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 Algorithm 5 (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 UniformReliableBroadcast-Component (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.

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

[1] the interval between sending two consecutive heartbeats in PFD.

should be removed. Update the 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 *pending* and *delivered* variables. Update the implementation of URB and describe the new algorithm in the report.

Exercise 3 The Lazy Probabilistic Broadcast (Algorithm 5) 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* 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).

```
upon event ⟨ bebBroadcast | m ⟩ do
    for all p_i ∈ Π do
        trigger ⟨pp2pSend | p_i, m ⟩;
    end for
end event

upon event ⟨ pp2pDeliver | p_i, m ⟩ do
    trigger ⟨bebDeliver | p_i, m ⟩;
end event
```

Algorithm 2 Unreliable Broadcast

Implements:

UnreliableBroadcast (un).

Uses:

FairLossPointToPointLinks (flp2p).

```
upon event ⟨ unBroadcast | m ⟩ do
    for all p_i ∈ Π do
        trigger ⟨flp2pSend | p_i, m ⟩;
    end for
end event

upon event ⟨ flp2pDeliver | p_i, m ⟩ do
    trigger ⟨unDeliver | p_i, m ⟩;
end event
```

Algorithm 3 Lazy Reliable Broadcast

Implements:

ReliableBroadcast (rb).

Uses:

BestEffortBroadcast (beb);

PerfectFailureDetector ($\mathcal{P}$).

```
upon event ⟨ Init ⟩ do
    delivered := ∅;
    correct := Π;
    for all p_i ∈ Π do
        from[p_i] := ∅;
    end for
end event

upon event ⟨ rbBroadcast | m ⟩ do
    trigger ⟨bebBroadcast | [DATA, self, m] ⟩;
end event

upon event ⟨ bebDeliver | p_i, [DATA, s_m, m] ⟩ do
    if (m ∉ delivered) then
        delivered := delivered ∪ {m};
        trigger ⟨rbDeliver | s_m, m ⟩;
        from[p_i] := from[p_i] ∪ {(s_m, m)};
        if (p_i ∉ correct) then
            trigger ⟨bebBroadcast | [DATA, s_m, m] ⟩;
        end if
    end if
end event

upon event ⟨ crash | p_i ⟩ do
    correct := correct \ {p_i};
    for all (s_m, m) ∈ from[p_i] do
        trigger ⟨bebBroadcast | [DATA, s_m, m] ⟩;
    end for
end event
```

Algorithm 4 All-Ack Uniform Reliable Broadcast

Implements:

UniformReliableBroadcast (urb).

Uses:

BestEffortBroadcast (beb);
PerfectFailureDetector ($\mathcal{P}$).

```
function canDeliver(m) returns boolean is
    return (correct ⊆ ack_m);
end function

upon event ⟨ Init ⟩ do
    delivered := pending := ∅;
    correct := Π;
    for all m do
        ack_m := ∅;
    end for
end event

upon event ⟨ urbBroadcast | m ⟩ do
    pending := pending ∪ {(self, m)};
    trigger ⟨bebBroadcast | [DATA, self, m] ⟩;
end event

upon event ⟨ bebDeliver | p_i, [DATA, s_m, m] ⟩ do
    ack_m := ack_m ∪ {p_i};
    if ((s_m, m) ∉ pending) then
        pending := pending ∪ {(s_m, m)};
        trigger ⟨bebBroadcast | [DATA, s_m, m] ⟩;
    end if
end event

upon event ⟨ crash | p_i ⟩ do
    correct := correct \ {p_i};
end event

upon exists (s_m, m) ∈ pending such that canDeliver(m) ∧ m ∉ delivered do
    delivered := delivered ∪ {m};
    trigger ⟨urbDeliver | s_m, m ⟩;
end
```

Algorithm 5 Lazy Probabilistic Broadcast

Implements:

ProbabilisticBroadcast (pb).

Uses:

FairLossPointToPointLinks (flp2p);
UnreliableBroadcast (un).

```
upon event ⟨ pbBroadcast | m ⟩ do
    lsn := lsn + 1;
    trigger ⟨unBroadcast | [DATA, self, m, lsn] ⟩;
end event
```