Royal Institute of Technology

MSc. Software Engineering of Distributed Systems

ID2203 Distributed Systems Advanced Course

Homework 1

Andrei Shumanski
andreish@kth.se
0046707761992

Trigonakis Vasileios
vtri@kth.se
0046707694420

Stockholm 2010

TABLE OF CONTENTS

EXERCISE 1

EXPERIMENTS

We tried the PFD with both 2 and 4 nodes in fully connected topologies.

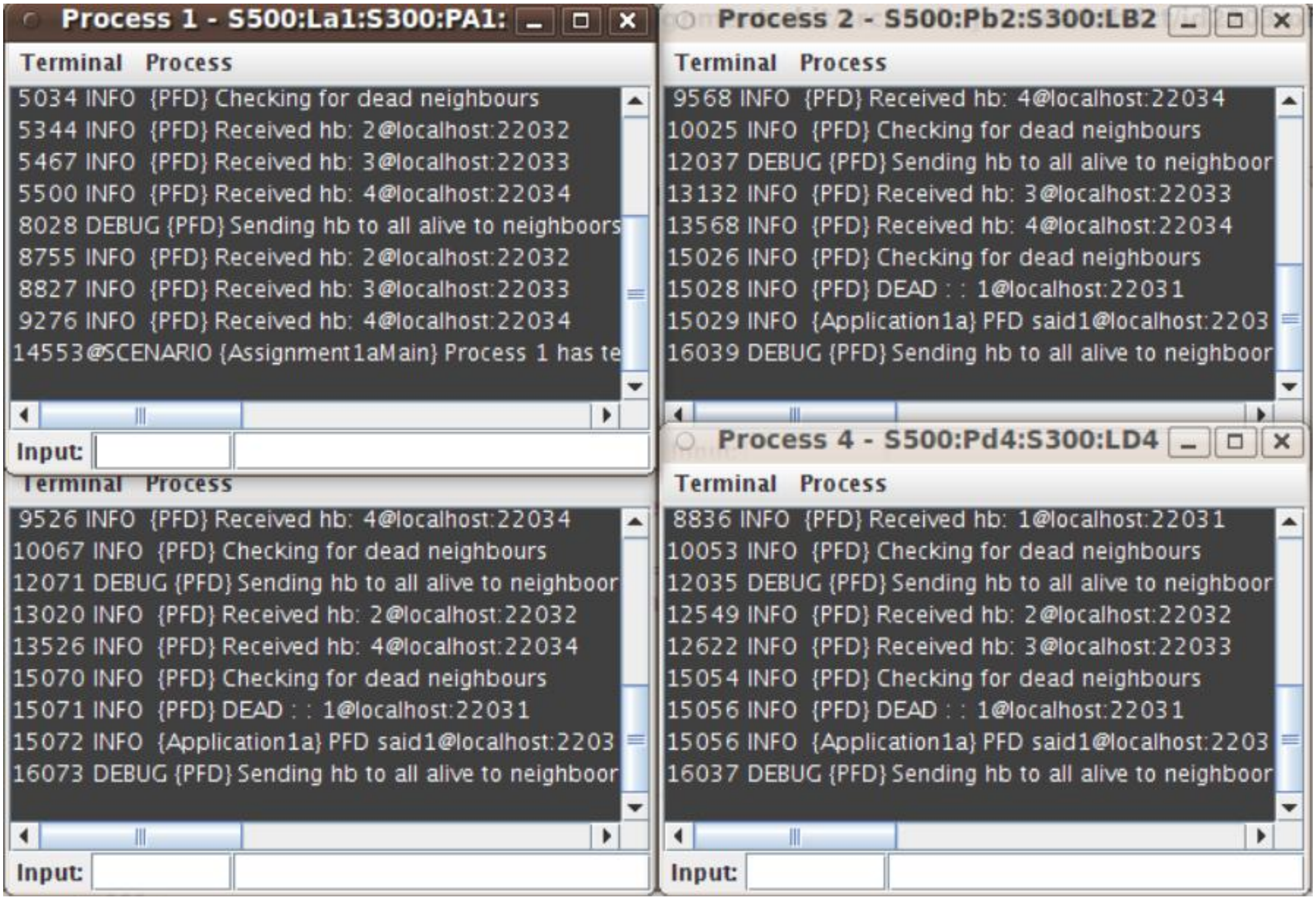
```
Topology topology1 = new Topology() {
            {
                  node(1, "127.0.0.1", 22031);
                  node(2, "127.0.0.1", 22032);
                  link(1, 2, 4000, 0).bidirectional();
            }
};
      Topology topology2 = new Topology() {
            {
                  node(1, "127.0.0.1", 22031);
                  node(2, "127.0.0.1", 22032);
                  node(3, "127.0.0.1", 22033);
                  node(4, "127.0.0.1", 22034);
                  defaultLinks(1000, 0);
            }
      };
```

