A Self-stabilizing Clustering Algorithm with Fault-containment Feature for Wireless Sensor Networks

Amirreza Ramtin, Vesal Hakami, Mehdi Dehghan
Department of Computer and IT Engineering
Amirkabir University of Technology
Tehran, Iran
{a_ramtin, vhakami, dehghan}@aut.ac.ir

***Abstract*—in this paper, the clustering of wireless sensor network has been equaled to constructing maximal independent set in graph theory, and a clustering algorithm with "self-stabilizing" and "fault containment" properties, which is considered as a critical feature in the issue of fault tolerance for distributed systems, is proposed. Existing comparable methods include no fault containment and their design is based on assuming a "centralized scheduler". The proposed algorithm is recovered from single fault configurations by space and time complexity of *O*(1), and work under policy of unfair distributed scheduler which has the maximum matching with operation environment of sensor networks. The "self-stabilization" and "fault containment" properties of algorithm will be proved by formal reasoning; Simulation results also show that regardless of the number and concentration of nods, the suggested method in addition to quick recovering against small-scale faults, will improve convergence time compared to previous methods. The creation of efficient clustering construction, reducing the numbers of updating messages and stabilizing by minimum change in the clustering topology structure, are other advantages of this algorithm.**

Keywords—self-stabilizing, clustering, wireless sensor networks (WSN), energy saving, fault containment

I. INTRODUCTION

Flat structures in WSN with large number of nodes, is not scalable and efficient in terms of energy consumption. In this regard, network clustering is considered as a successful and common solution to provide the ability of self-organizing and operation of hierarchical routing.

Generally, in the lack of constant substructure and considering the dynamicity of sensor network and the necessity of multi-hop communications, clustering methods having the better stabilizing in system and also having the ability of re-configuration without external interfere, will be more favorable.

Clustering with self-stabilizing feature [1], in addition to needing no initial configuration, provide the possibility of automatic recovering from transient faults due to environment changes, sensors failure or changing in their internal situation, rupture of communication structure, and finally asynchronous change in the configuration. Self-stabilizing is an approach to create the fault tolerance ability, especially in the dynamic distributed systems that has attracted special attention in sensor networks related studies recently.

Based on the definition, a self-stabilizing system guarantees that after finite steps, regardless of initial state, will automatically converge to a credible state without external interference. After any transient faults, the system is corrected by relying on local knowledge and needless to global information. These are considerable features of self-stabilizing systems [1].

Although the clustering with self-stabilizing feature in the sensor networks has been partly studied, the ability of stabilizing with fault containment [3], which has a more correspondence with these types of networks, has been rarely studied. In order to improve the performance of fault tolerance methods, ensuring faster recovery than all single fault configurations without effecting on final recovery ability of the system from more widespread faults, is of a considerable importance. Same policy of self-stabilizing clustering algorithm against many different types of faults, have unfavorable consequences such as out of service long time period, resulting in more energy wasting to return to stabilizing state or wider change in clustering topology structure.

In this work, the purpose is suggesting an algorithm with two features of self-stabilizing and fault containment for clustering of sensor networks with constructing maximal independent in the network graph. The proposed algorithm under policy of distributed scheduler prevents the expanding faults in the network at small-scale fault occurrence situation. To this purpose, fault containment theory [3], [9] that is covering approach in the fault containment discussion combined with self-stabilizing for space and time field, is the effect of used small-scale faults. This results in the reduction of numbers of updating and state transition and consequently reducing the numbers of broadcastings, power saving and increasing the network lifetime as well. Besides, by preventing the fault spreading in the network, the stabilizing time will be reduced. Using this algorithm, availability of system will be improved via removing intermediate state and reducing in stabilization time. Reducing the numbers of state transition prevents network topology tumbling and nodes might be remained in their clusters. Codifying stabilization rules in design of proposed algorithm is based on fault containment and creating more efficient clustering topology compared to previous method. Actually, the nature of implement rules for nodes is to reduce the numbers of useless cluster-heads in the resulted topology.

978-1-4799-5359-2/14/$31.00 ©2014 IEEE

Subjects of this paper: in part two, we will have a quick look on the basic concepts and reviewing previous researches. In part three, the proposed algorithm will be explained and the related theorems will be surveyed. In part four, the proposed algorithm will be evaluated and compared based on the results of simulation tests. In last part, we have the conclusion and the summery of study.

