Comparative simulation analysis of Bully leader election algorithms

CS 4005-730 Distributed Systems

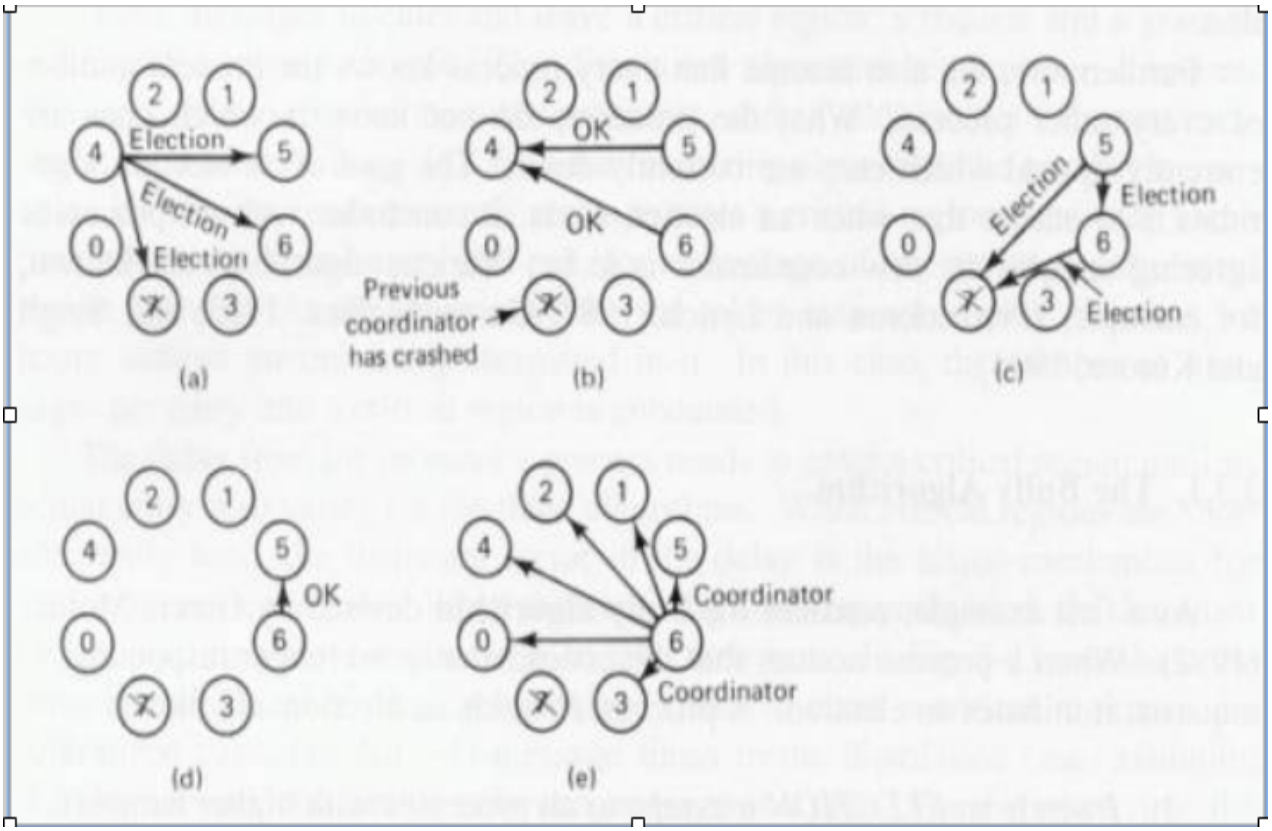
Prof. Alan Kaminsky

Graduate Project: Team Trinity

Members: Pooja Sunder
Parin Maru
Sharif Hdairis

Overview

In any distributed network, there is a leader node (also known as coordinator) that coordinates all the other nodes in the network. This leader node is responsible for proper maintenance and functioning of the network. There are various algorithms that elect a particular node as leader from all the different nodes in the distributed network. These algorithms are responsible for maintaining a leader in the distributed network. There can be various events like leader node going down, previous leader node recovering from a crash, etc. It is the duty of the leader election algorithms to always maintain a leader in the distributed network (by electing a new leader if current leader crashes). Thus the leader election algorithm has to be good enough to perform all its tasks efficiently. Many people in the past have come up with various different leader election algorithms. The most popular one is Bully algorithm. To understand how Bully algorithm works, let us see the following diagram :



Here fig (a) shows that current leader 7 has crashed and node 4 detected the crash, and it started an election by sending election messages to nodes having process id higher than itself ie nodes 5, 6 and 7. Nodes 5 and 6 reply to this election message of 4 by sending an OK message. After that, nodes 5 and 6 themselves start a new election by sending election messages as described above. Finally node 6 does not get any Ok message when it sends election message to nodes above it as there exists no node with process id higher than 6. Thus node 6 elects itself as the leader and sends coordinator messages to all the nodes in the network. Now if node 7 recovers and enters the network again, it will send Query message to nodes having process id higher than itself. Now again, since 7 is the highest process id, node 7 gets no Answer message back. Thus is elects itself as the leader, thereby bullying current leader node 6. This is how original Bully works.