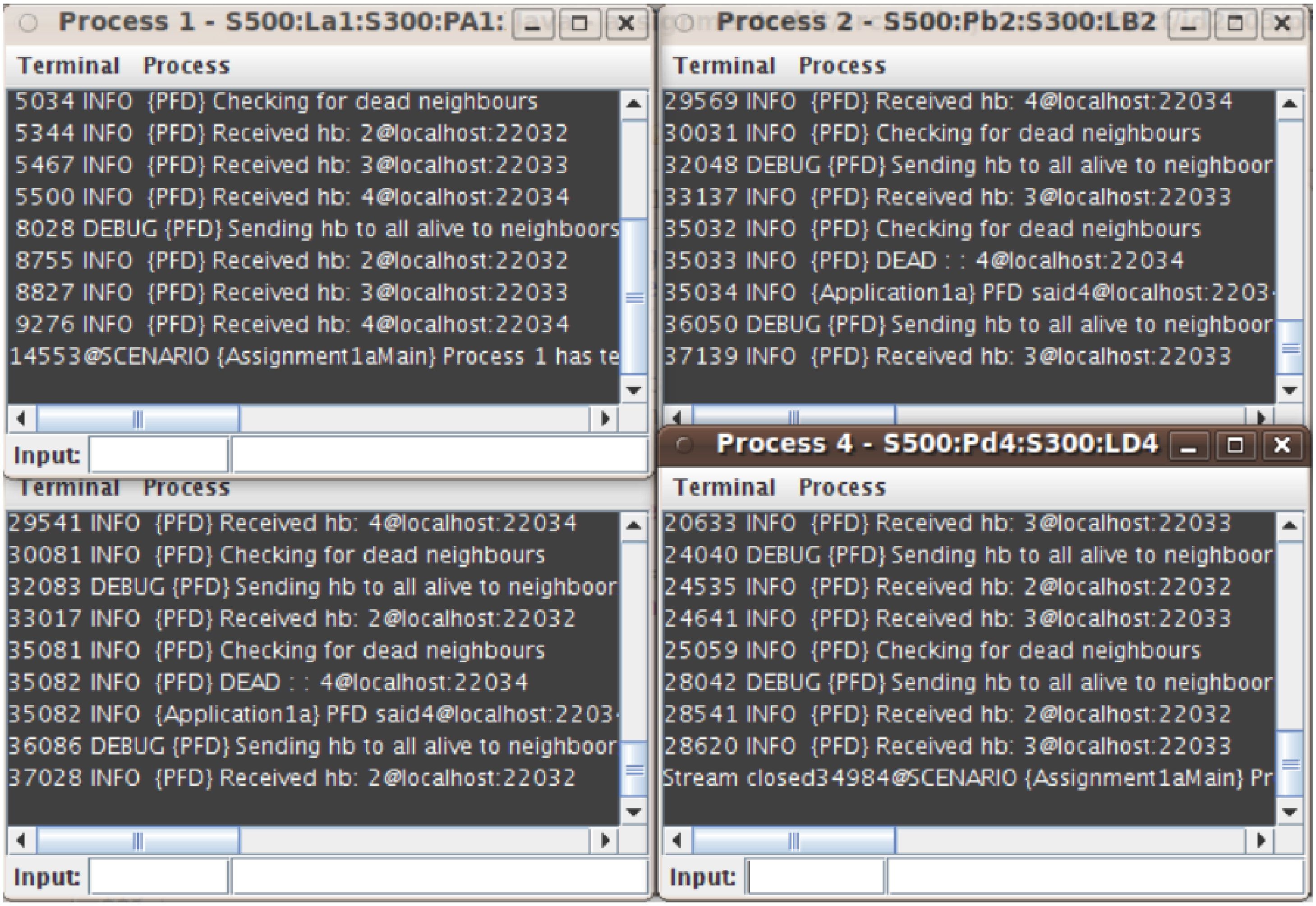
With the following values of γ and δ:

```
private static long GAMMA = 4000;
private static long DELTA = 1000;
```

OBSERVATIONS

The PFD worked as intended, reporting the crash of a node as you can observe in the following screenshots:


Process 1 - S500:La1:S300:PA1:
Terminal Process
5034 INFO {PFD} Checking for dead neighbours
5344 INFO {PFD} Received hb: 2@localhost:22032
5467 INFO {PFD} Received hb: 3@localhost:22033
5500 INFO {PFD} Received hb: 4@localhost:22034
8028 DEBUG {PFD} Sending hb to all alive to neighboors
8755 INFO {PFD} Received hb: 2@localhost:22032
8827 INFO {PFD} Received hb: 3@localhost:22033
9276 INFO {PFD} Received hb: 4@localhost:22034
14553@SCENARIO {Assignment1aMain} Process 1 has te
Input:
Process 2 - S500:Pb2:S300:LB2
Terminal Process
9568 INFO {PFD} Received hb: 4@localhost:22034
10025 INFO {PFD} Checking for dead neighbours
12037 DEBUG {PFD} Sending hb to all alive to neighboor
13132 INFO {PFD} Received hb: 3@localhost:22033
13568 INFO {PFD} Received hb: 4@localhost:22034
15026 INFO {PFD} Checking for dead neighbours
15028 INFO {PFD} DEAD : : 1@localhost:22031
15029 INFO {Application1a} PFD said1@localhost:2203
16039 DEBUG {PFD} Sending hb to all alive to neighboor
Terminal Process
9526 INFO {PFD} Received hb: 4@localhost:22034
10067 INFO {PFD} Checking for dead neighbours
12071 DEBUG {PFD} Sending hb to all alive to neighboor
13020 INFO {PFD} Received hb: 2@localhost:22032
13526 INFO {PFD} Received hb: 4@localhost:22034
15070 INFO {PFD} Checking for dead neighbours
15071 INFO {PFD} DEAD : : 1@localhost:22031
15072 INFO {Application1a} PFD said1@localhost:2203
16073 DEBUG {PFD} Sending hb to all alive to neighboor
Input:
Process 4 - S500:Pd4:S300:LD4
Terminal Process
8836 INFO {PFD} Received hb: 1@localhost:22031
10053 INFO {PFD} Checking for dead neighbours
12035 DEBUG {PFD} Sending hb to all alive to neighboor
12549 INFO {PFD} Received hb: 2@localhost:22032
12622 INFO {PFD} Received hb: 3@localhost:22033
15054 INFO {PFD} Checking for dead neighbours
15056 INFO {PFD} DEAD : : 1@localhost:22031
15056 INFO {Application1a} PFD said1@localhost:2203
16037 DEBUG {PFD} Sending hb to all alive to neighboor
Input:



CONCLUSIONS

Our test showed that the algorithm and the implementation of the PFD works correctly and is consistent to the completeness property of it.

