

ID2203 Tutorial 3

Cosmin Arad
icarad@kth.se

Handout: February 13, 2008
Due: February 20, 2008

1 Introduction

The goal of this tutorial is to understand and get accustomed to implementing various flavors of broadcast abstractions for the synchronous and asynchronous system models: *best-effort* broadcast (BEB), *regular reliable* broadcast (RB), *uniform reliable* broadcast (URB), and *probabilistic* broadcast (PB). The best-effort broadcast offers weak reliability guarantees and probabilistic broadcast abstraction does not offer deterministic reliability guarantees, but only with high probability. These abstractions assume an asynchronous system model. The regular and uniform reliable broadcast abstractions offer deterministic strong reliability guarantees but assume a synchronous model. They rely on a perfect failure detector.

2 Assignment

Using the TBN framework, you will implement the broadcast abstractions described in Algorithm 1 (BEB), Algorithm 2 (UN), Algorithm 3 (RB), Algorithm 4 (URB), and Algorithms 5, 6 and 7 (PB), as components.

You will use the same type of input as in Assignments 1 and 2, to specify your network of processes, i.e., a `topology.xml` file. You will reuse the communication component, the timer component, the delay component, the drop component, and the pfd component. You have to implement a `BestEffortBroadcastComponent` (BEB), an `UnreliableBroadcastComponent` (UN), a `ReliableBroadcastComponent` (RB), an `UniformReliableBroadcastComponent` (URB), and a `ProbabilisticBroadcastComponent` (PB). You have to implement the events that are accepted and triggered by these components: `BebBroadcastEvent`, `BebDeliverEvent`, `UnBroadcastEvent`, `UnDeliverEvent`, `RbBroadcastEvent`, `RbDeliverEvent`, `UrbBroadcastEvent`, `UrbDeliverEvent`, `PbBroadcastEvent`, and `PbDeliverEvent`, respectively, and the

messages that they exchange. The PB component used the timer component, so you will need a `TimeoutEvent` that extends the `TimerExpiredEvent` for this.

You will have a separate architecture for testing RB, URB, and PB, i.e. an `assignment3-rb.xml`, an `assignment3-urb.xml`, and an `assignment3-pb.xml`. When launching the `Assignment3.java` you pass a command line argument to specify which architecture to use:

```
java Assignment3 <topology.xml> <nodeId> rb|urb|pb.
```

You will slightly change the application component from the second assignment, to handle the `RbDeliverEvent`, the `UrbDeliverEvent`, and the `bDeliverEvent`. You use the same `InitEvent` to pass to the broadcast components the set of all processes (Π). Note that the algorithms assume that each process knows and can communicate with all other processes in Π . This corresponds to a fully connected topology.

These algorithms relying on the PFD, assume that all processes are started within γ^1 milliseconds. To start all your processes at once, you can use the following commands on Windows (cmd) and Linux (bash) respectively:

```
FOR /L %G IN (0, 1, 5) DO start java Assignment3 topology.xml %G ...
for ((i=0;i<=5;i++)) \
    do java Assignment3 topology.xml $i ... > p$i.log 2>&1 & done
```

Replace 5 with your number of processes minus one. For Linux, you can inspect the output of your processes in the `p*.log` files.

Implement the BEB, UN, RB, URB, and PB components and experiment with them as instructed in the following exercises. Describe your experiments in a written report. For each exercise include the topology descriptors used, and explain the behavior that you observe.

You need to read Section 3.8 of the textbook, Randomized Broadcast, in order to understand how the probabilistic broadcast algorithm works. The probabilistic algorithm given here (Algorithms 5, 6, and 7) is parameterized by: `TimeDelay`, `StoreThreshold`, `Fanout`, and `MaxRounds` parameters. Specify these parameters in a configuration file, `lpb.properties`, that you use to initialize the `ProbabilisticBroadcastComponent`. Here is an example `lpb.properties` file:

```
time.delay = 4000          # 4 seconds
store.threshold = 0.7      # 70% of nodes store messages
gossip.fanout = 3          # spread gossip to 3 other nodes
```

¹the interval between sending two consecutive heartbeats in PFD.

```

gossip.maxrounds = 3      # forward gossip maximum 3 times
pick-targets.seed = 0     # random seed for the pick-targets RNG
store-threshold.seed = 0 # random seed for the store-thres RNG

```

The assignment is due on February 20th. You have to send your source code and written report by email before the next tutorial session. During the tutorial session you will present the assignment on a given topology description. You can work in groups of maximum 2 students. Be prepared to answer questions about your process's system architecture and explain the behavior of the algorithms. Any questions are welcome on the mailing list.

Exercise 1 Modify Algorithm 3 such that it garbage collects the *delivered* set. Messages that no longer need to be maintained in the delivered set should be removed. Update your implementation of RB and describe the new algorithm in the report.

Exercise 2 Modify Algorithm 4 such that it only keeps track of the last message sent from each process, in the *delivered* variable, and it garbage collects the *pending* variable. Update your implementation of URB and describe the new algorithm in the report.