People have designed many different variations of Bully algorithm. In our project, we took two variations of Bully algorithm (optimized Bully and Stable Leader).

The difference between original Bully and optimized Bully is that in optimized Bully algorithm, the node that starts the election itself sends coordinator messages to all the other nodes in the network, informing them of the new leader. As opposed to original Bully where every node, with process id greater than the id of the node detecting the leader failure, will start a new election. Hence the name optimized bully algorithm.
Now in original Bully and optimized Bully algorithms, whenever the leader crashes and then recovers back, it starts a new election. This is not the case with Stable leader algorithm. In Stable leader, the current leader continues to serve as the leader, irrespective of the former leader recovering from a crash. Hence the name Stable leader algorithm.

We took these variations of the Bully algorithm, formulated an hypothesis concerning the number of messages required in the two algorithms to elect a leader, collected data and analyzed it to prove/disprove the hypothesis.

Hypothesis
The total number of elections, and hence messages, used to maintain a leader is fewer for the stable leader algorithm than the optimized Bully algorithm, given a fixed period of simulation time.

Approach
We used event driven simulation to collect data and analyze it. We wrote code to implement the two algorithms that we wanted to compare. We simulated the working of the two algorithms, created cluster for the two algorithms, then injected various node crash/recover events in the two networks. We collected the number of messages that pass when a leader crashes or recovers from a crash. These messages are required to elect a new leader or to maintain a leader in the network. We used t-test to analyze whether the data we collected satisfies our hypothesis model or not.

Analysis of first research paper
The first review paper is “Modified Bully Algorithm using Election Commission”. The authors in this paper talk about the original Bully algorithm. They also state and describe various other algorithms that apply tweakings and modification to the original Bully algorithm. Out of all these algorithms, we

took one variation of Bully algorithm known as optimized Bully algorithm. In our project, we compared this optimized Bully algorithm with the Stable leader algorithm and analyzed the data.

The main issue with the original Bully algorithm is that it causes too many messages. This causes network jitter during elections. Also, since it takes too many messages, it requires a significantly large amount of time in electing a leader. Along with this, network traffic is very large. All these issues are addressed in this paper.

This paper contributes some of the new ideas that increases efficiency while electing leader. These new ideas are :
- ability to trim the number of elections per election.

There is no chaining of elections in the optimized Bully algorithm, thus for electing a leader there is only one election and not many elections.
- ability to abort concurrent elections.

Whenever there are more than one elections going on, the election initiated by the node with the lowest process Id is continued, while all the other elections are aborted. Thus there is only one election that proceeds.

Now lets talk about what we adopted from this paper. We took the entire optimized Bully algorithm. We understood completely how it works, how the leader is elected and maintained in the distributed network. Using discrete event simulation, we implemented this algorithm and compared the number of messages with the number of messages in the Stable leader algorithm.

Analysis of second research paper

The second review paper is “Improved Algorithms for Leader Election in Distributed Systems”. This paper points out two problems with the earlier bully algorithm implementations and presents two separate solutions for the same. The two problems are :
1. Bully algorithm does not account for loss of messages.

2. Bully algorithm does not scale well with the number of concurrent nodes detecting leader failure.

The paper presents a fault tolerant bully algorithm to deal with the first problem.In this modification, when a node gets a co-ordinate message informing it that it has been selected to be the next leader, it itself sends an election message to all processes having higher process id than itself. This is done to verify that no other node with higher process id exists but failed to get detected because of loss of messages. This extra step gives the algorithm its fault tolerant behaviour.

To deal with the second problem, the paper describes leader election algorithm with heap tree. In this algorithm, the nodes in the network are stored as nodes in a heap tree maintained in an array data structure. The tree is a max heap tree, so every node in the tree will have a value less than its parent. The root of the tree is the leader in the network. So whenever a leader fails, the root is deleted. The detecting nodes send election messages to their parents. So if more than one node detects the failure, then when a parent receives more than one election messages from its children, it discards redundant election messages and considers only one election message coming from the node with higher process id. This way the heap tree algorithm deals with the problem of concurrent nodes detecting leader failure. At the end of the paper, performance analysis is done comparing the new algorithms with original Bully algorithm.