EXERCISE 2

EXPERIMENTS

We tried the EPFD with both link delay shorter than the initial heartbeat period and longer than this. Also, in the first case, we also killed one process to test that the other PFD is detecting the failure.

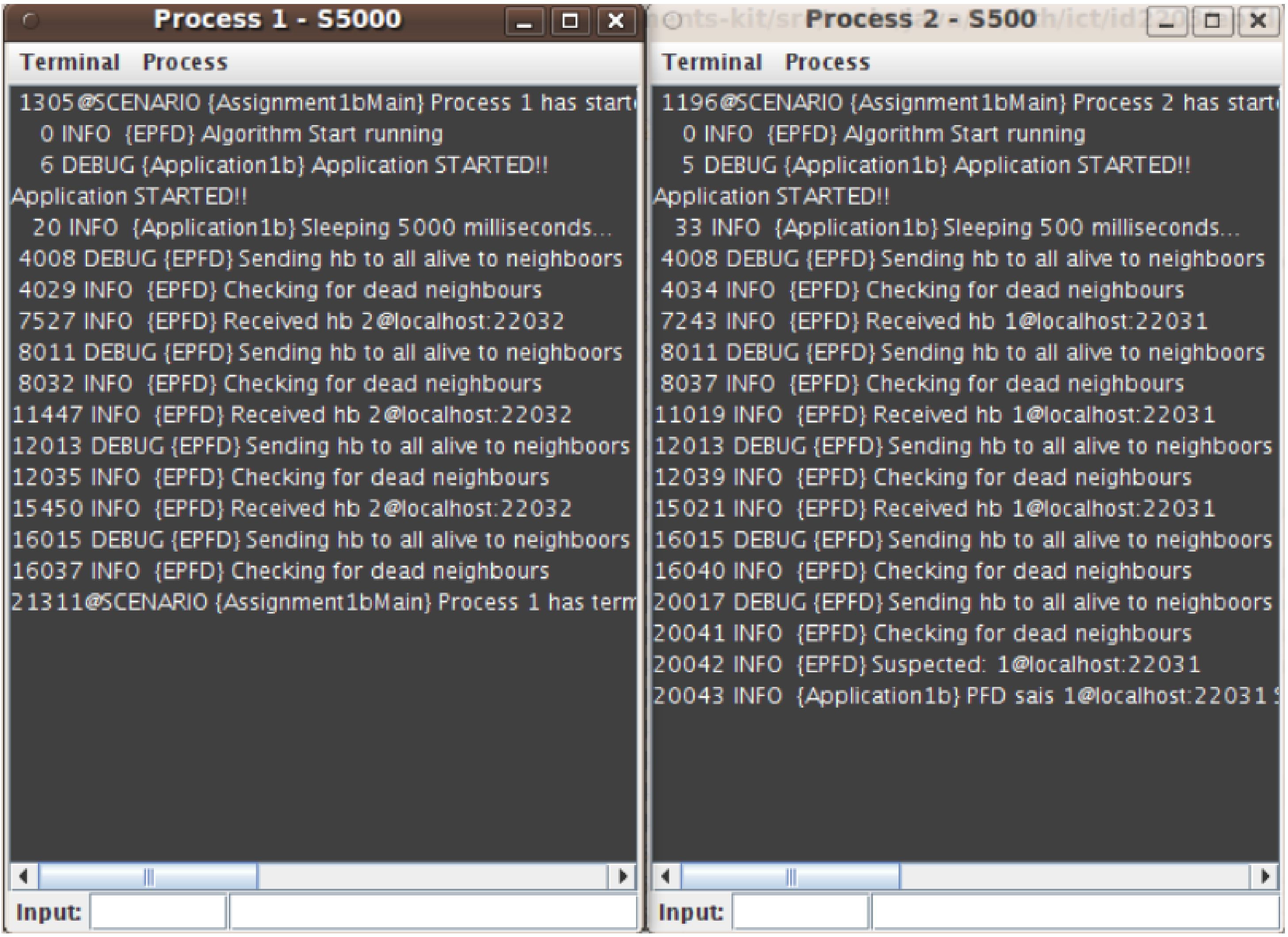
```
Topology topology1 = new Topology() {
                {
                        node(1, "127.0.0.1", 22031);
                        node(2, "127.0.0.1", 22032);
                        link(1, 2, 3213, 0).bidirectional();
                }
        };

        Topology topology2 = new Topology() {
                {
                        node(1, "127.0.0.1", 22031);
                        node(2, "127.0.0.1", 22032);
                        link(1, 2, 4000, 0).bidirectional();
                }
        };
```

