LUND UNIVERSITY

Broadcast Algorithms

BJÖRN A. JOHNSSON

Overview

- Best-Effort Broadcast
- (Regular) Reliable Broadcast (2)
- Uniform Reliable Broadcast (2)

- Stubborn Broadcast
- Logged Best-Effort Broadcast
- Logged Uniform Reliable Broadcast

- Probabilistic Broadcast (2)

LUND UNIVERSITY

Abstracting Processes

- Crash-stop
- ~~Omissions~~
- Crash-recovery
- ~~Eavesdropping faults~~
- Arbitrary-fault (Byzantine)

- *Faulty* or *correct*

LUND UNIVERSITY

Timing Assumptions

- Asynchronous System
- Synchronous System
- Partial Synchrony

LUND UNIVERSITY

Distributed-System Models

- Fail-stop
  - crash-stop, perfekt links, perfect failure detector (*P*)
- Fail-noisy
  - crash-stop, perfekt links, eventually *P* ($\diamond P$)
- Fail-silent
  - crash-stop, perfekt links, no failure detector
- Fail-recovery
  - crash-recovery, stubborn links, eventual leader detector (Ω)
- Fail-arbitrary
  - fail-arbitrary, authenticated perfekt links

LUND UNIVERSITY

Motivation

- *Client-server* scheme – *point-to-point* communication
  - Useful when *reliable*, e.g. TCP
- Bigger systems usually more than 2 processes
  - *Broadcast* abstractions convenient
  - Send to all processes, in a single one-shot op.
- Reliability req. of p2p not directly transposable
  - “No message lost or duplicated”
  - Complex for broadcast…

LUND UNIVERSITY

Best-Effort Broadcast

- Burden of ensuring reliability on sender:
  - Receivers unconcerned with enforcing reliability
  - No delivery guarantees if sender fails

Best-Effort Broadcast

Module 3.1: Interface and properties of best-effort broadcast

Module:

Name: BestEffortBroadcast, **instance** *beb*.

Events:

Request: $\langle$ *beb*, *Broadcast* | m $\rangle$: Broadcasts a message m to all processes.

Indication: $\langle$ *beb*, *Deliver* | p, m $\rangle$: Delivers a message m broadcast by process p.

Properties:

BEB1: *Validity:* If a correct process broadcasts a message m, then every correct process eventually delivers m.

PP2PL

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

BEB3: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

LUND UNIVERSITY

Best-Effort Broadcast

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 $\rangle$;

LUND UNIVERSITY

Best-Effort Broadcast

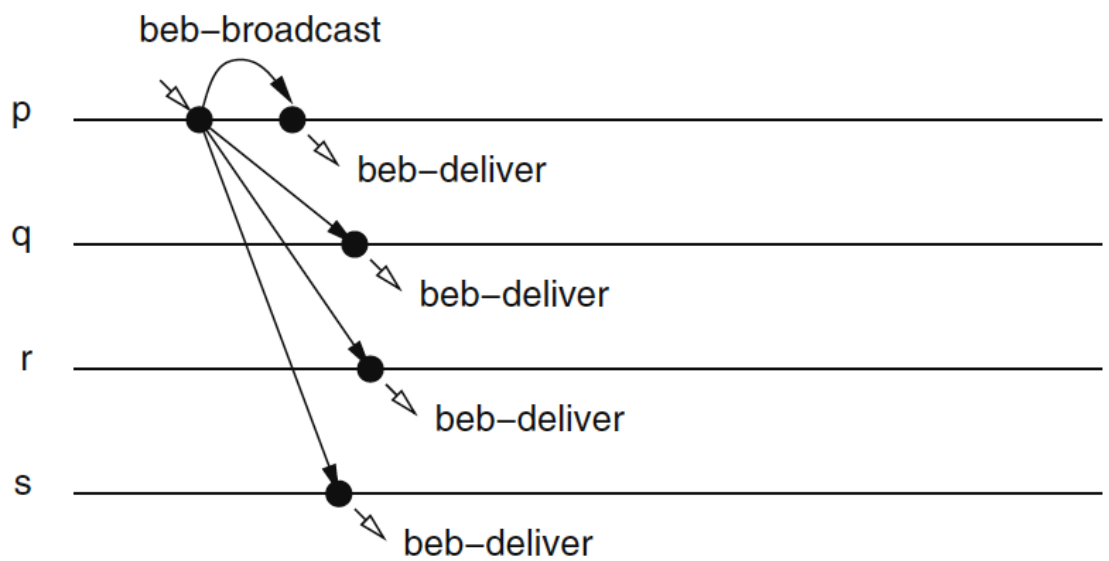


Figure 3.1: Sample execution of basic broadcast

LUND UNIVERSITY

(Regular) Reliable Broadcast

- Best-effort only ensures delivery if sender doesn't crash
  - Processes might not *agree* on message delivery
  - Even if all messages sent before sender crashes…
- (Regular) Reliable broadcast provides stronger notion of reliablity:
  - Ensures agreement even if sender fails.
  - Sender failure – no process delivers message