Although both the concepts discussed in the paper are very good, we decided not to adopt them in our implementation. We decided to keep the assumption of original Bully intact that communication links are reliable. We also decided against the use of heap tree in our algorithm just for simplification. However we did decide to adopt the analysis done in the paper. The performance analysis compares the number of messages exchanged in original Bully algorithm and modified versions. We are doing a similar analysis in our implementation.

Analysis of third research paper

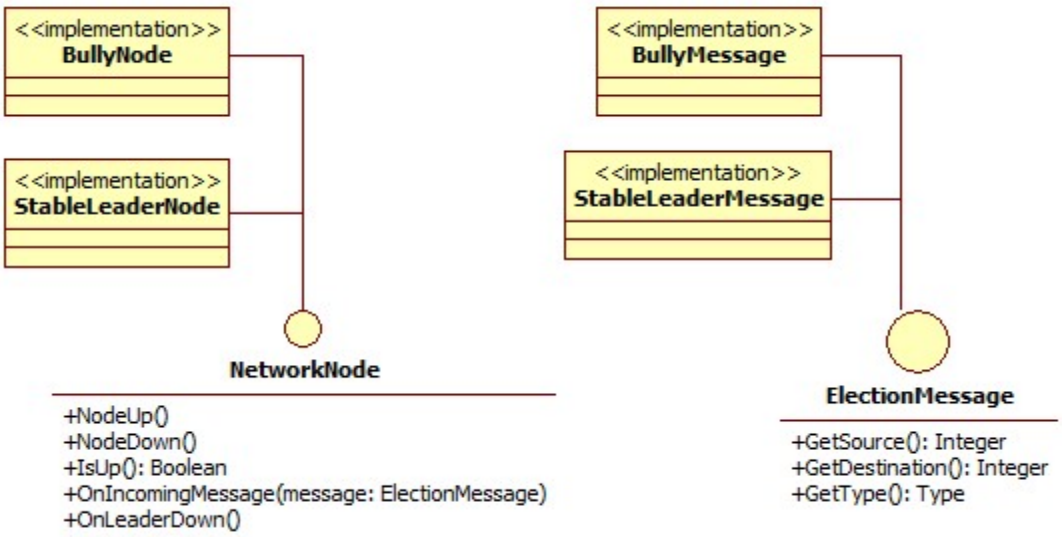
The third paper is “Stable Leader Election”. This paper introduces the concept of stable leader election. The paper first describes what a failure detector is and goes onto explain a failure detector example omega. In omega, a process p trusts a particular process q to be correct at a particular point of time. The failure detector ensures that eventually all the processes in the system trust the same process q. This concept of failure detector can be used for leader election where the process that is trusted by all the processes can be chosen to be the leader in the network. But this leader will be stable. This means a node or process that has been chosen to be the leader will remain a leader as long as it is up and running irrespective of other nodes entering or leaving the network. Thus it may happen that the current leader might have process ids less than other nodes in the network. This is all about the stable leader concept. The authors came up with this concept to curb the number of redundant elections caused due to nodes with higher process id entering the network. Thus the number of messages passed in the stable leader algorithm goes down by a large number. The paper then explains an algorithm to do the same. The algorithm is fault tolerant and communication efficient.

We adopted the stable leader concept in our implementation. We chose to compare the bully algorithm with the stable leader algorithm. However, we kept intact the assumption that communication links are reliable.

Design of our software

In our software we have following interfaces :