II. Theoretical History and Related Works

Prerequisite and enough condition for beneficiary from self-stabilizing ability is based on two features: ensuring the system convergence to global legitimate configuration after starting from any arbitrary situation (convergence), and ensuring the establishment of system legitimate configuration as long as the fault doesn't occur (closure) [1], [8]. Based on these two features, a self-stabilizing system does not need initialization and until no more fault condition is occurred, restores from one or more transient faults, automatically and without any external interference, and once again converge to the legitimate configuration (figure 1).

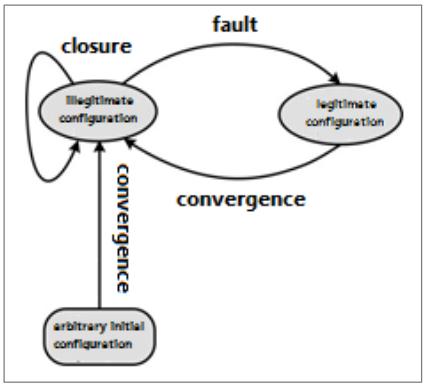

Fig. 1. State diagram of a self-stabilizing system

Many researches has been proposed in design of self-stabilizing algorithm for constructing independent and dominating sets of graph theory field [5]. However, most of these researches are designed base on reliable communication models (such as shared memory) and assuming the centralized scheduling policy [6], which actually impose limited condition on performance of sensor networks. Furthermore, some of these algorithms are not designed for special applications (such as clustering) and sometimes their topological forms are not suitable for sensor networks [7]. In related works with the idea of present study, the only algorithm, which has linear time complexity and by assuming distributed scheduling policy, devoted to problem solving, is the proposed method in [4]. Self-stabilizing ability in [4] has been done through embedding intermediate state in algorithm performance. However, this method is assumed as a fault resource and it could lead to a reduction in the availability of system.

The other disadvantage of all existing algorithms is lack of distinction in faults management based on their propagation. Occurring just one fault in the legitimate configuration could lead to algorithm response in term of multiple overall updates and thus the fault could affect large scale of the network until stabilization. Each updating in wireless networks is actually equivalent to a broadcast to the neighbor nodes. Considering the limitation of node energy, leads to reduce their lifetime and efficiency of network. The other problem is long time for stabilization, which the system does not work based on desired design and it is in the unreliable state until reaching to a stable state.

III. Proposed Algorithm

In this part, a high-level description of the clustering algorithm based on constructing maximal independent set, which has both properties of self-stabilizing and fault containment (MIS*fc*), will be proposed.

A. Description on design and functionality of algorithm

If it is assumed that single fault state with fault in node *v*, has been resulted from a change in one variable of node *v*. It could be simply shown that two conditions is satisfied in this state: (1) one of laws is active in node *v*. (2) It is possible to reach the legitimate configuration with implementation of the law in node *v* (in some cases, rule implementation in neighborhood nodes could have the same result). The purpose is identification and elimination of 1-fault states with rule implementation in that faulting nodes, and prevention from fault propagation with undesirable implementation in neighborhood nodes of *v*, *N(v)*. For simplicity in understanding the proposed algorithm, first, we define some propositions, which actually are precondition for running the operation of state transition in nodes, and numbers of units for better readability of algorithm pseudo code (figure 2).

$$\boldsymbol{inNeighbor}(\boldsymbol{v}) \equiv \exists w \in N(v): w.state = IN$$

$$\boldsymbol{conflictIn}(\boldsymbol{v}) \equiv v.state = IN \wedge \exists w \in N(v): w.state = IN$$

$$\boldsymbol{Pending}(\boldsymbol{v}) \equiv state.v = OUT \ \wedge \sim inNeighbor(v)$$

$$\boldsymbol{pendigNeighbors}(\boldsymbol{v}) \equiv w \in N(v): Pending(w)$$

$$\boldsymbol{inNeighborWithLowerId}(\boldsymbol{v}) \equiv \exists w \in N(v): w.state = IN \wedge w.id < v.id$$

$$\boldsymbol{solePending}(\boldsymbol{v}) \equiv Pending(v) \wedge \forall w \in N(v): \sim Pending(w)$$

$$\boldsymbol{canOut}(\boldsymbol{v}) \equiv conflictIn(v) \wedge execution\ of\ (v.state := OUT) \Rightarrow \{\sim pendingNeighbors(v) \wedge \sim conflictIn(v) \wedge (\forall w N(v): \sim conflictIn(w))\}$$

