Stubborn Point-to-Point Link: Implementation

Algorithm 2.1: Retransmit Forever

Implements:
StubbornPointToPointLinks, **instance** *sl*.

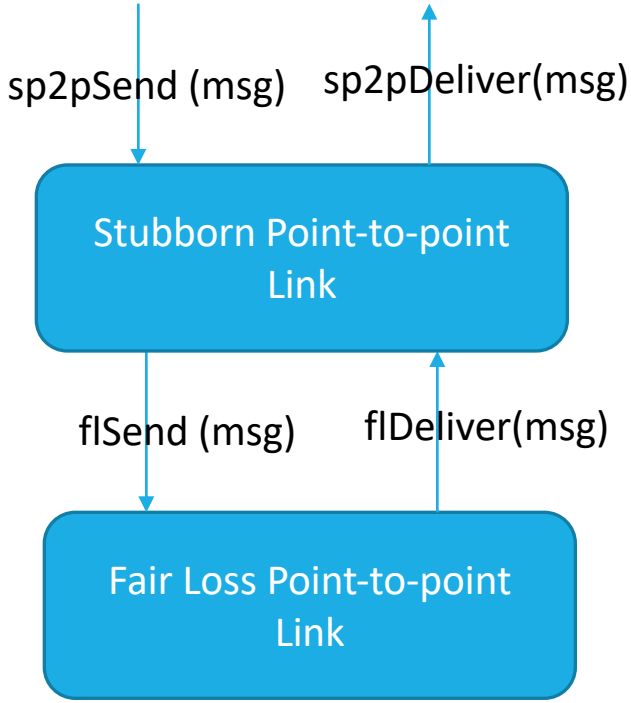
Uses:
FairLossPointToPointLinks, **instance** *fll*.

upon event ⟨ *sl*, *Init* ⟩ **do**
sent := ∅;
starttimer(Δ);

upon event ⟨ *Timeout* ⟩ **do**
forall $(q, m) \in sent$ **do**
trigger ⟨ *fll*, *Send* | q, m ⟩;
starttimer(Δ);

upon event ⟨ *sl*, *Send* | q, m ⟩ **do**
trigger ⟨ *fll*, *Send* | q, m ⟩;
sent := *sent* ∪ $\{(q, m)\}$;

upon event ⟨ *fll*, *Deliver* | p, m ⟩ **do**
trigger ⟨ *sl*, *Deliver* | p, m ⟩;



Perfect Point-to-Point Link: Implementation

Algorithm 2.2: Eliminate Duplicates

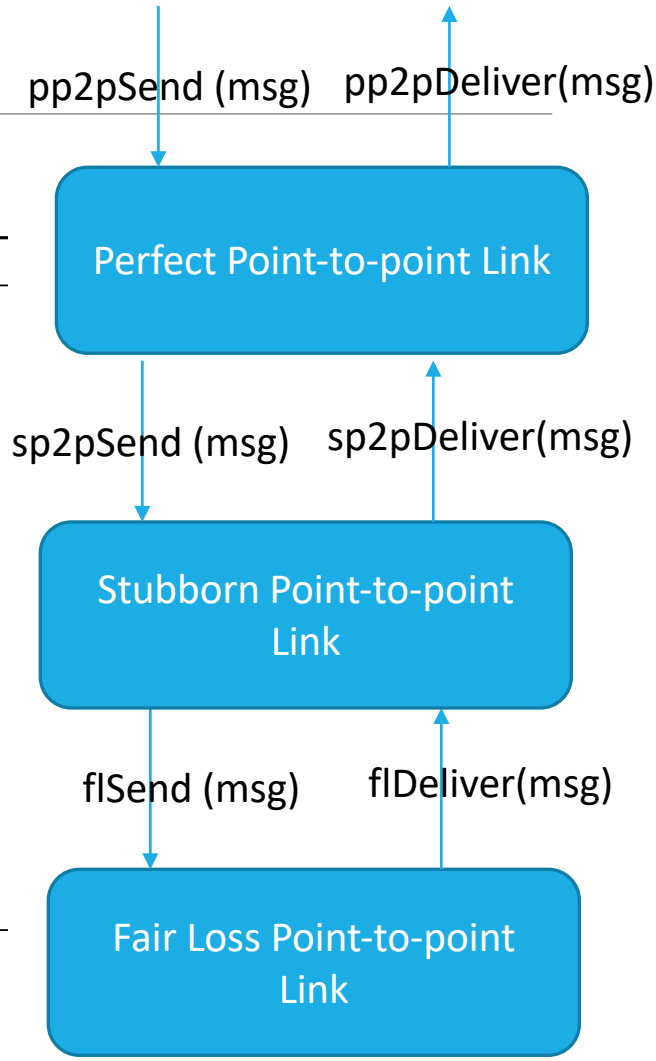
Implements:
PerfectPointToPointLinks, **instance** *pl*.

Uses:
StubbornPointToPointLinks, **instance** *sl*.

upon event ⟨ *pl*, *Init* ⟩ **do**
delivered := ∅;

upon event ⟨ *pl*, *Send* | *q*, *m* ⟩ **do**
trigger ⟨ *sl*, *Send* | *q*, *m* ⟩;

upon event ⟨ *sl*, *Deliver* | *p*, *m* ⟩ **do**
if $m \notin delivered$ **then**
$delivered := delivered \cup \{m\}$;
trigger ⟨ *pl*, *Deliver* | *p*, *m* ⟩;



Perfect failure detectors (P) Implementation

Algorithm 2.5: Exclude on Timeout

Implements:
PerfectFailureDetector, **instance** $\mathcal{P}$.

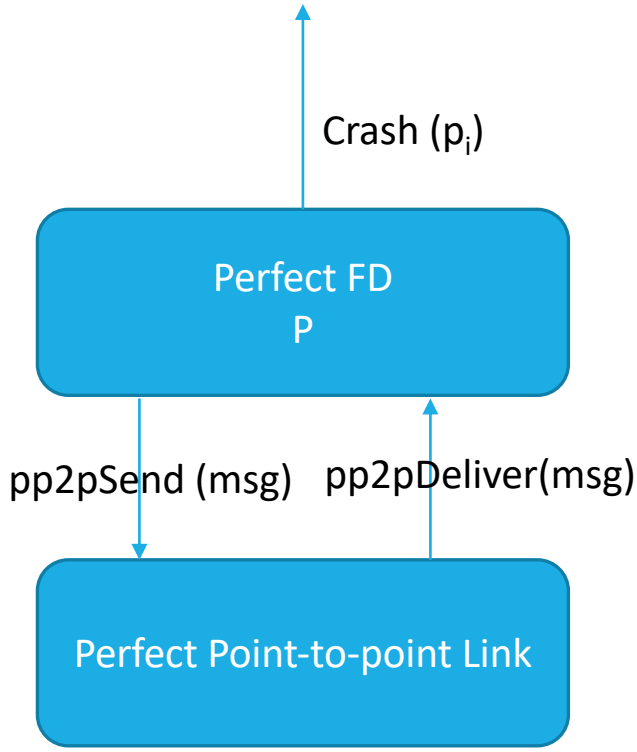
Uses:
PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle\ \mathcal{P}, \mathit{Init}\ \rangle$ **do**
  $\mathit{alive} := \Pi$;
  $\mathit{detected} := \emptyset$;
  starttimer(Δ);

upon event $\langle\ \mathit{Timeout}\ \rangle$ **do**
  forall $p \in \Pi$ **do**
    if $(p \notin \mathit{alive}) \wedge (p \notin \mathit{detected})$ **then**
      $\mathit{detected} := \mathit{detected} \cup \{p\}$;
      trigger $\langle\ \mathcal{P}, \mathit{Crash} \mid p\ \rangle$;
    trigger $\langle\ pl, \mathit{Send} \mid p, [\text{HEARTBEATREQUEST}]\ \rangle$;
  $\mathit{alive} := \emptyset$;
  starttimer(Δ);

upon event $\langle\ pl, \mathit{Deliver} \mid q, [\text{HEARTBEATREQUEST}]\ \rangle$ **do**
  trigger $\langle\ pl, \mathit{Send} \mid q, [\text{HEARTBEATREPLY}]\ \rangle$;

upon event $\langle\ pl, \mathit{Deliver} \mid p, [\text{HEARTBEATREPLY}]\ \rangle$ **do**
  $\mathit{alive} := \mathit{alive} \cup \{p\}$;



Eventually perfect failure detectors (◊P) Implementation

Algorithm 2.7: Increasing Timeout

Implements:
EventuallyPerfectFailureDetector, **instance** $\diamond\mathcal{P}$.

