Self-stabilizing algorithms of constructing virtual backbone in selfish wireless ad-hoc networks

Amirreza Ramtin
Department of Information Technology
Amirkabir University of Technology
Tehran, Iran
Email: a_ramtin@aut.ac.ir

Vesal Hakami
Department of Information Technology
Amirkabir University of Technology
Tehran, Iran
Email: vhakami@aut.ac.ir

Mehdi Dehghan
Department of Information Technology
Amirkabir University of Technology
Tehran, Iran
Email: dehghan@aut.ac.ir

Abstract— **A self-stabilizing system tolerates any transient faults and does not need any initialization. In wireless ad-hoc networks, a connected dominating set is the graph-theoretic counterpart of a virtual backbone. In this paper, we propose two distributed self-stabilizing algorithms for approximate minimum connected dominating set construction with perturbation-proof property in the sense that the legitimate configuration of the system gives rise to a Nash equilibrium, effectively discouraging the selfish nodes from perturbing the legitimate configuration by changing their valid states. The legitimate configuration of our first algorithm is always a Nash equilibrium regardless of the specifics of the nodes' utility functions, while the configurations reached in our second algorithm are Nash equilibria only if the selfish nodes explicitly prefer to be out of the virtual backbone. The second algorithm suits in particular for scenarios with multiple legitimate configurations, while the first algorithm always gives rise to a unique configuration. Both algorithms increase accessibility, reduce the number of update messages during convergence, and stabilize with minimum changes in the topological structure. Proofs are given for the self-stabilization and perturbation-proofness of the proposed algorithms. The simulation results show that our two algorithms outperform comparable schemes in terms of stabilization time and the number of state transitions.**

Keywords- Self-stabilization, Wireless ad-hoc network, virtual backbone, Selfishness, Perturbation, Nash equilibrium

I. INTRODUCTION

It is a well-established fact that the fundamental source of energy consumption in wireless ad hoc networks (WANETs) is the exchange of packets between nodes. Hence, when it comes to the design of routing mechanisms for WANETs, a key measure of efficiency is low communication overhead. A communication-efficient structure for supporting routing and multicast in WANETs is the virtual backbone architecture [1]. The virtual backbone approach to routing consists of two phases: a) creation and update of a virtual backbone substrate, b) finding and updating the paths. In this paper, we focus on the first phase. Concerning virtual backbone creation and maintenance, the two foremost desirable properties are stability and self-configuration without external intervention, given the dynamic and unstable topology of WANETs and the multi-hop nature of communications [4]. A promising approach to realize stability and self-configuration is to rely on the notion of self-stabilization in distributed fault-tolerance [8]. A self-stabilizing algorithm guarantees that the system eventually converges to the desirable state regardless of its initial configuration, and that it remains in the desirable configuration as long as no transient fault occurs. Hence, designing virtual backbones with the self-stabilization property has the advantages of automatic structuring, and robustness against: transient faults, node failures, changes in their internal states, and occasional breakages in the communication structure of the system [8].

However, in the majority of self-stabilizing protocols for wireless ad-hoc networks, it is routinely assumed that the network nodes will cooperate with each other so that the overall stabilization of the system is guaranteed. This is while in most practical settings, the nodes neither belong to the same authority, nor do they operate under a single administration domain. Hence, it might be the case that the nodes pursue some private goals that may be in conflict with the system-wide objective. Consider, in particular, virtual backbone construction using a self-stabilizing algorithm. Obviously, once the protocol stabilizes, the nodes serving in the backbone have to sacrifice more processing and communication resources to the benefit of the entire network. Hence, each backbone node faces a dilemma as to whether maintain its serving role in the constructed backbone, or alternatively, perturb the system hoping that the algorithm would re-stabilize this time into a new configuration where the node is a backbone client rather than a server.

Motivated by the impact of node selfishness on protocol stabilization in ad-hoc networks, in this paper, we deal with perturbation-proneness in the context of virtual backbone construction in WANETs. The problem is first translated into minimum connected dominating set (MCDS) construction in the topological graph of the network. We then propose self-stabilizing MCDS algorithms that prevent selfish nodes from post convergence perturbation of the system. A byproduct of our proposed scheme is faster recovery from all single-fault configurations with reduced message complexity, lower number of state transitions, and minimal topological re-structuring, which contributes to saving energy and increasing network life.