LUND UNIVERSITY

(Regular) Reliable Broadcast

Module 3.2: Interface and properties of (regular) reliable broadcast

Module:

Name: ReliableBroadcast, **instance** *rb*.

Events:

Request: $\langle$ *rb*, *Broadcast* | m $\rangle$: Broadcasts a message m to all processes.

Indication: $\langle$ *rb*, *Deliver* | p, m $\rangle$: Delivers a message m broadcast by process p.

Properties:

RB1: *Validity:* If a correct process p broadcasts a message m, then p eventually delivers m.

RB2: *No duplication:* No message is delivered more than once.

RB3: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

New! → **RB4:** *Agreement:* If a message m is delivered by some correct process, then m is eventually delivered by every correct process.

LUND UNIVERSITY

(Regular) Reliable Broadcast

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] $\rangle$;

array of sets

message descriptor

original source

LUND UNIVERSITY

(Regular) Reliable Broadcast

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] $\rangle$;

LUND UNIVERSITY

(Regular) Reliable Broadcast

Algorithm 3.3: Eager Reliable Broadcast

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

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

upon event $\langle$ *rb*, *Init* $\rangle$ **do**
delivered := $\emptyset$;

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 $m \notin$ *delivered* **then**
delivered := *delivered* $\cup \{m\}$;
trigger $\langle$ *rb*, *Deliver* | s, m $\rangle$;
trigger $\langle$ *beb*, *Broadcast* | [DATA, s, m] $\rangle$;

LUND UNIVERSITY

(Regular) Reliable Broadcast

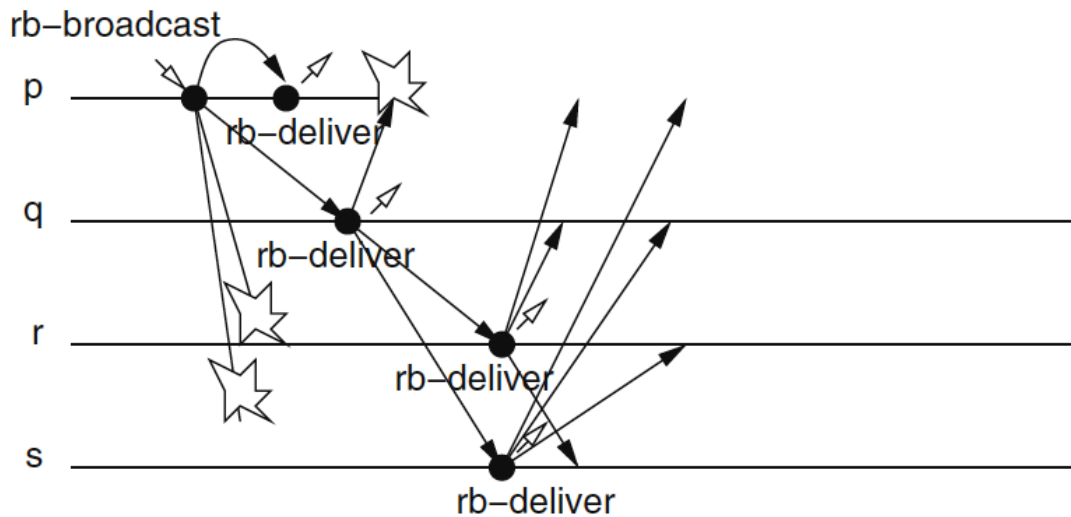


Figure 3.2: Sample execution of reliable broadcast with faulty sender

LUND UNIVERSITY

(Regular) Reliable Broadcast

- Problem: only requires the *correct* processes deliver the same set of messages