Exercise 3 The Lazy Probabilistic Broadcast (Algorithms 5, 6 and 7) presented here is different from Algorithm 3.10-3.11 from the textbook (pages 92-93). Analyze the differences between the two algorithms and discuss their implications in the written report.

Exercise 4 Experiment with the Lazy Probabilistic Broadcast by varying the loss rate of the links, the *fanout*, *threshold*, and *maxrounds* parameters of the gossip. Describe in your report how these parameters influence the reliability of the broadcast. Describe one execution where no message is lost, one execution where a message is lost by the unreliable broadcast but recovered by gossip, and one execution where a message is lost and not recovered by gossip, therefore permanently lost.

Algorithm 1 Basic Broadcast

Implements:

BestEffortBroadcast (beb).

Uses:

PerfectPointToPointLinks (pp2p).

```
1: upon event  $\langle \text{bebBroadcast} \mid m \rangle$  do
2:   for all  $p_i \in \Pi$  do
3:     trigger  $\langle \text{pp2pSend} \mid p_i, m \rangle$ ;
4:   end for
5: end event

6: upon event  $\langle \text{pp2pDeliver} \mid p_i, m \rangle$  do
7:   trigger  $\langle \text{bebDeliver} \mid p_i, m \rangle$ ;
8: end event
```

Algorithm 2 Unreliable Broadcast

Implements:

UnreliableBroadcast (un).

Uses:

FairLossPointToPointLinks (flp2p).

```
1: upon event  $\langle \text{unBroadcast} \mid m \rangle$  do
2:   for all  $p_i \in \Pi$  do
3:     trigger  $\langle \text{flp2pSend} \mid p_i, m \rangle$ ;
4:   end for
5: end event

6: upon event  $\langle \text{flp2pDeliver} \mid p_i, m \rangle$  do
7:   trigger  $\langle \text{unDeliver} \mid p_i, m \rangle$ ;
8: end event
```

Algorithm 3 Lazy Reliable Broadcast

Implements:

ReliableBroadcast (rb).

Uses:

BestEffortBroadcast (beb);

PerfectFailureDetector (\mathcal{P}).

```
1: upon event  $\langle Init \rangle$  do
2:   delivered :=  $\emptyset$ ;
3:   correct :=  $\Pi$ ;
4:   for all  $p_i \in \Pi$  do
5:     from[ $p_i$ ] :=  $\emptyset$ ;
6:   end for
7: end event

8: upon event  $\langle rbBroadcast \mid m \rangle$  do
9:   trigger  $\langle bebBroadcast \mid [DATA, self, m] \rangle$ ;
10: end event

11: upon event  $\langle bebDeliver \mid p_i, [DATA, s_m, m] \rangle$  do
12:   if ( $m \notin delivered$ ) then
13:     delivered := delivered  $\cup \{m\}$ ;
14:     trigger  $\langle rbDeliver \mid s_m, m \rangle$ ;
15:     from[ $p_i$ ] := from[ $p_i$ ]  $\cup \{(s_m, m)\}$ ;
16:     if ( $p_i \notin correct$ ) then
17:       trigger  $\langle bebBroadcast \mid [DATA, s_m, m] \rangle$ ;
18:     end if
19:   end if
20: end event

21: upon event  $\langle crash \mid p_i \rangle$  do
22:   correct := correct  $\setminus \{p_i\}$ ;
23:   for all  $(s_m, m) \in from[p_i]$  do
24:     trigger  $\langle bebBroadcast \mid [DATA, s_m, m] \rangle$ ;
25:   end for
26: end event
```

Algorithm 4 All-Ack Uniform Reliable Broadcast

Implements:

UniformReliableBroadcast (urb).

Uses:

BestEffortBroadcast (beb);

PerfectFailureDetector (\mathcal{P}).

```
1: function canDeliver( $m$ ) returns boolean is
2:   return (correct  $\subseteq$  ack $_m$ );
3: end function

4: upon event  $\langle \text{Init} \rangle$  do
5:   delivered := pending :=  $\emptyset$ ;
6:   correct :=  $\Pi$ ;
7:   for all  $m$  do
8:     ack $_m$  :=  $\emptyset$ ;
9:   end for
10: end event

11: upon event  $\langle \text{urbBroadcast} \mid m \rangle$  do
12:   pending := pending  $\cup$   $\{(\text{self}, m)\}$ ;
13:   trigger  $\langle \text{bebBroadcast} \mid [\text{DATA}, \text{self}, m] \rangle$ ;
14: end event

15: upon event  $\langle \text{bebDeliver} \mid p_i, [\text{DATA}, s_m, m] \rangle$  do
16:   ack $_m$  := ack $_m \cup \{p_i\}$ ;
17:   if  $((s_m, m) \notin \text{pending})$  then
18:     pending := pending  $\cup \{(s_m, m)\}$ ;
19:     trigger  $\langle \text{bebBroadcast} \mid [\text{DATA}, s_m, m] \rangle$ ;
20:   end if
21: end event

22: upon event  $\langle \text{crash} \mid p_i \rangle$  do
23:   correct := correct  $\setminus \{p_i\}$ ;
24: end event

25: upon exists  $(s_m, m) \in \text{pending}$  such that canDeliver( $m$ )  $\wedge m \notin$  de-
    livered do
26:   delivered := delivered  $\cup \{m\}$ ;
27:   trigger  $\langle \text{urbDeliver} \mid s_m, m \rangle$ ;
28: end
```

Algorithm 5 Lazy Probabilistic Broadcast (part 1)

Implements:

ProbabilisticBroadcast (pb).