- NetworkNode interface
- ElectionMessage interface

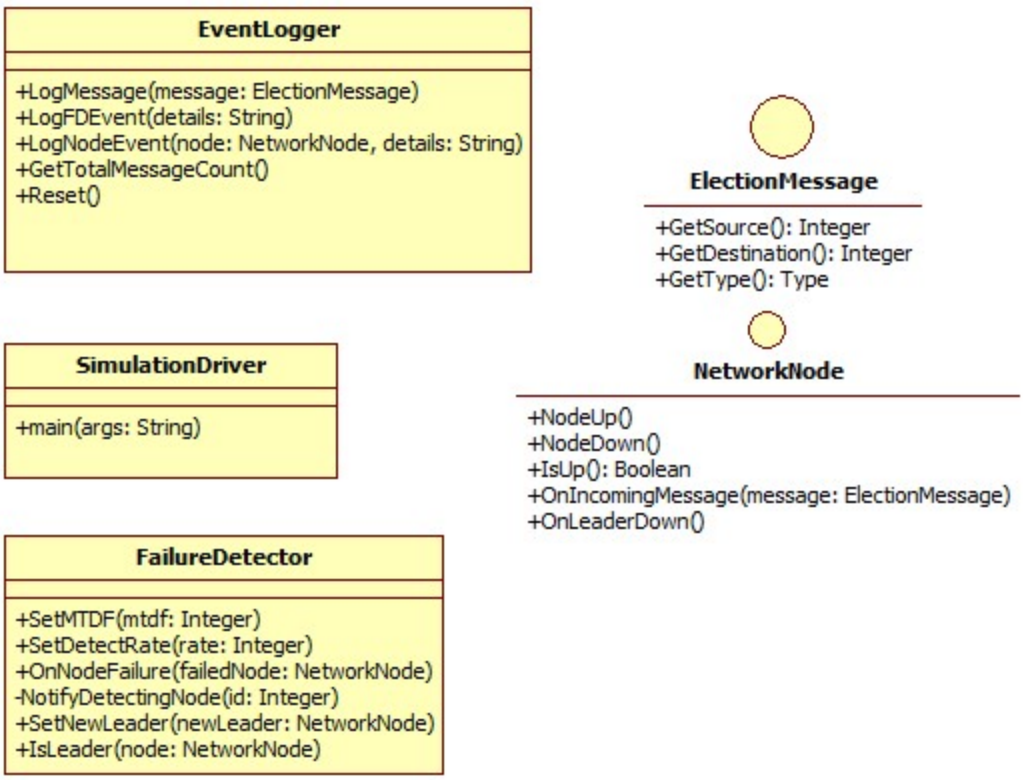


StableNode class and BullyNode class implements the NetworkNode interface. StableNode encapsulates nodes running stable leader election algorithm, while BullyNode encapsulates nodes running bully election algorithm.

StableMessage class and BullyMessage class implements the ElectionMessage interface. StableMessage encapsulates the messages that are passed when stable leader election algorithm is run, while BullyMessage encapsulates the messages that are passed when bully leader election algorithm is run.

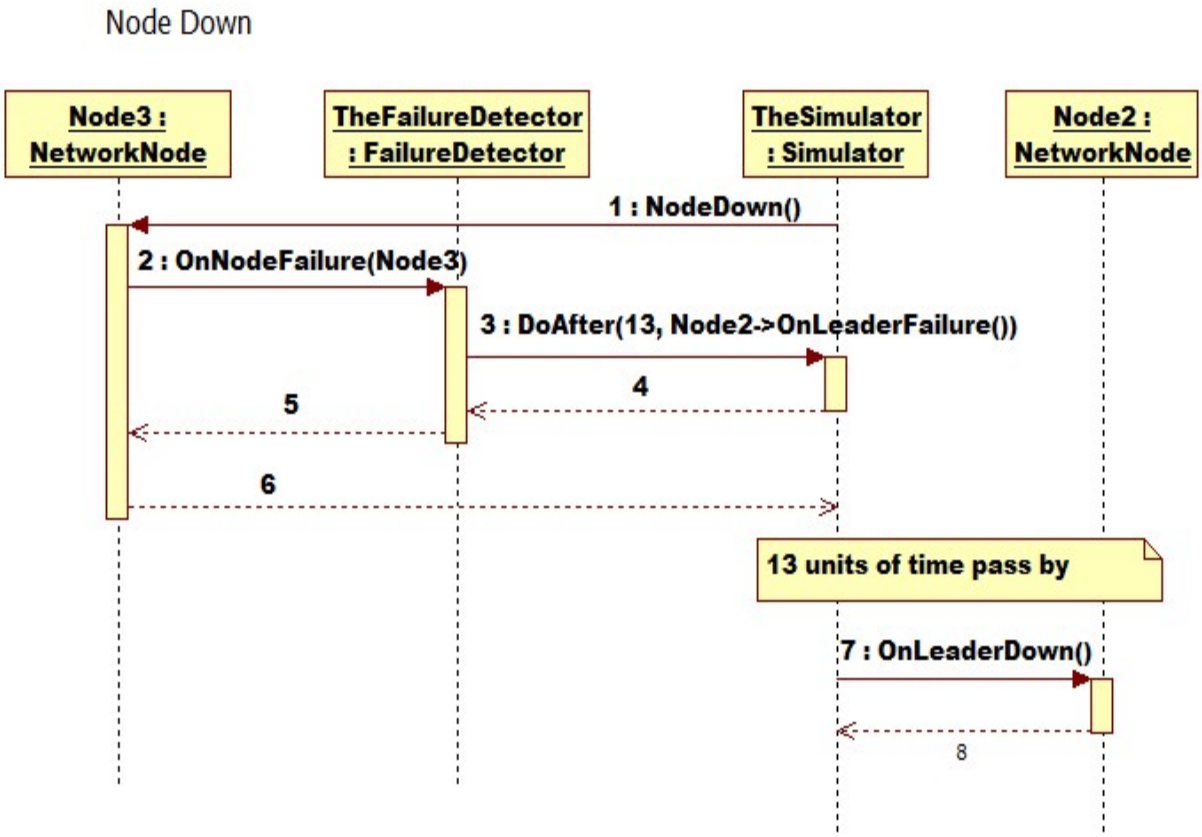
Apart from this, we have a FailureDetector class. This class does all the detection work, i.e. which node fails, which node detects the failure when a leader fails, etc. This class also keeps the track of the current leader in the network.