The rest of the paper is organized as follows: We briefly introduce the basic concepts and review the previous studies in section 2. In 3, the proposed algorithms are discussed and proofs are given to establish their correctness. Section 4 deals with the numerical evaluation of the algorithms and comparisons are made to contrast their performance against prior art. The paper ends with conclusions.

II. Theoretical background and relevant works

A system is self-stabilizing, if and only if, two conditions are satisfied: a) *convergence*: starting from an arbitrary initial state, the system converges to a legitimate global configuration after a finite number of state transitions, b) *closure*: the system remains in legitimate configuration until no transient fault happens [2].

In this paper, we are interested in forming a virtual backbone substrate for wireless ad-hoc networks with robustness properties against both transient systemic faults and deliberate perturbations. A popular abstraction in prior art [1] has been to translate the virtual backbone formation problem into the construction of a minimum connected dominating set (MCDS) in the topological graph of the underlying network. In graph theory, a CDS of graph G is a set D of nodes which satisfy two conditions: a) D is a connected sub-graph of G. b) any node of G is either in D or is adjacent to at least one node in D. A CDS of G is an MCDS, if it has the minimum cardinality among all possible CDSs of G. In recent years, several self-stabilizing algorithms have been proposed for constructing CDS. The majority of the existing self-stabilizing CDS algorithms, however, have been designed based on central daemon (scheduler) which is practically impossible to implement in wireless ad-hoc networks [3]. Furthermore, most of these works solely construct a CDS and their final product is not an approximation of MCDS [6]. Another drawback of all such algorithms is that they do not differentiate between faults with respect to their spread.

We set up our design on the self-stabilizing algorithm proposed in [7] which works under distributed scheduler with O(n^2) time complexity. We refer to this algorithm as MCDSss in the rest of this paper. The CDS Constructed by this algorithm is based on a sequential scheme [7] that produces an $8opt + 1$ approximation of MCDS in a given graph. We present two variants of MCDSss, which render the CDS construct immune to nodes' intentional state manipulations. Such manipulations are typically motivated by individual node utilities in the sense that the nodes would naturally prefer to be a CDS client rather than a CDS server. A general approach to realize the perturbation-proof property for self-stabilizing systems has been discussed in [5]. It is argued that the final states in a self-stabilizing system are analogous to fixed points of a game; hence, a fixed point, which is also a Nash equilibrium, is obviously immune against unilateral node deviations. With a perturbation-aware design, a self-stabilizing system can be made either absolutely or relatively perturbation-proof. A system is absolutely perturbation-proof, if all its fixed points are Nash equilibria for any set of utility functions. On the other hand, a system is relatively perturbation-proof, if all its fixed points are Nash equilibria for some specific set of utility functions.

III. Proposed algorithm

In this section, we first present two perturbation-proof variants of the MCDSss algorithm, namely MCDSpp and MCDSpp^*. Next, we provide proofs of their self-stabilization and perturbation-proof properties.

A. Discussion on design and functionality of algorithms

MCDSpp is designed based on the MCDSss algorithm [7]. MCDSss first constructs a breadth-first spanning (BFS) tree in the network graph. Next, starting from the root, a maximal independent set (MIS) is formed iteratively among the nodes of the same depth, which are not dominated by the nodes of lower depths. Integration of all these sets produces an MIS in network graph. It is proven that this MIS is a weakly connected dominating set (WCDS). Finally, a fully connected set is established by adding connecting nodes. Connecting nodes are the parents of the MIS members in the BFS tree. The final fully connected set is a CDS-tree. It is then proven that the CDS-tree is an 8opt+1 approximation of an MCDS.

We assume that the tree T is formed in the network graph through a self-stabilizing BFS tree algorithm. Let "l" denote the distance of each node from the root. The state of each node is specified by two variables *ind, dom* $\in \{IN, OUT\}$ in MCDS configuration. Each node in the legitimate configuration is in one of the three states: (IN, IN), (IN, OUT) and (OUT, OUT). The set of nodes in state (IN, IN) are MIS members. The union of nodes in state (IN, IN) and (IN, OUT) forms a CDS. In a legitimate configuration, any state transition can be deemed as a transient fault.

In a legitimate configuration of a self-stabilizing system, 1-fault situations correspond to the occurrence a single fault in a node v, which is induced by an undesirable change in its variables. It can be shown that two conditions apply a) one of the rules is active in v. b) it is possible to reach stability by execution of only one rule in v [9][10]. We aim to detect and resolve 1-fault states by restricting rule executions only on the faulty node, effectively preventing from error propagation by unwanted execution of rules in v's neighbors *N(v)*.

