Distributed Algorithms Coursework 1

Rishabh Jain (rj2315) and Vinamra Agrawal (va1215)

February 7, 2018

1 Setup

1.1 Local Setup

- Macbook Pro 2016 (macOS High Sierra).
- 2.6 GHz Intel Core i7.
- 16GB RAM.

1.2 Docker Setup

- Docker run on same machine as mentioned above.
- Version - 3.4
- Image - elixir:alpine

2 Tests and Deductions

2.1 System 1 - Elixir Broadcast

2.1.1 Test 1

Timeout = 3 sec; Max Broadcasts = 1000; #Peers = 5
Running System 1 with the mentioned conditions gives us perfect results where all the peers send/receive 1000 messages to/from every other peer without any loss. The results are the same when running locally or running on docker.

```
0: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
2: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
4: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
3: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
1: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
```

2.1.2 Test 2

Timeout = 3 sec; Max Broadcasts = 10_000_000; #Peers = 5
In this case, the timeout occurred before all the peers could finish broadcasting/receiving all the messages.

Local

Locally the number of messages received/sent was around 130_000 in the 3 second timeout window.

```
0: {146464, 146464} {146464, 172042} {146464, 143232} {146464, 164464} {146464, 160435}
2: {143232, 146463} {143232, 172042} {143232, 143232} {143232, 161523} {143232, 159694}
1: {172109, 146464} {172109, 172108} {172109, 143232} {172109, 181137} {172109, 161570}
4: {161570, 146463} {161570, 172042} {161570, 143232} {161570, 165173} {161570, 161569}
3: {181137, 146464} {181137, 172042} {181137, 143232} {181137, 181137} {181137, 161570}
```

Docker

On docker the number of messages received/sent was around 75_000, lesser than when run locally, because of higher latency involved between docker containers.

```
| 0: {75318, 75317} {75318, 67186} {75318, 57870} {75318, 39228} {75318, 54067}
| 3: {39692, 70021} {39692, 66892} {39692, 57223} {39692, 39691} {39692, 54067}
| 4: {54067, 55465} {54067, 42122} {54067, 52138} {54067, 35875} {54067, 54066}
| 2: {59523, 70022} {59523, 67186} {59523, 59523} {59523, 39258} {59523, 54067}
| 1: {67186, 70021} {67186, 67186} {67186, 53712} {67186, 39228} {67186, 54067}
```

2.1.3 Test 3

Timeout = 1 sec; Max Broadcasts = 1000; #Peers = 5
When run with even lower timeout of 1 second for 1000 messages, we were still able to communicate all the messages without any loss in any peer when run either locally or on docker.

```
0: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
2: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
4: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
3: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
1: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
```

2.2 System 2 - PL Broadcast

2.2.1 Test 1

Timeout = 3 sec; Max Broadcasts = 1000; #Peers = 5
Running System 2 with the mentioned conditions gives us perfect results where all the peers send/receive 1000 messages to/from every other peer without any loss. The results are the same when running locally or on docker.

```
0: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
2: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
4: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
3: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
1: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
```

2.2.2 Test 2

Timeout = 3 sec; Max Broadcasts = 10_000_000; #Peers = 5
In this case, the timeout occurred before all the peers could finish broadcasting/receiving all the messages. If compared to test 2 in System 1, messages send rate is similar however the receive rate is dropped dramatically. The main reason could be that messages now have to pass through more components(PL(s) in this case) before they reach their destination which needs extra time.

Local

Locally the number of messages sent was around 150_000 but received only about 1_000.

```
0: {153490, 342} {153490, 757} {153490, 370} {153490, 940} {153490, 969}
3: {165912, 343} {165912, 973} {165912, 374} {165912, 1183} {165912, 1039}
2: {161979, 376} {161979, 1090} {161979, 521} {161979, 1265} {161979, 1084}
1: {160507, 343} {160507, 1065} {160507, 381} {160507, 1220} {160507, 1039}
4: {153725, 342} {153725, 905} {153725, 373} {153725, 1049} {153725, 1039}
```

Docker

On docker the number of messages end was around 110_000 but received vary between 0 to 2_000.

```
| 1: {119453, 2284} {119453, 1421} {119453, 1} {119453, 1} {119453, 1}
| 4: {153691, 2158} {153691, 1316} {153691, 123} {153691, 0} {153691, 0}
| 2: {139013, 2764} {139013, 677} {139013, 361} {139013, 0} {139013, 0}
| 3: {119581, 2624} {119581, 436} {119581, 202} {119581, 1} {119581, 0}
| 0: {135899, 4064} {135899, 300} {135899, 1} {135899, 1} {135899, 1}
```

2.2.3 Test 3

Timeout = 1 sec; Max Broadcasts = 1000; #Peers = 5
When run with even lower timeout of 1 second for 1000 messages, we were still able to communicate all the messages without any loss in any peer when run locally similar to test 3 on System 1. However in the docker case, some of the packets could not be processed before the timeout, due to more latency when sending messages between docker containers.

