Game-Theoretic Approach to Self-Stabilizing Distributed Formation of Minimal Multi-Dominating Sets

Li-Hsing Yen *Member, IEEE* and Zong-Long Chen

Abstract—Dominating set is a subset of nodes called dominators in a graph such that every non-dominator nodes (called dominatee) is adjacent to at least one dominator. This paper considers a more general multi-dominating problem where each node i, dominator or dominatee, is required to have at least k_i neighboring dominators, and different node can have different k_i value. We first propose a game design toward this problem. This game is self-stabilizing (i.e., it always ends up with a legitimate state regardless of its initial configuration). The obtained result is guaranteed minimal (i.e., it contains no proper subset that is also a multi-dominating set) and Pareto optimal (we cannot increase the payoff of some player without sacrificing the payoff of any other). We then point out challenges when turning the design into a distributed algorithm using guarded commands. We present an algorithm that is proved weakly stabilizing. Simulation results show that the proposed game and algorithm produce smaller dominating sets, k-dominating sets, and multi-dominating sets in various network topologies when compared with prior approaches.

Index Terms—Dominating set, self-stabilization, distributed algorithm, game theory

1 Introduction

A distributed system consists of multiple processes interconnected by communication links. It can be modeled as a connected undirected graph where nodes represent processes and edges represent communication links between processes. Given a connected undirected graph $G = (V, E)$, a set $S \subseteq V$ is a *dominating set* if every node in $V - S$ is adjacent to some node in S. Nodes in S are *dominators* while those in $V - S$ are *dominatees*. Finding a dominating set with minimum cardinality (called a minimum dominating set) is NP-hard [1]. A dominating set S is *minimal* (called a *minimal dominating set* or MDS) if it contains no proper subset that is also a dominating set. Minimal dominating sets may not be minimum dominating sets. Dominators in a distributed system may represent servers or proxies that provide some type of services to adjacent dominatees. Typical services include message queuing or forwarding, data storage or backup, and sharing of computation load.

In some applications, a process may demand a service that is beyond the capacity of a single server or proxy. Even if a single server or proxy suffices, a process may seek for two or more neighboring servers to tolerate a crash of a single server. The demand of higher quality of service (QoS) leads to the definition of k-domination. A set of nodes is a *k-dominating set* if every dominatee is adjacent to at least k dominators, where k is a fixed constant for all dominatees[1] [3], [4]. When $k = 1$, k-domination reduces to the traditional 1-domination (i.e., the dominating set).

Harary and Haynes [5] introduced a variant of k-domination called *k-tuple dominating set*. A subset of nodes comprise a k-tuple dominating set if every node, dominator or dominatee, is dominated by at least k dominators. Here (and in the definition of multi-dominating set which will be introduced shortly) the domination by a dominator to itself counts. Therefore, each dominator is in fact required to have $k - 1$ neighboring dominators, and 1-tuple domination problem reduces to the classical single domination problem. In [6], Jia et al. further considered *multi-dominating set* problem, which extends the k-tuple domination problem by allowing different nodes to have different domination requirements. For a distributed system consisting of n nodes, multi-domination can be expressed as K-domination, where $K = (k_1, k_2, \ldots, k_n)$ is a tuple of integers that represent the domination requirement of every node. A K-dominating set with every k_i in K larger than or equal to k is also a k-dominating or k-tuple dominating set, but the opposite does not hold. In this regard, K-domination represents a more general QoS requirement. Fig. 1 shows an example that illustrates the mentioned variants of dominating set.

A distributed algorithm is *self-stabilizing* if starting from any initial state (possibly illegitimate), the algorithm eventually enters a legitimate state and remains in

• *Li-Hsing Yen and Zong-Long Chen are with the Department of Computer Science & Information Engineering, National University of Kaohsiung, Kaohsiung, Taiwan 811, R.O.C. (e-mail: lhyen@nuk.edu.tw, erice1211@gmail.com).*

1. This definition differs from that in [2], where every node is required to have a dominator at distance at most k from it.

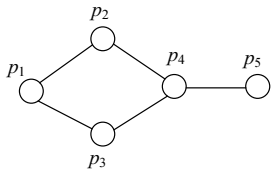


Fig. 1. A connected undirected graph, where $\{p_1, p_4\}$ is a dominating set, $\{p_2, p_3, p_5\}$ is a 2-dominating set, $\{p_1, p_3, p_4, p_5\}$ is a 2-tuple dominating set, and $\{p_1, p_3, p_5\}$ is a K-dominating set with $K = (2, 1, 2, 2, 1)$.

the set of legitimate states [7]. Self-stabilization provides a paradigm for designing a distributed algorithm that tolerates arbitrary transient faults. Several self-stabilizing protocols for the 1-dominating [8], [9], [10], [11], [12] and k-dominating [13], [14] set problem have been proposed (We refer the reader to [15] for a recent survey). However, to the best of the author's knowledge, there is no deterministic self-stabilizing algorithm for the K-domination problem[2].

This paper considers the minimal K-domination set problem under the framework of game theory. Game theory has been widely used in the study of strategies for players participating in a competition where players have conflicting benefits or goals. A game model consists of a player set, a strategy set for each player, and a utility function defined for each player. In the proposed approach, nodes modeled as autonomous players seek their own interest by making their strategies (decisions of being a dominator or not) based on local information. In this context, neighboring nodes have conflicting interest and one's strategy may cause subsequent responses (strategy changes) from neighboring nodes. Our goal is to design a decentralized self-stabilizing algorithm that identifies a minimal K-domination set through the use of incentives in the game. Although no centralized mechanism coordinates actions among these players, the mathematic framework provided by game theory helps us derive stability, correctness, and efficiency properties of the proposed game. Most existing self-stabilizing algorithms are expressed using Dijkstra's guarded command control structure [16], which imposes memory access constraints that make the realization of the game design as a distributed algorithm a challenging task. We identify the challenges and propose a solution. The proposed algorithm is proved weakly stabilizing [17] and can hopefully prevent the game from being trapped in inferior solutions. We conducted simulations to evaluate the proposed approach under various types of network topologies. Considered topologies include unit disk graph [18], the Erdös-Rényi model [19], the Watts-Strogatz model [20], and the Barabási-Albert model [21]. Simulation results indicate that the proposed approach outperforms existing 1- and k-domination algorithms in the size of dominating sets.

The contribution of our work is twofold. First, we devise a decentralized solution that finds a minimal multi-dominating set under the framework of game theory. The proposed approach is general in the sense that it also applies to single-domination, k-domination, and k-tuple domination problems. Second, we successfully convert the game design into guarded commands that are proved weakly stabilizing. This design paradigm would stimulate subsequent studies on related problems.

The remainder of this paper is organized as follows. Background knowledge and related work are presented in Section 2. Section 3 presents the proposed game-theoretic approach to the K-domination problem with rigorous proof. Section 4 discusses challenges when turning the game design into a distributed algorithm and presents a solution. In Section 5, performance evaluations of the proposed approach are presented in comparisons with other alternatives. Section 6 concludes this paper.