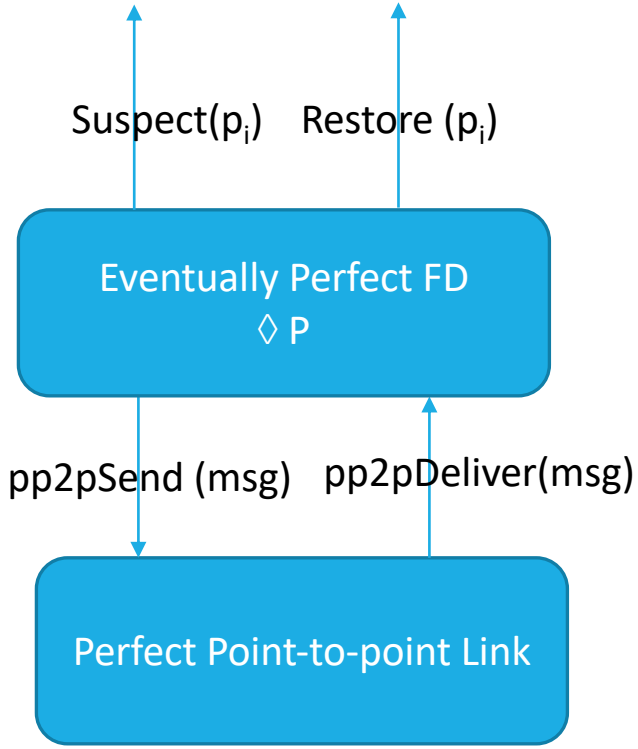
Uses:
PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle\ \diamond\mathcal{P}, \mathit{Init}\ \rangle$ **do**
  $\mathit{alive} := \Pi$;
  $\mathit{suspected} := \emptyset$;
  $\mathit{delay} := \Delta$;
  starttimer(*delay*);

upon event $\langle\ \mathit{Timeout}\ \rangle$ **do**
  if $\mathit{alive} \cap \mathit{suspected} \neq \emptyset$ **then**
    $\mathit{delay} := \mathit{delay} + \Delta$;
  forall $p \in \Pi$ **do**
    if $(p \notin \mathit{alive}) \wedge (p \notin \mathit{suspected})$ **then**
      $\mathit{suspected} := \mathit{suspected} \cup \{p\}$;
      trigger $\langle\ \diamond\mathcal{P}, \mathit{Suspect} \mid p\ \rangle$;
    else if $(p \in \mathit{alive}) \wedge (p \in \mathit{suspected})$ **then**
      $\mathit{suspected} := \mathit{suspected} \setminus \{p\}$;
      trigger $\langle\ \diamond\mathcal{P}, \mathit{Restore} \mid p\ \rangle$;
    trigger $\langle\ pl, \mathit{Send} \mid p, [\text{HEARTBEATREQUEST}]\ \rangle$;
  $\mathit{alive} := \emptyset$;
  starttimer(*delay*);

upon event $\langle\ pl, \mathit{Deliver} \mid q, [\text{HEARTBEATREQUEST}]\ \rangle$ **do**
  trigger $\langle\ pl, \mathit{Send} \mid q, [\text{HEARTBEATREPLY}]\ \rangle$;

upon event $\langle\ pl, \mathit{Deliver} \mid p, [\text{HEARTBEATREPLY}]\ \rangle$ **do**
  $\mathit{alive} := \mathit{alive} \cup \{p\}$;



Leader Election Implementation

Algorithm 2.6: Monarchical Leader Election

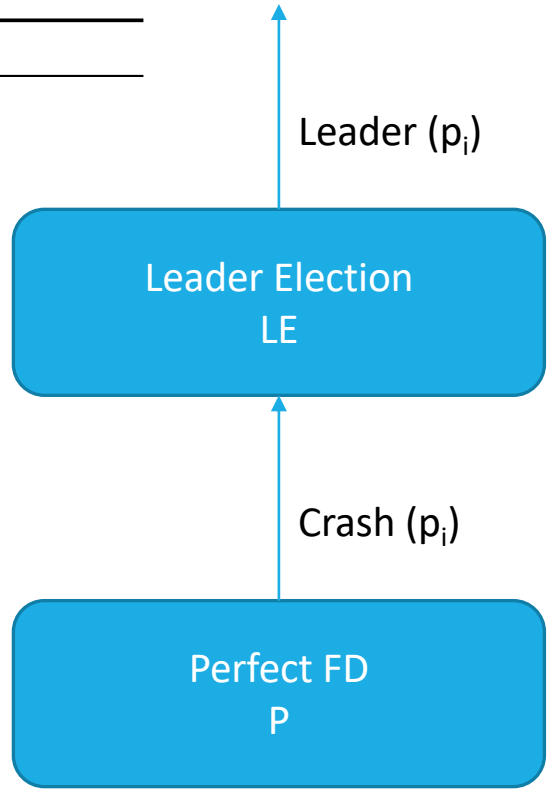
Implements:
LeaderElection, **instance** *le*.

Uses:
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ *le*, *Init* $\rangle$ **do**
suspected := $\emptyset$;
leader := $\perp$;

upon event $\langle$ $\mathcal{P}$, *Crash* | *p* $\rangle$ **do**
suspected := *suspected* $\cup$ {*p*};

upon *leader* $\neq$ *maxrank*(Π \ *suspected*) **do**
leader := *maxrank*(Π \ *suspected*);
trigger $\langle$ *le*, *Leader* | *leader* $\rangle$;



Best Effort Broadcast (BEB) Implementation

Algorithm 3.1: Basic Broadcast

Implements:
BestEffortBroadcast, **instance** *beb*.

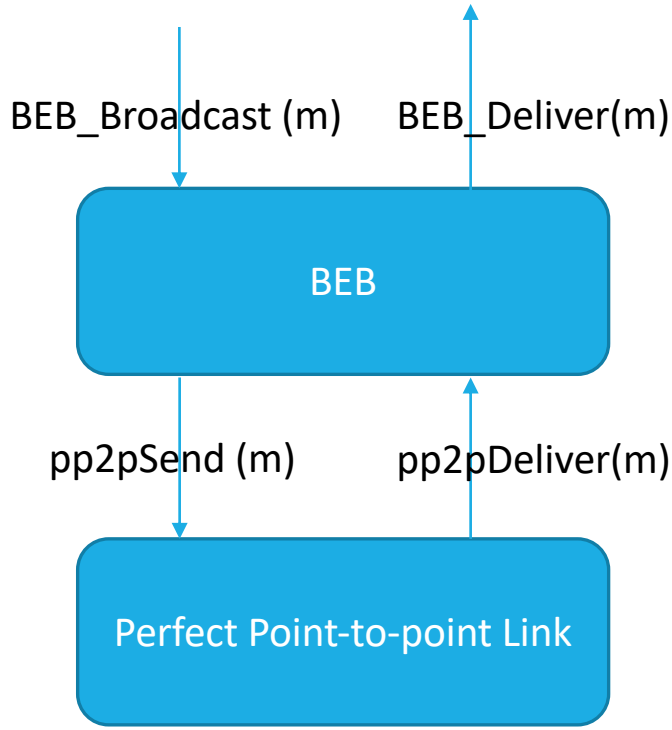
Uses:
PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle$ *beb*, *Broadcast* | m $\rangle$ **do**
forall $q \in \Pi$ **do**
trigger $\langle$ *pl*, *Send* | q, m $\rangle$;

upon event $\langle$ *pl*, *Deliver* | p, m $\rangle$ **do**
trigger $\langle$ *beb*, *Deliver* | p, m $\rangle$;

System model

- Asynchronous system
- perfect links
- crash failures



(Regular) Reliable Broadcast (RB) Implementation in Synchronous Systems

Algorithm 3.2: Lazy Reliable Broadcast

Implements:
ReliableBroadcast, **instance** *rb*.

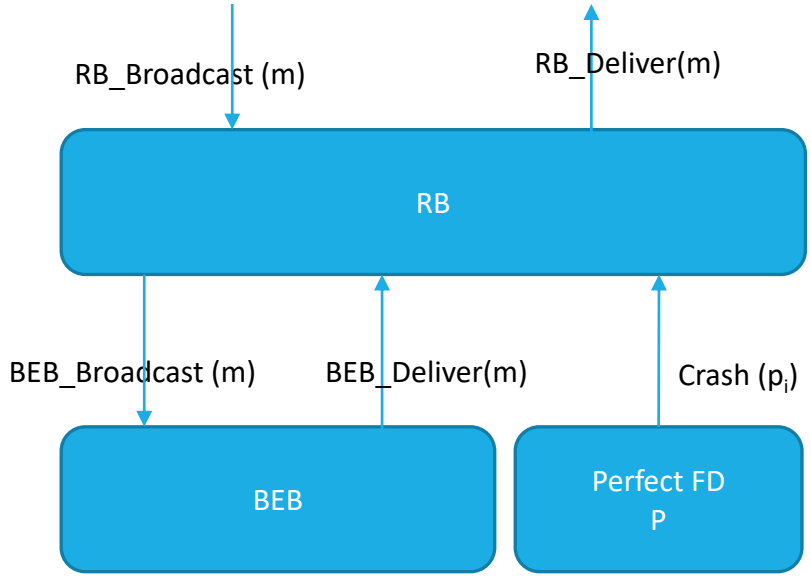
Uses:
BestEffortBroadcast, **instance** *beb*;
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ *rb*, *Init* $\rangle$ **do**
correct := Π;
from[p] := $[\emptyset]^N$;

upon event $\langle$ *rb*, *Broadcast* | m $\rangle$ **do**
trigger $\langle$ *beb*, *Broadcast* | [DATA, *self*, m] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, [DATA, s, m] $\rangle$ **do**
if $m \notin$ *from*[s] **then**
trigger $\langle$ *rb*, *Deliver* | s, m $\rangle$;
from[s] := *from*[s] $\cup \{m\}$;
if $s \notin$ *correct* **then**
trigger $\langle$ *beb*, *Broadcast* | [DATA, s, m] $\rangle$;

upon event $\langle$ $\mathcal{P}$, *Crash* | p $\rangle$ **do**
correct := *correct* $\setminus \{p\}$;
forall $m \in$ *from*[p] **do**
trigger $\langle$ *beb*, *Broadcast* | [DATA, p, m] $\rangle$;