Local

```
0: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
2: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
4: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
3: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
1: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
```

Docker

```
| 3: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
| 2: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
| 0: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
| 1: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 822} {1000, 956}
| 4: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
```

2.3 System 3 - Best Effort Broadcast

2.3.1 Test 1

Timeout = 3 sec; Max Broadcasts = 1000; #Peers = 5
As expected the output was every peer sent and received 1000 broadcasts to/from every other peer. This is same as test 1 in System 1 and 2.

```
0: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
2: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
4: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
3: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
1: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
```

2.3.2 Test 2

Timeout = 3 sec; Max Broadcasts = 10_000; #Peers = 5
In this case, the timeout occurred before all the peers could finish broadcasting/receiving all the messages. In this case one more component is added to the system further increasing the path of the message and therefore more time is required for messages to be received.

If compared to test 2 in System 2, messages send rate is similar however the receive rate is dropped dramatically. The main reason could be that messages now have to pass through more components(PL(s) in this case) before they reach their destination which need extra time.

Local

Locally the number of messages sent was around 150_000 but received only about 1_000.

```
0: {153490, 342} {153490, 757} {153490, 370} {153490, 940} {153490, 969}
3: {165912, 343} {165912, 973} {165912, 374} {165912, 1183} {165912, 1039}
2: {161979, 376} {161979, 1090} {161979, 521} {161979, 1265} {161979, 1084}
1: {160507, 343} {160507, 1065} {160507, 381} {160507, 1220} {160507, 1039}
4: {153725, 342} {153725, 905} {153725, 373} {153725, 1049} {153725, 1039}
```

Docker

On docker the number of messages end was around 110_000 but received vary between 0 to 2_000.

```
1: {119453, 2284} {119453, 1421} {119453, 1} {119453, 1} {119453, 1}
4: {153691, 2158} {153691, 1316} {153691, 123} {153691, 0} {153691, 0}
2: {139013, 2764} {139013, 677} {139013, 361} {139013, 0} {139013, 0}
3: {119581, 2624} {119581, 436} {119581, 202} {119581, 1} {119581, 0}
0: {135899, 4064} {135899, 300} {135899, 1} {135899, 1} {135899, 1}
```

2.4 System 4 - Unreliable Message Sending

2.4.1 Test 1

Timeout = 3 sec; Max Broadcasts = 1000; #Peers = 5; Reliability = 100%
With the above conditions, the Lossy PL behaves like a normal PL since the reliability is set to a 100 percent. Hence the results we get are the same as for the last system.

```
0: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
2: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
4: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
3: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
1: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
```

2.4.2 Test 2

Timeout = 3 sec; Max Broadcasts = 1000; #Peers = 5; Reliability = 50%
With a 50 percent reliability, almost half of the broadcasts sent by each app were actually received. The results were similar when run locally or on docker.

```
2: {1000, 527} {1000, 473} {1000, 520} {1000, 524} {1000, 495}
0: {1000, 521} {1000, 507} {1000, 489} {1000, 513} {1000, 492}
4: {1000, 506} {1000, 497} {1000, 514} {1000, 511} {1000, 485}
3: {1000, 497} {1000, 529} {1000, 502} {1000, 492} {1000, 523}
1: {1000, 502} {1000, 495} {1000, 473} {1000, 500} {1000, 516}
```

2.4.3 Test 3

Timeout = 3 sec; Max Broadcasts = 1000; #Peers = 5; Reliability = 0%
With a 0 percent reliability, none of the broadcasts sent by each app were received. The results were the similar when run locally or on docker.

```
1: {1000, 0} {1000, 0} {1000, 0} {1000, 0} {1000, 0}
0: {1000, 0} {1000, 0} {1000, 0} {1000, 0} {1000, 0}
2: {1000, 0} {1000, 0} {1000, 0} {1000, 0} {1000, 0}
3: {1000, 0} {1000, 0} {1000, 0} {1000, 0} {1000, 0}
4: {1000, 0} {1000, 0} {1000, 0} {1000, 0} {1000, 0}
```

2.5 System 5 - Faulty Process

2.5.1 Test 1

Timeout = 3 sec; Max Broadcasts = 1000; #Peers = 5; Reliability = 100%; Peer 3 timeout = 3ms
As we can see from the test below 3rd peer exits first. It is only able to send a few packets before termination which are received by other peers successfully. We witness similar results both when run locally and when in docker.