2 Background and Related Work

Distributed algorithms differ from each other in many aspects. One possible classification of distributed algorithms is based on the timing of events in the system [22]. Processes running a *synchronous* distributed algorithm perform communication and computation in perfect lock-step synchrony. On the other hand, processes executing an *asynchronous* distributed algorithm can take steps in an arbitrary order and at arbitrary relative speeds. Inter-process communication models also differ for different distributed algorithms. In the *shared memory* model, processes communicate with each other via common variables or registers. In the *message-passing* model, processes communicate by sending and receiving messages through communication channels.

Self-stabilization for a system can be defined with respect to a predicate over all states of the system [23]. The predicate under consideration specifies all correct or *legitimate* states of the system. A distributed algorithm is self-stabilizing with respect to the predicate if the following two conditions hold:

- *Convergence*. Starting from arbitrary state (possibly illegitimate), the algorithm eventually reaches a legitimate state.
- *Closure*. Any state following a legitimate state is also legitimate.

Self-stabilizing algorithms usually assume some type of execution models that can be characterized by the presence of a particular scheduler or daemon. In a *central daemon* execution model, only one process can execute at a time. With a *synchronous daemon*, all processes are scheduled to execute in parallel, which is a perfect match for a synchronous distributed algorithm. A *distributed daemon* subsumes the aforementioned models in the sense that any non-empty subset of processes

2. The approach proposed in [6] is a randomized algorithm and is not self-stabilizing.

can execute in parallel. Generally speaking, the major challenge in designing a distributed algorithm under a synchronous or distributed daemon lies in the handling of concurrent movements when such movements can break desired property (correctness or convergence) of the algorithm. A key to this handling is to break symmetry among nodes that are eligible to move, which can be done probabilistically [6] or by assigning a unique identifier to each node that represents its moving priority [10], [11], [12].

Many self-stabilizing algorithms for the minimal dominating set problem have been proposed in the literature. Due to space limitation, we present our literature review in Appendix A.

A game Γ can be formulated as $\Gamma = [P; \{S_i\}_{i=1}^n; \{u_i\}_{i=1}^n]$, where $P = \{p_1, p_2, \ldots, p_n\}$ is the *player set* consisting of all n participants in the game, $\{S_i\}$ defined for each p_i is p_i's *strategy set* (the collection of all p_i's feasible decisions), and u_i defined for each p_i is p_i's utility function with respect to some game result. The *strategy space* of the game $\Sigma = S_1 \times S_2 \times \cdots \times S_n$ is the Cartesian product of all strategy sets. A *strategy profile* $C = (c_1, c_2, \ldots, c_n) \in \Sigma$ is a tuple of n strategies, where $c_i \in S_i$. For a specific p_i, we may express C as $C = (c_i, C_{-i})$, where $C_{-i} = (c_1, c_2, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n)$ denotes the set of all player's strategies other than p_i's. Utility function $u_i(C)$ can be different for different player p_i, which represents p_i's payoff (utility) with respect to a particular strategy profile C. Players are selfish in the sense that the only goal of all the players is to maximize their own payoff. Therefore, the objective of the game can be expressed as $\max_{c_i \in S_i} u_i(c_i, C_{-i})$ for all $p_i \in P$.

The game under consideration is a *noncooperative dynamic* game. In a noncooperative game, players do not cooperate with each other to seek the system's benefit. A game is dynamic if players take turns to make their decisions. This is to model the asynchronous nature of process's execution and communication speeds. Here we assume that this game is of *perfect information*, meaning that every player knows what moves have already been made by all other players. Later we shall relax this assumption.

A Nash equilibrium is a strategy profile where no player can further increase its own utility by unilaterally changing its choice.

Definition 1 (Nash equilibrium): Given a game $\Gamma = [P; \{S_i\}_{i=1}^n; \{u_i\}_{i=1}^n]$, a strategy profile $C^* = (c_1^*, c_2^*, \ldots, c_n^*)$ is a Nash equilibrium if $\forall i \in \{1, 2, \ldots, n\} : \forall c_i \in S_i :: u_i(c_i^*, C_{-i}^*) \geq u_i(c_i, C_{-i}^*)$.

It is well known that Nash equilibria are not necessarily desired results. In fact, global optima in games may not even exist. Nevertheless, we can seek *Pareto optimal* results.

Definition 2 (Pareto optimal): A strategy profile $C = (c_1, c_2, \ldots, c_n)$ is Pareto optimal if and only if there exists no other strategy profile $C' = (c_1', c_2', \ldots, c_n')$ such that $\forall i \in \{1, 2, \ldots, n\} : u_i(C') \geq u_i(C)$ and $\exists j \in \{1, 2, \ldots, n\} : u_j(C') > u_j(C)$.

TABLE 1
Partial list of notations

Notation	Meaning
Γ	The proposed dominating game
n	Number of nodes
P	The set of all nodes; $P = \{p_1, p_2, \ldots, p_n\}$
K	The tuple of domination requirements; $K = (k_1, k_2, \ldots, k_n)$
N_i	The set of players that are p_i's neighbors
M_i	$M_i = \{p_i\} \cup N_i$
S_i	Strategy set of player p_i; $S_i = \{0, 1\}$ for all i
Σ	Strategy space; $\Sigma = S_1 \times S_2 \times \cdots \times S_n$
C	Strategy profile; $C = (c_1, c_2, \ldots, c_n) \in \Sigma$
$u_i(C)$	p_i's utility with respect to strategy profile C

Few studies in the literature blend game theory with self stabilization. Cohen et al. [24] proposed a distributed algorithm that allows selfish processes seeking their maximal payoffs to form a spanning tree. The proposed algorithm achieves weak stabilization, which means that starting from an arbitrary state, there exists at least one sequence of state transitions that leads the system to a stable state. Gouda [17] shows that weak stabilization of a system implies stabilization of the same system under some reasonable conditions.

3 Multi-Domination Game

3.1 Game Design

This section presents a game-theoretic distributed approach to the multi-dominating set problem. A realization of the proposed game in distributed systems shall be presented in the next section. The game design was inspired from and a generalization of our previous work on target coverage problem in wireless sensor networks [25]. We assume a distributed system consisting of n processes (nodes) interconnected by communication links. We define *multi-domination game* as follows. Let $P = \{p_1, p_2, \ldots, p_n\}$ denote the player set that contains all nodes in the system. Let $K = (k_1, k_2, \ldots, k_n)$ be an n-tuple of domination requirement, where $k_i \geq 1$ is the number of dominations required by node p_i. We assume that the domination requirement specified by K can be guaranteed if all nodes are dominators. This is a necessary condition for the existence of solutions to the multi-domination problem. A player's role of being a dominator or dominatee is represented by 1 or 0, respectively, so each player p_i has a strategy set $S_i = \{0, 1\}$. A strategy profile $(c_1, c_2, \ldots, c_n)$, where $c_i \in S_i$, can thus be coded as a bit vector $b_1 b_2 \cdots b_n$, where $b_i = c_i$, $1 \leq i \leq n$. The goal of the game design is to define $u_i(C)$ for every $p_i \in P$ such that the dominating game $\Gamma = [P; \{S_i\}_{i=1}^n; \{u_i\}_{i=1}^n]$ with objective $\max_{c_i \in S_i} u_i(c_i, C_{-i})$ for all $p_i \in P$ renders a self-stabilizing distributed approach that identifies a minimal multi-dominating set. For the ensuing discussions, most of the symbols used are summarized in Table 1.

