VIRTUAL BACKBONE FORMATION IN WIRELESS AD HOC NETWORKS

Hossein Kassaei

A thesis
in
The Department
of
Computer Science And Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

April 2010

Your file Votre référence
ISBN: 978-0-494-67117-7
Our file Notre référence
ISBN: 978-0-494-67117-7

Abstract

Virtual Backbone Formation in Wireless Ad Hoc Networks

Hossein Kassaei

We study the problem of virtual backbone formation in wireless ad hoc networks. A virtual backbone provides a hierarchical infrastructure that can be used to address important challenges in ad hoc networking such as efficient routing, multicasting/broadcasting, activity-scheduling, and energy efficiency. Given a wireless ad hoc network with symmetric links represented by a unit disk graph $G = (V, E)$, one way to construct this backbone is by finding a *Connected Dominating Set (CDS)* in G, which is a subset $V' \subseteq V$ such that for every node u, u is either in V' or has a neighbor in V' and the subgraph induced by V' is connected. In a wireless ad hoc network with asymmetric links represented by a directed graph $G = (V, E)$, finding such a backbone translates to constructing a *Strongly Connected Dominating and Absorbent Set (SCDAS)* in G. An SCDAS is a subset of nodes $V' \subseteq V$ such that every node u is either in V' or has an outgoing and an incoming neighbor in V', and the subgraph induced by V' is strongly connected. Based on most of its applications, minimizing the size of the virtual backbone is an important objective. Therefore, we are interested in constructing CDSs and SCDASs of minimal size.

We give efficient distributed algorithms with linear time and message complexities for the construction of the CDS in ad hoc networks with symmetric links. Since

topology changes are quite frequent in most ad hoc networks, we propose schemes to locally maintain the CDS in the face of such changes. We also give a distributed algorithm for the construction of the SCDAS in ad hoc networks with asymmetric links. Extensive simulations show that our algorithms outperform all previously known algorithms in terms of the size of the constructed sets.

Acknowledgments

I would like to thank all those who made this thesis possible with their perpetual support and encouragement

I owe my deepest gratitude to my supervisor, Dr. Lata Narayanan, for her invaluable guidance and insightful comments that helped me shape the direction of my research. She led many stimulating and thought-provoking discussions that were indispensable to the completion of my thesis.

I would like to express my sincere gratitude to Prof. Opatrny for his great contributions to this thesis. It was a great opportunity for me to interact with him during the course of my research and learn from his invaluable feedback and insightful reviews.

I wish to thank my good friend and colleague Mona Mehrandish for her close collaboration in the development of parts of the results in this thesis.

Finally, I would like to note that most of the results presented in Chapter 3 have been published in [27].

Contents

List of Figures

Chapter 1

Introduction

An ad hoc wireless network is an infrastructureless, peer-to-peer network of wireless nodes communicating with each other via multiple hops. By definition, an ad hoc network comes together when the need arises and achieves a goal without relying on any established infrastructure. It operates as a stand-alone network that may communicate with other networks or the Internet, but does not depend on them to accomplish its tasks. The devices participating in such a network may be of the same type or may be of different types as long as they all have the capability to wirelessly connect to other devices in the network. If two nodes are within the transmission range of each other, they can communicate directly; otherwise, a set of nodes between the two endpoints should forward their packets so they can communicate. This means that in an ad hoc network, any node must be able to play the role of a router in a conventional network. Although mobility is not explicitly part of the definition of a wireless ad hoc network, many appealing applications of ad hoc networking call for

the need to accommodate mobility. Thus, prominent classes of ad hoc networks, such as *Mobile Ad Hoc Networks (MANETs)* and *Vehicular Ad Hoc Networks (VANETs)* have emerged in which mobility is considered a key characteristic of the network. Another important characteristic of ad hoc networks is their resource constraints (bandwidth, computing power, battery lifetime, etc.) compared to traditional wired networks. The wide variety of envisioned applications as well as the highly challenging nature of this type of networking have resulted in a surge of interest in this field among researchers.

The numerous possible applications of ad hoc networks are expected to make them an indispensable part of our lives in the future. These applications include conferencing, home networking, emergency services, personal area networks, embedded computing, and sensor dust among others [41]. While some of these applications are predicted to emerge in the near future, others are thought to take much longer to become viable due to the large number of challenging issues that need to be addressed.

