* in oh, if the sum of the voting powers in PA is less than 1/3 of the voting powers in $oh.V$, that is,

$$\sum_{(v,p)\in PA} p < 1/3 \sum_{(v,p)\in oh.V} p$$

Proposition 1. The set of faulty processes

$$oh.V \setminus \{(v,p) : (v,p) \in oh.V \wedge correct(v, oh.bfttime + tp)\}$$

is a potential adversary.

Proof. By the LightClient invariant.

Definition 2. Let the *unknown validators* UV be the validators that appear in $nh.V$ and not in $oh.V$, that is,

$$UV = \{(v,p) : (v,p) \in nh.V \wedge \nexists (v,x) \in oh.V\}.$$

Theorem 1. If for all potential adversaries PA, in nh the combined voting powers of PA and UV is less than a third of the total voting power, then in nh, more than 2/3 of the voting power is held by correct processes. Formally, if for all PA

$$\sum_{\substack{(v,old) \in PA \wedge \\ (v,p) \in nh.V}} p + \sum_{(v,p) \in UV} p < 1/3 \sum_{(v,p) \in nh.V} p,$$

then

$$\sum_{\substack{(v,p) \in nh.V \wedge \\ correct(v, oh.bfttime + tp)}} p > 2/3 \sum_{(v,p) \in h.V} p$$

Proof. By the definition of PA, Proposition 1, and the LightClient invariant.

By Assumption 3, there is sufficient voting power to trust the new validator set. (And thus the validator set it signs in that block, for which the tp starts at the $bfttime$ of the header).

Below, we thus sketch a function that checks whether the premise of Theorem 1 holds. If the results is positive, we can trust nh, otherwise not.

An Algorithm

In pseudo go…

```
func CheckVS(oh, nh) bool {
  if oh.bfttime + unbonding_period < now { //
Observation 1
    return false // old header was once trusted but it
is expired
  }

  PAs := compute_all_PAs(oh)  // Definition 1
  PAs := reduce (PAs)         // remove every PA that
is a subset of another PA
```

```
  UV := compute_UV(oh,nh)     // Definition 2

  vpUV := votingpower(UV,nh)  // second sum in Theorem
1
  vpNH := votingpower(nh.V,nh) // right hand side of
premise of Theorem 1
  vpMaxPA := maximumvotingpower(PAs,nh) // voting
powers of all PA and big max

  return vpMaxPA + vpUV < 1/3 * vpNH  // Theorem 1. It
must be smaller for all
                                      // so it must
be smaller for the max
}
```

Remarks

Remark. Computing all PAs might be too expensive (all subsets of $oh.V$ that have a certain combined voting power in oh). Similarly, we then have to compute all voting powers of PAs in nh to get the maximum. This is disturbing, as right now, based on the examples, I expect that CheckVS will mostly return false, assuming that there are frequent changes in the validator sets. However, $oh.V = nh.V$ might be the common case.

To Do. The current invariant assumes that the 1/3 fault assumption is always satisfied. If this is not the case, and there is slashing, we should write the spec of the fault assumptions with temporary violations. Cf. fork accountability, slashing, “counter factual signing” etc. What is there?