Players in this game do not consider and plan for their opponent's reactions when choosing their strategies. The

reason is that the game does not presume any particular decision-making sequence among players to reflect the non-deterministic nature of event timing in an asynchronous distributed algorithm. Consequently, the number of possible reactions and sequences of subsequent responses corresponding to a particular strategy may be numerous. Therefore, it is not practical for players to perform backward induction. Instead, players in this game are *myopic* in the sense that a player chooses a strategy simply because that strategy increases its *current* utility. Formally, the *best response function* for player p_i is $r_i(C_{-i}) = \{c_i \in S_i | \forall c_i' \in S_i : u_i(c_i, C_{-i}) \geq u_i(c_i', C_{-i})\}$. Note that players can change their strategies whenever such changes increase their payoffs. It is theoretically possible that such strategy changes never stop (given some utility definition) and the game does not end up with a stable solution.

The proposed game design meets all the following requirements through the definition of a utility function for all players:

- *Self-stabilization*. Starting from any configuration, the game should always end up with a legitimate state (a state where the multi-domination requirement is met).
- *Small size*. The cardinality of the multi-domination set identified by the game should be as small as possible. The bottom line is: the resulting set should be minimal (i.e., it should not contain any subset that is also a multi-dominating set.)
- *Time efficiency*. The time for the game to stabilize should be reasonably short.

The task of defining $u_i(C)$ for every $p_i \in P$ centers on how much profit a dominator should gain. That profit should be sufficiently high to meet the domination requirement. On the other hand, the profit should be sufficiently low to minimize the size of the multi-domination set. Our design for the utility function attempts to capture the real contribution of a dominator. Formally, given $C = (c_1, c_2, \ldots, c_n)$, define

$$v_i(C) = \sum_{p_j \in M_i} c_j \tag{1}$$

for each player p_i, where $M_i = \{p_i\} \cup N_i$ is the *closed neighbor set* of p_i. Also define $g_i(C)$ as

$$g_i(C) = \begin{cases} \alpha & \text{if } v_i(C) \leq k_i \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where $\alpha > 0$ is a constant. The utility function of p_i is defined as

$$u_i(C) = \begin{cases} \left(\sum_{p_j \in M_i} g_j(C)\right) - \beta & \text{if } c_i = 1 \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where β is another constant such that $0 < \beta < \alpha$. Intuitively, a dominator p_i earns a profit of α from each closed neighbor of p_i when p_i's domination is really needed by that neighbor. What p_i gains from being a dominator is the total profit p_i earns minus a fixed cost β.

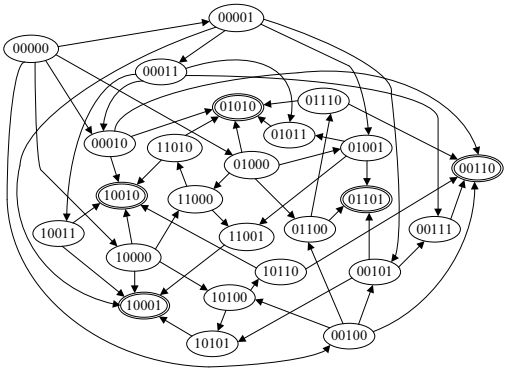


Fig. 2. Possible transitions of game states for the network topology of Fig. 1 with $K = (1, 1, 1, 1, 1)$

Since $\beta < \alpha$, p_i has the incentive to become a dominator as long as at least one closed neighbor of p_i needs p_i's domination. Clearly, the payoff of a player depends not only on its own choice, but also the choices of all its neighbors and neighbors of neighbors.

The dynamics of the game are quite complex even for small problem instances. For the network topology shown in Fig. 1, Fig. 2 shows all possible transitions of strategy profiles starting from 00000 with the defined utility function. The possible transitions of strategy profiles are also sensitive to the starting strategy profile. For example, transitions of strategy profiles starting from 11111 (not shown here) are different from those shown in Fig. 2. Starting from 00000, all transitions converge to five possible final results (10010, 10001, 01010, 01101, and 00110) and all these results are legitimate. Although these resulting sets are not of the same sizes (four sets are of size two while the other is of size three), these sets are all minimal. It is also not hard to verify that the maximum number of transitions is four, less than the number of nodes. Therefore, the self-stabilization, small size, and time efficiency requirements are all met in this particular example. The following subsections shall show whether these requirements are universally met with the designed utility function.

3.2 Proof of Self-Stabilization

The proposed approach corresponds to *silent* stabilizing [26], meaning that stabilization in Γ correspond to quiescent states. Therefore, the closure requirement for the proposed approach to be self-stabilization is trivially met. The convergence requirement corresponds to the following two properties:

P1 Starting from any strategy profile, Γ eventually ends up with a Nash equilibrium.

P2 Each Nash equilibrium in Γ is a K-dominating set.

We shall prove property P1 by showing that the proposed multi-domination game is an *exact potential game* [27].

Definition 3 (Exact potential game): $\Gamma = [P; \{S_i\}_{i=1}^n; \{u_i\}_{i=1}^n]$ is an exact potential game if it admits an *exact potential function* $\pi(c_i, C_{-i})$ such that

$$\forall p_i \in P : \forall c_i, c_i^* \in S_i, c_i \neq c_i^* :: \\ u_i(c_i^*, C_{-i}) - u_i(c_i, C_{-i}) = \pi(c_i^*, C_{-i}) - \pi(c_i, C_{-i}). \quad (4)$$

This definition indicates that we need to find an exact potential function to show that the proposed game is an exact potential game. This is the most difficult part of our analytic work. Note that a summation of all the player's payoffs $f(C) = \sum_{i=1..n} u_i(C)$ in the proposed game is not an exact potential function.

Theorem 1: The proposed multi-domination game $\Gamma = [P; \{S_i\}_{i=1}^n; \{u_i\}_{i=1}^n]$ is an exact potential game.

Proof: We prove the theorem by showing that the following function is an exact potential function:

$$\pi(C) = \left(\sum_{j=1}^{n} \sum_{k=0}^{v_j(C)} h_j(k) \right) - \beta \sum_{j=1}^{n} c_j, \quad (5)$$

where

$$h_i(k) = \begin{cases} \alpha & \text{if } 1 \leq k \leq k_i \\ 0 & \text{otherwise.} \end{cases}$$

Detailed proof is presented in Appendix B. □

Since the strategy space of Γ is finite, Γ is a finite exact potential game. It has been proved [27] that every finite exact potential game possesses a Nash equilibrium. Intuitively, any transition of strategy profile caused by the myopic behavior of a single player increases the value of the exact potential function $\pi(\cdot)$. Since the value of $\pi(\cdot)$ is upper bounded (refer to Theorem 5), the game eventually stops. Because we do not assume a particular initial configuration in the proof, *any* initial configuration of the game eventually leads to a Nash equilibrium.