The challenges facing ad hoc networks have a very wide range, varying from regulations regarding the use of radio spectrum currently in place to scalability, routing, energy efficiency, security and privacy. In other words, there are various issues to be addressed across all the layers of the protocol stack as well as new regulations to be made and approved in order for ad hoc networking to emerge as an influential technology. Furthermore, appropriate interfaces should be developed to make it possible for ad hoc networks to interact with other networks or the Internet where and when necessary. Therefore, compatibility and interoperability are other key issues

that may need to be considered in the development of ad hoc networks depending on the application.

While many of the aforementioned challenges are inherently different and need to be addressed separately, there are issues which are closely interdependent and the best solutions for such problems are the ones that take into account those common aspects. One such group of crucial problems are the intertwined issues of efficient routing, scalability and energy conservation. Suppose in a proposed proactive routing scheme, all the nodes in the network should act as routers and forward other nodes' packets. This implies that every node should maintain a routing table and be ready to receive and forward packets at all times. Since all the nodes are supposed to act as routers, the size of the routing tables grow linearly with the network size. Given the limited computing resources of the wireless nodes, this forces a limit on how large the network size can grow. Moreover, the fact that the nodes should keep their radio interfaces on at all times means they get depleted faster, which, in turn, results in decreased network lifetime. However, if an efficient hierarchical approach is adopted in which only a small subset of nodes act as routers, the size of the routing tables shrink considerably. Furthermore, by periodically changing the membership of nodes in that subset, they are not depleted as quickly. Clearly, the second approach is more scalable and more energy efficient.

As briefly mentioned above, adopting a hierarchical infrastructure in an inherently flat ad hoc network can provide a very good solution to several problems. Indeed, creating this hierarchy, which can be used by many other protocols as an underlying

infrastructure, is the main focus of this thesis.

1.1 Backbone formation in Wireless Ad Hoc Networks

Many conceivable applications of wireless ad hoc networks imply very large-scale deployments of nodes, possibly in the hundreds or thousands. Wireless Sensor Networks (WSNs), as a very important class of ad hoc networks which are expected to revolutionize information gathering and processing in the near future, have even more demanding design requirements. In addition to very large-scale deployment, sensor nodes might be deployed in environments that preclude physical access to them such as disaster recovery or other inhospitable terrain. Due to such characteristics, in many applications discussed for sensor networks, replacing batteries of depleted sensor nodes is not an option and a node is considered dead forever once it runs out of power. Such strict requirements call for the design of very efficient, scalable and robust protocols.

The key to scalability and efficiency in traditional networks is the hierarchical organization of the network infrastructure. However, due to the lack of such an infrastructure, ad hoc networks are inherently flat. In order to achieve the desired scalability and efficiency in such a flat architecture, a lot of algorithms and protocols have been designed to rely on a virtual infrastructure, which organizes nodes into a hierarchy. One of the most popular of such hierarchical schemes is to form a virtual

backbone. For many practical purposes, this backbone should be at most one hop away from all the nodes in the network as well as being connected, which translates to the concept of a *Connected Dominating Set (CDS)* in graph theory. The complete definition of a CDS and relevant explanations will be given in the following section, however, we would like to first have a look at some of the applications of virtual backbones in the remainder of this section.

A virtual backbone in ad hoc networks is used as an underlying infrastructure by many protocols for a wide range of key networking functions such as unicast, multicast, and broadcast routing as well as activity-scheduling and topology control.

One of the most significant applications of a virtual backbone is efficient routing. In general, the overhead for flat routing algorithms can grow faster than linearly as the network size increases, and in very large ad hoc networks, may result in serious scalability problems. The virtual backbone can efficiently narrow down the search space for a route to the nodes in the backbone and routing tables will be maintained only by those nodes, which will result in a significant reduction in message overhead associated with routing updates. The use of this approach has been extensively studied in several papers in the literature such as [7], [15], [16], [17], [42], [46], [48], and [50].

In position-based routing, messages are forwarded based on the geographical co-ordinates of the nodes. Intermediate nodes are selected based on their proximity to the message's destination. Using such a scheme, it is possible for a message to get stuck by reaching a local maximum; i.e. it might reach a node whose neighbors are

all farther from the destination than itself. In this case, the routing algorithm must use a recovery procedure in which it backtracks to find another route. The authors in [18] propose that if messages are forwarded to the nodes in the dominating set, the recovery phase can be performed more efficiently.