Local

```
3: {691, 0} {691, 0} {691, 0} {691, 0} {691, 0}
4: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 691} {1000, 1000}
1: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 691} {1000, 1000}
0: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 691} {1000, 1000}
2: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 691} {1000, 1000}
```

Docker

```
3: {597, 0} {597, 0} {597, 0} {597, 0} {597, 0}
0: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 597} {1000, 1000}
1: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 597} {1000, 1000}
4: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 597} {1000, 1000}
2: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 597} {1000, 1000}
```

2.5.2 Test 2

Timeout = 5 sec; Max Broadcasts = 1000; #Peers = 5; Reliability = 50%; Peer 3 Timeout = 250ms
In this test case Peer 3 have lot more time as compared to to test 1, hence when run locally as expected it was able to receive messages with 50 % reliability. However this time was not enough for Peers to send receive messages on docker because of increased latency while sending/receiving messages between containers. Therefore we observe Peer 3 to receive lesser than 50% messages when terminating.

Local

```
3: {1000, 500} {1000, 512} {1000, 480} {1000, 483} {1000, 495}
0: {1000, 516} {1000, 494} {1000, 486} {1000, 490} {1000, 517}
2: {1000, 517} {1000, 513} {1000, 498} {1000, 497} {1000, 506}
4: {1000, 515} {1000, 501} {1000, 513} {1000, 509} {1000, 470}
1: {1000, 510} {1000, 501} {1000, 522} {1000, 512} {1000, 508}
```

Docker

```
| 3: {1000, 3} {1000, 86} {1000, 0} {1000, 492} {1000, 2}
| 2: {1000, 472} {1000, 501} {1000, 505} {1000, 484} {1000, 507}
| 1: {1000, 485} {1000, 501} {1000, 499} {1000, 498} {1000, 496}
| 0: {1000, 499} {1000, 500} {1000, 484} {1000, 462} {1000, 512}
| 4: {1000, 495} {1000, 499} {1000, 503} {1000, 516} {1000, 502}
```

2.5.3 Test 3

Timeout = 5 sec; Max Broadcasts = 1000; #Peers = 5; Reliability = 100%; Peer 3 Timeout = 250ms; Sleep between sending = 1ms
This case is particularly interesting, we have observed that due to increased number of components, time taken to receive a message has increased considerably. After sending them they need to pass through more number of components before reaching the destination app.
As a result we were able to send all messages in a few milliseconds but took quite long after that to receive the first message. As more messages were created at a faster rate than they were sent, the time lag between them increased exponentially.
To make sending and receiving messages at a similar rate we put a process sleep of 1ms after each message send. This gave a chance to clear the pipeline and balance the send and receive rates.

Local

As expected with now balanced send/receive rate, Peer 3 was able to send/receive 125 messages to each peer before crashing.

```
3: {125, 124} {125, 124} {125, 124} {125, 124} {125, 124}
1: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 125} {1000, 1000}
0: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 125} {1000, 1000}
4: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 125} {1000, 1000}
2: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 125} {1000, 1000}
```

Docker

As expected, similar to local results, Peer 3 was able to send/receive about 110 messages to each peer before crashing.

```
| 3: {111, 115} {111, 109} {111, 102} {111, 110} {111, 99}
| 0: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 111} {1000, 1000}
| 2: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 111} {1000, 1000}
| 4: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 111} {1000, 1000}
| 1: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 111} {1000, 1000}
```

2.6 System 6 - Eager Reliable Broadcast

2.6.1 Test 1

Timeout = 3 sec; Max Broadcasts = 1000; #Peers = 5; Reliability = 100%
With the above conditions, since the reliability is set to a 100 percent we get all the messages as expected. Hence the results we get are the same as for the last system.

```
0: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
2: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
4: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
3: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
1: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
```

2.6.2 Test 2

Timeout = 3 sec; Max Broadcasts = 1000; #Peers = 5; Reliability = 50%
With a 50 percent reliability, almost half of the broadcasts sent by each app should be received (similar to System 4 test 2), however we received about 93% on average. This is due to the reliability property of the system. It rebroadcasts each new message it receives.
The messages in system is expected to increase by n (# of peers) times compared to System 4 however the reliability also increased (from 50 to 93% compared to System 4). The results were similar when run locally or on docker.