Property P2 is not difficult to prove, as the following theorem shows.

Theorem 2: Every Nash equilibrium in Γ is a K-dominating set.

Proof: We shall prove that all Nash equilibria are K-dominating sets by way of contradiction. Suppose that there exists some Nash equilibrium $C = (c_1, c_2, \ldots, c_n)$ in Γ that is not a K-dominating set. It follows that there must be some p_i in P for which $v_i(C) < k_i$. Since $C_0 = (1, 1, \ldots, 1)$ is a K-dominating set, this means $\exists p_j \in M_i : c_j = 0$. For any such p_j, changing c_j from 0 to 1 can cause a payoff change from 0 to at least $\alpha - \beta > 0$ (which comes from the new domination on p_i). Therefore, C cannot be a Nash equilibrium, which contradicts with the assumption. □

3.3 Set Size and Time Efficiency

We present two theorems to show that the proposed approach finds rather small multi-dominating sets. The first shows that all K-dominating sets found are minimal (Theorem 3). The second shows that these results are Pareto optimal (Theorem 4).

Theorem 3: Every Nash equilibrium in Γ is a minimal K-dominating set.

Proof: Theorem 2 already shows that each Nash equilibrium is a K-dominating set, so we need only prove that each Nash equilibrium is also minimal. By way of contradiction, suppose that $C = (c_1, c_2, \ldots, c_n)$ is a Nash equilibrium in Γ that contains some proper subset that is also a K-dominating set. It follows that there exists some node p_i such that $c_i = 1$ and $\forall p_j \in M_i : v_j(C) > k_j$. Therefore, $u_i(C) = -\beta$ and changing c_i from 1 to 0 can cause a utility gain of β. Clearly, C cannot be a Nash equilibrium. □

Although a minimum K-dominating set must be minimal as well, a minimal K-dominating set may not have the smallest size. Pareto optima provide another way to exhibit the quality of the game. With a Pareto optimum, we cannot increase the utility of some player without sacrificing the utility of any others. Since utility functions defined in the proposed game capture effective contribution of dominators (redundant domination does not count), Pareto optima indicate good and desired results.

Theorem 4: Every Nash equilibrium in Γ is Pareto optimal.

Proof: First note that for any player p_i, its payoff is either 0 (iff $c_i = 0$) or $m\alpha - \beta > 0$ (iff $c_i = 1$), where $m = 1, 2, \ldots, |M_i|$. Suppose, by way of contradiction, that $C = (c_1, c_2, \ldots, c_n)$ is a Nash equilibrium but not Pareto optimal. This implies that there exists some strategy profile $C' = (c_1', c_2', \ldots, c_n')$ such that $\forall i \in \{1, 2, \ldots, n\} : u_i(C') \geq u_i(C)$ and $\exists i \in \{1, 2, \ldots, n\} : u_i(C') > u_i(C)$. Consider any player p_i such that $u_i(C') > u_i(C)$. Since $u_i(C) \geq 0$, $u_i(C')$ must be greater than 0 and hence c_i' must be 1. The condition $u_i(C') > u_i(C)$ and $c_i' = 1$ implies that there must exist some $p_j \in M_i$ for which $g_j(C) = 0$ and $g_j(C') > 0$. The condition $g_j(C) = 0$ implies $v_j(C) > k_j$ while $g_j(C') > 0$ implies $v_j(C') \leq k_j$. It follows that $\exists p_k \in M_j : c_k = 1 \wedge c_k' = 0$. Because C is a Nash equilibrium, it is impossible that $\forall p_l \in M_k : g_l(C) = 0$ (otherwise p_k would rather choosing $c_k = 0$). Therefore, $u_k(C) \geq \alpha - \beta > 0 = u_k(C')$, which contradicts with our assumption that $\forall i \in \{1, 2, \ldots, n\} : u_i(C') \geq u_i(C)$. □

It is possible to show that minimum K-dominating sets are Pareto optimal by a proof similar to that given in Theorem 4. However, the opposite does not hold generally. As an example, all the five Nash equilibria shown in Fig. 2 are Pareto optimal. Four of the equilibria are also minimum dominating sets but the other one, 01101, is not.

Theorem 5: The number of strategy profile transitions for Γ to reach a Nash equilibrium starting from any strategy profile is upper bounded by

$$\frac{\alpha \sum_{j=1}^{n} k_j - \beta \max_{j=1}^{n} \{k_j\}}{\min(\alpha - \beta, \beta)}.$$

Proof: First note that the minimum value of $\pi(C)$ is 0 (which occurs when $C = (0, 0, \ldots, 0)$). For each p_j, the

maximal value of

$$\sum_{k=0}^{v_j(C)} h_j(k)$$

for any C is $k_j\alpha$. Furthermore, the minimal value of $\sum_{j=1}^{n} c_j$ is $\max_{j=1}^{n}\{k_j\}$. Therefore, the maximal value of $\pi(C)$ is $\alpha\sum_{j=1}^{n} k_j - \beta\max_{j=1}^{n}\{k_j\}$. Now consider the minimal increment of $\pi(C)$ brought by any player p_i's strategy change. If $(c_i, c_i^*) = (0,1)$, the value of $\pi(C^*) - \pi(C)$ is at least $\alpha - \beta$ as indicated by (10). If $(c_i, c_i^*) = (1,0)$, (13) indicates that the increment of $\pi(C)$ is β. Therefore, the maximal number of strategy changes for Γ to reach a Nash equilibrium starting from any strategy profile is $(\alpha\sum_{j=1}^{n} k_j - \beta\max_{j=1}^{n}\{k_j\})/\min(\alpha-\beta, \beta)$. □

For MDS problem (i.e., $k_i = 1$ for all i), Theorem 5 indicates that the maximal number of strategy changes is no more than $(n\alpha - \beta)/\min(\alpha-\beta, \beta)$, which is $2n-1$ if $\alpha = 2\beta$.

The multi-domination game can be reshaped for the k-dominating set problem. The concepts are similar, though. We put all the details in Appendix C due to space limitation.

4 From Game to Algorithm

4.1 Challenges and Models

We must provide a feasible game-playing environment for the realization of the game-theoretic design because such an environment is not readily available in all distributed systems. This task faces several challenges. The first is to provide an algorithm-execution framework for the dynamic game model. The dynamic game model implicitly assumes that no two or more players make their decisions simultaneously. This assumption can be ensured with a central daemon. However, the proposed game is a graphical game [28], [29], meaning that not every player has conflicting interest with every others. In fact, two player's utilities are related only if they are neighbors or have some common neighbor. Therefore, the game design allows decision parallelism among some players. Nevertheless, converting the game design into a distributed algorithm that runs under a distributed or synchronous daemon requires considerable efforts not directly related to the game itself. This study assumes central daemon to simplify the presentation of algorithms.