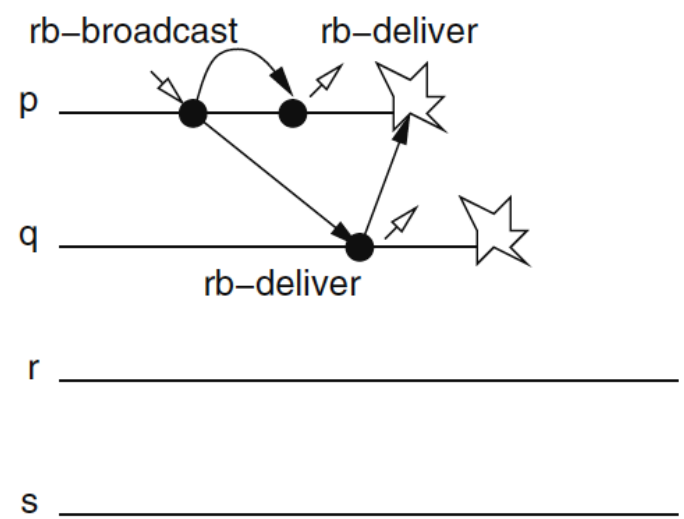


Figure 3.3: Nonuniform reliable broadcast

LUND UNIVERSITY

Uniform Reliable Broadcast

Module 3.3: Interface and properties of uniform reliable broadcast

Module:

Name: UniformReliableBroadcast, **instance** *urb*.

Events:

Request: $\langle$ *urb*, *Broadcast* | m $\rangle$: Broadcasts a message m to all processes.

Indication: $\langle$ *urb*, *Deliver* | p, m $\rangle$: Delivers a message m broadcast by process p.

Properties:

URB1–URB3: Same as properties RB1–RB3 in (regular) reliable broadcast (Module 3.2).

Different!

URB4: *Uniform agreement:* If a message m is delivered by some process (whether correct or faulty), then m is eventually delivered by every correct process.

LUND UNIVERSITY

Uniform Reliable Broadcast

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** $ack[m] := \emptyset$;

Infinite array for all possible message…

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

LUND UNIVERSITY

Uniform Reliable Broadcast

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 $\rangle$;

LUND UNIVERSITY

Uniform Reliable Broadcast

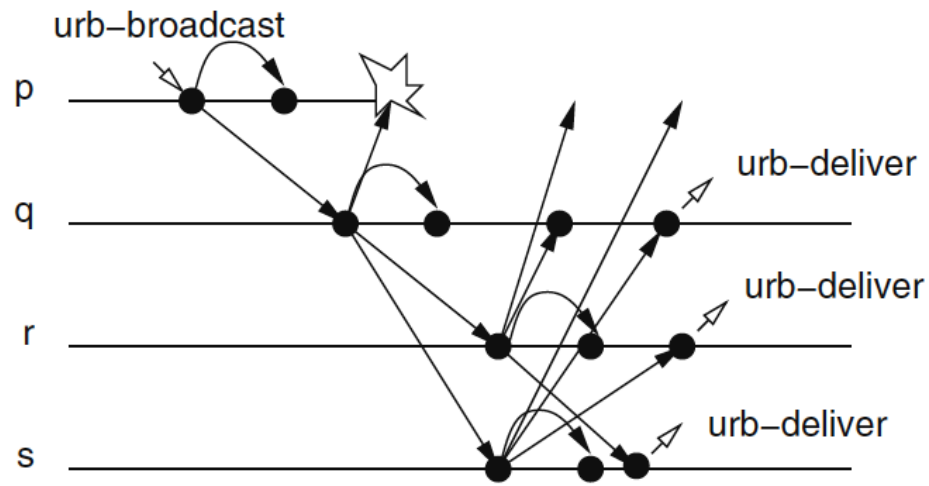


Figure 3.4: Sample execution of all-ack uniform reliable broadcast

LUND UNIVERSITY

Uniform Reliable Broadcast

- Requires $N > 2f$

Algorithm 3.5: Majority-Ack Uniform Reliable Broadcast

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

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

// Except for the function *candeliver*(·) below and for the absence of ⟨ *Crash* ⟩ events
// triggered by the perfect failure detector, it is the same as Algorithm 3.4.

function *candeliver*(m) **returns** Boolean **is**
return $\#(ack[m]) > N/2$;

Size of ack[m]

LUND UNIVERSITY

Stubborn Broadcast

Module 3.4: Interface and properties of stubborn best-effort broadcast

Module:

Name: StubbornBestEffortBroadcast, **instance** *sbeb*.