We first define some predicates, which will appear as preconditions to state transitions. We also introduce some sets to facilitate the readability of the pseudo-code (figure 1).

The sets PN, BN, MN, and CN denote the parent nodes (lower depth neighbors), sibling nodes (same depth neighbors), mature nodes (union of PN and BN), and child nodes (higher depth neighbors) of a node in the tree, respectively. The fifth expression identifies the parent of a given node. The parent of a node is its lowest *id* neighbor. The 6th and 7th expressions verifies whether a mature or parent neighbor is a member of MIS or not. The 8th term specifies if a node is pending. If neither a node nor its mature neighbors are members of MIS, that node is considered to be pending. The 9th term specifies if a node is in conflict. If a node and at least one of its mature neighbors are members of MIS, the conflict predicate is true in that node. The 10th predicate will hold in a node if at least one of its siblings is a member of MIS and its *id* is lower than that node. The 11th predicate indicates a conflict between a node and one of its parent neighbors. The rules of MCDS*pp* are depicted in figure 2.

1. $\boldsymbol{BN(v)} := \{w \in N(v) | l.w = l.v\}$
2. $\boldsymbol{PN(v)} := \{w \in N(v) | l.w < l.v\}$
3. $\boldsymbol{CN(v)} := \{w \in N(v) | l.w > l.v\}$
4. $\boldsymbol{MN(v)} := \{w \in N(v) | l.w <= l.v\}$
5. $\boldsymbol{father(v)} := \min\{id.w | w \in PN(v)\}$
6. $\boldsymbol{inMatureNeighbor(v)} \equiv \exists w \in MN(v): ind.w = IN$
7. $\boldsymbol{inParentNeighbor(v)} \equiv \exists w \in PN(v): ind.w = IN$
8. $\boldsymbol{Pending(v)} \equiv ind.v = OUT \wedge \sim inMatureNeighbor(v)$
9. $\boldsymbol{conflict(v)} \equiv ind.v = IN \wedge inMatureNeighbor(v)$
10. $\boldsymbol{inBrotherWithLowerId(v)} \equiv \exists w \in BN(v): ind.w = IN \wedge id.w < id.v$
11. $\boldsymbol{conflictWithParent(v)} \equiv ind.v = IN \wedge$

Figure 1. Set and Predicate definitions

R1. $l.v = 0 \wedge (ind.v = OUT \vee dom.v = OUT) \rightarrow ind.v := IN,\ dom.v := IN$

R2. $l.v = 1 \wedge ind.v = IN \rightarrow ind.v := OUT$

R3. $l.v \neq 0 \wedge l.v \neq 1 \wedge ind.v = OUT \wedge \sim inParentNeighbor(v) \wedge \forall w \in PN(v): \sim pending(w) \wedge \Big(\forall w \in BN(v): id.w > id.v \vee \big(ind.w = OUT \wedge \big(inParentNeighbor(w) \vee inBrotherWithLowerId(w)\big)\big)\Big) \rightarrow ind.v := IN, dom.v := IN$

R4. $l.v \neq 0 \wedge l.v \neq 1 \wedge conflictWithParent(v) \wedge \big(\forall w \in PN(v): \sim conflict(w)\big) \rightarrow ind.v := OUT$

R5. $l.v \neq 0 \wedge l.v \neq 1 \wedge ind.v = IN \wedge \sim conflictWithParent(v) \wedge (\forall w \in BN(v): ind.w = IN \wedge \sim conflictWithParent(w) \wedge \sim inBrotherWithLowerId(w) \wedge id.w < id.v) \rightarrow ind.v := OUT$

R6. $\sim R2 \wedge \sim R4 \wedge \sim R5 \wedge ind.v = IN \wedge dom.v = OUT \rightarrow dom.v := IN$

R7. $\sim R3 \wedge ind.v = OUT \wedge dom.v = OUT \wedge (\exists w \in CN(v): ind.w = IN \wedge \sim conflict(w) \wedge father(w) = v) \rightarrow dom.v := IN$

R8. $\sim R3 \wedge ind.v = OUT \wedge dom.v = IN \wedge (\forall w \in CN(v): father(w) \neq v \vee \big(ind.w = OUT \wedge \sim pending(w)\big)) \rightarrow dom.v := OUT$