The second challenge concerns the availability of perfect information in distributed systems. The assumption of perfect information in games states that whenever player p_k changes c_k, every other player p_i (every player p_i for which $M_i \cap M_k \neq \emptyset$ in our game) knows the latest value of c_k before p_i makes a potential response. This assumption may be reasonable in the message-passing inter-process communication model. For example, in a wireless ad hoc network, wireless nodes (as players) can broadcast their decisions with doubled transmission range to inform all neighbors of their neighbors within a reasonably short time. This model is similar to that discussed in [30]. However, most existing self-stabilizing algorithms are expressed in the form of guarded commands [16] that is grounded in the shared-variable model. Guarded commands impose two types of memory access constraints. The *read constraint* does not allow processes to read variables that are not owned by their closed neighbors. The *write constraint* does not allow processes to write variables that are not owned by them. Consequently, if $p_k \notin M_i$, c_k is not locally accessible to p_i (read constraint). One might argue that p_i can learn of c_k or the effect of c_k indirectly through some variable (say, x) that is owned by a common neighbor of p_i and p_k. However, the write constraint does not allow a consistent update of c_k and x at the same time. Consequently, players may not have up-to-date information when they make decisions. This limitation makes the proposed approach a game with *imperfect information* and may invalidate the stability property of the designed game.

4.2 Algorithm in Guarded Commands

We now present a distributed algorithm that realizes the multi-domination game using guarded commands. This algorithm runs under a central daemon and consists of five rules (R1 - R5) as shown below.

R1 $|M_i \cap \{p_j | x(j) = \textit{true}\}| < k_i \wedge g(i) \neq \text{UNDER} \rightarrow$
$g(i) := \text{UNDER}$

R2 $|M_i \cap \{p_j | x(j) = \textit{true}\}| = k_i \wedge g(i) \neq \text{EQUAL} \rightarrow$
$g(i) := \text{EQUAL}$

R3 $|M_i \cap \{p_j | x(j) = \textit{true}\}| > k_i \wedge g(i) \neq \text{OVER} \rightarrow$
$g(i) := \text{OVER}$

R4 $\exists p_j \in M_i : g(j) = \text{UNDER} \wedge x(i) \neq \textit{true} \rightarrow$
$x(i) := \textit{true}$

R5 $\forall p_j \in M_i : g(j) = \text{OVER} \wedge x(i) \neq \textit{false} \rightarrow$
$x(i) := \textit{false}$

Each guarded command specifies one rule that consists of a condition part (a Boolean expression) followed by an action part (statements). These two parts are separated by '$\rightarrow$'. A guarded command is *enabled* if its condition part is evaluated *true*. A process executes the action part of a command only when that command is enabled and the execution is scheduled by the daemon. When more than one commands are enabled, the daemon schedules one of them nondeterministically. Processes update their local states in action parts. The execution of the action part is assumed atomic, i.e., not interleaved with the execution of any other guarded command.

In the algorithm, each process p_i uses a local Boolean variable $x(i)$ to denote whether p_i chooses to be in the dominating set. It uses another variable $g(i)$ to deal with the read constraint. The value of $g(i)$ represents p_i's current knowledge of the quantitative relationship between k_i and the number of p_i's closed neighbors that

currently choose to be dominators. More explicitly,

$$g(i) = \begin{cases} \text{UNDER} & \text{if } |M_i \cap \{p_j | x(j) = \textit{true}\}| < k_i \\ \text{EQUAL} & \text{if } |M_i \cap \{p_j | x(j) = \textit{true}\}| = k_i \\ \text{OVER} & \text{if } |M_i \cap \{p_j | x(j) = \textit{true}\}| > k_i \end{cases} \quad (6)$$

Each player p_i decides $x(i)$ based on the $g(j)$ value of all its closed neighbor p_j. On the other hand, p_j updates $g(j)$ according to the $x(i)$ value of all its closed neighbors p_i. Because each rule updates only one local variable (either $x(i)$ or $g(i)$), and a central daemon schedules only one rule at a time, the values of $x(i)$'s and $g(i)$'s are not simultaneously and consistently updated. The inconsistency between local variables of the same process (for example, $x(i) = \textit{true}$ and $g(i) =$ UNDER for $k_i = 1$) can be prevented by rewriting the algorithm to consistently update all local variables in one command. However, inconsistencies between variables of different processes (for example, $x(i) = \textit{true}$ and $g(j) =$ UNDER for some $p_j \in N_i$ with $k_j = 1$) are inherent due to the write constraint. Consequently, this realization yields more states than the proposed game and allows many state transitions that are not possible in the proposed game. Note that the root cause of the inconsistencies comes from the read constraint. If p_i can directly read $x(k)$ for all $M_k \cap M_i \neq \emptyset$, then all $g(i)$'s are not needed and no extra states are introduced.

4.3 Stability and Other Properties

Although the proposed algorithm permits state transitions that are not possible in the proposed game, we can still prove weak stability [17] of the algorithm. Detailed proofs are given in Appendix D.

The introduction of extra state-transition paths in the algorithm may reduce the probability of the game being trapped in an inferior solution due to fast convergence of the game. Consider the example shown in Fig. 3. Suppose that $K = (1, 1, \ldots, 1)$ and nodes p_1, p_4, p_5, and p_7 have chosen to be dominators at some state of the game. In the proposed game, this configuration is a Nash equilibrium with a dominating set of size four. In the weakly-stabilizing algorithm, this configuration can further transit to another solution. In the example shown in Fig. 4, a possible state-transition path leads the system into a solution with only three dominators. Of course, it is also possible that the algorithm runs into an inferior solution. We conducted simulations to study this possibility. The results are presented in the next section.

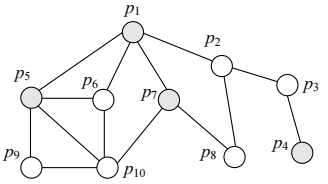


Fig. 3. A Nash equilibrium where nodes p_1, p_4, p_5, and p_7 have chosen to be dominators ($K = (1, 1, \ldots, 1)$).

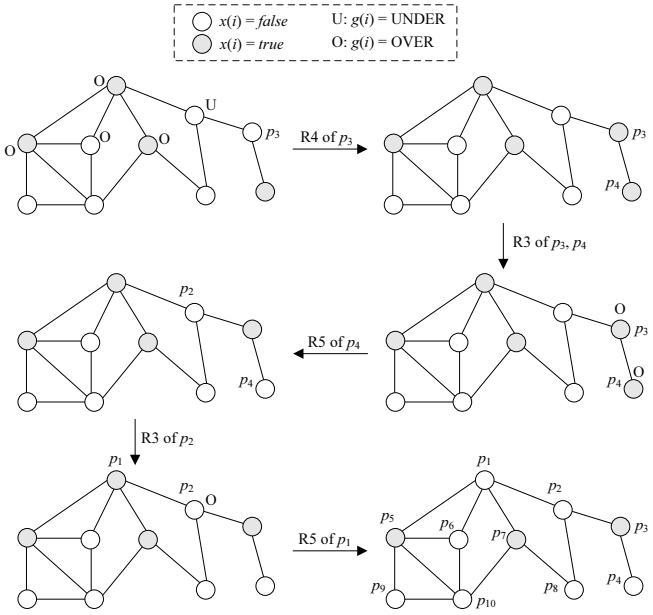


Fig. 4. A computation that changes the set of dominators ($K = (1, 1, \ldots, 1)$).