The algorithm is Lazy in the sense that it retransmits only when necessary

(Regular) Reliable Broadcast (RB) Implementation in Asynchronous Systems

Algorithm 3.3: Eager Reliable Broadcast

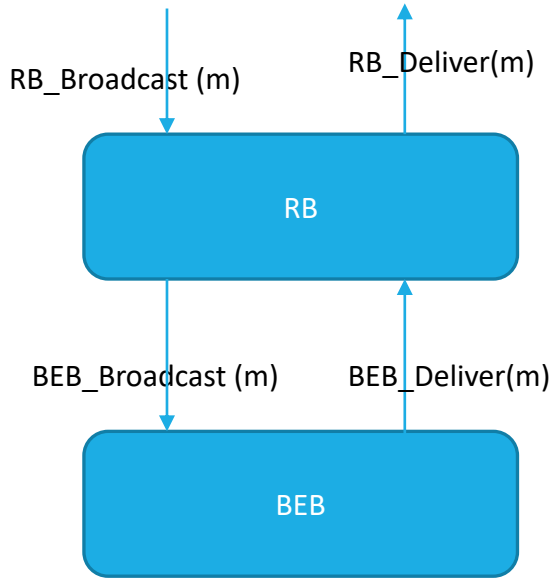
Implements:
ReliableBroadcast, **instance** *rb*.

Uses:
BestEffortBroadcast, **instance** *beb*.

upon event ⟨ *rb*, *Init* ⟩ **do**
delivered := ∅;

upon event ⟨ *rb*, *Broadcast* | m ⟩ **do**
trigger ⟨ *beb*, *Broadcast* | [DATA, *self*, m] ⟩;

upon event ⟨ *beb*, *Deliver* | p, [DATA, s, m] ⟩ **do**
if $m \notin$ *delivered* **then**
delivered := *delivered* ∪ {m};
trigger ⟨ *rb*, *Deliver* | s, m ⟩;
trigger ⟨ *beb*, *Broadcast* | [DATA, s, m] ⟩;



The algorithm is Eager in the sense that it retransmits every message

Uniform Reliable Broadcast (URB) Implementation in Synchronous System

Algorithm 3.4: All-Ack Uniform Reliable Broadcast

Implements:
UniformReliableBroadcast, **instance** *urb*.

Uses:
BestEffortBroadcast, **instance** *beb*.
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ *urb*, *Init* $\rangle$ **do**
delivered := $\emptyset$;
pending := $\emptyset$;
correct := Π;
forall m **do** *ack*$[m]$:= $\emptyset$;

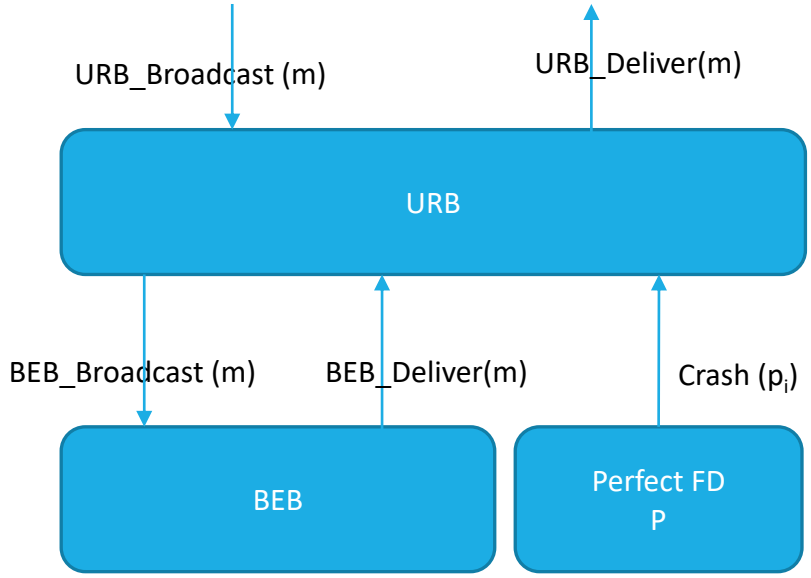
upon event $\langle$ *urb*, *Broadcast* | m $\rangle$ **do**
pending := *pending* $\cup$ $\{(self, m)\}$;
trigger $\langle$ *beb*, *Broadcast* | [DATA, *self*, m] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, [DATA, s, m] $\rangle$ **do**
ack$[m]$:= *ack*$[m]$ $\cup$ $\{p\}$;
if $(s, m) \notin$ *pending* **then**
pending := *pending* $\cup$ $\{(s, m)\}$;
trigger $\langle$ *beb*, *Broadcast* | [DATA, s, m] $\rangle$;

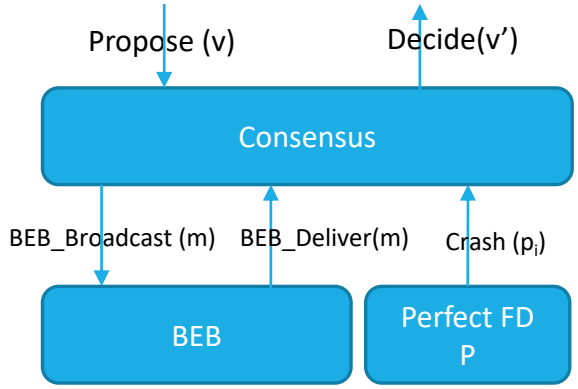
upon event $\langle$ $\mathcal{P}$, *Crash* | p $\rangle$ **do**
correct := *correct* $\setminus$ $\{p\}$;

function *candeliver*(m) **returns** Boolean **is**
return $(correct \subseteq ack[m])$;

upon exists $(s, m) \in$ *pending* such that *candeliver*$(m) \wedge m \notin$ *delivered* **do**
delivered := *delivered* $\cup$ $\{m\}$;
trigger $\langle$ *urb*, *Deliver* | s, m $\rangle$;



A Consensus Implementation in Synchronous Systems: Flooding Consensus



Algorithm 5.1: Flooding Consensus

Implements:
Consensus, **instance** c.

Uses:
BestEffortBroadcast, **instance** *beb*;
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle\ c, \mathit{Init}\ \rangle$ **do**
$\mathit{correct} := \Pi$;
$\mathit{round} := 1$;
$\mathit{decision} := \bot$;
$\mathit{receivedfrom} := [\emptyset]^N$;
$\mathit{proposals} := [\emptyset]^N$;
$\mathit{receivedfrom}[0] := \Pi$;

upon event $\langle\ \mathcal{P}, \mathit{Crash} \mid p\ \rangle$ **do**
$\mathit{correct} := \mathit{correct} \setminus \{p\}$;

