MODULE *IscpBatchTimestamp*

Let's assume the transaction currently being built is t_i and the previous one is t_{i-1}. The following requirements apply to the timestamp $t_i.ts$ of the transaction t_i:

1. Transaction timestamps are non-decreasing function in a chain, i.e.

$$t_i.ts \geq t_{i-1}.ts.$$

2. A transaction timestamp is not smaller than the timestamps of request transactions taken as inputs in t_i, i.e.

$$\forall r \in t_i.req : t_i.ts \geq t_i.req[r].tx.ts,$$

   where $t_i.req$ is a list of requests processed as inputs in the transaction t_i, $t_i.req[r]$ is a particular request and $t_i.req[r].tx$ is a transaction the request belongs to.

The initial attempt was to use the timestamp $t_i.ts$ as a median of timestamps proposed by the committee nodes accepted to participate in the transaction t_i by the ACS procedure. This approach conflicts with the rules of selecting requests for the batch (take requests that are mentioned in at least $F+1$ proposals). In this way it is possible that the median is smaller than some request transaction timestamp .
In this document we model the case, when we take maximal of the proposed timestamps excluding the F highest values. This value is close to the 66th percentile (while median is the 50th percentile). In this case all the requests selected to the batch will have timestamp lower than the batch timestamp IF THE BATCH PROPOSALS MEET THE CONDITION

$$\forall p \in batchProposals : \forall r \in p.req : p.req[r].tx.ts \leq p.ts.$$

It is possible that it can be not the case, because of the byzantine nodes. The specification bellow shows, that property (2) can be violated, in the case of byzantine node sending timestamp lower than the requests in the proposal.
The receiving node thus needs to check, if the proposals are correct. For this check it must have all the transactions received before deciding the final batch. The detected invalid batch proposals must be excluded from the following procedure. But that can decrease number of requests included into the final batch (because requests are included if mentioned in $F+1$ proposals). It is safe on the receiver side to "fix" such proposals by setting their timestamp to the maximal transaction timestamp of the requests in the proposal.

EXTENDS $Naturals$, $FiniteSets$, $TLAPS$

CONSTANT $Nodes$ — A set of node identifiers.

CONSTANT $Byzantine$ — A set of byzantine node identifiers.

CONSTANT $Time$ — A set of timestamps, represented as natural numbers to have $\leq$.

ASSUME $ConstantAssms \triangleq Byzantine \subseteq Nodes \land Time \subseteq Nat \land Time \neq \{\} \land Nodes \neq \{\}$

$Requests \triangleq Time$ — Assume requests are identified by timestamps of their TX only.

VARIABLE $proposed$ — Was the proposal made?

VARIABLE $npRq$ — Node proposal: A set of requests.

VARIABLE $npTS$ — Node proposal: Timestamp.

