Distributed Systems
Terminating Reliable Broadcast
Prof R. Guerraoui
Distributed Programming Laboratory
Terminating Reliable Broadcast
Terminating Reliable Broadcast

- Like reliable broadcast, terminating reliable broadcast (TRB) is a communication primitive used to disseminate a message among a set of processes in a reliable way.

- TRB is however strictly stronger than (uniform) reliable broadcast.
(Uniform) Reliable Broadcast

p1

broadcast(m)

p2

crash

deliver(m)

p3

deliver(m)
(Uniform) Reliable Broadcast

broadcast(m)

crash

??
Terminating Reliable Broadcast

p1

broadcast(m)

p2

p3

deliver(m)

crash

deliver(m)
Terminating Reliable Broadcast

$p_1$ 

broadcast($m$)

$p_2$ 

$crash$

$p_3$ 

deliver($\varphi$)

deliver($\varphi$)
Terminating Reliable Broadcast

- **Like** with reliable broadcast, correct processes in TRB agree on the set of messages they deliver.

- **Like** with (uniform) reliable broadcast, every correct process in TRB delivers every message delivered by any process.

- **Unlike** with reliable broadcast, every correct process delivers a message, even if the broadcaster crashes.
Terminating Reliable Broadcast

- The problem is defined for a specific broadcaster process \( p_i = \text{src} \) (known by all processes)
- Process src is supposed to broadcast a message \( m \) (distinct from \( \varphi \))
- The other processes need to deliver \( m \) if src is correct but may deliver \( \varphi \) if src crashes
Terminating Reliable Broadcast (pi)

**TRB1. Integrity:** If a process delivers a message \( m \), then either \( m \) is \( \varnothing \) or \( m \) was broadcast by \( src \)

**TRB2. Validity:** If the sender \( src \) is correct and broadcasts a message \( m \), then \( src \) eventually delivers \( m \)

**TRB3. (Uniform) Agreement:** For any message \( m \), if a correct (any) process delivers \( m \), then every correct process delivers \( m \)

**TRB4. Termination:** Every correct process eventually delivers exactly one message
Terminating Reliable Broadcast

**Events**

- Request: `<trbBroadcast, m>`
- Indication: `<trbDeliver, p, m>`

- **Properties:**
  - *TRB1, TRB2, TRB3, TRB4*
Algorithm (trb)

- **Implements**: trbBroadcast (trb).

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

- **upon event** `< Init >` **do**
  - `prop := ⊥;`
  - `correct := S;`
Algorithm (trb – cont’d)

- **upon event** \(< \text{trbBroadcast, m}> \) do
  - trigger \(< \text{bebBroadcast, m}> ;

- **upon event** \(< \text{crash, src}> \) and \((\text{prop} = \bot)\) do
  - prop := \(\varnothing\);
Algorithm (trb – cont’d)

upon event <bebDeliver, src, m> and (prop = ⊥) do
  prop := m;

• upon event (prop ≠ ⊥) do
  • trigger < Propose, prop >;

• upon event < Decide, decision> do
  • trigger < trbDeliver, src, decision>;
Algorithm (trb); src = p2

p1

broadcast(m)

p2

p3

crash

UCons(\varphi, \varphi - m)

deliver(\varphi - m)

UCons(m, \varphi - m)

deliver(\varphi - m)
Terminating Reliable Broadcast

- The TRB algorithm uses the perfect failure detector P (i.e., P is sufficient)
- Is P also necessary?
  - Is there an algorithm that implements TRB with a failure detector that is strictly weaker than P? (this would mean that P is not necessary)
  - Is there an algorithm that uses TRB to implement P (this would mean that P is necessary)
Terminating Reliable Broadcast

- We give an algorithm that implements $\mathbf{P}$ using TRB; more precisely, we assume that every process $\pi_i$ can use an infinite number of instances of TRB where $\pi_i$ is the sender src.

  1. Every process $\pi_i$ keeps on trbBroadcasting messages $m_i1$, $m_i2$, etc.
  2. If a process $\pi_k$ delivers $\varphi_i$, $\pi_k$ suspects $\pi_i$.

- NB. The algorithm uses (non-uniform) TRB.