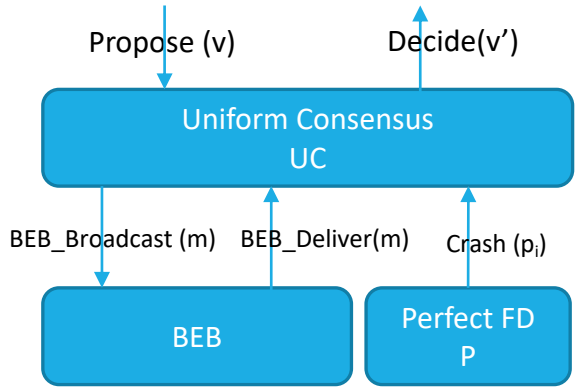
upon event $\langle\ c, \mathit{Propose} \mid v\ \rangle$ **do**
$\mathit{proposals}[1] := \mathit{proposals}[1] \cup \{v\}$;
trigger $\langle\ \mathit{beb}, \mathit{Broadcast} \mid [\text{PROPOSAL}, 1, \mathit{proposals}[1]]\ \rangle$;

upon event $\langle\ \mathit{beb}, \mathit{Deliver} \mid p, [\text{PROPOSAL}, r, ps]\ \rangle$ **do**
$\mathit{receivedfrom}[r] := \mathit{receivedfrom}[r] \cup \{p\}$;
$\mathit{proposals}[r] := \mathit{proposals}[r] \cup ps$;

upon $\mathit{correct} \subseteq \mathit{receivedfrom}[\mathit{round}] \wedge \mathit{decision} = \bot$ **do**
if $\mathit{receivedfrom}[\mathit{round}] = \mathit{receivedfrom}[\mathit{round} - 1]$ **then**
$\mathit{decision} := \min(\mathit{proposals}[\mathit{round}])$;
trigger $\langle\ \mathit{beb}, \mathit{Broadcast} \mid [\text{DECIDED}, \mathit{decision}]\ \rangle$;
trigger $\langle\ c, \mathit{Decide} \mid \mathit{decision}\ \rangle$;
else
$\mathit{round} := \mathit{round} + 1$;
trigger $\langle\ \mathit{beb}, \mathit{Broadcast} \mid [\text{PROPOSAL}, \mathit{round}, \mathit{proposals}[\mathit{round} - 1]]\ \rangle$;

upon event $\langle\ \mathit{beb}, \mathit{Deliver} \mid p, [\text{DECIDED}, v]\ \rangle$ **such that** $p \in \mathit{correct} \wedge \mathit{decision} = \bot$ **do**
$\mathit{decision} := v$;
trigger $\langle\ \mathit{beb}, \mathit{Broadcast} \mid [\text{DECIDED}, \mathit{decision}]\ \rangle$;
trigger $\langle\ c, \mathit{Decide} \mid \mathit{decision}\ \rangle$;

Uniform Consensus Implementation in Synchronous Systems



Algorithm 5.3: Flooding Uniform Consensus

Implements:
UniformConsensus, **instance** *uc*.

Uses:
BestEffortBroadcast, **instance** *beb*;
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ *uc*, *Init* $\rangle$ **do**
correct := Π;
round := 1;
decision := $\perp$;
proposalset := $\emptyset$;
receivedfrom := $\emptyset$;

upon event $\langle$ $\mathcal{P}$, *Crash* | p $\rangle$ **do**
correct := *correct* $\setminus$ $\{p\}$;

upon event $\langle$ *uc*, *Propose* | v $\rangle$ **do**
proposalset := *proposalset* $\cup$ $\{v\}$;
trigger $\langle$ *beb*, *Broadcast* | [PROPOSAL, 1, *proposalset*] $\rangle$;

No more related to the round

upon event $\langle$ *beb*, *Deliver* | p, [PROPOSAL, r, ps] $\rangle$ **such that** r = *round* **do**
receivedfrom := *receivedfrom* $\cup$ $\{p\}$;
proposalset := *proposalset* $\cup$ ps;

upon *correct* $\subseteq$ *receivedfrom* $\wedge$ *decision* = $\perp$ **do**
if *round* = N **then**
decision := min(*proposalset*);
trigger $\langle$ *uc*, *Decide* | *decision* $\rangle$;
else
round := *round* + 1;
receivedfrom := $\emptyset$;
trigger $\langle$ *beb*, *Broadcast* | [PROPOSAL, *round*, *proposalset*] $\rangle$;

Decision only at the end

Cleaned at the beginning of each round

FIFO Broadcast - Implementation

Algorithm 3.12: Broadcast with Sequence Number

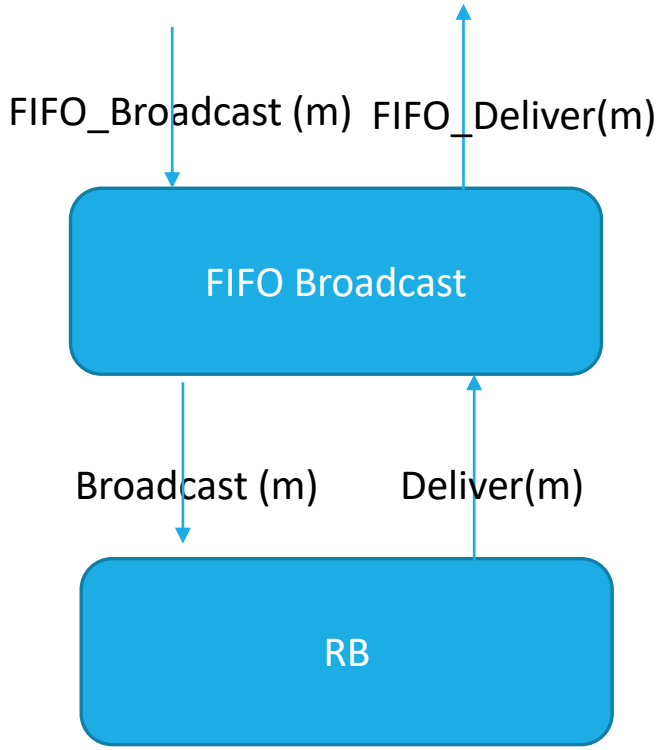
Implements:
FIFOReliableBroadcast, **instance** *frb*.

Uses:
ReliableBroadcast, **instance** *rb*.

upon event $\langle$ *frb*, *Init* $\rangle$ **do**
lsn := 0;
pending := $\emptyset$;
next := $[1]^N$;

upon event $\langle$ *frb*, *Broadcast* | m $\rangle$ **do**
lsn := *lsn* + 1;
trigger $\langle$ *rb*, *Broadcast* | [DATA, *self*, m, *lsn*] $\rangle$;

upon event $\langle$ *rb*, *Deliver* | p, [DATA, s, m, sn] $\rangle$ **do**
pending := *pending* $\cup \{(s, m, sn)\}$;
while exists $(s, m', sn') \in$ *pending* such that $sn' =$ *next*$[s]$ **do**
next$[s]$:= *next*$[s]$ + 1;
pending := *pending* $\setminus \{(s, m', sn')\}$;
trigger $\langle$ *frb*, *Deliver* | s, m' $\rangle$;



Causal Order Broadcast Implementation

Algorithm 3.15: Waiting Causal Broadcast

Implements:
CausalOrderReliableBroadcast, **instance** *crb*.

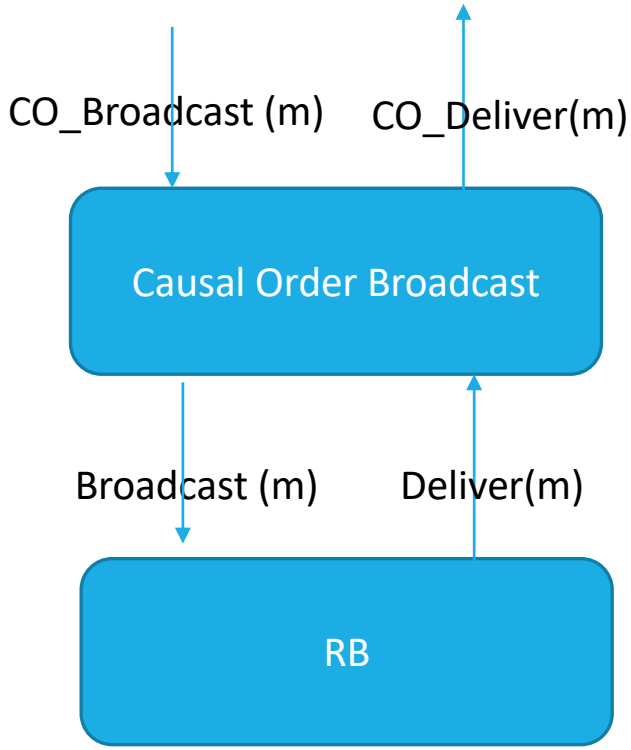
Uses:
ReliableBroadcast, **instance** *rb*.

upon event $\langle$ *crb*, *Init* $\rangle$ **do**
$V := [0]^N$;
$lsn := 0$;
$pending := \emptyset$;

upon event $\langle$ *crb*, *Broadcast* | m $\rangle$ **do**
$W := V$;
$W[rank(self)] := lsn$;
$lsn := lsn + 1$;
trigger $\langle$ *rb*, *Broadcast* | [DATA, W, m] $\rangle$;

upon event $\langle$ *rb*, *Deliver* | p, [DATA, W, m] $\rangle$ **do**
$pending := pending \cup \{(p, W, m)\}$;
while exists $(p', W', m') \in pending$ such that $W' \leq V$ **do**
$pending := pending \setminus \{(p', W', m')\}$;
$V[rank(p')] := V[rank(p')] + 1$;
trigger $\langle$ *crb*, *Deliver* | p', m' $\rangle$;

The function rank() associates an entry of the vector to each process



Causal Order Broadcast Implementation

Algorithm 3.13: No-Waiting Causal Broadcast

Implements:
CausalOrderReliableBroadcast, **instance** *crb*.

Uses:
ReliableBroadcast, **instance** *rb*.