We also have an EventLogger class. This class is our log. It logs all the events occurring in the system. All the messages, that are passed in the network to elect, maintain and query the leader, are logged by this class. We use this class to display our results.

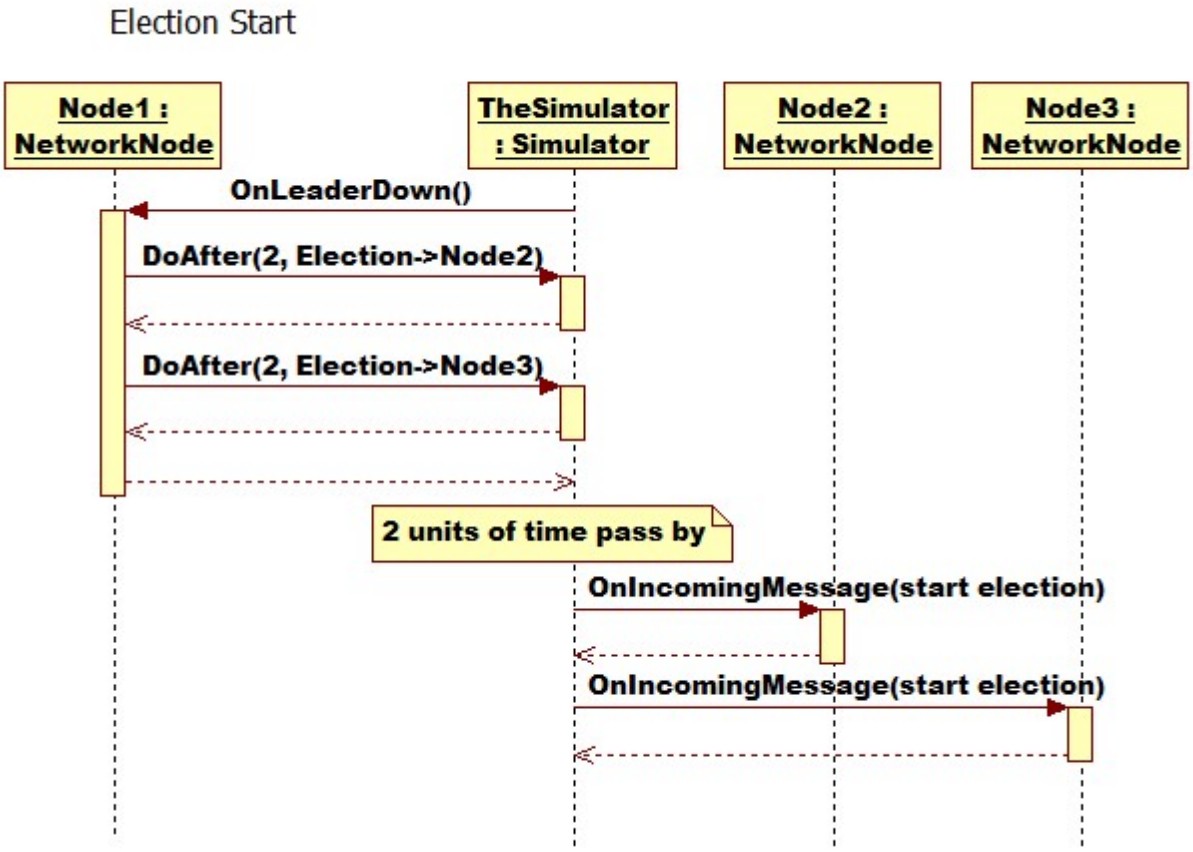


We have two driver classes, DemoSimDriver and SimulatorDriver. These two classes have the man program that runs the entire software.