Events:

Request: $\langle$ *sbeb*, *Broadcast* | m $\rangle$: Broadcasts a message m to all processes.

Indication: $\langle$ *sbeb*, *Deliver* | p, m $\rangle$: Delivers a message m broadcast by process p.

Properties:

SBEB1: *Best-effort validity:* If a process that never crashes broadcasts a message m, then every correct process delivers m an infinite number of times.

SBEB2: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

No duplication gone!

LUND UNIVERSITY

Stubborn Broadcast

Algorithm 3.6: Basic Stubborn Broadcast

Implements:
StubbornBestEffortBroadcast, **instance** *sbeb*.

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

upon event $\langle$ *sbeb*, *Recovery* $\rangle$ **do**
// do nothing

upon event $\langle$ *sbeb*, *Broadcast* $\mid m \rangle$ **do**
forall $q \in \Pi$ **do**
trigger $\langle$ *sl*, *Send* $\mid q, m \rangle$;

upon event $\langle$ *sl*, *Deliver* $\mid p, m \rangle$ **do**
trigger $\langle$ *sbeb*, *Deliver* $\mid p, m \rangle$;

LUND UNIVERSITY

Logged Best-Effort Broadcast

- First for Fail-recovery model
- Strongest model, uniform reliable, not enough
- Difficulty: crashing, recovery and never crashing again is *correct*
- Solution: stable storage, as seen in “logged perfect links"

LUND UNIVERSITY

Logged Best-Effort Broadcast

Module 3.5: Interface and properties of logged best-effort broadcast

Module:

Name: LoggedBestEffortBroadcast, **instance** *lbeb*.

Events:

Request: $\langle$ *lbeb*, *Broadcast* | m $\rangle$: Broadcasts a message m to all processes.

Indication: $\langle$ *lbeb*, *Deliver* | *delivered* $\rangle$: Notifies the upper layer of potential updates to variable *delivered* in stable storage (which log-delivers messages according to the text).

Properties:

LBEB1: *Validity:* If a process that never crashes broadcasts a message m, then every correct process eventually log-delivers m.

LBEB2: *No duplication:* No message is log-delivered more than once.

LBEB3: *No creation:* If a process log-delivers a message m with sender s, then m was previously broadcast by process s.

Logged Best-Effort Broadcast

Algorithm 3.7: Logged Basic Broadcast

Implements:
LoggedBestEffortBroadcast, **instance** *lbeb*.

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

ensures *Validity*

upon event ⟨ *lbeb*, *Init* ⟩ **do**
delivered := ∅;
store(*delivered*);

upon event ⟨ *lbeb*, *Recovery* ⟩ **do**
retrieve(*delivered*);
trigger ⟨ *lbeb*, *Deliver* | *delivered* ⟩;

upon event ⟨ *lbeb*, *Broadcast* | m ⟩ **do**
forall $q \in \Pi$ **do**
trigger ⟨ *sl*, *Send* | q, m ⟩;

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

Logged Uniform Reliable Broadcast

Module 3.6: Interface and properties of logged uniform reliable broadcast

Module:

Name: LoggedUniformReliableBroadcast, **instance** *lurb*.

Events:

Request: $\langle$ *lurb*, *Broadcast* | m $\rangle$: Broadcasts a message m to all processes.

Indication: $\langle$ *lurb*, *Deliver* | *delivered* $\rangle$: Notifies the upper layer of potential updates to variable *delivered* in stable storage (which log-delivers messages according to the text).

Properties:

LURB1–LURB3: Same as properties LBEB1–LBEB3 in logged best-effort broadcast (Module 3.5).

New!

LURB4: *Uniform agreement:* If a message m is log-delivered by some process (whether correct or faulty), then m is eventually log-delivered by every correct process.

LUND UNIVERSITY

Logged Uniform Reliable Broadcast

Algorithm 3.8: Logged Majority-Ack Uniform Reliable Broadcast

Implements:
LoggedUniformReliableBroadcast, **instance** *lurb*.