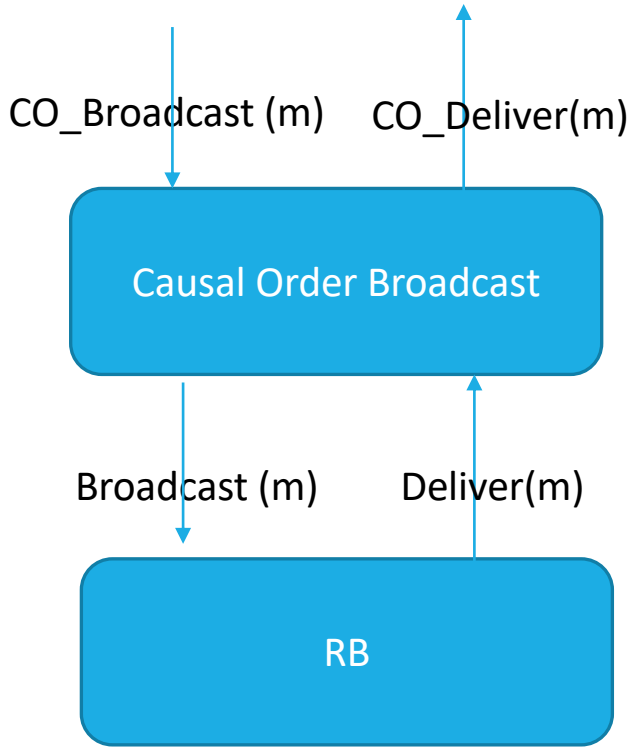
upon event ⟨ *crb*, *Init* ⟩ **do**
delivered := ∅;
past := [];

upon event ⟨ *crb*, *Broadcast* | m ⟩ **do**
trigger ⟨ *rb*, *Broadcast* | [DATA, *past*, m] ⟩;
append(*past*, (*self*, m));

upon event ⟨ *rb*, *Deliver* | p, [DATA, *mpast*, m] ⟩ **do**
if $m \notin$ *delivered* **then**
forall $(s, n) \in$ *mpast* **do**
if $n \notin$ *delivered* **then**
trigger ⟨ *crb*, *Deliver* | s, n ⟩;
delivered := *delivered* ∪ {n};
if $(s, n) \notin$ *past* **then**
append(*past*, (s, n));
trigger ⟨ *crb*, *Deliver* | p, m ⟩;
delivered := *delivered* ∪ {m};
if $(p, m) \notin$ *past* **then**
append(*past*, (p, m));

append(L, x) adds an element x at the end of list L

by the order in the list



Total Order Algorithm

Algorithm 6.1: Consensus-Based Total-Order Broadcast

Implements:
TotalOrderBroadcast, **instance** *tob*.

Uses:
ReliableBroadcast, **instance** *rb*;
Consensus (multiple instances).

upon event $\langle$ *tob*, *Init* $\rangle$ **do**
unordered := $\emptyset$;
delivered := $\emptyset$;
round := 1;
wait := FALSE;

upon event $\langle$ *tob*, *Broadcast* | m $\rangle$ **do**
trigger $\langle$ *rb*, *Broadcast* | m $\rangle$;

upon event $\langle$ *rb*, *Deliver* | p, m $\rangle$ **do**
if $m \notin$ *delivered* **then**
unordered := *unordered* $\cup \{(p, m)\}$;

upon *unordered* $\neq \emptyset \wedge$ *wait* = FALSE **do**
wait := TRUE;
Initialize a new instance *c.round* of consensus;
trigger $\langle$ *c.round*, *Propose* | *unordered* $\rangle$;

upon event $\langle$ *c.r*, *Decide* | *decided* $\rangle$ **such that** r = *round* **do**
forall $(s, m) \in$ *sort*(*decided*) **do** // by the order in the resulting sorted list
trigger $\langle$ *tob*, *Deliver* | s, m $\rangle$;
delivered := *delivered* $\cup$ *decided*;
unordered := *unordered* $\setminus$ *decided*;
round := *round* + 1;
wait := FALSE;

(1,N) regular register

Algorithm 4.1: Read-One Write-All

Implements:

$(1, N)$-RegularRegister, **instance** *onrr*.

Uses:

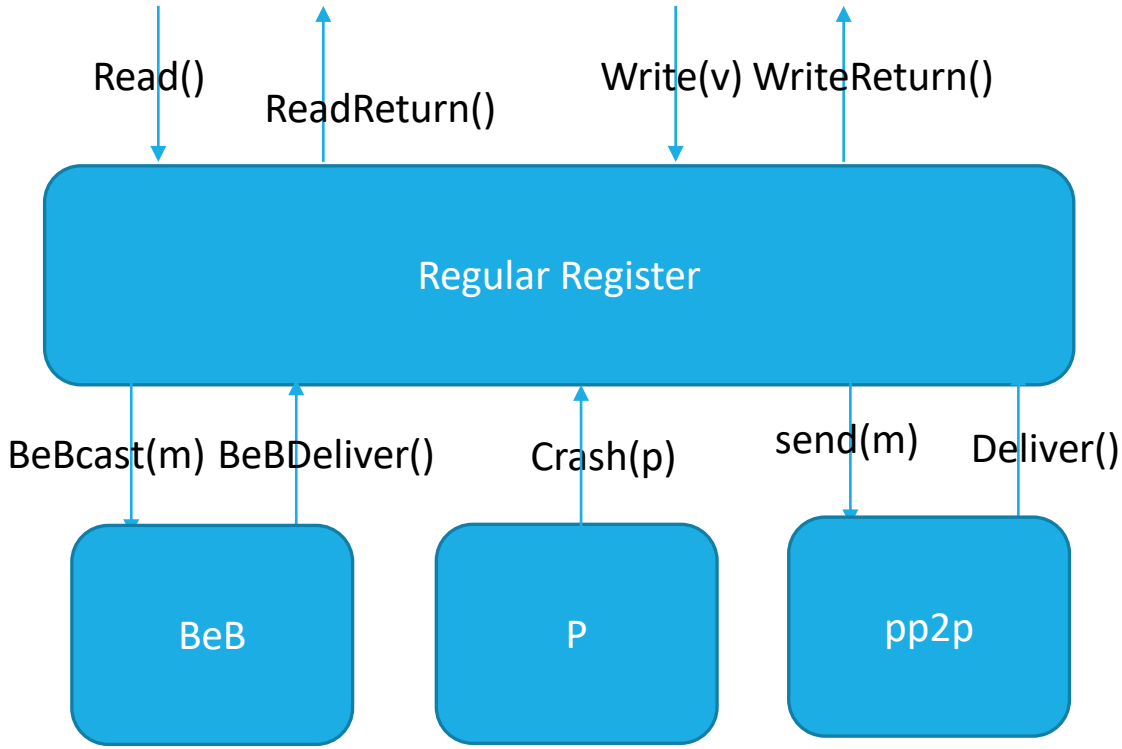
BestEffortBroadcast, **instance** *beb*;
PerfectPointToPointLinks, **instance** *pl*;
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ *onrr*, *Init* $\rangle$ **do**

$val := \perp$;
$correct := \Pi$;
$writeset := \emptyset$;

upon event $\langle$ $\mathcal{P}$, *Crash* | p $\rangle$ **do**

$correct := correct \setminus \{p\}$;



(1,N) regular register

upon event $\langle$ *onrr*, *Read* $\rangle$ **do**
    trigger $\langle$ *onrr*, *ReadReturn* | *val* $\rangle$;

upon event $\langle$ *onrr*, *Write* | *v* $\rangle$ **do**
    trigger $\langle$ *beb*, *Broadcast* | [WRITE, *v*] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | *q*, [WRITE, *v*] $\rangle$ **do**
    val := *v*;
    trigger $\langle$ *pl*, *Send* | *q*, ACK $\rangle$;

upon event $\langle$ *pl*, *Deliver* | *p*, ACK $\rangle$ **do**
    writeset := *writeset* $\cup$ {*p*};

upon *correct* $\subseteq$ *writeset* **do**
    writeset := $\emptyset$;
    trigger $\langle$ *onrr*, *WriteReturn* $\rangle$;

Read() operation Implementation

Write() operation Implementation

(1, N) Regular Register: write() operation

upon event $\langle$ *onrr*, *Write* | v $\rangle$ **do**
    $wts := wts + 1$;
    $acks := 0$;
    trigger $\langle$ *beb*, *Broadcast* | [WRITE, wts, v] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, [WRITE, ts', v'] $\rangle$ **do**
    if $ts' > ts$ **then**
        $(ts, val) := (ts', v')$;
    trigger $\langle$ *pl*, *Send* | p, [ACK, ts'] $\rangle$;

upon event $\langle$ *pl*, *Deliver* | q, [ACK, ts'] $\rangle$ **such that** $ts' = wts$ **do**
    $acks := acks + 1$;
    if $acks > N/2$ **then**
        $acks := 0$;
        trigger $\langle$ *onrr*, *WriteReturn* $\rangle$;

(1,N) regular register

Algorithm 4.2: Majority Voting Regular Register

Implements:

$(1, N)$-RegularRegister, **instance** *onrr*.