5 Simulation Results

We conducted simulations to study the performance of the proposed game-theoretic approach and compare it with those of existing methods. Since all approaches produced correct results, we were primarily concerned with the sizes of the dominating sets these methods generated. As the results heavily depend on network topology, we considered four representative types of network topologies in the simulations. These topologies are

- *Unit disk graph* (UDG) [18], which has been used to model network topologies of wireless ad hoc networks. In a UDG, two wireless nodes are neighbors only when they are within the transmission range of each other. In our simulations, n wireless nodes were randomly deployed in a 1000×1000 m^2 area. Each node has a default transmission range of 200 m.
- Random graph, which was introduced by Erdös and Rényi [19] (known as the Erdös-Rényi model or ER model) has been well studied for decades. In a random graph, whether an edge exists between two nodes is an independent event with probability p_e.
- The Watts-Strogatz model or WS model [20], which creates a small-world network that attempts to explain network dynamics such as six degrees of separation in a social network. In this model, each node has n_k links connecting to its n_k nearest neighbors, but each link has a probability p_r to be rewired to a randomly chosen node.
- The Barabási-Albert model or BA model [21], which generates a scale-free network that exhibits a power-

TABLE 2
Relative increases of the sizes of dominator sets with random initial states

Algorithm	UDG	ER	WS	BA
Hedetniemi et al. [8]	-2.1%	2.1%	-7.8%	-32.6%
Xu et al. [9]	-0.2%	-0.2%	-0.1%	0.6%
Kamei et al. [14]	-0.7%	-0.3%	-0.6%	-6.1%
Kakugawa et al. [10]	0.2%	0.1%	0.5%	0.7%
Turau [11]	0.0%	-0.1%	0.1%	0.2%
Goddard et al. [12]	0.3%	0.1%	0.1%	1.3%
WSA	0.1%	0.0%	0.1%	0.1%
Game	-0.3%	-0.3%	-0.2%	-0.2%

law distribution of node degrees. In this model, a network starts with a small number (m_0) of connected nodes. At every time step we add a new node x with m ($m \leq m_0$) edges. The probability of x being connected to a particular existing node y is proportional to the degree of y.

5.1 Single Domination

We considered several self-stabilizing distributed algorithms that find minimal dominating sets. These algorithms were respectively proposed by by Hedetniemi et al. [8], Xu et al. [9], Kakugawa et al. [10], Turau [11], and Goddard et al. [12]. We also tested the k-dominating set algorithm proposed by Kamei et al. [14] by setting k to 1. The proposed k-domination game and (hereafter denoted by Game) and the associated weakly-stabilizing algorithm (hereafter denoted by WSA), were also tested. Fig. 5 shows average sizes of dominating sets generated by each method in various types of topologies. Each average was obtained over 1000 runs.

Note that different algorithms might run under different types of daemons. Besides, Game demands perfect information while WSA achieves weak stabilization. Therefore, the results here were not obtained on a fair basis and should not be overstretched.

All classical self-stabilizing single-domination algorithms [8], [9], [10], [11], [12] performed nearly the same under all settings. These algorithms all generated more dominators than the k-dominating set algorithm proposed by Kamei et al. [14]. Kamei's algorithm is next to the proposed game-theoretic approaches (both Game and WSA). WSA outperformed Game in all settings (including those presented in the following). This confirms our conjecture that WSA is more immune to local optimal solutions than Game. The superiority of WSA over Game in the size of dominating set, however, comes at the cost of increased state transitions. Fig. 6 compares the average number of decisions made by a player between Game and WSA. The values of Game were all below 1.0 regardless of network topology. The values of WSA were all higher than those of Game, particularly in random graphs. This indicates the cost of WSA in identifying smaller dominating sets.

The above results were obtained by running each algorithm from scratch. That is, all nodes were dominatees initially and all variables were set to their default values at the beginning. Since all the algorithms are self-stabilizing, they can start from arbitrary states. We therefore repeated all the simulations but assigned randomly generated values to variables (nodes were randomly assigned the roles of dominators or dominatees and pointers were pointed to null or an arbitrary neighbor with equal probability). Table 2 lists the increases of the sizes of dominator sets in the new results relative to those in the old results in all settings. Almost all algorithms show little increases, indicating that these algorithms are not sensitive to initial conditions. The only exception is the algorithm proposed by Hedetniemi et al. [8], for which the increment can be up to -32.6% in the BA model. The algorithm by Kamei et al. [14] also show a significant difference in the BA model (-6.1%). The results indicate that the performance of these two algorithms is not independent of their initial states.

TABLE 3
Relative increases of the sizes of 2-dominator sets with random initial states

Algorithm	UDG	ER	WS	BA
Huang et al. (07) [31]	0.4%	-0.1%	0.4%	1.7%
Huang et al. (08) [32]	0.2%	0.3%	-0.1%	0.1%
Kamei et al. [14]	-0.5%	0.1%	-0.5%	-5.1%
Game	0.0%	0.3%	0.1%	-0.6%
WSA	0.2%	0.0%	0.1%	0.0%

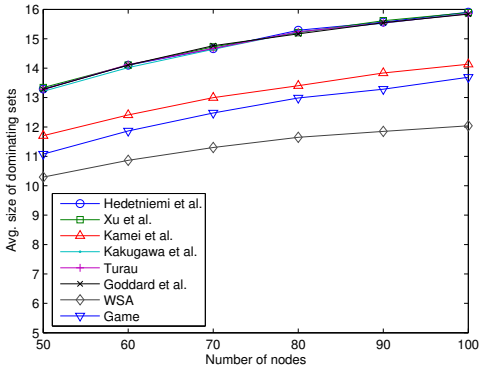
5.2 k-Domination

For k-dominating set problem, we have compared the proposed approaches with the algorithm proposed by Kamei et al. [14]. Fig. 8 in Appendix E shows the results for 2-dominating sets. Here the two algorithms proposed by Huang et al. [31], [32] are also considered. The results indicate that Huang's algorithm for central daemon [32] found more dominators than his algorithm for distributed daemon [31]. Kamei's algorithm performed better than Huang's algorithms but generally worse than the proposed approaches. The only exception is in the WS model, where Kamei's algorithm produced fewer dominators than Game when link rewiring probability p_r was less than 0.4. The superiority of WSA over Game is still observed in these results. We also tested these algorithms with random initial states. The result (Table 3) indicates that the performance of all algorithms (excepting the one by Kamei et al. in the BA model) is independent of initial states.