Figure 2. Rules of MCDS*pp* algorithm

The process of constructing MIS in the tree proceeds from root towards the last depth according to rules 1 to 5. The first rule determines the root's state. This node must become a member of MIS. The second rule determines the membership of the root neighboring nodes (first depth). Rules 3, 4, and 5 govern the membership of the nodes in MIS. A node may become a member by performing rule 3 and may cancel its membership by performing rule 4 or 5. While MIS forms, deeper nodes states has no effect on upper nodes states in T. The state of deeper nodes has no effect on the shallower ones. In order to break symmetry of nodes, we give priority of MIS membership to the nodes that have lower *id* than their siblings in the same depth of the tree. To detect 1-fault situations, each node needs to know the membership status of all its 2-hop neighbors. This information guarantees that if a 1-fault occurs in a given node's parent or sibling, no rule will become active on the node. According to rule 6, members of MIS i.e. the nodes for which the *ind* is in IN state, join MCDS. Rule 7 or 8 checks membership or none-membership of the remaining nodes in CDS, respectively. Nodes that are fathers of members of MIS, become members of MCDS by executing rule 7.

There is only one legitimate configuration in a system based on MCDS*pp* algorithm. In other words, it always terminates in a unique virtual backbone. However, if we assign weights to the nodes, the members of the final CDS always have the lowest weight among their neighbors, and in general, it is possible that there exist better MCDS approximations which are ignored by MCDS*pp*. To solve this problem, we design a new algorithm called MCDS*pp** which differs from MCDS*pp* in just the third rule.

In rule 3 of the MCDS*pp**, we address situations that the occurrence of 1-faults in members may spread to their neighbors. This situation occurs when a none-member node *v* has no pending sibling neighbor and has only one member sibling *w* whose *id* is greater than that of *v*'s. In this situation, if a fault happens in *w,* after re-convergence, node *v* will become member of MCDS instead of *w*. Therefore, 12-th predicate in rule 3 captures such occurrences in the legitimate configurations.

R'3. $l.v \neq 0 \wedge l.v \neq 1 \wedge ind.v = OUT \wedge \sim inParentNeighbor(v) \wedge \forall w \in PN(v): \sim pending(w) \wedge ((\forall w \in BN(v): ind.w = OUT \wedge (id.w > id.v \vee (inParentNeighbor(w) \vee inBrotherWithLowerId(w)))) \vee ((\forall w \in BN(v): \sim pending(w)\ ) \wedge oneInBrotherGreaterId(v))) \rightarrow ind.v := IN, dom.v := IN$

12. $\boldsymbol{oneInBrotherGreaterId(v)} \equiv (\exists w \in BN(v): ind.w = IN \wedge id.w > id.v) \wedge (|\{w \in BN(v) | ind.w = IN\}| = 1)$

Figure 3. The third rule of MCDS*pp** algorithm

B. Proof of correctness

In this part, we prove the correctness of MCDS*pp* through a sequence of lemmas and theorems. For the most parts, the proofs associated with MCDS*pp** proceed along the same lines, and are thus skipped here due to space limitations.

Lemma 1. Assume that the spanning tree T is valid up to the *i*-th depth and MIS is constructed up to (i-1)th depth by MCDS*pp* rules. It then holds that the MIS is constructed up to the ith depth after the maximum of *m* rounds. In addition, no node changes its state in the absence of transient faults.

Proof. The root becomes a member of MIS by executing rule 1 at the first round. It is obvious that this membership is permanent because rules 2-5 are not executed in the root. Similarly, neighbors of root (l=1) leave membership of MIS via rule 2 at the first round and this decision will be permanent. It is clear that the membership of deeper nodes has no effect on the membership of the ith depth in MIS according to rules 1-5. If a node gets out because of rule 4 or gets in because of rule 3, assuming that no '*inbrotherwithlowerid*' term has been active in rule 3, the new state of node will be permanent. The reason is that all predictions are related either to lower depth nodes for which the validation and stability are assumed, or to the base information like *id*. Yet in rule 3 or 5, there is the predicate '*inbrotherwithlowerid*', which is also related to the state of the same depth nodes. At the first round in all nodes that rule 4 is active, *ind* variable becomes equal to OUT. It is obvious that OUT state (non-membership in MIS) is permanent in these nodes. Following the first round, in the second round, *ind* variable in all nodes which rule 3 is active in them becomes equal to IN. After the second round, either *ind* variable value is permanently OUT in all nodes of ith depth or at least there is one node (v) that is in IN state, a permanent state. In the third round, neighboring nodes with the same depth of node v, which are in IN state switch to OUT state (rule 5). It can be shown that this state is permanent in those nodes and does not change in following rounds. In the next round, nodes that rule 5 is active in them get out and it is permanent. Then, rounds 3 and 4 will be repeated until there is still some nodes in which rules 3 or 5 are active. So, a number of rounds up to a maximum equal to the number of ith depth nodes are traversed until MIS is constructed at this depth.