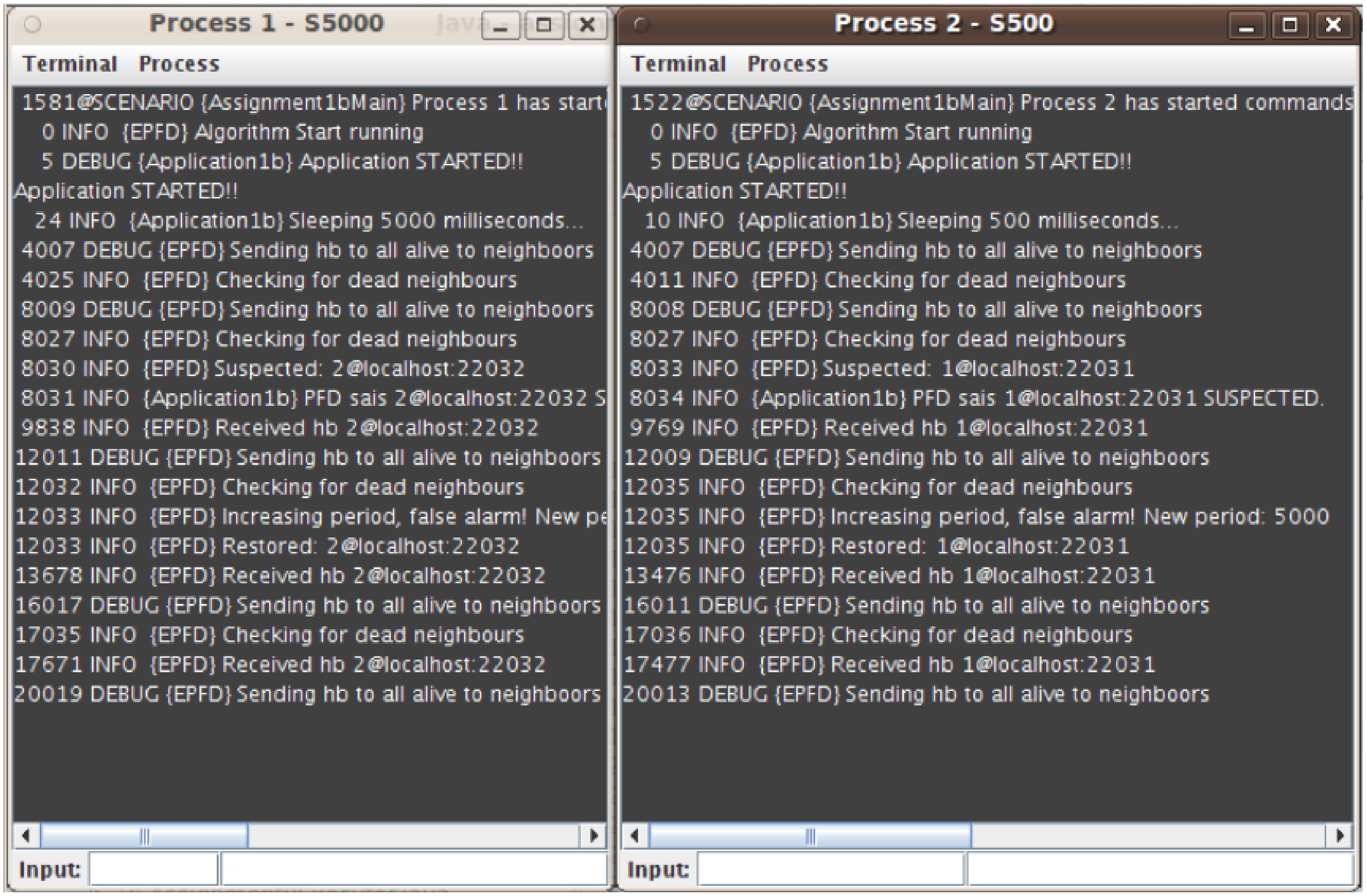
With the following values of TimeDelay and Δ:

```
    private static long TIMEDELAY = 4000;
    private static long DELTA = 1000;
```

OBSERVATIONS

The EPFD worked as intended, reporting the crash of a node in the 1st case and stabilizing in the 2nd case, as you can observe in the following screenshots:





CONCLUSIONS

Our test showed that the algorithm and the implementation of the EPFD in the specific test cases are not that good. What is happening is that after the 1st stabilization of the period time, the heartbeat events keep coming every TimeDelay ms, so a second stabilization does not happen. Actually, this is not exactly a problem, because the EPFD correctly do not suspect the nodes, since they are not crashed.