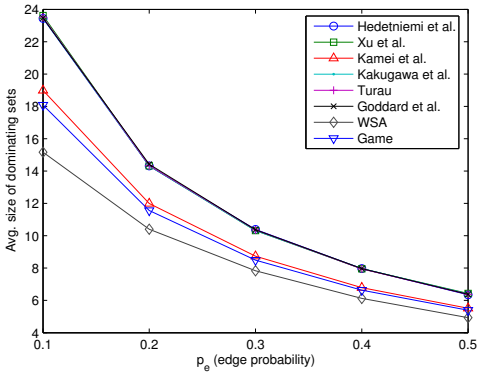
Figure 9 (Appendix E) compares Kamei's algorithm and the proposed approaches with k varied from 1 to 5. The result still shows the outperformance of WSA with a trend that Kamei's algorithm approaches the proposed ones with a large k.

5.3 Multi-Domination

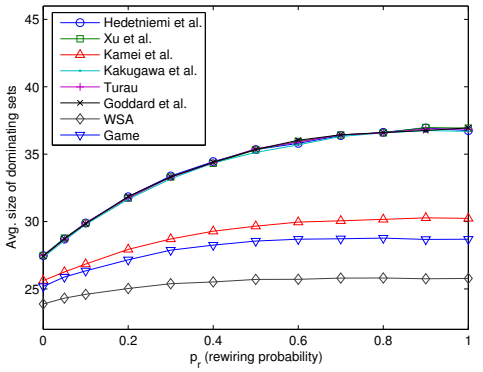
We ran the proposed approaches to find multi-dominating sets in various network topologies. The



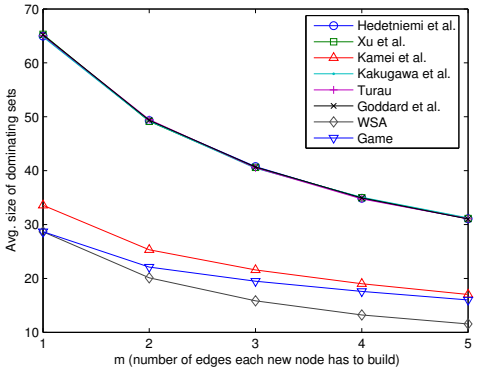
(a) UDG



(b) ER model (100 nodes)

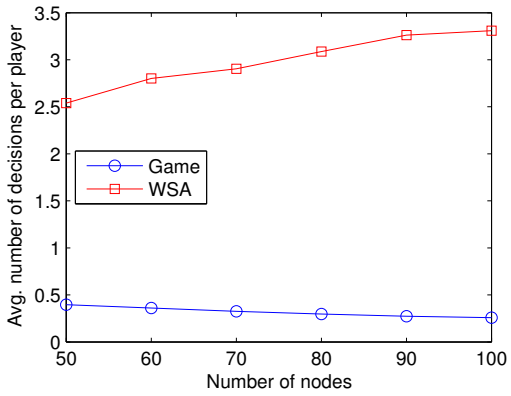


(c) WS model ($n_k = 4$; 100 nodes)

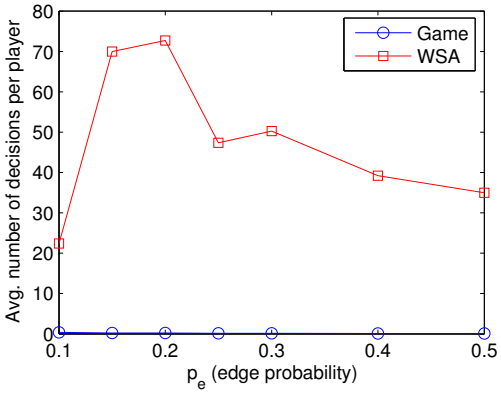


(d) BA model ($m_0 = 5$ forming a clique; 100 nodes)

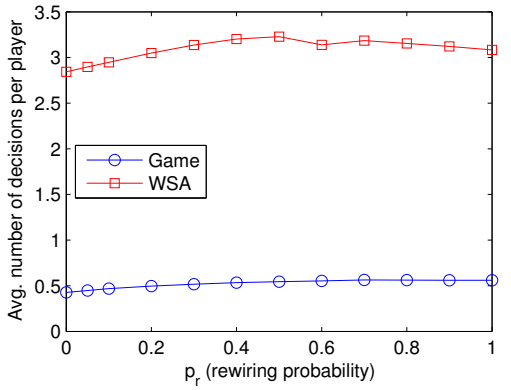
Fig. 5. Average sizes of 1-dominating sets in various types of topologies



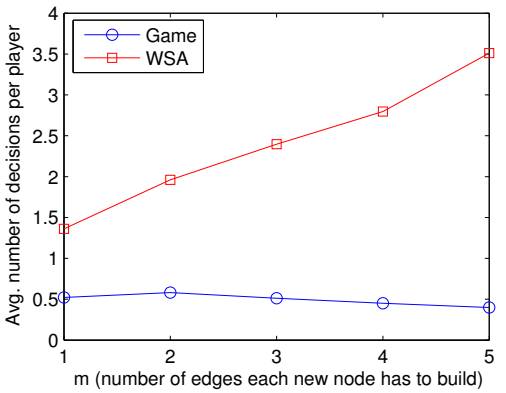
(a) UDG



(b) ER model (100 nodes)



(c) WS model ($n_k = 4$; 100 nodes)



(d) BA model ($m_0 = 5$ forming a clique; 100 nodes)

Fig. 6. Average number of decisions per player when generating 1-dominating sets in various types of topologies

domination requirement of each node p_i was varied by setting k_i to be a random integer within the range $[1, K_{\max}]$. Fig. 10 in Appendix E shows the average size of multi-dominating set with respect to the value of $K_{\max}$. As expected, the size of multi-dominating set increases with $K_{\max}$. The performance gap between Game and WSA increases with $K_{\max}$ in the WS model, while the gap remains unchanged or diminishes in other topologies.

6 Conclusions

We have proposed a game-theoretic approach to self-stabilizing algorithm that identifies multi-dominating sets in distributed systems. The utility function of each player (node) has been designed to guarantee that the proposed approach always ends up with a state of Nash equilibrium regardless of its initial states, any Nash equilibrium corresponds to a minimal multi-dominating set that is Pareto optimal, and the number of state transitions is bounded. However, game information is not fully available to distributed algorithms due to memory access constraints. A distributed algorithm that does not rely on the assumption of perfect information has been proposed and analyzed. This algorithm is proved weakly stabilizing and can hopefully prevent the game from being trapped in inferior results. Extended simulations were done, which considered four representative network topologies: UDG, the ER model, the WS model, and the BA model. The results show that the proposed approaches outperform existing algorithms in identifying smaller dominating sets, k-dominating sets, and multi-dominating sets. The algorithm finds smaller sets than the game at the cost of increased state transition times. In short, this paper has demonstrated the feasibility and efficiency of applying game theory to the design of self-stabilizing algorithms.

References

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman, 1979.

[2] S. Kutten and D. Peleg, "Fast distributed construction of small k-dominating sets and applications," *Journal of Algorithms*, vol. 28, no. 1, pp. 40–66, Jul. 1998.