Uses:

FairLossPointToPointLinks (flp2p);

UnreliableBroadcast (un).

```
1: upon event  $\langle Init \rangle$  do
2:   for all  $p_i \in \Pi$  do
3:     delivered[ $p_i$ ] := 0;
4:   end for
5:   lsn := 0; pending := stored :=  $\emptyset$ ;
6: end event

7: function pick-targets(count) returns set of processes is
8:   targets :=  $\emptyset$ ;
9:   while |targets| < count do
10:    targets := targets  $\cup$  {random( $\Pi \setminus$  targets  $\setminus$  {self})};
11:   end while
12:   return targets;
13: end function

14: procedure deliver-pending( $s$ ) is
15:   while exists [DATA,  $s$ ,  $x$ ,  $sn_x$ ]  $\in$  pending such that
16:      $sn_x = \text{delivered}[s] + 1$  do
17:       delivered[ $s$ ] := delivered[ $s$ ] + 1;
18:       pending := pending  $\setminus$  {[DATA,  $s$ ,  $x$ ,  $sn_x$ ]};
19:       if ( $x \neq \text{NIL}$ )
20:         trigger  $\langle pbDeliver \mid s, x \rangle$ ;
21:       end if
22:   end while
23: end procedure

24: procedure gossip( $msg$ ) is
25:   for all  $t \in \text{pick-targets}(\text{Fanout})$  do
26:     trigger  $\langle flp2pSend \mid t, msg \rangle$ ;
27:   end for
28: end procedure
```

Algorithm 6 Lazy Probabilistic Broadcast (part 2)

```
1: upon event  $\langle pbBroadcast \mid m \rangle$  do
2:    $lsn := lsn + 1$ ;
3:   trigger  $\langle unBroadcast \mid [DATA, self, m, lsn] \rangle$ ;
4: end event

5: upon event  $\langle unDeliver \mid p_i, [DATA, s_m, m, sn_m] \rangle$  do
6:   if ( $random() > StoreThreshold$ ) then
7:      $stored := stored \cup \{[DATA, s_m, m, sn_m]\}$ ;
8:   end if
9:   if ( $sn_m = delivered[s_m] + 1$ ) then
10:     $delivered[s_m] := delivered[s_m] + 1$ ;
11:    trigger  $\langle pbDeliver \mid s_m, m \rangle$ ;
12:  else if ( $sn_m > delivered[s_m] + 1$ ) then
13:     $pending := pending \cup \{[DATA, s_m, m, sn_m]\}$ ;
14:    for all  $seqnb \in [delivered[s_m] + 1, sn_m - 1]$  do
15:       $gossip([REQUEST, self, s_m, seqnb, MaxRounds - 1])$ ;
16:       $startTimer(TimeDelay, p_i, seqnb)$ ;
17:    end for
18:  end if
19: end event

20: upon event  $\langle flp2pDeliver \mid p_j, [REQUEST, p_i, s_m, sn_m, r] \rangle$  do
21:   if ( $[DATA, s_m, m, sn_m] \in stored$ ) then
22:     trigger  $\langle flp2pSend \mid p_i, [DATA, s_m, m, sn_m]$ 
23:   else if ( $r > 0$ ) then
24:      $gossip([REQUEST, p_i, s_m, sn_m, r - 1])$ ;
25:   end if
26: end event

27: upon event  $\langle flp2pDeliver \mid p_j, [DATA, s_m, m, sn_m] \rangle$  do
28:   if ( $sn_m = delivered[s_m] + 1$ ) then
29:      $delivered[s_m] := delivered[s_m] + 1$ ;
30:     trigger  $\langle pbDeliver \mid s_m, m \rangle$ ;
31:      $deliver-pending(s_m)$ ;
32:   else
33:      $pending := pending \cup \{[DATA, s_m, m, sn_m]\}$ ;
34:      $pending := pending \setminus \{[DATA, s_m, NIL, sn_m]\}$ ;
35:   end if
36: end event
```

Algorithm 7 Lazy Probabilistic Broadcast (part 3)

```
1: upon event  $\langle \textit{Timeout} \mid s, sn \rangle$  do  
2:   if ( $sn = \text{delivered}[s]+1$ ) then  
3:      $\text{delivered}[s] := \text{delivered}[s]+1$ ;  
4:      $\text{deliver-pending}(s)$ ;  
5:   else if ( $sn > \text{delivered}[s]+1 \wedge \forall x \neq \text{NIL} [\text{DATA}, s, x, sn] \notin \text{pending}$ )  
   then  
6:      $\text{pending} := \text{pending} \cup \{[\text{DATA}, s, \text{NIL}, sn]\}$ ;  
7:   end if  
8: end event
```