EXERCISE 3

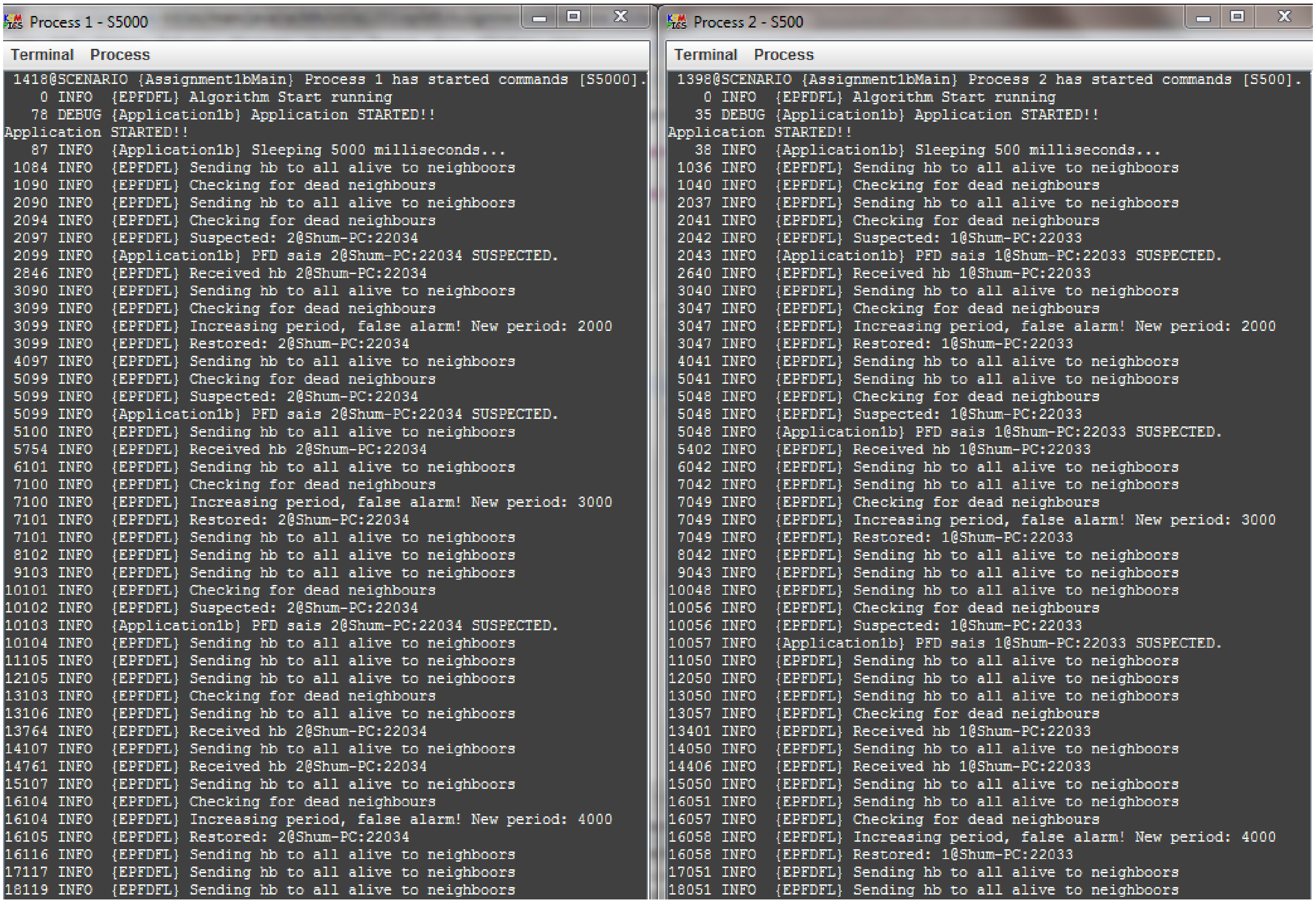
In this exercise we used EPFD with FairLoss links to simulate loss of the messages. We used the following topology:

```
Topology topology1 = new Topology() {
                  {
                        node(1, "127.0.0.1", 22033);
                        node(2, "127.0.0.1", 22034);
                        link(1, 2, 1500, 0.7).bidirectional();
                  }
            };
```

Latency = 1500 Ms;
LossRate = 0.7;
TimeDelay = 1000 Ms;
Delta = 1000 Ms.

We used big loss rate to make adjustment of TimeDelay more often and visible.

OBSERVATIONS



Process 1 - S5000

Terminal Process

```
 1418@SCENARIO {Assignment1bMain} Process 1 has started commands [S5000].
    0 INFO  {EPFDFL} Algorithm Start running
   78 DEBUG {Application1b} Application STARTED!!
Application STARTED!!
   87 INFO  {Application1b} Sleeping 5000 milliseconds...
 1084 INFO  {EPFDFL} Sending hb to all alive to neighboors
 1090 INFO  {EPFDFL} Checking for dead neighbours
 2090 INFO  {EPFDFL} Sending hb to all alive to neighboors
 2094 INFO  {EPFDFL} Checking for dead neighbours
 2097 INFO  {EPFDFL} Suspected: 2@Shum-PC:22034
 2099 INFO  {Application1b} PFD sais 2@Shum-PC:22034 SUSPECTED.
 2846 INFO  {EPFDFL} Received hb 2@Shum-PC:22034
 3090 INFO  {EPFDFL} Sending hb to all alive to neighboors
 3099 INFO  {EPFDFL} Checking for dead neighbours
 3099 INFO  {EPFDFL} Increasing period, false alarm! New period: 2000
 3099 INFO  {EPFDFL} Restored: 2@Shum-PC:22034
 4097 INFO  {EPFDFL} Sending hb to all alive to neighboors
 5099 INFO  {EPFDFL} Checking for dead neighbours
 5099 INFO  {EPFDFL} Suspected: 2@Shum-PC:22034
 5099 INFO  {Application1b} PFD sais 2@Shum-PC:22034 SUSPECTED.
 5100 INFO  {EPFDFL} Sending hb to all alive to neighboors
 5754 INFO  {EPFDFL} Received hb 2@Shum-PC:22034
 6101 INFO  {EPFDFL} Sending hb to all alive to neighboors
 7100 INFO  {EPFDFL} Checking for dead neighbours
 7100 INFO  {EPFDFL} Increasing period, false alarm! New period: 3000
 7101 INFO  {EPFDFL} Restored: 2@Shum-PC:22034
 7101 INFO  {EPFDFL} Sending hb to all alive to neighboors
 8102 INFO  {EPFDFL} Sending hb to all alive to neighboors
 9103 INFO  {EPFDFL} Sending hb to all alive to neighboors
10101 INFO  {EPFDFL} Checking for dead neighbours
10102 INFO  {EPFDFL} Suspected: 2@Shum-PC:22034
10103 INFO  {Application1b} PFD sais 2@Shum-PC:22034 SUSPECTED.
10104 INFO  {EPFDFL} Sending hb to all alive to neighboors
11105 INFO  {EPFDFL} Sending hb to all alive to neighboors
12105 INFO  {EPFDFL} Sending hb to all alive to neighboors
13103 INFO  {EPFDFL} Checking for dead neighbours
13106 INFO  {EPFDFL} Sending hb to all alive to neighboors
13764 INFO  {EPFDFL} Received hb 2@Shum-PC:22034
14107 INFO  {EPFDFL} Sending hb to all alive to neighboors
14761 INFO  {EPFDFL} Received hb 2@Shum-PC:22034
15107 INFO  {EPFDFL} Sending hb to all alive to neighboors
16104 INFO  {EPFDFL} Checking for dead neighbours
16104 INFO  {EPFDFL} Increasing period, false alarm! New period: 4000
16105 INFO  {EPFDFL} Restored: 2@Shum-PC:22034
16116 INFO  {EPFDFL} Sending hb to all alive to neighboors
17117 INFO  {EPFDFL} Sending hb to all alive to neighboors
18119 INFO  {EPFDFL} Sending hb to all alive to neighboors
```

