

Distributed Algorithms

Fall 2020

Reliable & Causal Broadcast
1st exercise session, 28/09/2020

Matteo Monti <matteo.monti@epfl.ch>
Jovan Komatovic <jovan.komatovic@epfl.ch>

Reliable broadcast

Specification:

- **Validity:** If a *correct* process broadcasts m , then it eventually delivers m .
- **Integrity:** m is delivered by a process at most once, and only if it was previously broadcast.
- **Agreement:** If a correct process delivers m , then all correct processes eventually deliver m .

Algorithm: 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 $\langle rb, Broadcast \mid m \rangle$ **do**

trigger $\langle beb, Broadcast \mid [DATA, self, m] \rangle$;

upon event $\langle beb, Deliver \mid p, [DATA, s, m] \rangle$ **do**

if $m \notin from[s]$ **then**

trigger $\langle rb, Deliver \mid s, m \rangle$;

from[*s*] := *from*[*s*] $\cup \{m\}$;

if $s \notin correct$ **then**

trigger $\langle beb, Broadcast \mid [DATA, s, m] \rangle$;

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

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

forall $m \in from[p]$ **do**

trigger $\langle beb, Broadcast \mid [DATA, p, m] \rangle$;

Strong accuracy:

No correct process is ever suspected:

$$\forall F, \forall H, \forall t \in \mathcal{T}, \forall p \in correct(F), \forall q : p \notin H(q, t)$$

Strong completeness:

Eventually, every faulty process is permanently suspected by **every** correct process:

$$\forall F, \forall H, \exists t \in \mathcal{T}, \forall p \in crashed(F), \forall q \in correct(F), \forall t' \geq t : p \in H(q, t')$$

Where:

- $crashed(F)$ is the set of crashed processes.
- $correct(F)$ is the set of correct processes.
- $H(p, t)$ is the output of the failure detector of process p at time t .

Exercise 1

Implement a reliable broadcast algorithm without using any failure detector, i.e., using only *BestEffort-Broadcast(BEB)*.

Exercise 2

The reliable broadcast algorithm presented in class has the processes continuously fill their different buffers without emptying them.

✦ **Implements:** ReliableBroadcast (rb).

✦ **Uses:**

- ✦ BestEffortBroadcast (beb).
- ✦ PerfectFailureDetector (P).

✦ **upon event** < Init > **do**

- ✦ `delivered := ∅;`
- ✦ `correct := S;`
- ✦ **forall** `pi ∈ S` **do** `from[pi] := ∅;`

✦ **upon event** < rbBroadcast, m > **do**

- ✦ `delivered := delivered U {m};`
- ✦ **trigger** < rbDeliver, self, m >;
- ✦ **trigger** < bebBroadcast, [Data,self,m] >;

✦ **upon event** < crash, pi > **do**

- ✦ `correct := correct \ {pi};`
- ✦ **forall** `[pj,m] ∈ from[pi]` **do**
- ✦ **trigger** < bebBroadcast,[Data,pj,m] >;

✦ **upon event** < bebDeliver, pi, [Data,pj,m] > **do**

- ✦ **if** `m ∉ delivered` **then**
- ✦ `delivered := delivered U {m};`
- ✦ **trigger** < rbDeliver, pj, m >;
- ✦ **if** `pi ∉ correct` **then**
- ✦ **trigger** < bebBroadcast,[Data,pj,m] >;
- ✦ **else**
- ✦ `from[pi] := from[pi] U {[pj,m]};`

Modify it to remove (i.e. garbage collect) unnecessary messages from the buffers:

- A. *from*, and
- B. *delivered*

Uniform reliable broadcast

Specification:

- **Validity:** If a *correct* process broadcasts m , then it eventually delivers m .
- **Integrity:** m is delivered by a process at most once, and only if it was previously broadcast.
- **Uniform Agreement:** If a ~~correct~~ process delivers m , then all correct processes eventually deliver m .

Algorithm: All-Ack Uniform Reliable Broadcast

Implements:

UniformReliableBroadcast, **instance** *urb*.

Uses:

BestEffortBroadcast, **instance** *beb*.

PerfectFailureDetector, **instance** \mathcal{P} .

upon event $\langle \textit{urb}, \textit{Init} \rangle$ **do**

delivered := \emptyset ;

pending := \emptyset ;

correct := Π ;

forall *m* **do** *ack*[*m*] := \emptyset ;

upon event $\langle \textit{urb}, \textit{Broadcast} \mid m \rangle$ **do**

pending := *pending* \cup $\{(self, m)\}$;

trigger $\langle \textit{beb}, \textit{Broadcast} \mid [DATA, self, m] \rangle$;

upon event $\langle \textit{beb}, \textit{Deliver} \mid p, [DATA, s, m] \rangle$ **do**

ack[*m*] := *ack*[*m*] \cup $\{p\}$;

if $(s, m) \notin \textit{pending}$ **then**

pending := *pending* \cup $\{(s, m)\}$;

trigger $\langle \textit{beb}, \textit{Broadcast} \mid [DATA, s, m] \rangle$;