[3] J. F. Fink and M. S. Jacobson, "n-domination in graphs," in *Graph Theory with Applications to Algorithms and Computer Science*. New York: John Wiley & Sons, 1985, pp. 283–300.

[4] ——, "On n-domination, n-dependence and forbidden subgraphs," in *Graph Theory with Applications to Algorithms and Computer Science*. New York: John Wiley & Sons, 1985, pp. 301–311.

[5] F. Harary and T. W. Haynes, "Double domination in graphs," *Ars Combinatoria*, vol. 55, pp. 201–213, Apr. 2000.

[6] L. Jia, R. Rajaraman, and T. Suel, "An efficient distributed algorithm for constructing small dominating sets," *Distributed Computing*, vol. 15, no. 4, pp. 193–205, 2002.

[7] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control," *Comm. ACM*, vol. 17, no. 11, pp. 643–644, Nov. 1974.

[8] S. M. Hedetniemi, S. Hedetniemi, D. P. Jacobs, and P. K. Srimani, "Self-stabilizing algorithms for minimal dominating sets and maximal independent sets," *Computers & Mathematics with Applications*, vol. 46, no. 5-6, pp. 805–811, Sep. 2003.

[9] Z. Xu, S. T. Hedetniemi, W. Goddard, and P. K. Srimani, "A synchronous self-stabilizing minimal domination protocol in an arbitrary network graph," in *Lecture Notes in Computer Science 2918*. Springer-Verlag, 2003, pp. 26–32.

[10] H. Kakugawa and T. Masuzawa, "A self-stabilizing minimal dominating set algorithm with safe convergence," in *Int'l Parallel and Distributed Processing Symposium*, Apr. 2006.

[11] V. Turau, "Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler," *Inform. Process. Lett.*, vol. 103, no. 3, pp. 88–93, 2007.

[12] W. Goddard, S. T. Hededtniemi, D. P. Jacobs, P. K. Srimani, and Z. Xu, "Self-stabilizing graph protocols," *Inform. Process. Lett.*, vol. 18, no. 1, pp. 189–199, 2008.

[13] S. Kamei and H. Kakugawa, "A self-stabilizing algorithm for the distributed minimal k-redundant dominating set problem in tree network," in *Proc. 4th International Conference on Parallel and Distributed Computing, Applications and Technologies*, August 2003.

[14] ——, "A self-stabilizing approximation algorithm for the distributed minimum k-domination," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, no. 5, pp. 1109–1116, 2005.

[15] N. Guellati and H. Kheddouci, "A survey on self-stabilizing algorithms for independence, domination, coloring, and matching in graphs," *J. Parallel Distrib. Comput.*, vol. 70, pp. 406–415, 2010.

[16] E. W. Dijkstra, "Guarded commands, nondeterminacy, and formal derivation of programs," *Comm. ACM*, vol. 18, no. 8, pp. 453–457, Aug. 1975.

[17] M. G. Gouda, "The theory of weak stabilization," in *Lecture Notes in Computer Science 2194*, A. Datta and T. Herman, Eds. Springer-Verlag, 2001, pp. 114–123.

[18] M. L. Huson and A. Sen, "Broadcast scheduling algorithms for radio networks," in *Proc. IEEE MILCOM*, 1995, pp. 647–651.

[19] P. Erdös and A. Rényi, "On random graphs I," *Publications Mathematicae, Debrecen*, vol. 6, pp. 290–297, 1959.

[20] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, pp. 440–442, Jun. 1998.

[21] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, Oct. 1999.

[22] N. A. Lynch, *Distributed Algorithms*. San Francisco, USA: Morgan Kaufmann Publishers, Inc., 1996.

[23] A. D. Kshemkalyani and M. Singhal, *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge, UK: Cambridge University Press, 2008, p. 634.

[24] J. Cohen, A. Dasgupta, S. Ghosh, and S. Tixeuil, "An exercise in selfish stabilization," *ACM Trans. on Autonomous and Adaptive Systems*, vol. 3, no. 4, Nov. 2008.

[25] L.-H. Yen, C.-M. Lin, and V. C. Leung, "Distributed lifetime-maximized target coverage game," *ACM Trans. on Sensor Networks*, vol. 9, no. 4, Jul. 2013.

[26] S. Dolev, M. G. Gouda, and M. Schneider, "Memory requirements for silent stabilization," *Acta Informatica*, vol. 36, pp. 447–462, 1999.

[27] D. Monderer and L. S. Shapley, "Potential games," *Games and Economic Behavior*, vol. 14, pp. 124–143, 1996.

[28] M. J. Kearns, M. L. Littman, and S. P. Singh, "Graphical models for game theory," in *Proc. 17th Conf. in Uncertainty in Artificial Intelligence*, 2001.

[29] D. K. D. Vickrey, "Multi-agent algorithm for solving graphical games," in *P The AAAI Conference on Artificial Intelligence*, 2002.

[30] T. Herman, "Models of self-stabilization and sensor networks," in *Lecture Notes in Computer Science 2918*, S. R. Das and S. K. Das, Eds. Springer-Verlag, 2003, pp. 205–214.

[31] T. C. Huang, J. C. Lin, C. Y. Chen, and C. P. Wang, "A self-stabilizing algorithm for finding a minimal 2-dominating set assuming the distributed demon model," *Computers & Mathematics with Applications*, vol. 54, no. 3, pp. 350–356, 2007.

[32] T. C. Huang, C. Y. Chen, and C. P. Wang, "A linear-time self-stabilizing algorithm for the minimal 2-dominating set problem in general networks," *Journal of Information Science and Engineering*, vol. 24, no. 1, pp. 175–187, 2008.

[33] E. J. Cockayne, R. M. Dawes, and S. T. Hedetniemi, "Total domination in graphs," *Networks*, vol. 10, no. 3, pp. 211–219, 1980.

[34] W. Goddard, S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani, "A self-stabilizing distributed algorithm for minimal total domination in an arbitrary system graph," in *Proc. 17th Int'l Parallel and Distributed Processing Symp.*, Apr. 2003.

Li-Hsing Yen received the BS (1989), MS (1991), and PhD (1997) degrees in computer science, all from National Chiao Tung University, Taiwan. He was with the Department of Computer Science and Information Engineering at Chung Hua University, Taiwan, from 1998 to 2006. Dr. Yen has been with the Department of Computer Science and Information Engineering, National University of Kaohsiung, Taiwan, since 2006 and is currently a full professor. His research interests include mobile computing, wireless networking, and distributed algorithms. Dr. Yen has won IEEE WCNC 2013 Best Paper Award. He has served on the editor boards of Springer's Wireless Networks and Hindawi's International Journal of Distributed Sensor Networks.

Zong-Long Chen was born in Kaohsiung, Taiwan, in 1988. He received the B.S. degree in computer science from National Pingtung University of Education, Pingtung, Taiwan, in 2011 and the M.S. degree in computer science from the University of Kaohsiung, Kaohsiung, Taiwan, in 2013. His research interests include distributed algorithms and game theory.