Fig. 2. propositions and sets that are used in MIS*fc* algorithm pseudo code

There are two 1-fault states for maximal independent set: 1- there is a fault from member node leaving the set (i.e. IN to OUT), and the fault from entering a non-member node into the set (i.e. OUT to IN); the considerable feature of proposed algorithm is removing the intermediate state, which has been introduced at [30]. MIS*fc* algorithm rules with two features of self-stabilizing and fault containment is showed in figure 3.

Rules 1 to 3 are for the management of node state transition in any IN to OUT 1-fault scenario. First two rules are for identification of single-fault state and third rule controls more than one fault conditions in node v and its neighbors.

confilictIn(v) proposition is a prerequisite for establishing each of three rules. At first rule, with review of *canOut(v)*, single-fault state is just for node *v*, not in its neighbors, which means: $\forall w \in N(v): \sim canOut(w)$, we identify that under these

conditions to reach the stabilized state, node *v* should change its state variable and leave the set. Second rule will be active when single-fault state is at node *v* and also its neighbors like *w*, i.e. each one of two nodes (*v* and *w*) leave the set and we reach to steady state. However, it should be considered that simultaneous leaving of these two nodes caused the unsteady state, thus for disrupting occurred symmetric condition and preventing simultaneous running, in 2nd rule, if *v* node *id* is larger than its neighbor nodes, it will leave the set. In fact, the priority of remaining in independent maximal set with the equal conditions is with the node that has a smaller node *id*. In the following, rule 4th and 5th are implemented for management of changing in nodes condition at single-fault scenario of OUT to IN. Establishment of *pending(v)* proposition is the prerequisite for activation of two these rules. In the fourth rule, *v* will be membered of the set when the number of members of *pendingNeighbors* set is larger than this proposition in its neighbors.

Actually, the purpose is state transition of a node, which has the most problem solving among itself and its neighbors (removes the *pending* state from the most numbers of nodes). If this number is equal for current node and its neighbor, the node *id* will be used for disrupting the symmetric condition. Fifth rule check the state that *v* is the only pending node between itself and its neighbor.

R1. $canOut(v) \wedge \left(\forall w \in N(v): \sim canOut(w)\right) \rightarrow v.state \coloneqq OUT$

R2. $canOut(v) \wedge \left(\forall w \in N(v): (canOut(w) \wedge v.id > w.id)\right) \rightarrow v.state \coloneqq OUT$

R3. $conflictIn(v) \wedge \sim canOut(v) \wedge \left(\forall w \in N(v): \sim canOut(w)\right) \wedge inNeighborWithLowerId(v) \rightarrow v.state \coloneqq OUT$

R4. $Pending(v) \wedge \Big(|pendingNeighbors(v)| > \max_{\forall w \in pendingNeighbors(v)} |pendingNeighbors(w)|\Big) \vee \Big(\big(|pendingNeighbors(v)| = \max_{\forall w \in pendingNeighbors(v)} |pendingNeighbors(w)|\big) \wedge (v.id < w.id)\Big) \rightarrow v.state \coloneqq IN$

Fig. 3. Rules of MIS*fc* algorithm

B. *Proof of correctness*

Lemma 1 (Closure condition): In each configuration that no rule in any node is active, $I = \{v | v.state = IN\}$ set will construct a maximal independent set.

This trick will be simply proved by contradiction; the description of this proof is ignored in this part because of space limitation.

Lemma 2: During the running of algorithm, node *v* with any arbitrary conditions will run one of $OUT \rightarrow IN$،$IN \rightarrow OUT$ or $IN \rightarrow OUT \rightarrow IN$ series, and after that, it will not follow any other rule.

Proof: possible states are showed in figure 4 which are reviewed in detail below.