Process 2 - S500

Terminal Process

```
 1398@SCENARIO {Assignment1bMain} Process 2 has started commands [S500].
    0 INFO  {EPFDFL} Algorithm Start running
   35 DEBUG {Application1b} Application STARTED!!
Application STARTED!!
   38 INFO  {Application1b} Sleeping 500 milliseconds...
 1036 INFO  {EPFDFL} Sending hb to all alive to neighboors
 1040 INFO  {EPFDFL} Checking for dead neighbours
 2037 INFO  {EPFDFL} Sending hb to all alive to neighboors
 2041 INFO  {EPFDFL} Checking for dead neighbours
 2042 INFO  {EPFDFL} Suspected: 1@Shum-PC:22033
 2043 INFO  {Application1b} PFD sais 1@Shum-PC:22033 SUSPECTED.
 2640 INFO  {EPFDFL} Received hb 1@Shum-PC:22033
 3040 INFO  {EPFDFL} Sending hb to all alive to neighboors
 3047 INFO  {EPFDFL} Checking for dead neighbours
 3047 INFO  {EPFDFL} Increasing period, false alarm! New period: 2000
 3047 INFO  {EPFDFL} Restored: 1@Shum-PC:22033
 4041 INFO  {EPFDFL} Sending hb to all alive to neighboors
 5041 INFO  {EPFDFL} Sending hb to all alive to neighboors
 5048 INFO  {EPFDFL} Checking for dead neighbours
 5048 INFO  {EPFDFL} Suspected: 1@Shum-PC:22033
 5048 INFO  {Application1b} PFD sais 1@Shum-PC:22033 SUSPECTED.
 5402 INFO  {EPFDFL} Received hb 1@Shum-PC:22033
 6042 INFO  {EPFDFL} Sending hb to all alive to neighboors
 7042 INFO  {EPFDFL} Sending hb to all alive to neighboors
 7049 INFO  {EPFDFL} Checking for dead neighbours
 7049 INFO  {EPFDFL} Increasing period, false alarm! New period: 3000
 7049 INFO  {EPFDFL} Restored: 1@Shum-PC:22033
 8042 INFO  {EPFDFL} Sending hb to all alive to neighboors
 9043 INFO  {EPFDFL} Sending hb to all alive to neighboors
10048 INFO  {EPFDFL} Sending hb to all alive to neighboors
10056 INFO  {EPFDFL} Checking for dead neighbours
10056 INFO  {EPFDFL} Suspected: 1@Shum-PC:22033
10057 INFO  {Application1b} PFD sais 1@Shum-PC:22033 SUSPECTED.
11050 INFO  {EPFDFL} Sending hb to all alive to neighboors
12050 INFO  {EPFDFL} Sending hb to all alive to neighboors
13050 INFO  {EPFDFL} Sending hb to all alive to neighboors
13057 INFO  {EPFDFL} Checking for dead neighbours
13401 INFO  {EPFDFL} Received hb 1@Shum-PC:22033
14050 INFO  {EPFDFL} Sending hb to all alive to neighboors
14406 INFO  {EPFDFL} Received hb 1@Shum-PC:22033
15050 INFO  {EPFDFL} Sending hb to all alive to neighboors
16051 INFO  {EPFDFL} Sending hb to all alive to neighboors
16057 INFO  {EPFDFL} Checking for dead neighbours
16058 INFO  {EPFDFL} Increasing period, false alarm! New period: 4000
16058 INFO  {EPFDFL} Restored: 1@Shum-PC:22033
17051 INFO  {EPFDFL} Sending hb to all alive to neighboors
18051 INFO  {EPFDFL} Sending hb to all alive to neighboors
```

CONCLUSION.

We see that in this scenario TimeDelay increases to accommodate large delays when messages are lost. We observed that the larger is probability of loosing messages the faster grows TimeDelay. In the run shown above after 137 second TimeDelay increased to 8000 Ms.

