

Specification of the signature-free *Sailfish* consensus algorithm at a high level of abstraction.

We use a number of abstractions and simplifying assumptions in order to expose the high-level principles of the algorithm clearly and in order to make model-checking of interesting configurations tractable:

- 1) Nodes read and write a global *DAG*. When a node transitions into a new round, it is provided with an arbitrary quorum of vertices from the previous round (except that, after *GST*, some additional assumptions apply).
- 2) We do not model timeouts. Instead, we assume that, every round r after *GST*, each correct node votes for the previous leader.
- 3) Byzantine nodes are allowed to create new *DAG* vertices arbitrarily, but only one per round.
- 4) We do not explicitly model committing based on $2f + 1$ first *RBC* messages.
- 5) There are no weak edges.

EXTENDS *Digraph, Integers, FiniteSets, Sequences*

CONSTANTS

- N The set of all nodes
- , F The set of *Byzantine* nodes
- , R The set of rounds
- , $IsQuorum(-)$ Whether a set is a quorum (*i.e.* cardinality $\geq n-f$)
- , $IsBlocking(-)$ Whether a set is a blocking set (*i.e.* cardinality $\geq f + 1$)
- , $Leader(-)$ operator mapping each round to its leader
- , GST the first round in which the system is synchronous

ASSUME $\exists n \in R : R = 1 \dots n$ useful rounds start at 1

For our purpose of checking safety and liveness of *Sailfish*, we do not need to model blocks of transactions. Instead, *DAG* vertices just consist of a node and a round.

$V \triangleq N \times R$ the set of possible *DAG* vertices
 $Node(v) \triangleq v[1]$
 $Round(v) \triangleq v[2]$

Next we define how we order *DAG* vertices when we commit a leader vertice

$LeaderVertice(r) \triangleq \langle Leader(r), r \rangle$

RECURSIVE $OrderSet(-)$ arbitrarily order the members of the set S

$OrderSet(S) \triangleq$ IF $S = \{\}$ THEN $\langle \rangle$ ELSE
 LET $e \triangleq$ CHOOSE $e \in S : \text{TRUE}$
 IN $Append(OrderSet(S \setminus \{e\}), e)$

NOTE: CHOOSE is deterministic in TLA+,
i.e. CHOOSE $e \in S : \text{TRUE}$ is always the same e if S is the same

```

RECURSIVE CollectLeaders( $\_, \_, \_$ )
CollectLeaders( $vs, r, dag$ )  $\triangleq$  IF  $vs = \{\}$  THEN  $\langle \rangle$  ELSE
  LET  $children \triangleq$  UNION  $\{Children(v, dag) : v \in vs\}$ 
  IN IF  $LeaderVertice(r) \in vs$ 
    THEN Append(
      CollectLeaders( $Children(LeaderVertice(r), dag), r - 1, dag$ ),
       $LeaderVertice(r)$ )
    ELSE CollectLeaders( $children, r - 1, dag$ )

RECURSIVE OrderVertices( $\_, \_$ )
OrderVertices( $dag, leaderVertices$ )  $\triangleq$ 
  IF  $leaderVertices = \langle \rangle$  THEN  $\langle \rangle$  ELSE
  LET  $l \triangleq Head(leaderVertices)$ 
     $toOrder \triangleq Descendants(\{l\}, dag)$ 
     $prefix \triangleq OrderSet(toOrder)$ 
     $remainingVertices \triangleq Vertices(dag) \setminus (toOrder \cup \{l\})$ 
     $remainingEdges \triangleq \{e \in Edges(dag) : \{e[1], e[2]\} \subseteq remainingVertices\}$ 
     $remainingDAG \triangleq \langle remainingVertices, remainingEdges \rangle$ 
  IN  $prefix \circ \langle l \rangle \circ OrderVertices(remainingDAG, Tail(leaderVertices))$ 

CommitLeader( $v, dag$ )  $\triangleq$ 
  LET  $leaderVertices \triangleq CollectLeaders(\{v\}, Round(v), dag)$ 
  IN OrderVertices( $dag, leaderVertices$ )