$vars \triangleq \langle proposed, npRq, npTS \rangle$

$F \triangleq Cardinality(Byzantine)$

$N \triangleq Cardinality(Nodes)$

ASSUME $ByzantineAssm \triangleq N \geq 3 * F + 1$

$F1Quorums \triangleq \{q \in \text{SUBSET } Nodes : Cardinality(q) = F + 1\}$

$NFQuorums \triangleq \{q \in \text{SUBSET } Nodes : Cardinality(q) = N - F\}$

```
BatchRq(rq)  ≜ ∃ q ∈ F1Quorums : ∀ n ∈ q : rq ∈ npRq[n]
BatchRqs     ≜ {rq ∈ Requests : BatchRq(rq)}

SubsetTS(s)  ≜ {npTS[n] : n ∈ s}
BatchTS(ts)  ≜ ∃ q ∈ NFQuorums :
                    ∧ ts ∈ SubsetTS(q)
                    ∧ ∀ x ∈ SubsetTS(q) : ts ≥ x
                    ∧ ∀ x ∈ SubsetTS(Nodes \ q) : ts ≤ x

ProposalValid(n) ≜ ∀ rq ∈ npRq[n] : rq ≤ npTS[n]
Propose ≜ ¬proposed ∧ proposed′ = TRUE
  ∧ npRq′ ∈ [Nodes → (SUBSET Requests) \ {{}}]      Some node non-empty proposals.
  ∧ npTS′ ∈ [Nodes → Time]                           Some timestamps.
  ∧ ∀ n ∈ (Nodes \ Byzantine) : ProposalValid(n)′    Fair node proposals are valid.
────────────────────────────────────────────────────────────────────────────
Init ≜   Dummy values, on init (to make TLC faster), see Propose instead.
  ∧ proposed = FALSE
  ∧ npRq = [n ∈ Nodes ↦ {}]
  ∧ npTS = [n ∈ Nodes ↦ 0]
Spec ≜ Init ∧ □[Propose]_vars   For model checking in TLC.

TypeOK ≜
  ∧ proposed ∈ BOOLEAN
  ∧ npRq ∈ [Nodes → SUBSET Requests]
  ∧ npTS ∈ [Nodes → Time ∪ {0}]

Invariant ≜
  proposed ⇒ ∀ ts ∈ Time, rq ∈ BatchRqs : BatchTS(ts) ⇒ rq ≤ ts

THEOREM Spec ⇒ □TypeOK ∧ □Invariant
  PROOF OMITTED   Checked with TLC.
────────────────────────────────────────────────────────────────────────────
THEOREM SpecTypeOK ≜ Spec ⇒ □TypeOK
  ⟨1⟩ Init ⇒ TypeOKBY  DEF Init, TypeOK
  ⟨1⟩ TypeOK ∧ [Propose]_vars ⇒ TypeOK′
      ⟨2⟩ SUFFICES ASSUME TypeOK, ProposePROVE  TypeOK′BY  DEF vars, TypeOK
      ⟨2⟩ QED BY  DEF TypeOK, Propose
  ⟨1⟩ QED BY PTL DEF Spec

THEOREM Byzantine = {} ∧ Spec ⇒ □Invariant
  ⟨1⟩ SUFFICES ASSUME Byzantine = {}PROVE  Spec ⇒ □InvariantOBVIOUS
  ⟨1⟩1. Init ⇒ InvariantBY  DEF Init, Invariant
  ⟨1⟩2. TypeOK ∧ TypeOK′ ∧ Invariant ∧ [Propose]_vars ⇒ Invariant′
    ⟨2⟩ SUFFICES ASSUME TypeOK, TypeOK′, Invariant, ProposePROVE  Invariant′
          BY  DEF vars, Invariant, BatchRq, BatchRqs, BatchTS, ProposalValid, SubsetTS
    ⟨2⟩ SUFFICES ASSUME proposed′
                 PROVE (∀ ts ∈ Time, rq ∈ BatchRqs : BatchTS(ts) ⇒ rq ≤ ts)′
```

```
        BY DEF Invariant
    ⟨2⟩ TAKE ts ∈ Time, rq ∈ BatchRqs′
    ⟨2⟩ HAVE BatchTS(ts)′
    ⟨2⟩ ∀ n ∈ Nodes \ Byzantine : (∀ rqx ∈ npRq[n] : rqx ≤ npTS[n])′
        BY DEF Propose, ProposalValid
    ⟨2⟩ ∀ n ∈ Nodes : (∀ rqx ∈ npRq[n] : rqx ≤ npTS[n])′
        OBVIOUS
    ⟨2⟩q. QED PROOF OMITTED  TODO
          BY DEF BatchTS, BatchRq, BatchRqs, SubsetTS
          ,Requests, NFQuorums, F1Quorums, N, F
  ⟨1⟩q. QED BY ⟨1⟩1, ⟨1⟩2, PTL, SpecTypeOK DEF Spec, vars
```

Counter-example with $Nodes = 101\,..\,104$, $Byzantine = \{104\}$, $Time = 1\,..\,3$:

PropposedRq: (101 :> {1} @@ 102 :> {1} @@ 103 :> {2} @@ 104 :> {2}),
PropposedTS: (101 :> 1 @@ 102 :> 1 @@ 103 :> 2 @@ 104 :> 1),
BatchRq: {1, 2},
BatchTS: 1