Uses:
StubbornBestEffortBroadcast, **instance** *sbeb*.

upon event ⟨ *lurb*, *Init* ⟩ **do**
  delivered := ∅;
  pending := ∅;
  forall m **do** *ack*[m] := ∅;
  store(*pending*, *delivered*);

upon event ⟨ *lurb*, *Recovery* ⟩ **do**
  retrieve(*pending*, *delivered*);
  trigger ⟨ *lurb*, *Deliver* | *delivered* ⟩;
  forall $(s, m) \in$ *pending* **do**
    trigger ⟨ *sbeb*, *Broadcast* | [DATA, s, m] ⟩;

LUND UNIVERSITY

Logged Uniform Reliable Broadcast

```
upon event ⟨ lurb, Broadcast | m ⟩ do
    pending := pending ∪ {(self, m)};
    store(pending);
    trigger ⟨ sbeb, Broadcast | [DATA, self, m] ⟩;

upon event ⟨ sbeb, Deliver | p, [DATA, s, m] ⟩ do
    if (s, m) ∉ pending then
        pending := pending ∪ {(s, m)};
        store(pending);
        trigger ⟨ sbeb, Broadcast | [DATA, s, m] ⟩;
    if p ∉ ack[m] then
        ack[m] := ack[m] ∪ {p};
        if #(ack[m]) > N/2 ∧ (s, m) ∉ delivered then
            delivered := delivered ∪ {(s, m)};
            store(delivered);
            trigger ⟨ lurb, Deliver | delivered ⟩;
```

LUND UNIVERSITY

Probabilistic Broadcast

- No deterministic broadcast guarantees
- Offers “cost” reduction at the price of lower reliability

a. Reliability not scalable – *ack implosion* problem

b. Possible solution – requires configuration

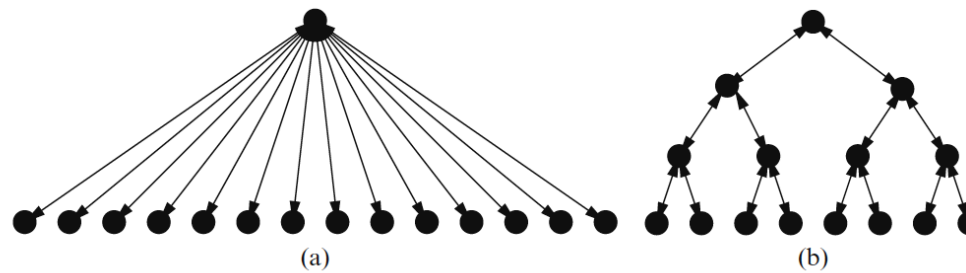


Figure 3.5: Direct vs. hierarchical communication for sending messages and receiving acknowledgments

- *Epidemic dissemination* – rumor spreading, gossiping

LUND UNIVERSITY

Probabilistic Broadcast

Module 3.7: Interface and properties of probabilistic broadcast

Module:

Name: ProbabilisticBroadcast, **instance** *pb*.

Events:

Request: $\langle$ *pb*, *Broadcast* | m $\rangle$: Broadcasts a message m to all processes.

Indication: $\langle$ *pb*, *Deliver* | p, m $\rangle$: Delivers a message m broadcast by process p.

Properties:

Weaker than *validity*

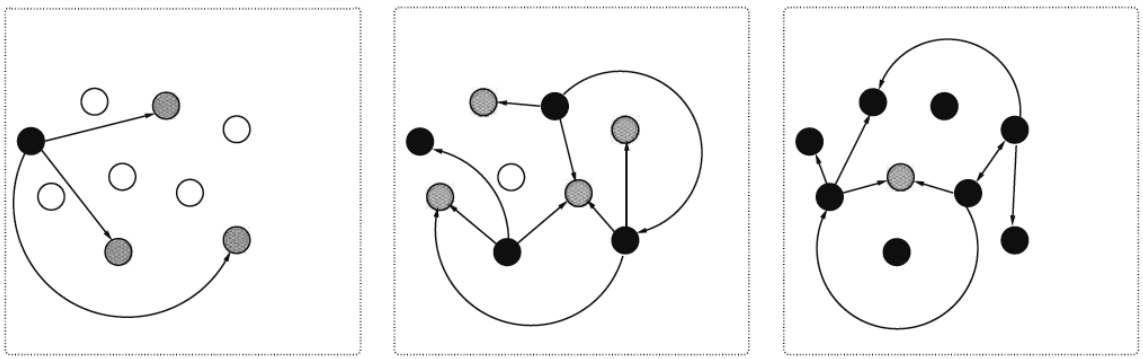
PB1: *Probabilistic validity:* There is a positive value ε such that when a correct process broadcasts a message m, the probability that every correct process eventually delivers m is at least $1 - \varepsilon$.

PB2: *No duplication:* No message is delivered more than once.