```

Now we specify the algorithm in the *PlusCal* language.

```

--algorithm Sailfish{
  variables
     $vs = \{\}$ ,   the vertices of the DAG
     $es = \{\}$ ;  the edges of the DAG
  define {
     $dag \triangleq \langle vs, es \rangle$ 
    NoLeaderVoteQuorum( $r, delivered, add$ )  $\triangleq$ 
      LET  $NoLeaderVote \triangleq \{v \in delivered : LeaderVertice(r - 1) \notin Children(v, dag)\}$ 
      IN IsQuorum( $\{Node(v) : v \in NoLeaderVote\} \cup add$ )
  }
  process (  $correctNode \in N \setminus F$  )
    variables
       $round = 0$ ,   current round; 0 means the node has not started execution
       $log = \langle \rangle$ ;  delivered log
    {
l0:  while ( TRUE ) {
      if (  $round = 0$  ) { start the first round  $r = 1$ 
         $round := 1$ ;
         $vs := vs \cup \{\langle self, 1 \rangle\}$ 
      }
    }
  }

```

```

else { start a round  $r > 1$ 
  with (  $r \in \{r \in R : r > round\}$  )
  with (  $delivered \in \text{SUBSET } \{v \in vs : Round(v) = r - 1\}$  ) {
    await  $IsQuorum(\{Node(v) : v \in delivered\})$ ;
    await  $LeaderVertice(r - 1) \in delivered \Rightarrow$ 
       $\vee LeaderVertice(r - 2) \in Children(LeaderVertice(r - 1), dag)$ 
       $\vee NoLeaderVoteQuorum(r - 1, delivered, \{\})$ ;
    if (  $Leader(r) = self$  )
      await  $\vee LeaderVertice(r - 1) \in delivered$ 
       $\vee NoLeaderVoteQuorum(r, delivered, \{self\})$ ;
    round :=  $r$ ;
    with (  $newV = \langle self, round \rangle$  ) {
       $vs := vs \cup \{newV\}$ ;
       $es := es \cup \{\langle newV, pv \rangle : pv \in delivered\}$ ;
    } ;
    commit if there is a quorum of votes for the leader of  $r - 2$ :
    if (  $round > 1$  )
      with (  $votesForLeader = \{pv \in delivered : \langle pv, LeaderVertice(round - 2) \rangle \in es\}$  )
      if (  $IsBlocking(\{Node(pv) : pv \in votesForLeader\})$  )
        log :=  $CommitLeader(LeaderVertice(round - 2), dag)$ 
      }
    }
  }
}

```

Next comes our model of *Byzantine* nodes. Because the real protocol disseminates *DAG* vertices using reliable broadcast, *Byzantine* nodes cannot equivocate and cannot deviate much from the protocol (lest their messages be ignored).

```

process (  $byzantineNode \in F$  )
{
l0: while ( TRUE ) {
  with (  $r \in R$  )
  with (  $newV = \langle self, r \rangle$  ) {
    when  $newV \notin vs$ ; no equivocation
    if (  $r = 1$  )
       $vs := vs \cup \{newV\}$ 
    else
      with (  $delivered \in \text{SUBSET } \{v \in vs : Round(v) = r - 1\}$  ) {
        await  $IsQuorum(\{Node(v) : v \in delivered\})$ ; ignored otherwise
         $vs := vs \cup \{newV\}$ ;
         $es := es \cup \{\langle newV, pv \rangle : pv \in delivered\}$ 
      }
    }
  }
}