EXERCISE 4

EXPERIMENTS

We implemented the doRecover method, that is being run when a process recovers from a failure. It just prints an informational message to the logger. The topology and scenario that we used:

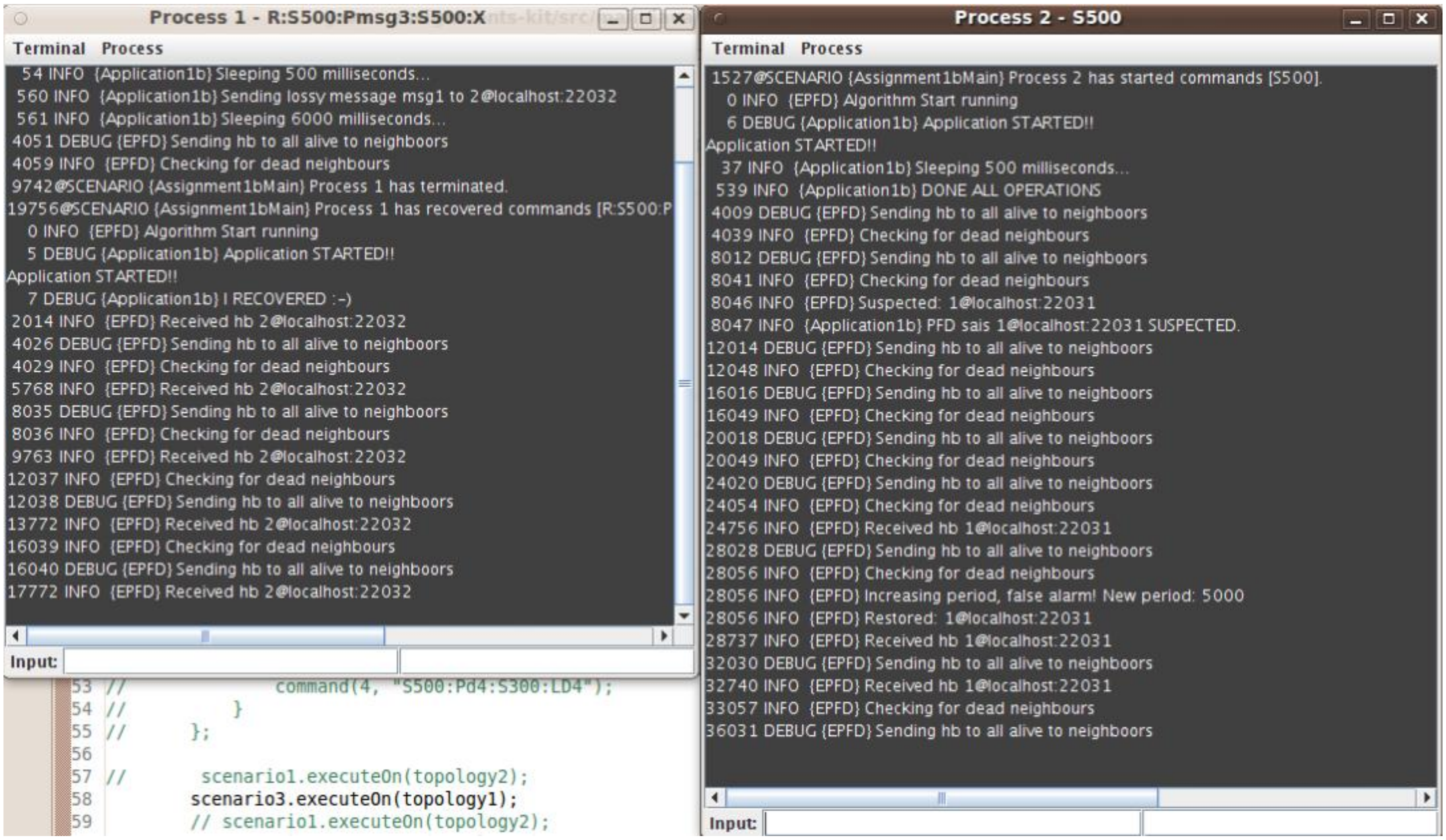
```
Topology topology1 = new Topology() {
                  {
                        node(1, "127.0.0.1", 22031);
                        node(2, "127.0.0.1", 22032);
                        link(1, 2, 3213, 0).bidirectional();
                  }
            };

      Scenario scenario3 = new Scenario(Assignment1bMain.class) {
                  {
            command(1, "S500:Lmsg1:S6000:X").recover("R:S500:Pmsg3:S500:X",
      10000);
                        command(2, "S500");
                  }
            };
```

With the following values of TimeDelay and Δ:

```
      private static long TIMEDELAY = 4000;
      private static long DELTA = 1000;
```

OBSERVATIONS

The results can be seen in the following screenshot:

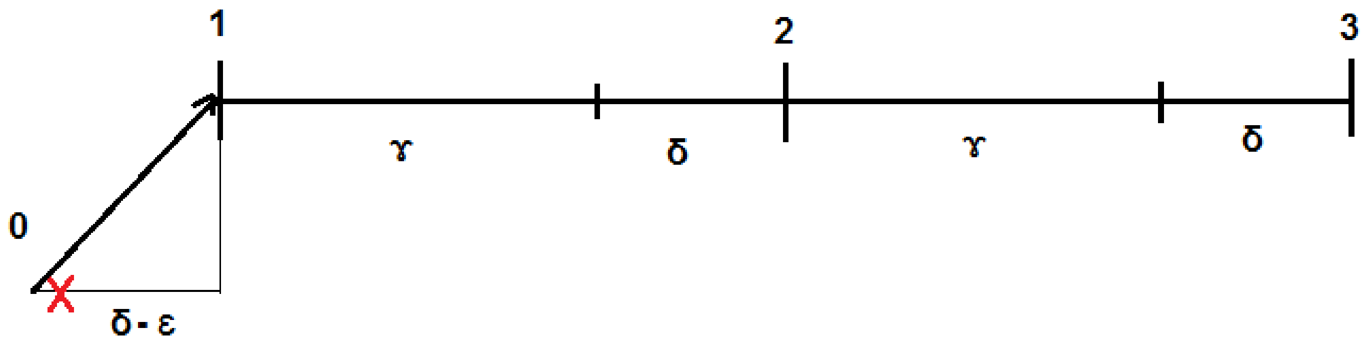


CONCLUSIONS

As we expected, the EPFD correctly suspected the process that failed. This is absolutely normal, since in exercise 2 we tested that the EPFD detected a crashed node. Moreover, we can see that after the node recovers, it starts sending heartbeats again, and the other node calls a “false alarm” about the suspected event. Finally, we can conclude that the nodes that suspect the crashed node, when this node recovers, they are not able to distinguish between a node failure and recovery, an omission failure or just if the period of checking was not set to the correct value.

