Distributed systems

Reliable Broadcast

Prof R. Guerraoui
Lpdwww.epfl.ch
Broadcast

A → broadcast

m → deliver

B

C
Broadcast abstractions

- Best-effort broadcast
- Reliable broadcast
- Uniform broadcast
Modules of a process

Applications

(request)

indication

(request)

indication

(request)

indication

(request)

(indication)

(request)

(indication)

(request)

((B-U)Reliable broadcast)

Failure detector

Channels

(indication)

(indication)

(indication)

(indication)

(request)

(request)

(request)

(request)
Intuition

Broadcast is useful for instance in applications where some processes subscribe to events published by other processes (e.g., stocks).

The subscribers might require some reliability guarantees from the broadcast service (we say sometimes quality of service – QoS) that the underlying network does not provide.
Overview

We shall consider three forms of reliability for a broadcast primitive

1. Best-effort broadcast
2. (Regular) reliable broadcast
3. Uniform (reliable) broadcast

We shall give first specifications and then algorithms
Best-effort broadcast (beb)

**Events**

- Request: `<bebBroadcast, m>`
- Indication: `<bebDeliver, src, m>`

- **Properties:** BEB1, BEB2, BEB3
Best-effort broadcast (beb)

Properties

BEB1. Validity: If pi and pj are correct, then every message broadcast by pi is eventually delivered by pj

BEB2. No duplication: No message is delivered more than once

BEB3. No creation: No message is delivered unless it was broadcast
Best-effort broadcast

p1

m delivery

p2

m delivery

p3

delivery
Best-effort broadcast

p1  delivery  m1  crash

p2  delivery  m2

p3  delivery  m1
Reliable broadcast (rb)

*Events*

- Request: `<rbBroadcast, m>`
- Indication: `<rbDeliver, src, m>`

- **Properties:** `RB1, RB2, RB3, RB4`
Reliable broadcast (rb)

**Properties**

- **RB1 = BEB1.**
- **RB2 = BEB2.**
- **RB3 = BEB3.**

**RB4. Agreement:** For any message m, if a correct process delivers m, then every correct process delivers m
Reliable broadcast

\[ \text{delivery} \quad \text{delivery} \quad \text{delivery} \]
Reliable broadcast
Reliable broadcast

p1

m1

delivery

delivery

crash

m2

delivery

delivery

crash

m2

m1

delivery

p2

p3
Uniform broadcast (urb)

**Events**
- Request: `<urbBroadcast, m>`
- Indication: `<urbDeliver, src, m>`

- **Properties:** `URB1, URB2, URB3, URB4`
Uniform broadcast (urb)

**Properties**

- **URB1 = BEB1.**
- **URB2 = BEB2.**
- **URB3 = BEB3.**

**URB4. Uniform Agreement:** For any message m, if a process delivers m, then every correct process delivers m
Uniform reliable broadcast
Uniform reliable broadcast

p1

m1

delivery

p2

m2

delivery

p3

m1

delivery

p1

crash

p2

crash

p3
Overview

We consider three forms of reliability for a broadcast primitive

1. *Best-effort broadcast*
2. *Regular* (reliable) broadcast
3. *Uniform* (reliable) broadcast

We give first *specifications* and then *algorithms*
Algorithm (beb)

- **Implements:** BestEffortBroadcast (beb).
- **Uses:** PerfectLinks (pp2p).

upon event < bebBroadcast, m> do
  forall pi ∈ S do
    trigger < pp2pSend, pi, m>;
  upon event < pp2pDeliver, pi, m> do
    trigger < bebDeliver, pi, m>;
Algorithm (beb)

p1

m
delivery

p2

m
delivery

p3
Algorithm (beb)

Proof (sketch)

BEB1. Validity: By the validity property of perfect links and the very facts that (1) the sender sends the message to all and (2) every correct process that pp2pDelivers a message bebDelivers it

