**Available Online at www.ijcsmc.com**

**International Journal of Computer Science and Mobile Computing**

**A Monthly Journal of Computer Science and Information Technology**

**ISSN 2320–088X**
**IMPACT FACTOR: 5.258**

***IJCSMC, Vol. 5, Issue. 3, March 2016, pg.782 – 788***

# DESIGN and IMPLEMENTATION of LEADER ELECTION SYSTEM

**Puja Koramkar[1], Priyanka Khobragade[2], Khushali Raut[3], Rangeeta Pancheshwar[4], Prof. Charan Pote[5]**

Computer Technology, PCE Nagpur, India[12345]

pujakoramkar@gmail.com [1], prkhobragade10@gmail.com [2], khushaliraut3@gmail.com [3]
rangeetapancheshwar10@gmail.com [4], scrpote@gmail.com [5]

***Abstract— Leader election is an important problem in distributed computing. Garcia-Molina's Bully Algorithm is a classic solution to leader election in synchronous systems with crash failures. This paper shows that the Bully Algorithm can be easily adapted for use in asynchronous systems. First, we re-write the Bully Algorithm to use a failure detector, instead of explicit time-outs; this yields a modular solution to leader election in synchronous systems. Second, we show that minor modifications to that algorithm yield a simple and efficient solution to leader election in asynchronous systems with crash failures. Bully algorithm in which new leader will be elected to avoid the loss of data or information. We point out a in Garcia-Molina's specification of leader election in asynchronous systems, propose a revised specification, and show that the modified Bully Algorithm satisfies this speciation. In leader election algorithm the higher priority node will be selected that do the election with the lower priority node.***

***We propose a new specification for leader election in asynchronous systems, which requires that the number of groups not exceed the number of fully-connected components needed to cover the reach ability graph. This specification allows nodes that cannot directly communicate to be in the same group, but never forces this. Finally, we adapt the Bully Algorithm for use in asynchronous systems. The effect of asynchrony is reflected by considering failure detectors that sometimes raise false alarms" (i.e., report failure of an operational node). Only minor modifications are needed in the other parts of the algorithm to deal with these false alarms. We show that the modified algorithm, which we call the Asynchronous Bully Algorithm, satisfies our specification.***

***Keywords— Bully Algorithm, Leader Election, Crash failure***

## I. INTRODUCTION

In distributed systems, shared memory or message passing technique are used to nodes communicate with each other. In distributed system, for executing the distributed task the nodes required coordination. In distributed system, every node has to communicate with the rest of the nodes in the network to make an proper decision.

In system during the decision of coordinator, not all nodes make the same decision, thus the communication between nodes time-consuming and decision-making process. Coordination among nodes becomes difficult when consistency is needed among

all nodes. In Centralized system to reduce the complexity of decision making nodes can be selected from the group of available nodes.

Many distributed algorithms require one node to act as coordinator, initiator, or otherwise perform some special role. Leader election is a technique that can be used to break the symmetry of distributed systems. In order to determine a central controlling node in a distributed system, a node is elected from the group of nodes as the leader to serve as the centralized controller for that decentralized system. The purpose of leader election is to choose a node that will coordinate activities of the system. In any leader election algorithm, a leader is usually chosen based on some criterion such as choosing the node with the largest identifier. Once the leader is elected, the nodes reach a certain state known as terminated state. In leader election algorithms, such states are partitioned into elected states and non-elected states. When a node enters either state, it always remain in that state .The bully algorithm is a method in distributed computing for dynamically selecting a coordinator by process ID number.

Assumptions :

1. The system is synchronous and uses timeout for Identifying process failure

Message Types :

2. Election Message : Sent to announce the election
3. Answer Message : Reply to the election message
4. Coordinator message: sent to announce the identity elected
5. process

Compare with Ring algorithm :

1. Assumes that system is synchronous
2. Uses timeout to detect process failure/crash
3. Each processor knows which processor has the Higher identifier
4. Number and communicates with that.