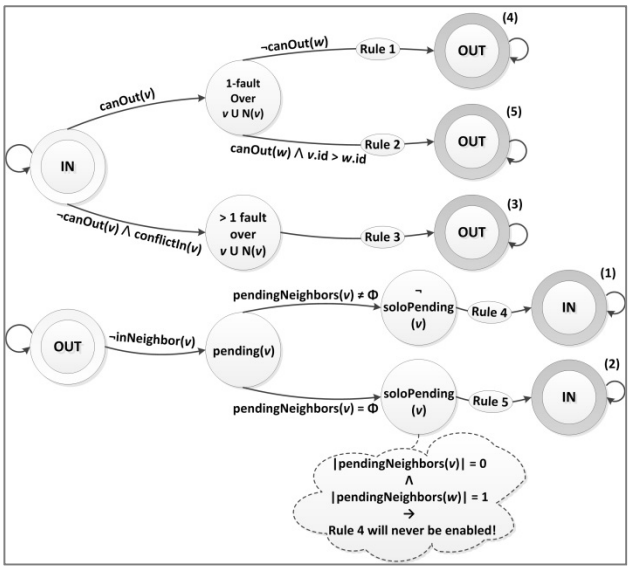

Fig. 4. State diagram of the MIS*fc* algorithm

Condition (1): The procedure of *v* to reach this state as shown in figure 4, starts with initial state of OUT and under terms of $Pending(v)$ and lack of $soloPending(v)$. After this round with becoming *v* as a member (IN), by considering that *v* does not have any neighbor with IN state ($\sim inNeighbor(v)$), $confilictIn(v)$ will never be established. In other words, all neighbors of *v*, *N(v)*, will be remained in the OUT state.

Condition (2: The procedure of *v* to reach this state as shown in figure 4, starts with initial state of OUT and under terms of *Pending(v)* and *soloPending(v)*. After this round, node *v* has changed to IN and all its neighbors are in OUT condition with at least two IN neighbors (out of set with at least 2 member neighbors); therefore, nodes of $\{v\} \cup N(v)$ set will never execute any other rules for changing condition.

Conditions (3), (4), and (5). starting from initial state of IN, executing rules corresponding with above diagram in figure 4 leads to place in one of conditions of (3), (4) or (5) with OUT condition for the node. In these conditions, starting from OUT state, if this state is not the last state, condition 1 or 2 will be held. Although, it could be proved that there will be no more state change after condition 4 and 5. Because these two conditions are held after 1-fault situation (lemma 3). Only after condition 3, it may be a new round of OUT → IN.

Theorem 1 (convergence condition): starting from any arbitrary configuration and after finite number of steps, any node is no longer active and $I = \{v | v.state = IN\}$ is a maximal independent set.

Proof: starting from any arbitrary configuration, it has been proved in lemma 2 that any node after maximum three steps will have no active rule. Besides, based on lemma 1, I is a maximal independent set in the configuration with no active node.

Lemma 3 (fault containment feature): in a legitimate configuration and in the event of a fault in a condition of an

arbitrary node *j*, we return to a legitimate condition with only $O(1)$ step.

Proof: two conditions can occur.

(A) with occurring a fault, node *j* changes from OUT condition to IN condition (there is $confilictIn(j)$): 1-(A) *j* has more than IN neighbor: In this case, there is $canOut(j)$, but for none of *j* neighbors, like *i*, $canOut(i)$ will not be established, because despite of *i* exit from the set, $confilictIn(j)$ could be established yet. Hence, in this condition, rule 1 could be only established for node *j* and with changing condition, we return to legitimate state. 2-(A) *j* has just one IN neighbor, like *i*: In this case, as regards that $canOut(j)$ necessarily leads to establish a legitimate condition, all following propositions will be established: $pendingNeighbors(j) = \emptyset$ and $\sim confilictIn(j)$ and $\forall k \in N(j): \sim confilictIn(k)$, which finally lead to establish $canOut(j)$. In these conditions, if $canOut(k)$ is not established, *j* is the only node changes its condition to OUT by considering rule 1 and system return to legitimate conditions. In the condition with $canOut(k)$, rule 2 will be activated just for one of *j* or *k* (node with large node *id*), and therefore, we will return to legitimate condition with only one changing condition.