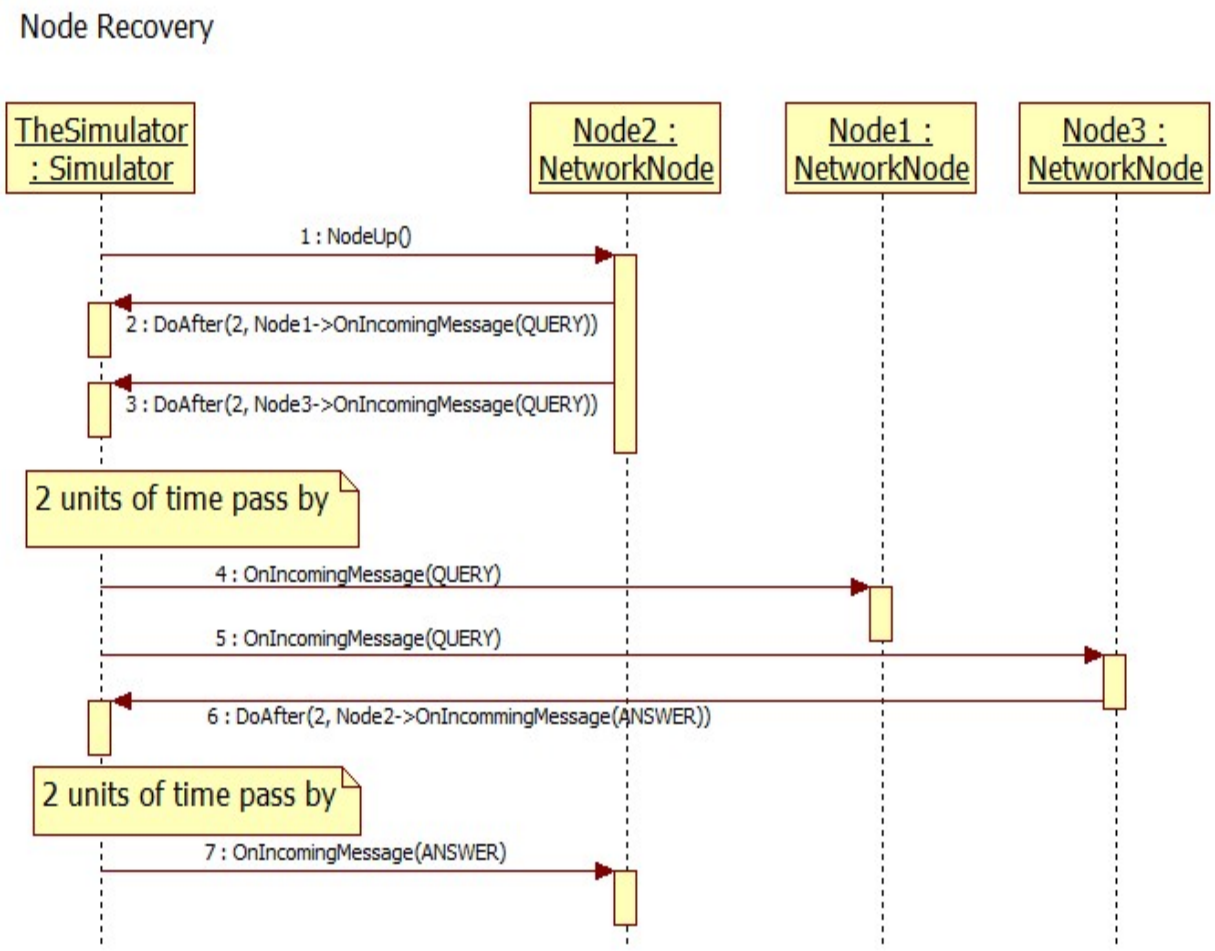
The following sequence diagrams help put these classes together, the first sequence represents the sequence that occurs as the result of a node crashing and becoming non-operational.



When a node is notified that the current leader is inoperable, the node starts an election, the startup sequence looks like this.



After the nodes go through an election, a new leader is selected. At some point, the failed node will be serviced and return to the network. This node-up event, starts a sequence as depicted in the next diagram.



Developer's manual

The project's programming language of choice is Java, version 1.5. The project requires the Parallel Java Library available from the RIT Computer Science department.

To get started, extract the contents of the tar file into a working directory, there should be several *.java source files and a single input *.txt file.

