iDetect - Lab 1

Your established client, Mugscript, has been experiencing more failures as its data center grows. A failure detection system for its servers is needed for continuity of service and notification. You have been directed to build iDetect, a distributed crash-stop failure detection utility.

Functional Requirements

A1. iDetect shall support a minimum of 4 networked nodes.
A2. The failure of any single node shall be reported by at least one non-faulty node within 8 seconds. It may be assumed that RTCs among the nodes are synchronized (e.g., using NTP).
A3. The failure of any two nodes simultaneously shall be reported by at least one working node within 10 seconds.
A4. iDetect shall produce no false positive reports of failure when no packets are lost in network communications.
A5. iDetect shall produce no more than 1 false positive report in any 5 second period when 20% of packets are lost at random in network communications.
A6. The network overhead bandwidth used for failure detection should be minimized.

Requirements Test

The following lists the methods used to verify the requirements set forth by the instructor

Test - A1

1. Start 5 iDetect nodes with the following settings:
    a. Node 1 port: 5001
    b. Node 2 port: 5002
    c. Node 3 port: 5003
    d. Node 4 port: 5004
    e. Node 5 port: 5005
2. Start each iDetect node:

```
java iDetect 5001 127.0.0.1:5002 127.0.0.1:5003 127.0.0.1:5004
127.0.0.1:5005

java iDetect 5002 127.0.0.1:5001 127.0.0.1:5003 127.0.0.1:5004
127.0.0.1:5005

java iDetect 5003 127.0.0.1:5001 127.0.0.1:5002 127.0.0.1:5004
127.0.0.1:5005
```

```
java iDetect 5004 127.0.0.1:5001 127.0.0.1:5002 127.0.0.1:5003
127.0.0.1:5005

java iDetect 5005 127.0.0.1:5001 127.0.0.1:5002 127.0.0.1:5003
127.0.0.1:5004
```
3. Validate that each node is running and monitoring the other hosts

Test - A2
1. Begin with the end test setup for Test - A1
2. Choose node 5
3. Kill node 5's process
4. Verify that nodes 1-4 eventually detect node 5's failure (within 8 seconds) via a similar message:
    a. === !!! HOST 127.0.0.1:5005 has failed !!! ===

Test - A3
1. Begin with the end test setup for Test - A1
2. Choose node 5
3. Kill node 5's process
4. Chose node 4
5. Kill node 4's process
6. Verify that nodes 1-3 eventually detect node 4 & node 5's failure (within 10 seconds) via a similar message:
    a. === !!! HOST 127.0.0.1:5005 has failed !!! ===

Test - A4
1. Begin with end test setup for Test - A1
2. Allow the system to run for 3 minutes
3. Verify that there are no messages such as the following received:
    a. INFO: timeout received while connecting to host

Test - A5
1. Begin with end test setup for Test - A1
2. Chose node 5
3. Kill node 5's process
4. Wait 2 seconds
5. Start node 5's process
    a. NOTE: At 20% packet loss, this is 1 out of every 5 packets being lost; based on the current time values, 2 seconds has been chosen to be the most accurate test scenario; however a tool such as **tc** in linux could be used to generate the same effect.
6. Verify that nodes 1-4 indicated there was packet loss to node 5 but do not indicate that

node 5 has failed with a similar message:

a. === !!! HOST 127.0.0.1:5005 has failed !!! ===

Test - A6

Requirement A6 is not testable; however, UDP was chosen for this implementation because it requires less packets on the wire per transaction than a TCP connection. The current protocol is ASCII string based for easy of review; however, it could be improved to us binary data to reduce the amount of network traffic further. Additionally, the current implementation statically monitors hosts (and for the test setup, monitors ALL hosts). This can be improved by generating an announcement protocol and adjusting the messaging between nodes so that a given node is monitored by no less than X, but no more than X+N nodes for redundancy. These are all methods that can be used to reduce the network overhead; however, were not implemented yet.