Distributed Algorithms

Reliable & Causal Broadcast - solutions
2nd exercise session

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* | 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] $\rangle$;

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 1 (Solution)

Use a step of all-to-all communication.

In particular, very process that gets a message relays it immediately.

Recall that in the original algorithm, processes were relaying messages from a process p only if p crashes.

```
upon initialization do
        delivered := {}

upon RB-broadcast(m) do
        send(m) to Π \ {p}
        RB-deliver(m)

upon BEB-receive(m) from q do
        if not m ∈ delivered
                send (m) to Π \ {p, q}
                RB-deliver(m)
                delivered := delivered ∪ m
```

Agreement: Before RB-delivering m, a correct process p forwards m to all processes. By the properties of perfect channels and the fact that p is correct, all correct processes will eventually receive m and RB-deliver it.

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*

Exercise 2 (Solution)

A. The *from* buffer is used only to store messages that are relayed in the case of a failure. Therefore, messages from the *from* buffer can be removed as soon as they are relayed.

B. Messages from the *delivered* array cannot be removed. Consider this scenario: If a process crashes and its messages are retransmitted by two different processes, then a process might RB-deliver the same message twice if it empties the *delivered* buffer in the meantime. This is a violation of the "no duplication" property.

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 ⟨ *urb*, *Init* ⟩ **do**
delivered := $\emptyset$;
pending := $\emptyset$;
correct := Π;
forall m **do** *ack*[m] := $\emptyset$;

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

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

upon event ⟨ $\mathcal{P}$, *Crash* | p ⟩ **do**
correct := *correct* \ {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 ⟨ *urb*, Deliver | s, m ⟩;

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 3 (Solution 1/2)

Reliable broadcast:

1. Suppose that accuracy is violated. Then, the processes might be relaying messages when this is not really necessary. This wastes resource, but does not impact correctness.
2. Suppose that completeness is violated. Then, the processes might not be relaying messages they should be relaying. This may violate agreement. For instance, assume that only a single process p_1 BEB-delivers (hence RB-delivers) a message m from a crashed process p_2. If a failure detector (at p_1) does not ever suspect p_2, no other correct process will deliver m (agreement is violated).

Exercise 3 (Solution 2/2)

Uniform Reliable broadcast:

Consider a system of three processes p_1, p_2 and p_3. Assume that p_1 URB-broadcasts the message m.

1. Suppose that accuracy is violated. Assume that p_1 falsely suspects p_2 and p_3 to have crashed. p_1 eventually URB-delivers m. Assume that p_1 crashes afterwards. It may happen that p_2 and p_3 never BEB-deliver m and have no knowledge about m (uniform agreement is violated).
2. Suppose that completeness is violated. p_1 might never URB-deliver m if either p_2 or p_3 crashes and p_1 never detects their crash. Hence, p_1 would wait indefinitely for p_2 and p_3 to relay m (validity property violation)

Exercise 4

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

Exercise 4 (Solution)

Just modify the “candeliver” function.

```
Function candeliver(m) returns Boolean is
        return #(ack[m]) > N / 2
```

Uniform agreement:
Suppose that a correct process delivers m. That means that at least one correct process p “acknowledged” m (rebroadcast m using BestEffortBroadcast). Consequently, all correct processes eventually deliver m from BestEffortBroadcast broadcast by p and rebroadcast m themselves (if they have not done that yet). Hence, every correct process eventually collects at least N/2 acknowledgements and delivers m.