

---

 MODULE *Sailfish*


---

This is a high-level specification of the *Sailfish* and *Sailfish++* (also called signature-free *Sailfish*) algorithms. At the level of abstraction of this specification, the differences between the two algorithms are not visible.

EXTENDS *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  $\exists n \in R : R = 1 \dots n$  rounds start at 1; 0 is used as default placeholder

INSTANCE *BlockDag* Import definitions related to *DAGs* of blocks

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

```
--algorithm Sailfish{
    variables
        vs = {Genesis}, the vertices of the DAG
        es = {} ; the edges of the DAG
    define {
        dag  $\triangleq$  ⟨vs, esNoLeaderVoteQuorum(r, vertices, add)  $\triangleq$ 
            LET NoLeaderVote  $\triangleq$  {v  $\in$  vertices : LeaderVertex(r-1)  $\notin$  Children(dag, v)}
            IN IsQuorum({Node(v) : v  $\in$  NoLeaderVote}  $\cup$  add)
    }
    process ( correctNode  $\in$  N \ F )
        variables
            round = 0, current round; 0 means the node has not started execution
            log = ⟨⟩ ; delivered log
        {
            l0: while ( TRUE ) {
                if ( round = 0 ) { start the first round r = 1
                    round := 1 ;
                    vs := vs  $\cup$  {⟨self, 1⟩} ;
                    es := es  $\cup$  {⟨⟨self, 1⟩, Genesis} }
                }
                else { start a round r > 1
                    with ( r  $\in$  {r  $\in$  R : r > round} )
                    with ( deliveredVertices  $\in$  SUBSET {v  $\in$  vs : Round(v) = r-1} ) {
                        we enter a round only if we have a quorum of vertices:
                    }
                }
            }
        }
}
```

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

```

Basic type invariant:

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

Next we define the safety and liveness properties

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

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