PB3: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

Eager Probabilistic Broadcast

- Sends to *k* random processes – the *fanout*
- A *round of gossiping* = receiving and resending message
- *R* rounds of gossiping per message
- *R* and *k* determine efficiency of algorithm



(a) round 1 (b) round 2 (c) round 3

Figure 3.6: Epidemic dissemination or gossip (with fanout 3)

LUND UNIVERSITY

Eager Probabilistic Broadcast

Algorithm 3.9: Eager Probabilistic Broadcast

Implements:
ProbabilisticBroadcast, **instance** *pb*.

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

upon event $\langle$ *pb*, *Init* $\rangle$ **do**
delivered := $\emptyset$;

Returns *k* processes from *Π \ {self}*

procedure *gossip*(*msg*) **is**
forall $t \in$ *picktargets*(k) **do trigger** $\langle$ *fll*, *Send* | t, *msg* $\rangle$;

upon event $\langle$ *pb*, *Broadcast* | m $\rangle$ **do**
delivered := *delivered* $\cup \{m\}$;
trigger $\langle$ *pb*, *Deliver* | *self*, m $\rangle$;
gossip([GOSSIP, *self*, m, R]);

upon event $\langle$ *fll*, *Deliver* | p, [GOSSIP, s, m, r] $\rangle$ **do**
if $m \notin$ *delivered* **then**
delivered := *delivered* $\cup \{m\}$;
trigger $\langle$ *pb*, *Deliver* | s, m $\rangle$;
if $r > 1$ **then** *gossip*([GOSSIP, $s, m, r-1$]);

UND
IVERSITY

Lazy Probabilistic Broadcast

- EPB too eager; consumes resources and causes redundant transmissions

1. Gossip until e.g. N/2 processes infected (push-phase)
2. Missed processes ask for message (pull-phase)

LUND UNIVERSITY

Lazy Probabilistic Broadcast

Algorithm 3.10: Lazy Probabilistic Broadcast (part 1, data dissemination)

Implements:
ProbabilisticBroadcast, **instance** *pb*.

Uses:
FairLossPointToPointLinks, **instance** *fll*;
ProbabilisticBroadcast, **instance** *upb*. // an *unreliable* implementation

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

procedure *gossip*(*msg*) **is**
forall $t \in$ *picktargets*(k) **do trigger** $\langle$ *fll*, *Send* | t, *msg* $\rangle$;

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

Lazy Probabilistic Broadcast

upon event $\langle$ *upb*, *Deliver* | p, [DATA, s, m, sn] $\rangle$ **do**
  if *random*([0, 1]) $> \alpha$ **then**
    stored := *stored* $\cup$ {[DATA, s, m, sn]};
  if $sn =$ *next*[s] **then**
    next[s] := *next*[s] + 1;
    trigger $\langle$ *pb*, *Deliver* | s, m $\rangle$;
  else if $sn >$ *next*[s] **then**
    pending := *pending* $\cup$ {[DATA, s, m, sn]};
    forall *missing* $\in$ [*next*[s], ..., $sn - 1$] **do**
      if no m' exists such that [DATA, s, m', *missing*] $\in$ *pending* **then**
        gossip([REQUEST, *self*, s, *missing*, $R - 1$]);
    starttimer(Δ, s, sn);

LUND UNIVERSITY

Lazy Probabilistic Broadcast

upon event $\langle$ *fll*, *Deliver* | p, [REQUEST, q, s, sn, r] $\rangle$ **do**
  if exists m such that [DATA, s, m, sn] $\in$ *stored* **then**
    trigger $\langle$ *fll*, *Send* | q, [DATA, s, m, sn] $\rangle$;
  else if $r > 0$ **then**
    gossip([REQUEST, $q, s, sn, r - 1$]);

upon event $\langle$ *fll*, *Deliver* | p, [DATA, s, m, sn] $\rangle$ **do**
  pending := *pending* $\cup$ {[DATA, s, m, sn]};

upon exists [DATA, s, x, sn] $\in$ *pending* such that $sn = next[s]$ **do**
  $next[s] := next[s] + 1$;
  pending := *pending* $\setminus$ {[DATA, s, x, sn]};
  trigger $\langle$ *pb*, *Deliver* | s, x $\rangle$;

upon event $\langle$ *Timeout* | s, sn $\rangle$ **do**
  if $sn > next[s]$ **then**
    $next[s] := sn + 1$;

LUND
UNIVERSITY