Uses:

BestEffortBroadcast, **instance** *beb*;

PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle$ *onrr*, *Init* $\rangle$ **do**

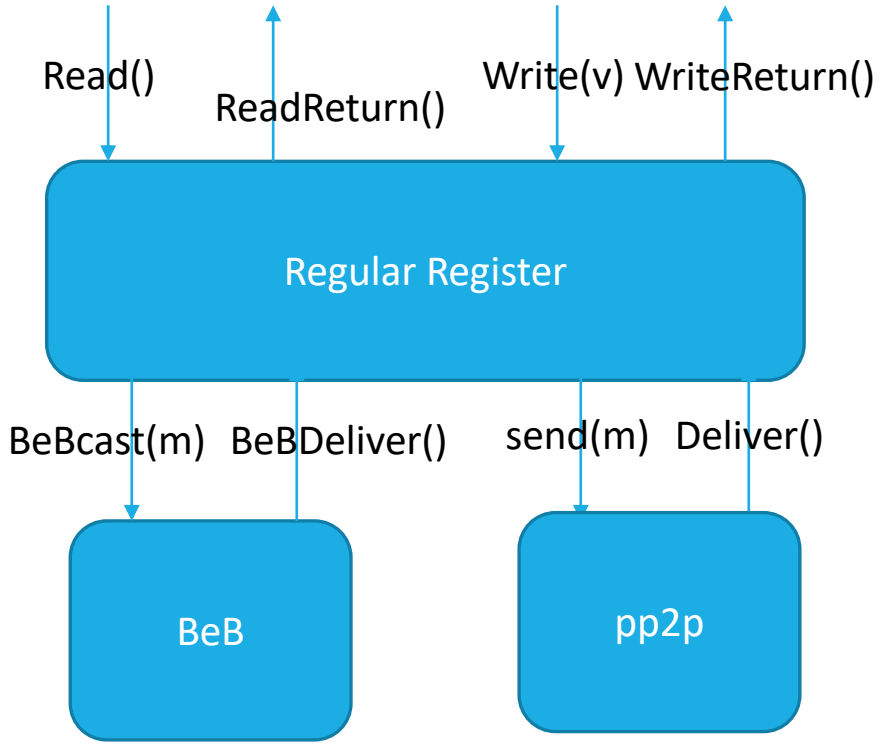
$(ts, val) := (0, \perp)$;

$wts := 0$;

$acks := 0$;

$rid := 0$;

$readlist := [\perp]^N$;



(1,N) regular register: Read() operation

upon event $\langle$ *onrr*, *Read* $\rangle$ **do**
    $rid := rid + 1$;
    $readlist := [\perp]^N$;
    trigger $\langle$ *beb*, *Broadcast* | [READ, *rid*] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, [READ, r] $\rangle$ **do**
    trigger $\langle$ *pl*, *Send* | p, [VALUE, r, ts, val] $\rangle$;

upon event $\langle$ *pl*, *Deliver* | q, [VALUE, r, ts', v'] $\rangle$ **such that** $r = rid$ **do**
    $readlist[q] := (ts', v')$;
    if $\#(readlist) > N/2$ **then**
        $v := highestval(readlist)$;
        $readlist := [\perp]^N$;
        trigger $\langle$ *onrr*, *ReadReturn* | v $\rangle$;

(1,N)Regular Register→ (1,1)Atomic Register

Algorithm 4.3: From $(1, N)$ Regular to $(1, 1)$ Atomic Registers

Implements:
$(1, 1)$-AtomicRegister, **instance** *ooar*.

Uses:
$(1, N)$-RegularRegister, **instance** *onrr*.

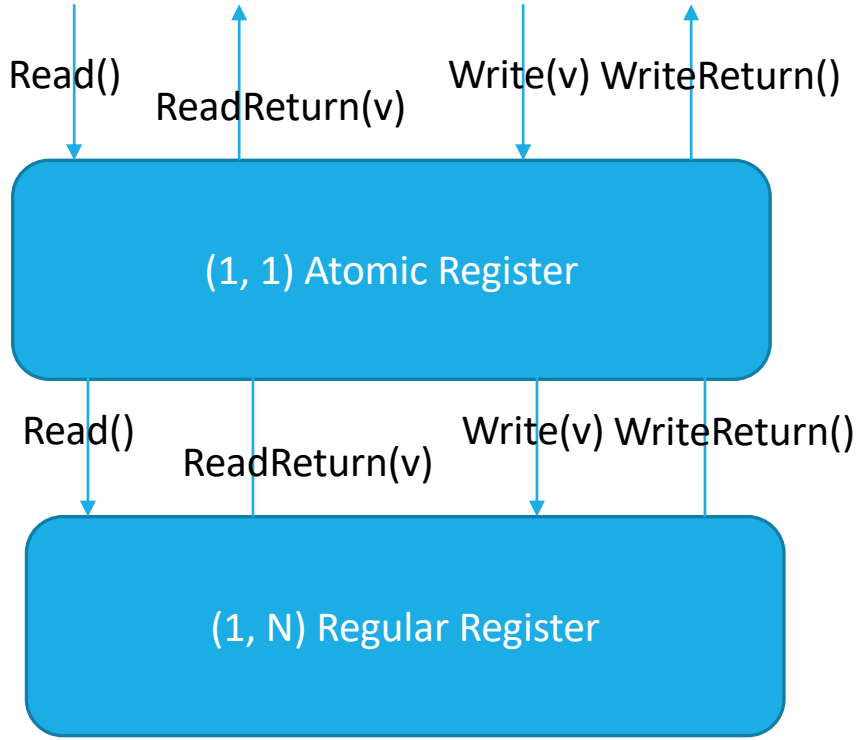
upon event $\langle$ *ooar*, *Init* $\rangle$ **do**
$(ts, val) := (0, \perp)$;
$wts := 0$;

upon event $\langle$ *ooar*, *Write* | v $\rangle$ **do**
$wts := wts + 1$;
trigger $\langle$ *onrr*, *Write* | (wts, v) $\rangle$;

upon event $\langle$ *onrr*, *WriteReturn* $\rangle$ **do**
trigger $\langle$ *ooar*, *WriteReturn* $\rangle$;

upon event $\langle$ *ooar*, *Read* $\rangle$ **do**
trigger $\langle$ *onrr*, *Read* $\rangle$;

upon event $\langle$ *onrr*, *ReadReturn* | (ts', v') $\rangle$ **do**
if $ts' > ts$ **then**
$(ts, val) := (ts', v')$;
trigger $\langle$ *ooar*, *ReadReturn* | val $\rangle$;



(1,1)Atomic Register → (1,N) Atomic Register: Phase 2

upon event $\langle$ *onar*, *Write* | v $\rangle$ **do**
    $ts := ts + 1$;
    writing := TRUE;
    forall $q \in \Pi$ **do**
        trigger $\langle$ *ooar.q.self*, *Write* | (ts, v) $\rangle$;

upon event $\langle$ *ooar.q.self*, *WriteReturn* $\rangle$ **do**
    acks := *acks* + 1;
    if *acks* = N **then**
        acks := 0;
        if *writing* = TRUE **then**
            trigger $\langle$ *onar*, *WriteReturn* $\rangle$;
            writing := FALSE;
        else
            trigger $\langle$ *onar*, *ReadReturn* | *readval* $\rangle$;

(1,1)Atomic Register → (1,N) Atomic Register: Phase 2

upon event $\langle$ *onar*, *Read* $\rangle$ **do**
    forall $r \in \Pi$ **do**
        trigger $\langle$ *ooar.self.r*, *Read* $\rangle$;

upon event $\langle$ *ooar.self.r*, *ReadReturn* | (ts', v') $\rangle$ **do**
    readlist$[r]$:= (ts', v');
    if $\#(readlist) = N$ **then**
        $(maxts, readval)$:= *highest*$(readlist)$;
        readlist := $[\perp]^N$;
        forall $q \in \Pi$ **do**
            trigger $\langle$ *ooar.q.self*, *Write* | $(maxts, readval)$ $\rangle$;

Read-Impose Write-All (1,N) Atomic Register

Algorithm 4.5: Read-Impose Write-All

Implements:

$(1, N)$-AtomicRegister, **instance** *onar*.