Lemma 2. MIS structure in *T* is formed after *n* rounds. *n* is number of tree nodes.

Proof. We use induction to prove this lemma. In lemma 1, it has been shown that the root and the second-depth nodes of T enter to valid state of MIS just in one round (basis: statement holds for d=1,2). Using lemma 1, inductive step will be proven for d>1. Due to lemma 1, if MIS is formed up to the ith depth, after m_i rounds, it will be formed up to (i+1)th depth. Therefore time complexity of MIS construction is $o(\sum_{i=2}^{D} m_i)$ which is equal to *o(n)*. D is depth of T.

Lemma 3 (convergence). The MCDS*pp* algorithm constructs MCDS after R_T+n+1 rounds.

Proof. T in R_T and then MIS in n rounds are constructed. According to rules 6-8, members of MIS and connecting nodes join to MCDS. The MCDS members with active rule #8 exit in one round. Since all terms of those three rules depend on *id* and *ind* variables, and not on *dom*, final states are permanent.

Lemma 4 (closure).

Proof. We prove this lemma by contradiction. Suppose that the closure condition does not hold; hence, at least one rule is active in legitimate configuration. This is while due to lemmas 1-3, the final states are permanent and no rules will be executed in the legitimate configuration.

C. *Proof of perturbation-proof feature*

Lemma 5. Occurrence of 1-fault in *ind* variable of a node in the ith depth has no effect on the state of upper or lower depth nodes.

Proof. Since the MIS is formed prior the 1-fault incident, either of the pending or conflict predicates will hold. The state of parents affects the preconditions of rules 3-5 in a node. To be sure that those rules will not be activated by the 1-fault incident in lower depth nodes, some terms are added to them, checking whether the pending or conflict predicates are active in the parent neighbors or not. Similarly, in rules 7-8, those predicates are checked for upper depth nodes given that the states of children affect the preconditions for those rules. It is obvious that preconditions of rules 1, 2 and 6 have no relevance to the states of the neighbors.

Lemma 6. In a system based on MCDS*pp* algorithm, occurrence of 1-fault in *ind* variable of an ith depth node has no effect on the other ith depth nodes.

Proof. It is obvious that the change in *ind* variable of a node has no impact on the *dom* variables of its siblings. Hence, we only focus on the 1-faults in q node v and its impact on the *ind* variables of the ith depth 1-hop neighbor z and 2-hop neighbor k.

If 1-fault (IN to OUT) happens in v, the only rule that might be active in z is rule 3. Note that state of z is OUT. If node z has a parent in IN state or its *id* is greater than v, rule 3 does not activate. Otherwise it is evident that in the valid states, rule 3 did not execute in z because of another brother like w that had a lower id than z and was in IN state. Because 1-fault happens in z, not w, rule 3 still do not activate in z. If state of k is IN, the only rule that might be active in that node is rule 5. However, in rule 5, even if term '*~inbrotherwithlowerid*' is active, term '*ind*.w=IN' must be active concurrently either, but in the previous paragraph we show that 1-hop brother of v remains in OUT state. If node k is in OUT state, rule 3 certainly cannot be active in it, because there is no preconditions in that rule that holds with occurrence of 1-fault.

Assuming that 1-fault (OUT to IN) happens in v, if state of z is OUT, it cannot activate any rule in z. If state of z is IN, the only rule that might be active is rule 5. Because valid state of v had been OUT, there were some preconditions of rule 3 that had not hold. It is not possible that node v can activate rule 5 in another node because of those preconditions. In legitimate configuration, MIS membership states in two-hop neighborhood of v (k z v) is one of these three cases: (010,100,000). In the first case, the only rule that might be active is rule 3, but term '*ind*.w=OUT' must hold if rule 3 is active. Therefore, 1-fault cannot activate rule 3 in k, because state of z is still IN. In the second case, the only rule that can be active is rule 5, but as the term '*ind*.w=IN' exists in rule 5, it cannot activate, because z is in OUT state. In the last case, although it seems that rule 3 can activate in node k, but a brother or a parent in IN state has existed and they still do not allow rule 3 being active in node k.