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

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

function *candeliver*(*m*) **returns** Boolean **is**

return (*correct* \subseteq *ack*[*m*]);

upon exists $(s, m) \in \textit{pending}$ such that *candeliver*(*m*) \wedge $m \notin \textit{delivered}$ **do**

delivered := *delivered* \cup $\{m\}$;

trigger $\langle \textit{urb}, \textit{Deliver} \mid s, m \rangle$;

Exercise 3

What happens in the reliable broadcast and uniform reliable broadcast algorithms if the:

- A. accuracy, or
- B. completeness

property of the failure detector is violated?

Exercise 4

Implement a **uniform** reliable broadcast algorithm without using any failure detector, i.e., using only *BestEffort-Broadcast(BEB)*.

Causal Broadcast

Definition (Happens-before):

We say that an event e happens-before an event e' , and we write $e \rightarrow e'$, if one of the following three cases holds (is true):

$\exists p_i \in \Pi$ s.t. $e = e_i^r, e' = e_i^s, r < s$ (e and e' are executed by the same process)

$e = \text{send}(m, *) \wedge e' = \text{receive}(m)$ (e and e' are send/receive events of a message respectively)

$\exists e''$ s.t. $e \rightarrow e'' \rightarrow e'$ (i.e. \rightarrow is transitive)

Causal Broadcast

Specification:

It has the same specification of reliable broadcast, with the additional ordering constraint of causal order.

More precisely (causal order):

$$\text{broadcast}_p(m) \rightarrow \text{broadcast}_q(m') \Rightarrow \text{deliver}_r(m) \rightarrow \text{deliver}_r(m')$$

Which means that:

If the broadcast of a message m *happens-before* the broadcast of a message m' , then no process delivers m' unless it has previously delivered m .

Exercise 5

Can we devise a broadcast algorithm that does **not** ensure the causal delivery property **but only** (in) its non-uniform variant:

No correct process p_i delivers a message m_2 unless p_i has already delivered every message m_1 such that $m_1 \rightarrow m_2$?

Exercise 6

Suggest a memory optimization of the garbage collection scheme of the following algorithm:

No-Waiting Causal Broadcast

Implements:

CausalOrderReliableBroadcast, **instance** *crb*.

Uses:

ReliableBroadcast, **instance** *rb*.

upon event $\langle crb, Init \rangle$ do

```
delivered :=  $\emptyset$ ;  
past := [];
```

upon event $\langle crb, Broadcast \mid m \rangle$ do

```
trigger  $\langle rb, Broadcast \mid [DATA, past, m] \rangle$ ;  
append(past, (self, m));
```

upon event $\langle rb, Deliver \mid p, [DATA, mpast, m] \rangle$ do

```
if  $m \notin delivered$  then  
  forall  $(s, n) \in mpast$  do // by the order in the list  
    if  $n \notin delivered$  then  
      trigger  $\langle crb, Deliver \mid s, n \rangle$ ;  
      delivered := delivered  $\cup \{n\}$ ;  
      if  $(s, n) \notin past$  then  
        append(past, (s, n));  
  trigger  $\langle crb, Deliver \mid p, m \rangle$ ;  
  delivered := delivered  $\cup \{m\}$ ;  
  if  $(p, m) \notin past$  then  
    append(past, (p, m));
```

Garbage-Collection of Causal Past in the “No-Waiting Causal Broadcast”

Implements:

CausalOrderReliableBroadcast, **instance** *crb*.

Uses:

ReliableBroadcast, **instance** *rb*;

PerfectFailureDetector, **instance** \mathcal{P} .

// Except for its $\langle Init \rangle$ event handler, the pseudo code on the left is // part of this algorithm.

upon event $\langle crb, Init \rangle$ do

```
delivered :=  $\emptyset$ ;  
past := [];  
correct :=  $\Pi$ ;  
forall  $m$  do ack[m] :=  $\emptyset$ ;
```

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

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

upon exists $m \in delivered$ such that $self \notin ack[m]$ do

```
ack[m] := ack[m]  $\cup \{self\}$ ;  
trigger  $\langle rb, Broadcast \mid [ACK, m] \rangle$ ;
```

upon event $\langle rb, Deliver \mid p, [ACK, m] \rangle$ do

```
ack[m] := ack[m]  $\cup \{p\}$ ;
```

upon correct $\subseteq ack[m]$ do

```
forall  $(s', m') \in past$  such that  $m' = m$  do  
  remove(past, (s', m));
```

Exercise 7

Can we devise a Best-effort Broadcast algorithm that satisfies the causal delivery property, *without* being a causal broadcast algorithm, i.e., without satisfying the *agreement* property of a reliable broadcast?