ID2203 Tutorial 2

Cosmin Arad
`icarad@kth.se`

Handout: February 6, 2008
Due: February 13, 2008

1 Introduction

The goal of this tutorial is to understand and get accustomed to implementing failure detectors for the synchronous and partially synchronous system models.

On pages 5 and 6 you have the algorithms for two failure detectors. Algorithm 1 is a correct perfect failure detector proposed by Ghodsi and Haridi. Algorithm 2 is the correct eventually perfect failure detector algorithm from the textbook (page 54). Both algorithms use perfect point to point links.

2 Assignment

Using the TBN framework, you will implement the two failure detectors described in Algorithm 1 and Algorithm 2.

You will use the same type of input as in Assignment 1, to specify your network of processe, i.e., a topology.xml file. You will reuse the communication component, the timer component, and the delay component. You have to implement a PerfectFailureDetectorComponent (PFD) and an EventuallyPerfectFailureDetectorComponent (EPFD).

The PFD component implements Algorithm 1. Periodically, it sends heartbeat messages and expects to receive heartbeat messages from other processes. If it detects that a process has crashed, the PFD component triggers a CrashEvent. Thus, You need to implement a HeartbeatMessage and a CrashEvent.

The EPFD component implements Algorithm 2. Periodically, it sends heartbeat messages and expects to receive heartbeat messages from other processes. If it suspects that a process has crashed, the EPFD component triggers a SuspectEvent. If it detects that some process that is suspected is in fact still alive, the EPFD component triggers a RestoreEvent. Thus, You need to implement a SuspectEvent and a RestoreEvent.

The PFD and EPFD components use the timer component to schedule their periodical checks and sending of heartbeat messages. According to the algorithms, you need to implement three events extending the Timer-ExpiredEvent: a HeartbeatTimeoutEvent and a CheckTimeoutEvent, to be handled by the PFD component, and a TimeoutEvent to be handled by the EPFD component.

You will slightly change the application component from the first assignment, to handle the CrashEvent, the SuspectEvent, and the RestoreEvent. You use the same InitEvent to pass to the PFD and EPFD 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.

You will have a separate architecture for each one of the two failure detectors, i.e. an assignment2-pfd.xml and an assignment2-epfd.xml. When launching the Assignment2.java you pass as a command line argument to specify which failure detector to use, so each failure detector is tested separately: `java Assignment2 <topology.xml> <nodeId> pfd|epfd`.

Both algorithms use some configuration parameters: γ and δ for PFD, $TimeDelay$ (the initial value of period) and Δ for EPFD. All of these represent time periods and are given in milliseconds. They are specified in the configuration files of the components. For the PFD component you'll have a `pfd.properties` file looking like this:

```
gamma = 2000
delta = 4000
```

In your `assignment2-pfd.xml` you specify the configuration file like this:

```
<tns:component>
    <tns:factoryName>PFDComponentFactory</tns:factoryName>
    <tns:componentName>PFDComponent</tns:componentName>
    <tns:initFile>pfd.properties</tns:initFile>
</tns:component>
```

For the EPFD component you'll have a `epfd.properties` file looking like this:

```
period = 1000     # TimeDelay
increment = 1000  # Delta
```

In your `assignment2-epfd.xml` you specify the configuration file like this:

```
<tns:component>
    <tns:factoryName>EPFDComponentFactory</tns:factoryName>
    <tns:componentName>EPFDComponent</tns:componentName>
    <tns:initFile>epfd.properties</tns:initFile>
</tns:component>
```

You have to implement `init` methods for the PFD and EPFD components, that read these properties and initialize local variables. This is similar to the initialization of the ApplicationComponent.

Beware that the δ in PFD has to be larger than every link delay specified in your topology file, as it represents an upper bound on the transmission delay.

The values of the properties given as example mean that PFD shall send heartbeats every 1 second ($\gamma = 1000$) and check received heartbeats every 5 seconds ($\gamma + \delta = 5000$). The upper bound on the transmission delay is $\delta = 4000$. Initially, EPFD will send heartbeats and check received heartbeats every 1 second ($period = TimeDelay = 1000$). Suppose you have a bidirectional link between proces 0 and process 1 with delay 2000. As process 0 does not receive a heartbeat from process 1 earlier than 2 seconds, 0 will suspect 1. Later 0 revises its suspicion and increases its $period$ to 2 (increments it by 1, as $\Delta = 1$), and so on.