Lemma 7. In a system based on MCDS*pp** algorithm, occurrence of 1-fault in *ind* variable of a ith depth member node (IN to OUT) has no effect on the other ith depth nodes.

Proof. It is obvious that *ind* variable change in a node has no effect on the *dom* variables of its brothers. Hence, we investigate effect of 1-fault in node v on *ind* variables of the ith depth 1-hop neighbor z. For a 2-hop neighbor, it is completely like lemma 6.

If 1-fault (IN to OUT) happens in v, the only rule that might be active in z is rule 3. Note that state of z is OUT. If node z has a parent in IN state, its *id* is greater than *v* or is still pending, rule 3 does not activate. Now assume that node z is pending and its *id* is lower than *v*. It is concluded that before occurrence of 1-fault in *v*, node z have had only one member neighbor with higher id. Because system has been in legitimate configuration, all neighbors of z were in none-pending state. However, with considering the terms of rule 3, before occurrence of 1-fault in *v*, rule 3 has been activated in z that is in contradiction with definition of legitimate configuration.

Lemma 8. 1-faults in the *dom* variable of an ith depth node have no effect on the states of is neighbors.

Proof. Since in preconditions of rules 1-8 do not refer to *dom* variables of neighbors, it is obvious that the change of *dom* variable in a node has no effect on the others.

Theorem 1. If a 1-fault occurs in the system, faulty nodes and only that node enters to the valid state that it was in before.

Proof. The convergence property of an algorithm explains that the system converges from an illegitimate configuration to a legitimate one. We also showed in lemmas 5-8 that the occurrence of 1-faults has no effect on the neighbors. With these in mind, it is easy to see that with the execution of the self-stabilizing rules in the faulty node, the system will return to a legitimate configuration.

Theorem 2. If the self-stabilizing rules cause that after perturbation of any selfish node in a legitimate configuration, the system returns to that legitimate configuration, that configuration is a Nash equilibrium.

Proof. Consider the definition of a Nash equilibrium: a legitimate configuration of a self-stabilizing system is a Nash equilibrium, if no node can profit by unilateral deviations from its state. The main drive for a node to induce perturbations in a self-stabilizing system is the possible convergence of the algorithm into an alternative legitimate configuration so that its utility increases in the new configuration. In a legitimate configuration of a self-stabilizing system, perturbation of a node is analogous to the occurrence of a 1-fault in that node. Given that the rules in MCDS*pp* guarantee that after any 1-fault in a given node, the system converges back to the same legitimate configuration, no node will have any incentive to deviate from its valid state, and thus the algorithm, once stabilizes, gives rise to a Nash equilibrium configuration.

Theorem 3. A system based on MCDS*pp* algorithm is absolutely perturbation-proof.

Proof. According to theorem 1, in a system based on MCDS*pp* algorithm, after 1-fault incident in legitimate configuration, system will return to that legitimate configuration again only by one move. In theorem 2, we said that if self-stabilizing rules force system to return to the previous legitimate configuration after perturbation of a selfish node, that configuration is in Nash equilibrium. Therefore, stable states of a self-stabilizing system based on MCDS*pp* algorithm are in Nash equilibrium for any utility functions. It means that the MCDS*pp* algorithm is absolutely perturbation-proof.

Theorem 4. A system based on MCDS*pp** algorithm is relatively perturbation-proof.

Proof. In a system based on MCDS*pp** algorithm, after any 1-fault in *dom* variables and IN to OUT 1-faults in *ind* variables, the system will return to the previous configuration only by one move. In theorem 2, we said that if the self-stabilizing rules force the system back to the pre-perturbation configuration, this configuration is a Nash equilibrium. Therefore, the stable states of a self-stabilizing system based on MCDS*pp**algorithm are Nash equilibria with respect to the utility functions that drive a member node to perturb and get out of the virtual backbone construction. It means that the MCDS*pp* algorithm is relatively perturbation-proof and no member node has an incentive to exit from the membership of the virtual backbone construction.