QUESTION 1

The worst time is 2ɤ + 3δ. This situation illustrates following picture:



Let process p1 sends heartbeat at 0 and immediately dies. The message is delivered in δ – maximum transmission dely. The time between 0 and 1 is (δ- ε) where ε → 0 is very small. Process p2 checks for heartbeats at 1, 2 ,3. As heartbeat from p1 was delivered at 1+ ε process p2 at 2 will not understand that p1 is dead and understands it only at 3. It means after (2ɤ + 3δ- ε), but as ε is very small we can consider that it is after (2ɤ + 3δ).

The best case is δ. If the node p1 died less than δ before the check it means that the previous its heartbeat was less than (δ+ ɤ) before and was received during that check round so the node will not be detected as dead during this check.

QUESTION 2

Can you improve the algorithm for PFD, such that it improves the worst case failure detection time?

Yes. Solutions:

More timers (more computations)

Keep a specific timer for every neighbor.

```
upon event < pp2pDeliver | Pi, [HEARTBEAT]> do

        cancel timer for Pi;

        startTimer (Pi);


upon event 〈 Timeout | Pi 〉 do

        detected := detected ∨ {Pi};

        trigger 〈 crash | Pi 〉;
```

By this way, we have a worst case of:

Pi sends a heartbeat and dies after $\varepsilon \rightarrow 0$, that arrives to Pj after δ

Pj cancels the timer and starts a new one, that will trigger and detect Pi after $\gamma + \delta$

So, worst case of $\gamma + 2\delta$

More messages

A process can broadcast a heartbeat request to the other processes every γ and then wait for 2δ (2 * max network delay) for the response. If one process does not response in this time, it has crashed.

By this way, we have a worst case of:

Pi sends a heartbeat request to Pj (and all other nodes), request that arrives after $\varepsilon \rightarrow 0$

Pj responses after $\zeta \rightarrow 0$

Pi checks for crashes after $\mathbf{2\delta}$, Pj looks ok

Pi sends a heartbeat request to Pj (and all other nodes) after γ

Pi detects the crash of Pj after $\mathbf{2\delta}$

So, worst case of $\mathbf{2\delta + \gamma + 2\delta = \gamma + 4\delta}$

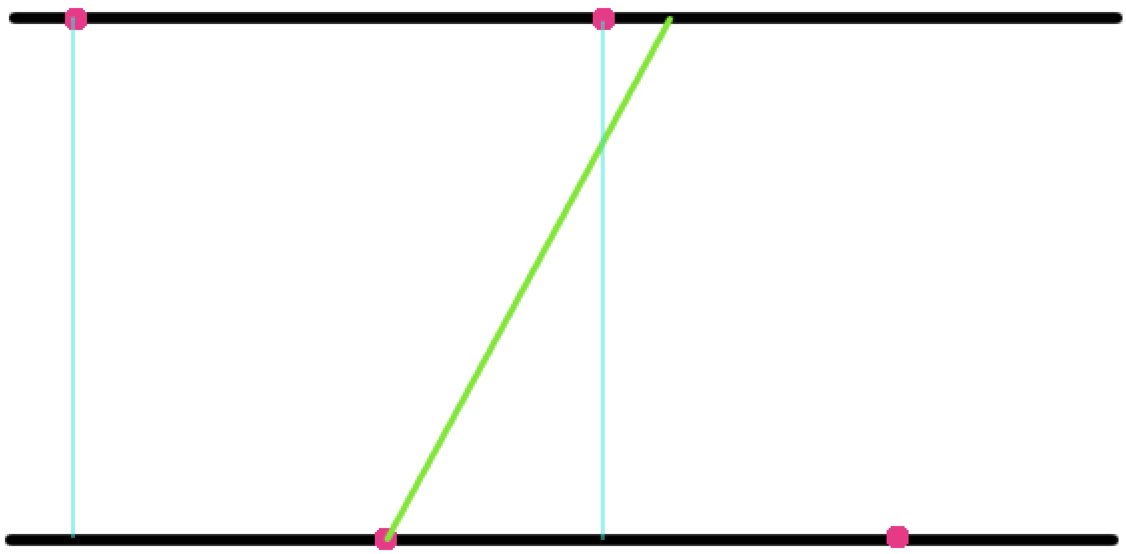
QUESTION 3

Can you improve the algorithm for PFD, such that it requires a single timer?

If we use a heuristic solution, we can answer yes. In the algorithm that we used for this homework, we can use **only one timer** with period **δ** and select a value for γ, such that $\gamma = \nu * \delta$, where $\nu \in I$. Then we can simply use a counter and two ifs to select when each part of the algorithm will execute.

Else, if we want to use an algorithm that is making use of only one timer, we would have to use the book's algorithm, which we will prove that is wrong in a specific case.

Assume that the TimeDelay that is used in the book's algorithm can be written as:



TimeDelay = $\gamma + \delta$, where δ is the maximum network delay. If we do not assume that the cycles in every node are synchronized, then there is the chance that the following execution could appear:

Which would obviously cause a false alarm.