The PFD component triggers events of type CrashEvent, specifying the NodeReferece of the crashed process. The EPFD component triggers events of type SuspectEvent and RestoreEvent, specifying the NodeReferece of the process that is suspected or for which the suspicion is revised, respectivelly, and the value of the variable $period$ in Algorithm 2. These events are handled by the application component which outputs them to the user. So the $period$ value is output with each suspect or restore event.

These two algorithms assume that all processes are started within γ 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 Assignment2 topology.xml %G ...
for ((i=0;i<=5;i++)) \
   do java Assignment2 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 PFD and EPFD 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 13th. 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 PFD and EPFD algorithms. Any questions are welcome on the mailing list.

Exercise 1   Verify the completeness of the failure detectors by killing one or more processes and wait for crash/suspect events to be triggered in the remaining processes.

Exercise 2   For EPFD, initialize $TimeDelay$ to a value smaller than you link delays, and observe how it is adjusted to accomodate larger transmission delays, like in the example above. Explain the behaviour of your processes (the execution) step by step.

Exercise 3   Remember that these algorithms are for the crash-stop process failure model. You can try to kill a process and restart it after the EPFD component has triggered a suspicion event. Observe how a restore event is triggered and the fact that other processes believe that the crashed and recovered process did not actually crash. Therefore, sometimes, in a crash-recovery model, one needs to distinguish between different incarnations of the same process.

Algorithm 1 Perfect Failure Detector

Implements: PerfectFailureDetector ($\mathcal{P}$).

Uses: PerfectPointToPointLinks (pp2p).

1: **upon event** $\langle\ Init\ \rangle$ **do**
2:     alive := Π;
3:     detected := $\emptyset$;
4:     *startTimer* (γ, HEARTBEAT);
5:     *startTimer* ($\gamma + \delta$, CHECK);
6: **end event**

7: **upon event** $\langle\ Timeout\ |$ HEARTBEAT $\rangle$ **do**
8:     **for all** $p_i \in \Pi$ **do**
9:         **trigger** $\langle pp2pSend\ |\ p_i,$ [HEARTBEAT] $\rangle$;
10:     **end for**
11:     *startTimer* (γ, HEARTBEAT);
12: **end event**

13: **upon event** $\langle\ Timeout\ |$ CHECK $\rangle$ **do**
14:     **for all** $p_i \in \Pi$ **do**
15:         **if** $(p_i \notin$alive$) \wedge (p_i \notin$detected$)$ **then**
16:             detected := detected $\cup$ { p_i };
17:             **trigger** $\langle crash\ |\ p_i \rangle$;
18:         **end if**
19:     **end for**
20:     alive := $\emptyset$;
21:     *startTimer* ($\gamma + \delta$, CHECK);
22: **end event**

23: **upon event** $\langle\ pp2pDeliver\ |$ src, [HEARTBEAT] $\rangle$ **do**
24:     alive := alive $\cup$ { src };
25: **end event**

Algorithm 2 Eventually Perfect Failure Detector

Implements: EventuallyPerfectFailureDetector ($\lozenge\mathcal{P}$).

Uses: PerfectPointToPointLinks (pp2p).

1: **upon event** $\langle$ $Init$ $\rangle$ **do**
2:     alive := Π;
3:     suspected := $\emptyset$;
4:     period := TimeDelay;
5:     $startTimer$ (period);
6: **end event**

7: **upon event** $\langle$ $Timeout$ $\rangle$ **do**
8:     **if** (alive $\cap$ suspected) $\neq \emptyset$ **then**
9:         period := period + Δ;
10:     **end if**
11:     **for all** $p_i \in \Pi$ **do**
12:         **if** ($p_i \notin$ alive) $\wedge$ ($p_i \notin$ suspected) **then**
13:             suspected := suspected $\cup$ { p_i };
14:             **trigger** $\langle suspect \mid p_i \rangle$;
15:         **else if** ($p_i \in$ alive) $\wedge$ ($p_i \in$ suspected) **then**
16:             suspected := suspected $\setminus$ { p_i };
17:             **trigger** $\langle restore \mid p_i \rangle$;
18:         **end if**
19:         **trigger** $\langle pp2pSend \mid p_i,$ [HEARTBEAT] $\rangle$;
20:     **end for**
21:     alive := $\emptyset$;
22:     $startTimer$ (period);
23: **end event**

24: **upon event** $\langle$ $pp2pDeliver \mid$ src, [HEARTBEAT] $\rangle$ **do**
25:     alive := alive $\cup$ { src };
26: **end event**