BEB2. No duplication: By the no duplication property of perfect links

BEB3. No creation: By the no creation property of perfect links
Algorithm (beb)
Algorithm \( (rb) \)

- **Implements:** ReliableBroadcast \( (rb) \).
- **Uses:**
  - BestEffortBroadcast \( (beb) \).
  - PerfectFailureDetector \( (P) \).
- **upon event** \( < \text{Init} > \) **do**
  - delivered := \( \emptyset \);
  - correct := \( S \);
  - **forall** \( pi \in S \) **do** from[\( pi \)] := \( \emptyset \);
Algorithm (rb – cont’d)

\begin{verbatim}
\textbf{upon event} < rbBroadcast, m> \textbf{do}
\hspace{1em} delivered := delivered U \{m\};
\hspace{1em} \textbf{trigger} < rbDeliver, self, m>;
\hspace{1em} \textbf{trigger} < bebBroadcast, [Data,self,m]>;
\end{verbatim}
Algorithm (rb – cont’d)

upon event < crash, pi > do
  correct := correct \ {pi};
  forall [pj,m] ∈ from[pi] do
    trigger < bebBroadcast,[Data,pj,m]>;
Algorithm (rb – cont’d)

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]};
Algorithm (rb)

p1

p2

delivery

m

p3

delivery

m
Algorithm (rb)

$p1$

$p2$

crash

delivery

$p3$

delivery

$\text{delivery}$
Algorithm (rb)

**Proof (sketch)**

- **RB1. RB2. RB3:** as for the 1st algorithm

- **RB4. Agreement:** Assume some correct process pi rbDelivers a message m rbBroadcast by some process pk. If pk is correct, then by property BEB1, all correct processes bebDeliver and then rebDeliver m. If pk crashes, then by the completeness property of P, pi detects the crash and bebBroadcasts m to all. Since pi is correct, then by property BEB1, all correct processes bebDeliver and then rebDeliver m.
Algorithm (urb)

**Implements:** uniformBroadcast (urb).

**Uses:**
- BestEffortBroadcast (beb).
- PerfectFailureDetector (P).

**upon event** < Init > do
- correct := S;
- delivered := forward := ∅;
Algorithm (urb – cont’d)

upon event < crash, pi > do
  correct := correct \ {pi};

upon event < urbBroadcast, m> do
  forward := forward U {[self,m]};
  trigger < bebBroadcast, [Data,self,m]>;
Algorithm (urb – cont’d)

upon event <bebDeliver, pi, [Data,pj,m]> do
  ack[m] := ack[m] U {pi};
  if [pj,m] ∉ forward then
    forward := forward U {[pj,m]};
  trigger < bebBroadcast,[Data,pj,m]>;
upon event (for any \([pj, m] \in \text{forward}\)
<correct \(\subseteq\) ack\([m]\)> and \(<m \notin \text{delivered}\>) do

\[\text{delivered := delivered U \{m\};}\]

\[\text{trigger < urbDeliver, pj, m>;}\]
Algorithm (urb)
Algorithm (urb)

p1

p2

p3

delivery

crash

m

m

m

m

m

m

suspicion


Algorithm (urb)

Proof (sketch)

**URB2. URB3:** follow from BEB2 and BEB3

**A simple lemma:** If a correct process pi bebDelivers a message m, then pi eventually urbDelivers m.

Any process that bebDelivers m bebBroadcasts m. By the completeness property of the failure detector and property BEB1, there is a time at which pi bebDelivers m from every correct process and hence urbDelivers m.
Algorithm (urb)

Proof (sketch)

URB1. Validity: If a correct process pi urbBroadcasts a message m, then pi eventually bebBroadcasts and bebDelivers m: by our lemma, pi urbDelivers m.

URB4. Agreement: Assume some process pi urbDelivers a message m. By the algorithm and the completeness and accuracy properties of the failure detector, every correct process bebDelivers m. By our lemma, every correct process will urbDeliver m.