Once the java CLASSPATH environment variable is setup to point to an available copy of pj.jar, the project is ready to be compiled.

To compile, execute:

javac *.java

The main method is located in SimulationDriver.java, to run the program with the

sample input file, execute:
java SimulationDriver.java SimulationsRun.txt

when the simulation is done, a graph similar to the one presented in this paper will show on the screen.

User's manual

The main class for the simulation program is in SimulationDriver.java. This program takes just one argument: a filename containing simulation parameters. A sample input file is provided that contains the parameters used in this paper's simulation. (see SimulationRun.txt).
The input file consists of several lines, each of which contains parameters for a single simulation run. A detailed description of the fields follows.
<SimType (Bully/Stable)> - the type of simulation to run, Bully or Stable
<Node Count> - the number of nodes in the network
<MTTF (MTBF)> - the average time between node failures in the network
<MTTR> - the average time a node spends in the 'crashed' state.
<Mean Detection Rate (%)> - the percentage of nodes that will detect that the leader has failed.
<Mean Time To Detect Failure> - the average amount of time it will take for the nodes to detect leader failure.

The main program runs the simulation once for each configuration. The only limitation right now is that for each Bully simulation, a Stable leader simulation needs to be run with the same network size, this is so that the t-tests executed at the end for each node size can be computed.

The command to run the program is as follows:
java SimulationDriver <input file>

Measurement data

For 50 different random number seeds, 5000 node failures :

Input table :

SimType (Bully/Stable)	Node Count	MTTF (MTBF)	MTTR	Mean Detection Rate (%)	Mean Time To Detect Failure
Bully	5	100	20	2	3
Bully	10	100	20	2	3
Bully	15	100	20	2	3
Bully	20	100	20	2	3
Bully	30	100	20	2	3
Bully	40	100	20	2	3
Bully	50	100	20	2	3
Bully	75	100	20	2	3
Bully	100	100	20	2	3
Bully	120	100	20	2	3
Stable	5	100	20	2	3
Stable	15	100	20	2	3
Stable	20	100	20	2	3
Stable	30	100	20	2	3
Stable	40	100	20	2	3
Stable	50	100	20	2	3
Stable	75	100	20	2	3
Stable	100	100	20	2	3
Stable	120	100	20	2	3

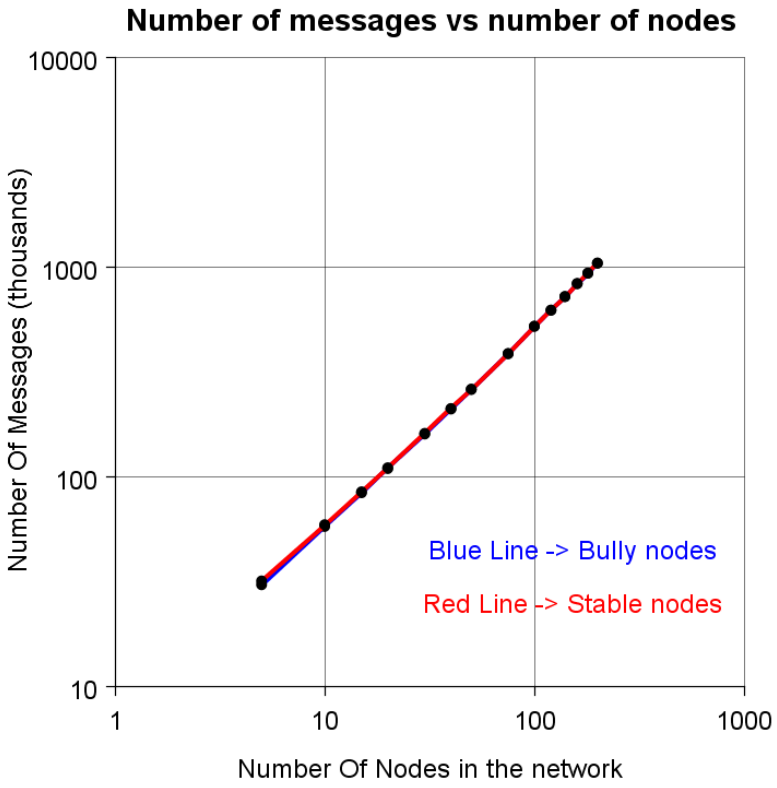
Output table :

Node count	Bully Mean	Bully Std deviation	Stable Mean	Stable Std deviation	T value	P value
5	30432.86	259.19	31728.41	173.52	-29.075	0.000