Uses:

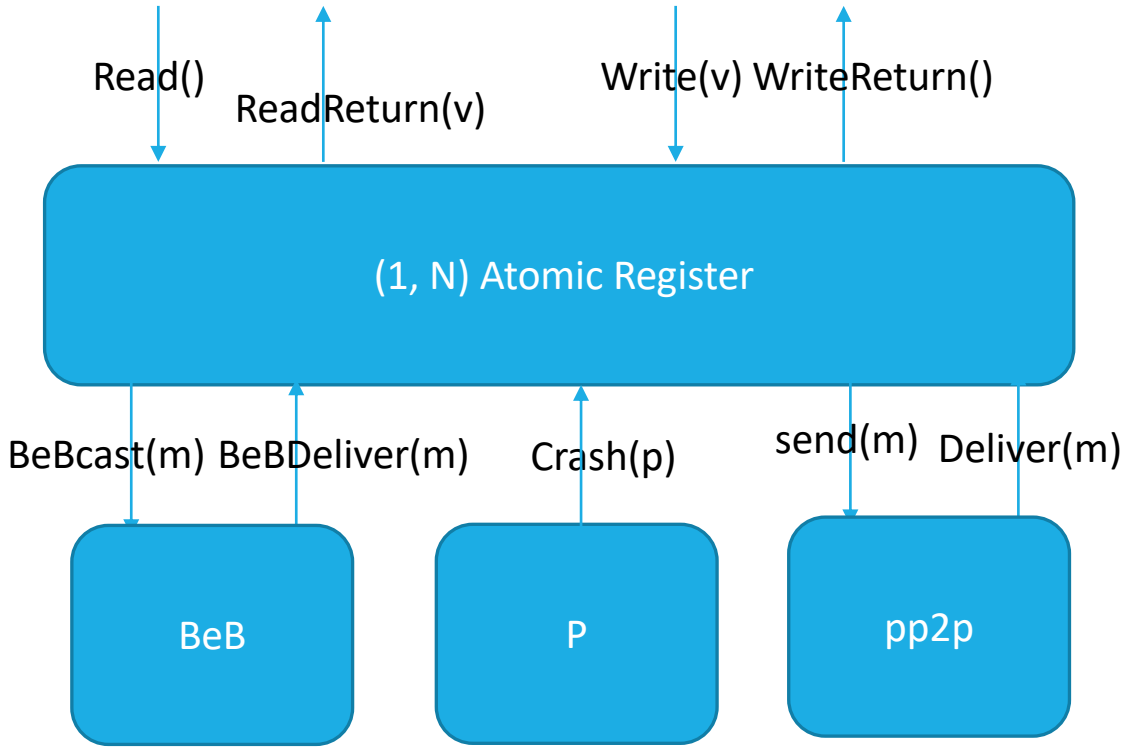
BestEffortBroadcast, **instance** *beb*;
PerfectPointToPointLinks, **instance** *pl*;
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ *onar*, *Init* $\rangle$ **do**

$(ts, val) := (0, \perp)$;
correct := Π;
writeset := $\emptyset$;
readval := $\perp$;
reading := FALSE;

upon event $\langle$ $\mathcal{P}$, *Crash* | p $\rangle$ **do**

correct := *correct* $\setminus$ $\{p\}$;



Read-Impose Write-All (1,N) Atomic Register

```
upon event ⟨ onar, Read ⟩ do
    reading := TRUE;
    readval := val;
    trigger ⟨ beb, Broadcast | [WRITE, ts, val] ⟩;

upon event ⟨ onar, Write | v ⟩ do
    trigger ⟨ beb, Broadcast | [WRITE, ts + 1, v] ⟩;

upon event ⟨ beb, Deliver | p, [WRITE, ts', v'] ⟩ do
    if ts' > ts then
        (ts, val) := (ts', v');
    trigger ⟨ pl, Send | p, [ACK] ⟩;

upon event ⟨ pl, Deliver | p, [ACK] ⟩ then
    writeset := writeset ∪ {p};

upon correct ⊆ writeset do
    writeset := ∅;
    if reading = TRUE then
        reading := FALSE;
        trigger ⟨ onar, ReadReturn | readval ⟩;
    else
        trigger ⟨ onar, WriteReturn ⟩;
```

Read() operation Implementation

Write() operation Implementation

Read-Impose Write-Majority (1,N) Atomic Register

Algorithm 4.6: Read-Impose Write-Majority (part 1, read)

Implements:

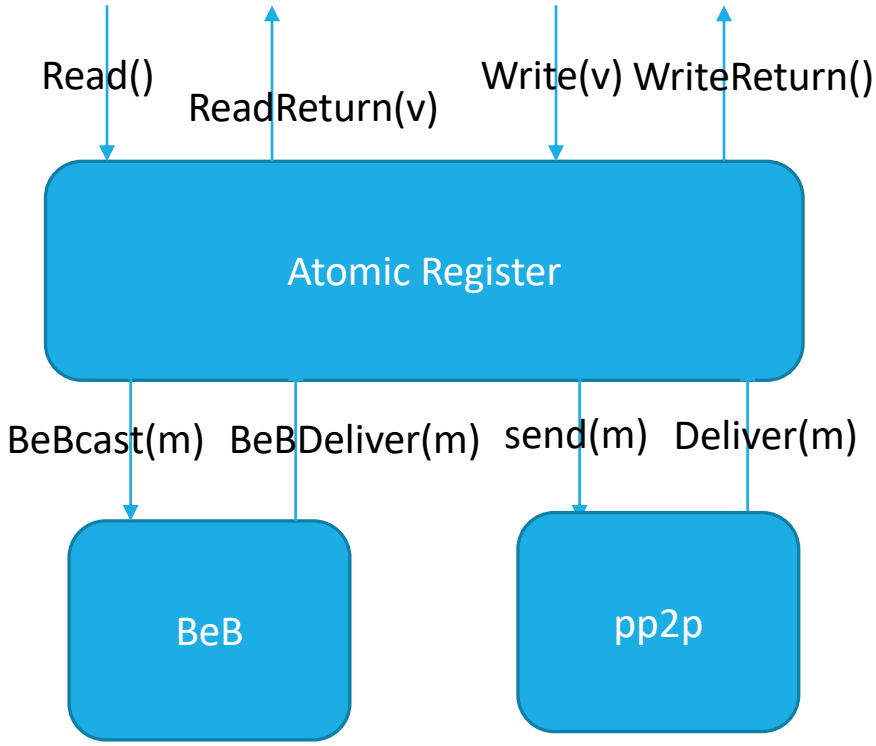
$(1, N)$-AtomicRegister, **instance** *onar*.

Uses:

BestEffortBroadcast, **instance** *beb*;
PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle$ *onar*, *Init* $\rangle$ **do**

$(ts, val) := (0, \perp)$;
$wts := 0$;
$acks := 0$;
$rid := 0$;
$readlist := [\perp]^N$;
$readval := \perp$;
$reading :=$ FALSE;



Read-Impose Write-Majority (1,N) Atomic Register

upon event $\langle$ *onar*, *Read* $\rangle$ **do**
    $rid := rid + 1$;
    $acks := 0$;
    $readlist := [\bot]^N$;
    $reading :=$ TRUE;
    trigger $\langle$ *beb*, *Broadcast* | [READ, *rid*] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, [READ, r] $\rangle$ **do**
    trigger $\langle$ *pl*, *Send* | p, [VALUE, r, ts, val] $\rangle$;

upon event $\langle$ *pl*, *Deliver* | q, [VALUE, r, ts', v'] $\rangle$ **such that** $r = rid$ **do**
    $readlist[q] := (ts', v')$;
    if $\#(readlist) > N/2$ **then**
        $(maxts, readval) :=$ *highest*$(readlist)$;
        $readlist := [\bot]^N$;
        trigger $\langle$ *beb*, *Broadcast* | [WRITE, *rid*, *maxts*, *readval*] $\rangle$;

Read-Impose Write-Majority (1,N) Atomic Register

upon event $\langle$ *onar*, *Write* | v $\rangle$ **do**
    $rid := rid + 1$;
    $wts := wts + 1$;
    $acks := 0$;
    trigger $\langle$ *beb*, *Broadcast* | [WRITE, rid, wts, v] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, [WRITE, r, ts', v'] $\rangle$ **do**
    if $ts' > ts$ **then**
        $(ts, val) := (ts', v')$;
    trigger $\langle$ *pl*, *Send* | p, [ACK, r] $\rangle$;

upon event $\langle$ *pl*, *Deliver* | q, [ACK, r] $\rangle$ **such that** $r = rid$ **do**
    $acks := acks + 1$;
    if $acks > N/2$ **then**
        $acks := 0$;
        if *reading* $=$ TRUE **then**
            reading := FALSE;
            trigger $\langle$ *onar*, *ReadReturn* | *readval* $\rangle$;
        else
            trigger $\langle$ *onar*, *WriteReturn* $\rangle$;

Byzantine consistent broadcast implementation

Algorithm 3.16: Authenticated Echo Broadcast

Implements:
ByzantineConsistentBroadcast, **instance** *bcb*, with sender s.