IV. PERFORMANCE EVALUATION

In this section, we conduct a number of experiments to compare the performance of our two virtual backbone construction algorithms MCDS*pp* and MCDS*pp** with that of MCDS*ss* [7]. The comparisons are made in terms of the number of update packets (overhead) and stabilization time. We simulate the algorithms under two operational scenarios: arbitrary configuration (*ind* and *dom* variables take on random values from the set {IN,OUT}) and multiple fault configuration. All experiments are implemented with OMnet++ simulator under an unfair scheduler, and the reported data points are the average of 100 tests in each scenario. The MAC configuration adheres to IEEE 802.11 and the channel model is simple path loss. Each node asynchronously notifies its neighbors of its current state by broadcasting update packets. In our proposed algorithms, each node needs to be notified of the states of its neighbors. This can increase message overhead dramatically. To solve this problem, a self-stabilizing synchronization algorithm has been designed that manages notifications based on prediction changes (figure 4).

```
1. $state.v = [OUT \rightarrow IN] \longrightarrow send(sync\ packet, date = 1XX)$
2. $state.v = [IN \rightarrow OUT] \longrightarrow send(sync\ packet, date = 1XX)$
3. $state.v = OUT\ \wedge [pending(v) \rightarrow {\sim}pending(v)] \longrightarrow$
   $send(sync\ packet, date = 00X)$
4. $state.v = OUT\ \wedge [{\sim}pending(v) \rightarrow pending(v)] \longrightarrow$
   $send(sync\ packet, date = 01X)$
5. $state.v = IN\ \wedge [conflict(v) \rightarrow {\sim}conflict(v)] \longrightarrow$
   $send(sync\ packet, date = 10X)$
6. $state.v = IN\ \wedge [{\sim}conflict(v) \rightarrow conflict(v)] \longrightarrow$
   $send(sync\ packet, date = 11X)$
7. $\big[\big({\sim}inBrotherWithLowerId(v) \wedge {\sim}inParentNeighbor(v)\big) \rightarrow$
   $\big(inBrotherWithLowerId(v) \vee inParentNeighbor(v)\big)\big] \longrightarrow$
   $send(sync\ packet, date = XX1)$
8. $[(inBrotherWithLowerId(v) \vee inParentNeighbor(v)) \rightarrow$
   $({\sim}inBrotherWithLowerId(v) \wedge {\sim}inParentNeighbor(v))] \longrightarrow$
   $send(sync\ packet, date = XX0)$
```

Figure 4. Self-stabilizing synchronization algorithm

In figures 5 and 6, the number of update packets (overhead) and stabilization time of MCDS*pp*, MCDS*pp** and MCDS*ss* are reported, respectively. Average connectivity degree is 8 and we have varied the number of nodes. As can be seen, the performance superiority of MCDS*pp* and MCDS*pp** over MCDS*ss* becomes even more apparent as the number of nodes increases. Overall, MCDS*pp* has the best performance among the three algorithms.

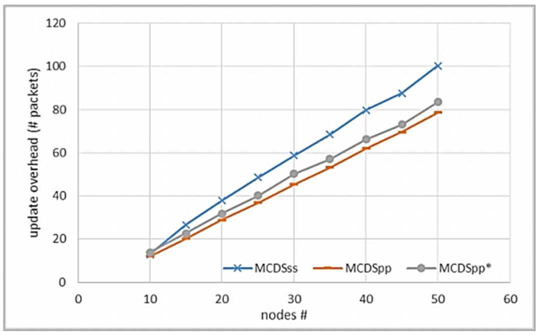

Figure 5. The impact of the number of nodes on overhead.

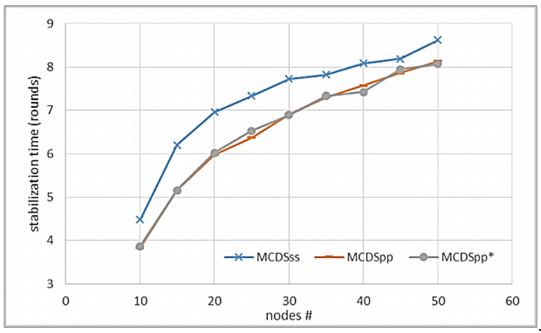

Figure 6. The impact of the number of nodes on stabilization time.