Local

```
3: {1000, 929} {1000, 920} {1000, 921} {1000, 919} {1000, 932}
1: {1000, 924} {1000, 926} {1000, 914} {1000, 925} {1000, 926}
4: {1000, 927} {1000, 912} {1000, 904} {1000, 918} {1000, 942}
0: {1000, 922} {1000, 915} {1000, 920} {1000, 919} {1000, 941}
2: {1000, 918} {1000, 914} {1000, 908} {1000, 924} {1000, 935}
```

Docker

```
2: {1000, 936} {1000, 908} {1000, 947} {1000, 918} {1000, 921}
1: {1000, 941} {1000, 914} {1000, 933} {1000, 914} {1000, 916}
0: {1000, 926} {1000, 916} {1000, 933} {1000, 917} {1000, 928}
4: {1000, 924} {1000, 915} {1000, 924} {1000, 927} {1000, 908}
3: {1000, 934} {1000, 894} {1000, 926} {1000, 916} {1000, 921}
```

2.6.3 Test 3

Timeout = 30 sec; Max Broadcasts = 1000; #Peers = 10; Reliability = 50%
This test case is similar to previous test case with 50% reliability, however we have increased number of peers in this case. This has resulted in ever higher reliability of overall message delivery. This could be explained because of the structure of eager reliable broadcast. Increase in peers would result in more overall re-propagation of a message which would result in more chances of receiving a message form lossy PLs. The following result show the following, the results are similar locally and on docker.

Local

```
1: {1000, 997} {1000, 999} {1000, 1000} {1000, 998} {1000, 999} {1000, 998} {1000, 996} {1000, 1000} {1000, 998} {1000, 998}
7: {1000, 998} {1000, 998} {1000, 997} {1000, 996} {1000, 999} {1000, 998} {1000, 998} {1000, 998} {1000, 998} {1000, 999}
8: {1000, 998} {1000, 998} {1000, 1000} {1000, 998} {1000, 999} {1000, 999} {1000, 996} {1000, 1000} {1000, 997} {1000, 996}
0: {1000, 998} {1000, 1000} {1000, 999} {1000, 993} {1000, 997} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 996} {1000, 999}
2: {1000, 997} {1000, 999} {1000, 999} {1000, 998} {1000, 999} {1000, 998} {1000, 998} {1000, 998} {1000, 999} {1000, 999}
6: {1000, 998} {1000, 1000} {1000, 1000} {1000, 998} {1000, 999} {1000, 998} {1000, 999} {1000, 999} {1000, 997} {1000, 996}
5: {1000, 998} {1000, 999} {1000, 997} {1000, 998} {1000, 999} {1000, 999} {1000, 998} {1000, 1000} {1000, 997} {1000, 998}
9: {1000, 998} {1000, 999} {1000, 999} {1000, 997} {1000, 998} {1000, 999} {1000, 999} {1000, 999} {1000, 998} {1000, 999}
3: {1000, 998} {1000, 999} {1000, 998} {1000, 996} {1000, 999} {1000, 1000} {1000, 1000} {1000, 998} {1000, 998} {1000, 998}
4: {1000, 997} {1000, 1000} {1000, 1000} {1000, 998} {1000, 998} {1000, 999} {1000, 1000} {1000, 1000} {1000, 998} {1000, 999}
```