When a process P determines that the current coordinator is down because of message timeouts or failure of the coordinator to initiate a handshake, it performs the following sequence of actions:

1. P broadcasts an election message (inquiry) to all other processes with higher process IDs.
2. If P hears from no process with a higher process ID than it, it wins the election and broadcasts victory.
3. If P hears from a process with a higher ID, P waits a certain amount of time for that process to broadcast itself as the leader. If it does not receive this message in time, it re-broadcasts the election message.
4. If P gets an election message (inquiry) from another process with a lower ID it sends an "I am alive" message back and starts new elections.

Note that if P receives a victory message from a process with a lower ID number, it immediately initiates a new election. This is how the algorithm gets its name - a process with a higher ID number will bully a lower ID process out of the coordinator position as soon as it comes online.

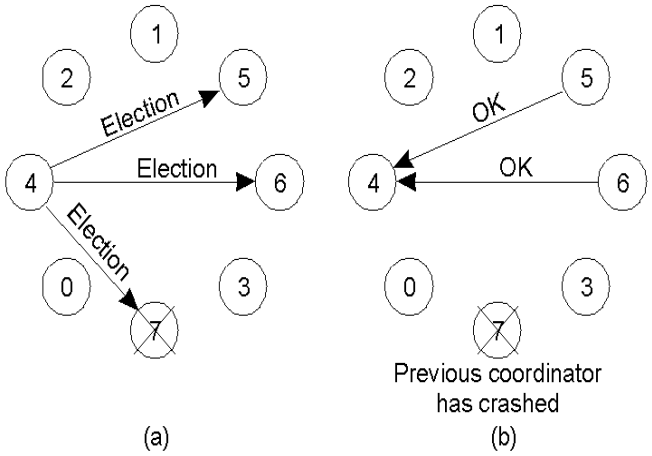
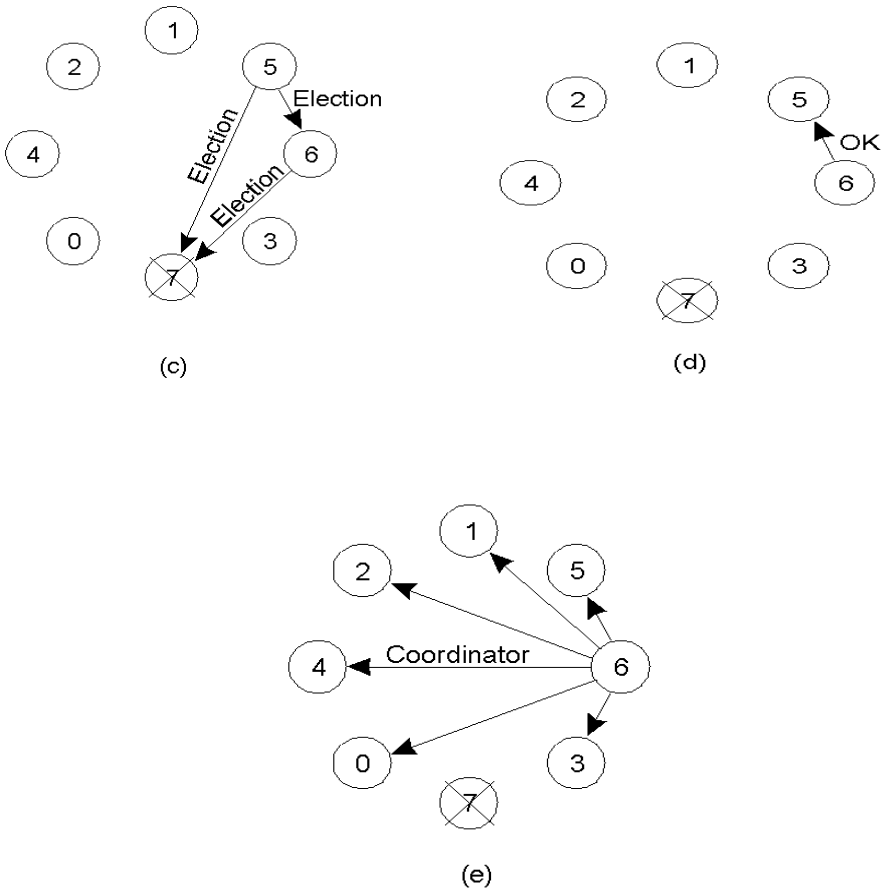



Fig.1.Bully Algorithm: (a) process 4 detects coordinator is failed and announce an election, (b) process 5 and 6 respond to 4 to stop election




(c) each of 5 and 6 announce election now, (d) process 6 responds to 5 to stop election, (e) process 6 winds and announce to all.