(B) with occurring a fault, *j* changes its condition from IN to OUT (*Pending(j)* is established): 1-(B) all neighbors of *j* have a neighbor with IN condition: In this case, proposition of *Pending(j)* is established and therefore, rule 5 will be just activated and we will return to a legitimate configuration with only one changing condition. Rule 4 will not be activated because |*pendingNeighbors(v)*| is always smaller than |*pendingNeighbors(w)*|, *w*∈*N(v)*. 2-(B) There are neighbor or neighbors of *j* that exiting *j* leads to cause a pending condition; for simplicity, we assume that only node *i* in the neighbor of *j* has this condition. In this case, *Pending(j)* and *Pending(i)* propositions are established. We have 3 states: 1- |*pendingNeighbors(j)*|>|*pendingNeighbors(i)*|: only in node *j*, rule 4 will be activated and we will return to legitimate condition with only one move. 2- |*pendingNeighbors(j)*|<|*pendingNeighbors(i)*| : only in node *i* , rule 4 will be activated and we will return to legitimate condition with only one move,. 3- |*pendingNeighbors(j)*|=|*pendingNeighbors(i)*| : In the node with smaller node *id*, rule 4 will be activated and we will return to legitimate condition with only one move.

IV. Performance analysis

In this part, performance of the MIS*fc* clustering algorithm based on numbers of state transitions, numbers of necessary time cycles for stabilizing, and structure of topological clustering will be compared with presented algorithm at [4] (MIS). General procedure of simulations is based on starting from initial configuration (all nodes are in OUT state) and starting from multi-fault, which will be tested under policy of unfair distributed scheduler. All of simulation tests will be done with MATLAB software and results report for 100 times of tests in each scenario.

Figures of 5 and 6 show performance comparison of MIS*fc* and MIS under policy of unfair distributed scheduler with initial configuration of all non-member which is assumed that the average of connected degree is a constant value of 7 and numbers of nodes are variable.

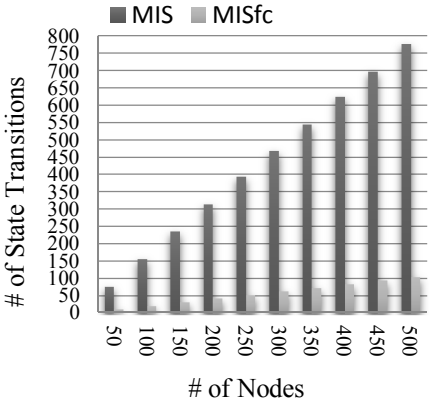

Fig. 5. Number of state taransitions in MSI and MIS*fc*

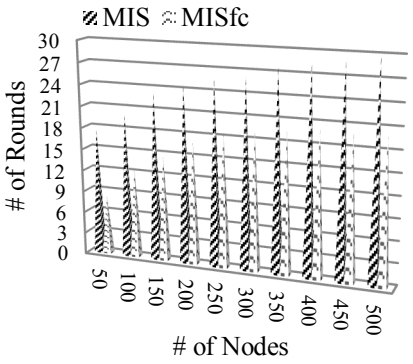

Fig. 6. Number of time cycles (rounds) in MSI and MIS*fc*

According to the diagrams, it can be concluded that performance ratio of MIS*fc* to MIS will increase with increasing in the numbers of nodes. Moreover, regardless of connected degree, MIS always has constant number of 2n state transitions (n: node number). Whereas, MIS*fc* has higher performance in denser topologies.

In another scenario, performance of these two algorithms has been compared by fault injection to the stabilizing configuration. In this test, which its diagram has been showed at figure 7, number of faults is varied between f 1 to 7 on a topology with 100 nodes and average connected degree is seven.

MISfc algorithm stabilizes single-fault states only with one state transition and one time cycle. However, MIS requires three state transitions and three time cycles on the average. With increasing in fault number, performance ratio of MIS*fc* to MIS will be reduced about 2 times in state transition numbers and 1.5 times in time cycle numbers.

Figure 8 shows clustering structure from running two algorithms of MIS*fc* and MIS on an initial topology with 50 nodes. Cluster numbers from running algorithm MIS are 27 in which 9 clusters of them are single member and consist of just one cluster-head. While in response to running MIS*fc* algorithm, Cluster number is reduced to 15 and we have only two single member cluster.