10	57650.82	484.10	58835.71	441.83	-12.655	0.000
15	83489.57	871.06	84433.96	505.24	-6.565	0.000
20	108997.92	1086.77	109860.59	596.04	-4.872	0.000
30	159123.18	1452.36	160256.90	911.41	-4.628	0.000
40	209363.22	1809.86	210615.02	1033.56	-4.204	0.000
50	259377.20	1960.12	260676.76	1074.44	-4.070	0.000
75	384544.43	2952.47	385651.78	1189.83	-2.435	0.018
100	519245.92	3935.00	520310.37	2168.06	-1.658	0.101
120	619538.39	4841.46	620362.65	2504.41	-1.059	0.293

Data Analysis

The data clearly shows that there is a significant difference in the average number of messages passed in Bully election algorithm and Stable leader election algorithm. We performed t-test to see whether the difference is significant enough. The t-value and p-value collected from the T test confirms that the the difference is significant enough. However, we found that the number of messages passed in Bully algorithm is less than that in Stable leader algorithm. This is in contrast to our hypothesis. Our hypothesis stated that the number of messages in Stable leader algorithm will be less than bully algorithm. Thus the data that we collected disproved the hypothesis. Also the difference was found to be significant as proved by t-test.



We formulated the hypothesis that Stable leader election algorithm will result in lesser number of messages being passed as compared to Bully algorithm. This was because it reduced the number of elections as the leader remained stable as long as the leader is up irrespective of other nodes entering or leaving the network. However we noticed that whenever a node with highest process id recovers from a failure, it sends out n-1 coordinate messages (to all the nodes other than itself) in Bully algorithm. However, in stable leader, in the same scenario, the leader sends n-1 query messages (to all the nodes other than itself) and gets 1 answer message (from the current leader) which sums up to n messages. This is more than the number of messages in Bully algorithm. Similarly, when a process with a process id lower than current leader recovers from a failure, it sends out approximately (n-1)/2 query messages (to all nodes with process id higher than itself) in Bully. It gets back (n-1)/2 replies. This sums up to n-1 messages. In Stable leader, in such a scenario, the recovered node sends out n-1 query messages (to all the nodes other than itself) and gets back 1 reply (from the current leader), which sums to n messages. Again the number of messages in stable leader is

more. In the long run, this difference between the number of messages between the two algorithm increases by a significant number. Having lesser number of elections in the stable leader algorithm does not affect the total number of messages in the long run. Hence, the hypothesis was proved false.

Future work

Currently we assume that the communication links are reliable. So our future work would include making the algorithm fault tolerant so as to deal with loss of messages. This would include adopting the modifications discussed in [2].
Also handling the number of elections because of concurrent nodes detecting the failure is another area where we would like to improve on. Again for this, we would like to adopt the modifications described in [2]. Use of heap tree leader election in the algorithm would solve the concurrent election problem.

Lessons learnt

We learnt a significant amount of stuff from this project. Before this project, we had never done anything like formulating a hypothesis and proving or disproving it. But this project taught us how to do it. Also, we had never done discrete event simulation. In this project, to provide the different behavior and functionality in the network, we learnt and used discrete event simulation. Finally to analyze our data and to prove mathematically whether our collected data satisfies our hypothesis model or not, we used t-test. We did not know what t-test actually does and how to use it, thus we learnt it from this project.

Each individual's contribution

The project can be broken down into different components :

1. Formulation of hypothesis
2. Software design of simulation program
3. Implementation of simulation program
4. Collection of data
5. Documentation in the form of report and presentations

All the team members collaboratively contributed to each of the project's software components and deliverables. The deliverables include four team presentations and this final report.

References

[1] Muhammad Mahbubur Rahman, Afroza Nahar, "Modified Bully Algorithm using Election Commission",
MASAUM Journal of Computing(MJC),Vol.1 No.3,pp.439-446,October 2009, ISSN 2076-0833

[2] M. EffatParvar, N. Yazdani, M. EffatParvar, A. Dadlani, and A. Khonsari, "Improved Algorithms for Leader Election in Distributed Systems", The 2nd International Conference on Computer Engineering and Technology (ICCET 2010), Vol. 2, April, 2010, pp. 6-10.

[3] Tanenbaum, A.S., and Steen M.V.: "Distributed Systems Principles and Paradigms," Prentice-Hall International, Inc, 2002. Sample Chapter

[4]M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, S. Toueg. "Stable Leader Election", *International Symposium on Distributed Computing,* October 2001.

[5] Garcia-Molina, H., "Elections in Distributed Computing System," IEEE Transaction Computers, Vol.C-1,pp.48-59,Jan.1982.