## II. Literature Survey

Many algorithms for leader election, clustering, and hierarchy construction have been proposed. Leader election algorithms for mobile ad hoc networks have been proposed in [6, 10, 7]. But the algorithms in [6, 10] do not consider security issues and are not extrema-finding algorithms.

The leader election algorithms proposed in [7], although extrema-finding, are not suitable to the applications we discussed earlier because they require nodes to move and exchange information with other nodes in order to elect a leader. The clustering/hierarchy construction proposals in [9, 11,12, 13] use a bottom-up approach that we have adapted for SEFA and SPLEA. The algorithm in [9] constructs a hierarchy in a sensor network of low power devices by promoting nodes with higher energy up the hierarchy. A Landmark hierarchy is constructed in a bottom-up fashion in [11] for routing purposes. In [12], a structure called an *AC Hierarchy* is constructed that logically organizes processors into various levels. In [13], sensors organize themselves into clusters and cluster heads are elected based on localized coordination and control. In addition, cluster heads are randomly rotated, with each node serving as a head only for a fixed duration. In order to minimize energy spent in communicating with the remote base station, cluster members communicate to the base station through their cluster heads.

As we will see, two of our secure leader election algorithms, SEFA and SPLEA, use the idea of bottom-up hierarchy construction, and are similar to the clustering proposals in that respect. However, none of the algorithms described above consider the notion of *secure* leader election.

Indeed, it is not clear whether security can easily be built into their protocols. In this paper, we develop two secure leader election algorithms, called SEFA and SPLEA, and prove that these algorithms are cheat-proof. However, these algorithms require lock-step execution of the election algorithm and that nodes remain static for the duration of election. Both these assumptions are relaxed in our Asynchronous Extrema Finding Algorithm (AEFA), which is a non-secure election algorithm. AEFA can tolerate multiple, concurrent topological changes during the process of election itself. It guarantees that when topological changes stop for a sufficiently long period of time, the node with the extreme is elected as the leader.

Leader election is a fundamental problem and has been studied in various models. Leader (coordinator) election is an influential problem in distributed computing systems since performance of all nodes in system depending on leader. Depending on a network topology, many algorithms have been presented for electing leader in distributed systems Such as Gerard Le Lann introduced AUDITOR is one of the first protocols to coordinate the leader election problem.

There are two basic strategies in the leader election algorithms. One of strategy is to make the system, temporarily halt normal operation and take a time to reorganize the system. During the reorganization period, the components of the system can be evaluated and compete with each other. The other strategy is to let the system contain software which can operate continuously and correctly when failure occurs and can recover the system.

In this protocol, each node contains an "auditor" and there is an ordered ranking of auditors. The highest-ranking auditor select 'audit coordinator' which responsible for detection of failures. A disadvantage of this approach is that, if multiple node failures (or partitioning), it may take many sequential executions of the promotion protocol before a candidate is successful in reaching coordinator rank. As Leader election is an important problem in distributed computing systems. In this problem, when the leader is crashed, other nodes must elect another leader.

This paper shows that the Bully Algorithm can be easily adapted for use in asynchronous systems. The resulting algorithm, which we call the Asynchronous Bully efficient: in common cases, it uses half as many messages as the Invitation Algorithm. We obtain the Asynchronous Bully Algorithm in two steps. First, we re-write the Bully Algorithm to use a failure detector [CT94], instead of explicit time-outs. A failure detector is a module that reports crashes of other nodes. Re-writing the Bully Algorithm in this way. Here we used the bully algorithm with the use of socket theory which is more suitably used for networking.