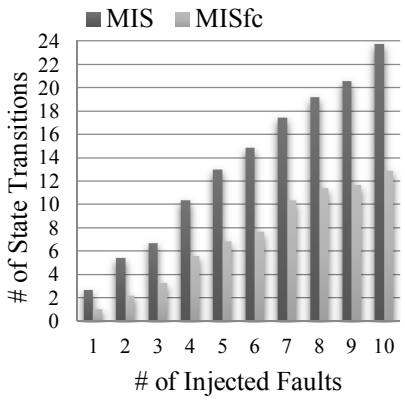

Fig. 7. Number of time cycles (rounds) in MSI and MIS*fc* (after fault injection)

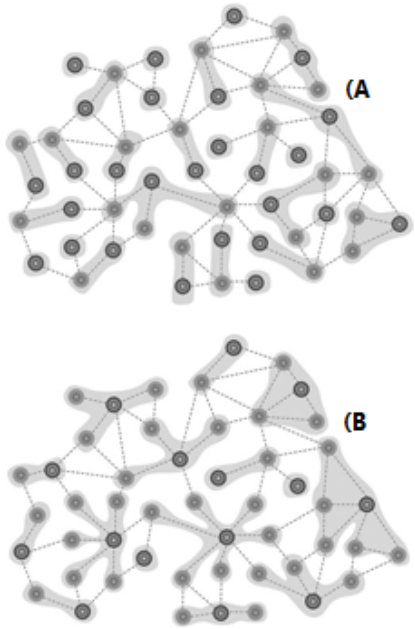

Fig. 8. Final Topology. a) MIS, b) MIS*fc*

V. Conclusion And Future Works

Regarding the susceptibility of node fault, the possibility of environmental changes and lack of accessibility to structure after placement of sensor network, clustering methods lead to higher stabilization of the system and its enjoyment from re-configuration without external interference is more desirable. In this work, a clustering method has been presented based on constructing a maximal independent set with two feature of self-stabilizing and fault containment. In addition to fast stabilization in small scale faults, it will improve necessary time for stabilization in start from any arbitrary initial configuration.

In other view which is based on cluster constructing and forming minimum dominated set, condition of non-neighbor for cluster-head nodes is not necessary and thus, in scenarios due to the moving of nodes after placement, it is possible to form cluster-heads in neighborhood of each other and numbers of state transitions for reaching to a legitimate configuration is lower. It is noteworthy that this algorithm has been designed and will be presented in future works.

References

[1] S. Tixeuil, "Algorithms and Theory of Computation Handbook, Second Edition, chapter Self-stabilizing Algorithms," CRC Press, Taylor & Francis Group, 2009.

[2] V. Turau and C. Weyer,"Fault tolerance in wireless sensor networks through self-stabilisation," *Int. J. Communication Networks and Distributed Systems*, Vol.2, No. 1, pp. 78-98, 2009.

[3] S. Ghosh, A. Gupta and S.V. Pemmaraju, "Fault-containing network protocols," *Proc. of the ACM Symposium on Applied Computing*, pp. 431-437, USA, 1997.

[4] V. Turau, "Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler," *Information Processing Letters*, Vol. 103, pp. 88-93, 2007.

[5] N. Guellati and H. Kheddouci, "A Survey on Self-Stabilizing Algorithms for Independence, Domination, Coloring, and Matching in Graphs," *J. of Parallel and Distributed Computing*, Vol. 70, No. 4, pp. 406-415, 2010.

[6] J. C. Lin and T. C. Huang, "An Efficient Fault-Containing Self-Stabilizing Algorithm for Finding a Maximal Independent Set," *IEEE Trans. Parallel and Distributed Systems*, Vol. 14, pp. 742-754, 2003.

[7] S. M. Hedetniemi, et al., "Self-stabilizing Algorithms for Minimal Dominating Sets and Maximal Independent Sets," *Computers & Mathematics with Applications*, Vol. 46, pp. 805-811, 2003.

[8] S. Dolev, "Self-stabilization," MIT Press, 2000.

[9] A. Dasgupta, "Extensions and Refinements of Stabilization," PhD Dissertation, University of Iowa, 2009.