Docker

```
6: {1000, 998} {1000, 999} {1000, 999} {1000, 997} {1000, 999} {1000, 997} {1000, 997} {1000, 997} {1000, 999} {1000, 998}
3: {1000, 997} {1000, 997} {1000, 1000} {1000, 998} {1000, 997} {1000, 998} {1000, 997} {1000, 998} {1000, 999} {1000, 999}
7: {1000, 997} {1000, 999} {1000, 1000} {1000, 997} {1000, 999} {1000, 998} {1000, 998} {1000, 998} {1000, 1000} {1000, 999}
2: {1000, 997} {1000, 998} {1000, 1000} {1000, 999} {1000, 999} {1000, 997} {1000, 997} {1000, 998} {1000, 999} {1000, 1000}
0: {1000, 997} {1000, 999} {1000, 1000} {1000, 998} {1000, 998} {1000, 998} {1000, 997} {1000, 998} {1000, 997} {1000, 999}
1: {1000, 996} {1000, 999} {1000, 998} {1000, 999} {1000, 1000} {1000, 997} {1000, 997} {1000, 998} {1000, 1000} {1000, 1000}
4: {1000, 998} {1000, 999} {1000, 1000} {1000, 999} {1000, 1000} {1000, 998} {1000, 998} {1000, 999} {1000, 999} {1000, 1000}
5: {1000, 998} {1000, 998} {1000, 999} {1000, 998} {1000, 998} {1000, 997} {1000, 997} {1000, 998} {1000, 999} {1000, 999}
9: {1000, 996} {1000, 998} {1000, 999} {1000, 996} {1000, 999} {1000, 998} {1000, 998} {1000, 999} {1000, 998} {1000, 1000}
8: {1000, 998} {1000, 999} {1000, 1000} {1000, 997} {1000, 999} {1000, 996} {1000, 998} {1000, 999} {1000, 999} {1000, 998}
```

2.7 System 7 - Lazy Reliable Broadcast

2.7.1 Test 1

Timeout = 10 sec; Max Broadcasts = 1000; #Peers = 5; Reliability = 100%; Heartbeat Rate = 50ms

The system behaves just like System 6 with these conditions. The heartbeat rate here is set to 50ms because a round trip heartbeat request and response can take that much time in our system. The results were the same when run locally or on docker.

```
0: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
2: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
4: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
3: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
1: {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000} {1000, 1000}
```

2.7.2 Test 2

Timeout = 10 sec; Max Broadcasts = 1000; #Peers = 5; Reliability = 50%; Heartbeat Rate = 50ms

The results seem to be interesting in this case since, the PL2s (used for heartbeat communications) in Lazy Reliable Broadcast are lossy as well. Hence, a lot of the times, peers get pfd_crash updates for peers that are actually alive. This results in rebroadcasts of the received messages from that peer. Thus, overall, all the peers end up receiving more messages than they should have received (based on the reliability of the PLs).

Therefore, in this test case, the peers tend to receive around 92% of the broadcasted messages (even though the reliability is set to 50%).

```
4: {1000, 929} {1000, 916} {1000, 920} {1000, 922} {1000, 918}
2: {1000, 927} {1000, 916} {1000, 923} {1000, 925} {1000, 930}
3: {1000, 921} {1000, 923} {1000, 922} {1000, 929} {1000, 924}
1: {1000, 932} {1000, 928} {1000, 918} {1000, 924} {1000, 930}
0: {1000, 924} {1000, 933} {1000, 906} {1000, 926} {1000, 925}
```

2.7.3 Test 3

Timeout = 10 sec; Max Broadcasts = 10; #Peers = 5; Reliability = 100%; Heartbeat Rate = 5ms; Peer 3 Timeout = 10ms; Artificial delay between each message sent = 5ms

We added this test case to test the functionality of Lazy Reliable Broadcast. Since the timeout for peer 3 is just 10ms, it crashes after sending its first broadcast to only the first peer. Then, as we can see in the figure below, the peers get notified about peer 3 crashing. Eventually, the first broadcast message from peer 3 gets re-broadcasted and is received by everyone else. Thus, proving the functionality of the LRB system.

```
#PID<0.117.0>: pfd_crash received for #PID<0.120.0>
#PID<0.118.0>: pfd_crash received for #PID<0.120.0>
#PID<0.119.0>: pfd_crash received for #PID<0.120.0>
#PID<0.121.0>: pfd_crash received for #PID<0.120.0>
#PID<0.117.0>: Re-broadcasting {:broadcast_msg, 0} from #PID<0.120.0>
#PID<0.118.0>: Re-broadcasting {:broadcast_msg, 0} from #PID<0.120.0>
#PID<0.119.0>: Re-broadcasting {:broadcast_msg, 0} from #PID<0.120.0>
#PID<0.121.0>: Re-broadcasting {:broadcast_msg, 0} from #PID<0.120.0>
4: {10, 10} {10, 10} {10, 10} {10, 1} {10, 10}
0: {10, 10} {10, 10} {10, 10} {10, 1} {10, 10}
2: {10, 10} {10, 10} {10, 10} {10, 1} {10, 10}
1: {10, 10} {10, 10} {10, 10} {10, 1} {10, 10}
```