Garcia-Molina's Bully Algorithm is a classic solution to cope with this problem and one of the most applicable elections algorithms. In a classic paper, Garcia-Molina specifies the leader election problem for synchronous and asynchronous distributed systems with crash failures and gives an elegant algorithm for each type of system; this algorithms are called the Bully Algorithm and Invitation Algorithm, respectively [GM82]. Leader election is an important problem in fault-tolerant distributed computing. It is closely related to the primary-backup approach (since choosing a primary replica is like electing efficient form of passive replication. It is also closely related to group communication [Pow96], which (among other uses) provides a powerful basis for implementing active replication. For example, the group communication system in Amoeba [KT91, KT92] uses Garcia-Molina's figure a group after crashes. As another example, the group membership algorithms in

Horus [FvR95, vRBM96] and Ensemble [Hay97] can be seen as a combination of Garcia-Molina's Bully Algorithm (for handling crashes) and Invitation Algorithm (for merging partitions of a group).

## III. Proposed Work

### 1. Route Discovery

In the chosen subsystem, there are three protocols. The first protocol involves the communication between the console and the election members. The execution starts with the console sending a start command to every member. Each member, upon reception of this command, initializes all its internal components. If the console then sends a message to simulate a failure, the member executes the reversed sequence of actions, sending commands to every component to stop. At this point, a recovery message causes the member to restart its components, whereas a command to close terminates the execution.

The first criterion in wireless medium is to discover the available routes and establish them before transmitting. To understand this better let us look at the example below. The below architecture consists of 11 nodes in which two being source and destination others will be used for data transmission. The selection of path for data transmission is done based on the availability of the nodes in the region using the ad-hoc on demand distance vector routing algorithm. By using the Ad hoc on Demand Distance Vector routing protocol, the routes are created on demand, i.e. only when a route is needed for which there is no "fresh" record in the routing table. In order to facilitate determination of the freshness of routing information, AODV maintains the time since when an entry has been last utilized. A routing table entry is "expired" after a certain predetermined threshold of time. Consider all the nodes to be in the position. Now the shortest path is to be determined by implementing the Ad hoc on Demand Distance Vector routing protocol in the wireless simulation environment for periodically sending the messages to the neighbors and the shortest path.

In the MANET (Mobile ADHOC Network), the nodes are prone to undergo change in their positions. Hence the source should be continuously tracking their positions. By implementing the AODV protocol in the simulation scenario it transmits the first part of the video through the below shown path. After few seconds the nodes move to new positions.

Fig 2; Route Discovery in Network

### 2. Route Maintenance

The next step is the maintenance of these routes which is equally important. The source has to continuously monitor the position of the nodes to make sure the data is being carried through the path to the destination without loss. In any case, if the position of the nodes change and the source doesn't make a note of it then the packets will be lost and eventually have to be resent.

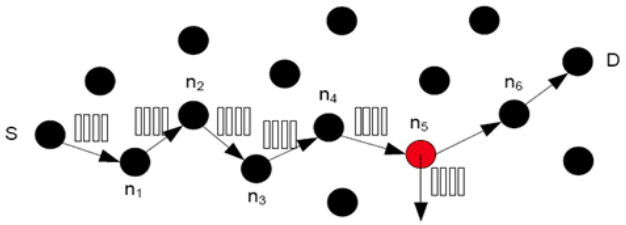



Fig3: Node Failure Detection

The modified proposed algorithm
Threshold = 50%; success = 0; cutoff = 10%
A := S;
Repeat
If g(A) >= threshold then
B := A;
Let A be neighbor of B that minimizes
pc(B,A) = power-cost(B,A) + v(s)f`(A);
Send message to A;
success = 1;
Until A = D (* Destination reached *)
or if success <> 1 then
if threshold > cutoff then
threshold = threshold /2;
or A = B ( * Delivery failed *);

**3. Data Transmission**

The path selection, maintenance and data transmission are consecutive process which happen in split seconds in real-time transmission. Hence the paths allocated priory is used for data transmission. The first path allocated previously is now used for data transmission. The data is transferred through the highlighted path. The second path selected is now used for data transmission. The data is transferred through the highlighted path. The third path selected is used for data transmission. The data is transferred through the highlighted path.