```

Next we define the safety and liveness properties

$$\begin{aligned} \text{Compatible}(s1, s2) &\triangleq \text{whether the sequence } s1 \text{ is a prefix of the sequence } s2, \text{ or vice versa} \\ \text{LET } \text{Min}(n1, n2) &\triangleq \text{IF } n1 \geq n2 \text{ THEN } n2 \text{ ELSE } n1 \text{ IN} \\ &\quad \forall i \in 1 \dots \text{Min}(\text{Len}(s1), \text{Len}(s2)) : s1[i] = s2[i] \end{aligned}$$

$$\text{Agreement} \triangleq \forall n1, n2 \in N \setminus F : \text{Compatible}(\log[n1], \log[n2])$$

$$\begin{aligned} \text{Liveness} &\triangleq \forall r \in R : r \geq \text{GST} \wedge \text{Leader}(r) \notin F \Rightarrow \exists B \in \text{SUBSET } (N \setminus F) : \\ &\quad \wedge \text{IsBlocking}(B) \\ &\quad \wedge \forall n \in B : \text{round}[n] \geq r + 2 \Rightarrow \exists i \in \text{DOMAIN } \log[n] : \log[n][i] = \text{LeaderVertice}(r) \end{aligned}$$

Finally we make a few auxiliary definitions used for model-checking with *TLC*

Basic typing invariant:

$$\begin{aligned} \text{TypeOK} &\triangleq \\ &\quad \wedge \forall v \in vs : \text{Node}(v) \in N \wedge \text{Round}(v) \in \text{Nat} \setminus \{0\} \\ &\quad \wedge \forall e \in es : \\ &\quad \quad \wedge e = \langle e[1], e[2] \rangle \\ &\quad \quad \wedge \{e[1], e[2]\} \subseteq vs \\ &\quad \quad \wedge \text{Round}(e[1]) > \text{Round}(e[2]) \\ &\quad \wedge \forall n \in N \setminus F : \text{round}[n] \in \text{Nat} \end{aligned}$$

Synchrony assumption: for each round r from *GST* onwards, if the leader of r is correct then every correct node votes for the round- r leader vertex in round $r + 1$

$$\begin{aligned} \text{Synchrony} &\triangleq \forall r \in R : r \geq \text{GST} \wedge \text{Leader}(r) \notin F \Rightarrow \\ &\quad \forall n \in N \setminus F : \\ &\quad \quad \text{LET } v \triangleq \langle n, r + 1 \rangle \\ &\quad \quad \text{IN } \quad \wedge v \in vs \\ &\quad \quad \quad \wedge \quad \vee r = 1 \\ &\quad \quad \quad \vee \text{LeaderVertice}(r - 1) \in \text{Children}(\text{LeaderVertice}(r), \text{dag}) \\ &\quad \quad \quad \vee \text{NoLeaderVoteQuorum}(r, \{v2 \in vs : \text{Round}(v2) = r\}, \{\}) \\ &\quad \quad \Rightarrow \text{LeaderVertice}(r) \in \text{Children}(v, \text{dag}) \end{aligned}$$

We add the synchrony assumption to the specification

$$\begin{aligned} \text{SyncNext} &\triangleq (\exists \text{self} \in N \setminus F : \text{correctNode}(\text{self}) \wedge \text{Synchrony}') \\ &\quad \vee (\exists \text{self} \in F : \text{byzantineNode}(\text{self})) \\ \text{SyncSpec} &\triangleq \text{Init} \wedge \Box[\text{SyncNext}]_{\text{vars}} \end{aligned}$$

Next we define a constraint to stop the model-checker.

$$\begin{aligned} \text{Max}(S) &\triangleq \text{CHOOSE } x \in S : \forall y \in S : y \leq x \\ \text{StateConstraint} &\triangleq \forall n \in N \setminus F : \text{round}[n] \in 0 \dots \text{Max}(R) \end{aligned}$$

Finally, we give some properties we expect to be violated (useful to get the model-checker to print interesting executions).

$$Falsy1 \triangleq \neg(\forall n \in N \setminus F : round[n] = Max(R))$$

$$Falsy2 \triangleq \neg(\exists n \in N \setminus F : Len(log[n]) > 1)$$