In another scenario, we evaluate the performance of the algorithms when faults are injected into the legitimate configuration (see figures 7 and 8). In this scenario, the number of fault injections varies from 1 to 20. The network consists of 25 nodes with average connectivity degree of 3. MCDS*pp* stabilizes from single faults using only one notification packet. While MCDS*pp** and MCDS*ss* need on average 1.2 and 4.4 notification packets, respectively. With more fault injections, the performance gain of MCDS*pp* over MCDS*ss* decreases.

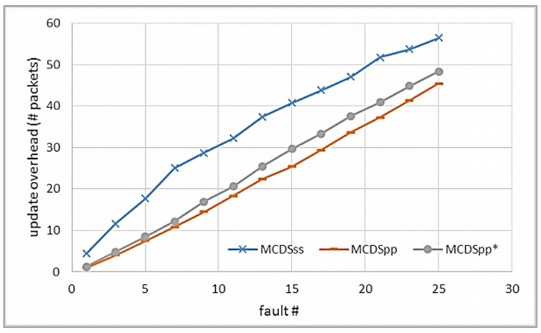

Figure 7. The impact of the number of fault injections on overhead.

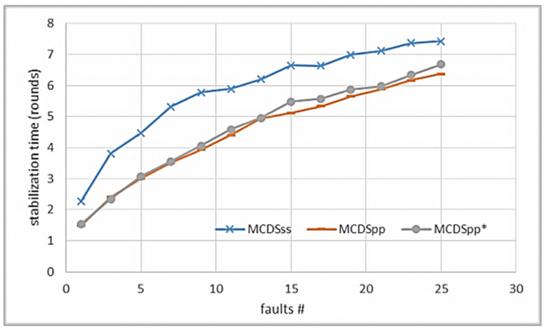

Figure 8. The impact of the number of fault injections on stabilization time.

V. Conclusion

In this paper, two distributed virtual backbone construction algorithms have been proposed for wireless ad-hoc networks based on the notion of MCDS in graph theory. The proposed algorithms are self-stabilizing against transient faults and topology changes. We also proved that the stable configuration of our algorithms gives rise to a Nash equilibrium, and thus, selfish nodes have no motivation to perturb the constructed backbone once the system converges. The other merit featured by our algorithms is fast convergence from single fault configurations. We plan to extend these algorithms to accommodate situations where nodes may also exhibit selfish behavior during convergence.

References

[1] B. Das and V. Bharghavan, "Routing in ad-hoc networks using minimum connected dominating sets," in *Communications, 1997. ICC 97 Montreal,'Towards the Knowledge Millennium'. 1997 IEEE International Conference on*, 1997, vol. 1, pp. 376–380.

[2] A. Dasgupta, S. Ghosh, and S. Tixeuil, "Selfish stabilization," *Stabilization, Safety, and Security of Distributed Systems*, pp. 231–243, 2006.

[3] D. Dubhashi, A. Mei, A. Panconesi, J. Radhakrishnan, and A. Srinivasan, "Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons," *Journal of Computer and System Sciences*, vol. 71, no. 4, pp. 467–479, 2005.

[4] L. Gao, M. Li, B. Li, and W. Zhou, "Virtual backbone routing structures in wireless ad-hoc networks," *Global journal of computer science and technology*, vol. 10, no. 4, 2010.

[5] M. Gouda and H. Acharya, "Nash equilibria in stabilizing systems," *Stabilization, Safety, and Security of Distributed Systems*, pp. 311–324, 2009.

[6] A. Jain and A. Gupta, "A distributed self-stabilizing algorithm for finding a connected dominating set in a graph," in *Parallel and Distributed Computing, Applications and Technologies, 2005. PDCAT 2005. Sixth International Conference on*, 2005, pp. 615–619.

[7] S. Kamei and H. Kakugawa, "A self-stabilizing distributed approximation algorithm for the minimum connected dominating set," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007, pp. 1–8.

[8] S. Tixeuil, "Self-stabilizing algorithms," in *Algorithms and theory of computation handbook*, M. J. Atallah and M. Blanton, Eds. Chapman & Hall/CRC, 2010, pp. 26–26.

[9] S. Köhler and V. Turau, "Fault-Containing Self-Stabilization in Asynchronous Systems with Constant Fault-Gap," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, 2010, pp. 418–427.

[10] S. Ghosh, A. Gupta, T. Herman, and S. V. Pemmaraju, "Fault-containing self-stabilizing distributed protocols," *Distributed Computing*, vol. 20, no. 1, pp. 53–73, Jun. 2007.