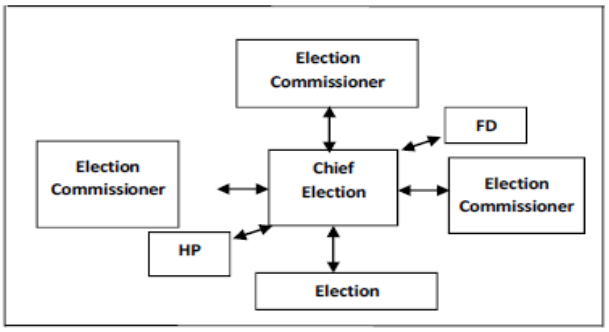



Fig4..Data Flow of the Proposed System

## IV. CONCULSION

We tried to oreduced limitations of original election algorithm and implement the bully algorithm . In this project, we modified bully algorithm using a new concept Election Commission. Our discussion section prove that our algorithm is more efficient than election algorithm and bully algorithm in respect of message passing, redundant election and network traffic. The case studies confirmed the application and importance of the idea of context when extracting models. Moreover, the possibility of creating user-defined actions permitted us a high degree of customization of the models, so that relevant parts of the code without method calls could also be represented in the model. User-defined attributes extended this customization to the possible expansion of the system state by defining expressions over existing attributes.

This was particularly useful in the leader election application, where we would like to know the number of members involved in the execution but we only had the attribute which contained the data structure storing the list of members. An expression over the contents of this structure (method size) provided us with the necessary information.

It is important to mention that we assume that the systems from which we extract models are deterministic. This guarantees that every execution using a certain sequence of inputs will result in the generation of the same trace. In non-deterministic systems,

one can obtain different outcomes for the same sequence of inputs. This means that the same test case would have to be executed several times in order to produce traces for each possible output. Even so, guaranteeing coverage of all possibilities could make it impracticable. Moreover, checking whether an error trace is real or not would also require multiple execution

**ACKNOWLEDGEMENT**

We thank to our project guide prof. charan pote sir for so much effort on building Design and Implementation of Leader election.

## REFERENCES

[1] E.W. Dijkstra and C.S. Scholten. Termination detection for diffusing computations. In Information Processing Letters, vol. 11, no. 1,pp. 1-4, August 1980.

[2] E. Gafni and D. Bertsekas. Distributed Algorithms for generatingloop-free routes in networks with frequently changing topology. In IEEE Transactions on Communications,C-29(1):11-18, 1981

[3] N. Lynch. Distributed Algorithms. 1996, Morgan Kaufmann Publishers, Inc.

[4] C. Wong, M. Gouda and S. Lam. Secure Group Communication using Key Graphs. In Proceedings of ACM SIGCOMM'98, September 1998.

[5] P C. Wong, M. Gouda and S. Lam. Secure Group Communication using Key Graphs. In Proceedings of ACM SIGCOMM '98, September 1998.

[6] D. Estrin, R. Govindan, J. Heidemann and S. Kumar. Next Century Challenges : Scalable Coordination in Sensor Networks. In Proceedings of ACM MobiComm,August 1999.

[7] E. Royer and C. Perkins. Multicast Operations of the AdHoc On-Demand Distance Vector Routing Protocol. In Proceedings of Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM), pages 207-218, August 15-20, 1999

[8] G. Tel. Introduction to Distributed Algorithms. Second Edition, 2000, Cambridge University Press.

[9] N. Malpani, J. Welch and N. Vaidya. Leader Election Algorithms for Mobile Ad Hoc Networks. In Fourth International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, Boston, MA, August 2000.

[10] B. DeCleene et al. Secure Group Communication for Wireless Networks. In Proceedings of MILCOM 2001, VA, October 2001.