Uses:
AuthPerfectPointToPointLinks, **instance** *al*.

upon event $\langle$ *bcb*, *Init* $\rangle$ **do**
sentecho := FALSE;
delivered := FALSE;
echos := $[\perp]^N$;

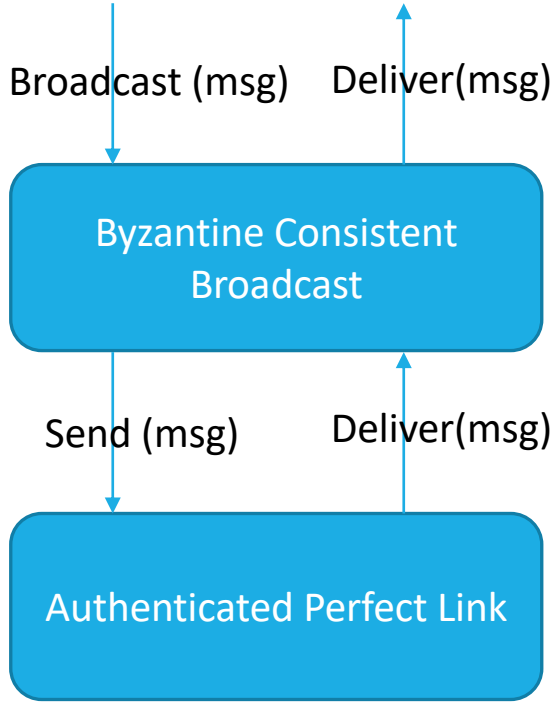
upon event $\langle$ *bcb*, *Broadcast* | m $\rangle$ **do** // only process s
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [SEND, m] $\rangle$;

upon event $\langle$ *al*, *Deliver* | p, [SEND, m] $\rangle$ **such that** $p = s$ **and** *sentecho* $=$ FALSE **do**
sentecho := TRUE;
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [ECHO, m] $\rangle$;

upon event $\langle$ *al*, *Deliver* | p, [ECHO, m] $\rangle$ **do**
if $echos[p] = \perp$ **then**
$echos[p] := m$;

upon exists $m \neq \perp$ such that $\#(\{p \in \Pi \mid echos[p] = m\}) > \frac{N+f}{2}$
and *delivered* $=$ FALSE **do**
delivered := TRUE;
trigger $\langle$ *bcb*, *Deliver* | s, m $\rangle$;

Correctness is ensured if N>3f



Byzantine Reliable Broadcast implementation

Algorithm 3.18: Authenticated Double-Echo Broadcast

Implements:
ByzantineReliableBroadcast, **instance** *brb*, with sender s.

Uses:
AuthPerfectPointToPointLinks, **instance** *al*.

upon event $\langle$ *brb*, *Init* $\rangle$ **do**
sentecho := FALSE;
sentready := FALSE;
delivered := FALSE;
echos := $[\bot]^N$;
readys := $[\bot]^N$;

upon event $\langle$ *brb*, *Broadcast* | m $\rangle$ **do** // only process s
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [SEND, m] $\rangle$;

upon event $\langle$ *al*, *Deliver* | p, [SEND, m] $\rangle$ **such that** $p = s$ **and** *sentecho* $=$ FALSE **do**
sentecho := TRUE;
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [ECHO, m] $\rangle$;

upon event $\langle$ *al*, *Deliver* | p, [ECHO, m] $\rangle$ **do**
if *echos*$[p] = \bot$ **then**
echos$[p] := m$;

upon exists $m \neq \bot$ such that $\#(\{p \in \Pi \mid echos[p] = m\}) > \frac{N+f}{2}$
and *sentready* $=$ FALSE **do**
sentready := TRUE;
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [READY, m] $\rangle$;

upon event $\langle$ *al*, *Deliver* | p, [READY, m] $\rangle$ **do**
if *readys*$[p] = \bot$ **then**
readys$[p] := m$;

upon exists $m \neq \bot$ such that $\#(\{p \in \Pi \mid readys[p] = m\}) > f$
and *sentready* $=$ FALSE **do**
sentready := TRUE;
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [READY, m] $\rangle$;

upon exists $m \neq \bot$ such that $\#(\{p \in \Pi \mid readys[p] = m\}) > 2f$
and *delivered* $=$ FALSE **do**
delivered := TRUE;
trigger $\langle$ *brb*, *Deliver* | s, m $\rangle$;

Algorithm 4.14: Byzantine Masking Quorum

Implements:
$(1, N)$-ByzantineSafeRegister, **instance** *bonsr*, with writer w.
Uses:
AuthPerfectPointToPointLinks, **instance** *al*.

upon event $\langle$ *bonsr*, *Init* $\rangle$ **do**
$(ts, val) := (0, \perp)$;
$wts := 0$;
acklist $:= [\perp]^N$;
rid $:= 0$;
readlist $:= [\perp]^N$;

upon event $\langle$ *bonsr*, *Write* | v $\rangle$ **do** // only process w
$wts := wts + 1$;
acklist $:= [\perp]^N$;
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [WRITE, wts, v] $\rangle$;

upon event $\langle$ *al*, *Deliver* | p, [WRITE, ts', v'] $\rangle$ **such that** $p = w$ **do**
if $ts' > ts$ **then**
$(ts, val) := (ts', v')$;
trigger $\langle$ *al*, *Send* | p, [ACK, ts'] $\rangle$;

byzhighestval (·): selects the value from the pair that occurs more than f time and with the highest timestamp. If no pair exists, the reader selects a default value v0 from the domain of the register.

Assumption
N>4f

upon event $\langle$ *al*, *Deliver* | q, [ACK, ts'] $\rangle$ **such that** $ts' = wts$ **do**
1 *acklist*[q] := ACK;
if $\#(acklist) > (N + 2f)/2$ **then**
acklist $:= [\perp]^N$;
trigger $\langle$ *bonsr*, *WriteReturn* $\rangle$;

upon event $\langle$ *bonsr*, *Read* $\rangle$ **do**
rid := *rid* + 1;
readlist $:= [\perp]^N$;
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [READ, *rid*] $\rangle$;

upon event $\langle$ *al*, *Deliver* | p, [READ, r] $\rangle$ **do**
trigger $\langle$ *al*, *Send* | p, [VALUE, r, ts, val] $\rangle$;

upon event $\langle$ *al*, *Deliver* | q, [VALUE, r, ts', v'] $\rangle$ **such that** $r = rid$ **do**
readlist[q] := (ts', v');
2 **if** $\#(readlist) > \frac{N+2f}{2}$ **then**
v := byzhighestval(*readlist*);
readlist $:= [\perp]^N$;
trigger $\langle$ *bonsr*, *ReadReturn* | v $\rangle$;

Regular Register
Implementation with cryptographic assumptions

Algorithm 4.15: Authenticated-Data Byzantine Quorum

Implements:
$(1, N)$-ByzantineRegularRegister, **instance** *bonrr*, with writer w.
Uses:
AuthPerfectPointToPointLinks, **instance** *al*.

upon event $\langle$ *bonrr*, *Init* $\rangle$ **do**
$(ts, val, \sigma) := (0, \perp, \perp)$;
$wts := 0$;
acklist $:= [\perp]^N$;
rid $:= 0$;
readlist $:= [\perp]^N$;

upon event $\langle$ *bonrr*, *Write* | v $\rangle$ **do** // only process w
$wts := wts + 1$;
acklist $:= [\perp]^N$;
$\sigma := sign(self, bonrr \| self \| \text{WRITE} \| wts \| v)$;
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [WRITE, wts, v, σ] $\rangle$;

upon event $\langle$ *al*, *Deliver* | p, [WRITE, ts', v', σ'] $\rangle$ **such that** $p = w$ **do**
if $ts' > ts$ **then**
$(ts, val, \sigma) := (ts', v', \sigma')$;
trigger $\langle$ *al*, *Send* | p, [ACK, ts'] $\rangle$;

upon event $\langle$ *al*, *Deliver* | q, [ACK, ts'] $\rangle$ **such that** $ts' = wts$ **do**
acklist[q] := ACK;
if $\#(acklist) > (N + f)/2$ **then**
acklist $:= [\perp]^N$;
trigger $\langle$ *bonrr*, *WriteReturn* $\rangle$;

i.e. > 2f

Assumption
N>3f

upon event $\langle$ *bonrr*, *Read* $\rangle$ **do**
rid := *rid* + 1;
readlist $:= [\perp]^N$;
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [READ, *rid*] $\rangle$;

upon event $\langle$ *al*, *Deliver* | p, [READ, r] $\rangle$ **do**
trigger $\langle$ *al*, *Send* | p, [VALUE, r, ts, val, σ] $\rangle$;

upon event $\langle$ *al*, *Deliver* | q, [VALUE, r, ts', v', σ'] $\rangle$ **such that** $r = rid$ **do**
if $verifysig(q, bonrr \| w \| \text{WRITE} \| ts' \| v', \sigma')$ **then**
readlist[q] := (ts', v');
if $\#(readlist) > \frac{N+f}{2}$ **then**
$v := highestval(readlist)$;
readlist $:= [\perp]^N$;
trigger $\langle$ *bonrr*, *ReadReturn* | v $\rangle$;

i.e. > 2f