A great challenge in multicast/broadcast routing and flooding is that many intermediate nodes unnecessarily forward a message. This redundancy which results in increased contention and collision in the network is referred to as the *broadcast storm problem* [38]. However, by using a virtual backbone, a large percentage of such redundant broadcasts can be eliminated. This approach has been investigated in several papers such as [7], [30], [31], [34], [44], [47], [51], and [52]. In [31], it is shown that the minimum cost flooding tree problem is similar to the MCDS (Minimum CDS) problem.

Nodes in wireless ad hoc networks are often battery-powered and thus have a limited energy supply. In many conceivable applications of ad hoc sensor networks, it is not even possible to replace batteries. Such restrictions necessitate the use of energy-aware protocols that minimize energy consumption and prolong network lifetime. CDSs play an important role in power management. They have been used to increase the number of nodes that can switch to sleep mode while preserving the connectivity of the network so that it can perform key functions such as routing. Several papers including [8], [19], [51], [52], and [56] have investigated this application of CDSs.

In very dense WSNs, a virtual backbone that includes the sink(s) can be used for

data gathering and dissemination as well as routing, activity-scheduling and topology information extraction. Additionally, in-network processing or data aggregation has been proposed as a way of conserving energy by reducing the volume of exchanged messages in sensor networks [35]. CDS nodes are ideal candidates to be used as data aggregation points in sensor networks.

Finally, the virtual backbone formed by the CDS can be used to propagate "link quality" information for route selection to provide quality of service in ad hoc networks [43]. In their work, this backbone is referred to as the *core* and the core extraction relies on CDS formation.

All these applications imply that in many cases the fundamental problem of constructing a backbone should be solved before anything else can be accomplished by the nodes in the network. However, the best algorithms proposed so far for CDS construction suffer from at least one of the following two problems: (i) poor scalability, and (ii) large size of the constructed CDS. This observation has formed the basic motivation of this thesis.

1.2 Problem Statement

In wireless networks, a wireless node A can directly communicate with another wireless node B if B lies in the transmission range of A. If the wireless nodes use omnidirectional antennas, then the network can be modeled as a disk graph in which nodes are the wireless devices. In this model, the disks represents the transmission range of nodes and there exists an edge from node u to node v if node v lies in the disk centered at node u.

If the wireless devices are homogeneous; i.e. have the same transmission range, the graph becomes a *Unit Disk Graph* (UDG). In such a context, the virtual backbone discussed above can be created by finding a *Connected Dominating Set* (CDS) in the underlying graph. Given an undirected graph $G = (V, E)$, a subset $V' \subset V$ is a CDS of G if for every node $u \in V$, u is either in V' or there exists a node v such that $(u, v) \in E$ and the graph induced by V' is connected. In other words, we need to find a subset S of the nodes in a graph such that every node in the graph is either in S or has a neighbor in S and the graph induced by S is connected. It is always desirable to minimize the size of the CDS. For example, in routing applications, a smaller sized CDS translates to lower routing information overhead. Similarly, in activity scheduling applications, this results in enabling more nodes to switch to sleep mode and conserve energy.

In reality, nodes in a network may not necessarily have the same transmission range. This might simply occur when the network consists of various kinds of wireless

devices with different powers and different functionalities. Even when the network consists of similar nodes, these nodes may need to adjust their transmission ranges for many reasons. For example, in many power control schemes, nodes adjust their transmission power to save energy, reduce collisions and so on. Similarly, in a topology control scheme, nodes adjust their transmission ranges in order to maintain a certain number of neighbors with the goal of improving spatial reuse. All of these scenarios result in introducing asymmetric links in the network. In such cases, the wireless network can be modeled as a *Disk Graph (DG)* rather then a UDG. In a disk graph $G = (V, E)$ a node $v_i \in V$ has a transmission range $r_i \in [r_{min}, r_{max}]$. If $d(v_i, v_j)$ denotes the Euclidean distance between the two nodes v_i and v_j, then there exists a directed edge $(v_i, v_j) \in E$ iff $d(v_i, v_j) < r_i$. In other words, there is a directed link from v_i to v_j only if v_j lies in the disk centered at v_i. An edge (v_i, v_j) is unidirectional if $(v_i, v_j) \in E$, but $(v_j, v_i) \notin E$. If $((v_i, v_j) \in E$ and also $(v_j, v_i)) \in E$, then the edge (v_i, v_j) is bidirectional. Figure 1 illustrates an example of a DG representing such a wireless network in which both unidirectional and bidirectional links exist. The dotted circles represent the transmission range of nodes, directed edges indicate unidirectional links, and undirected edges represent bidirectional links.

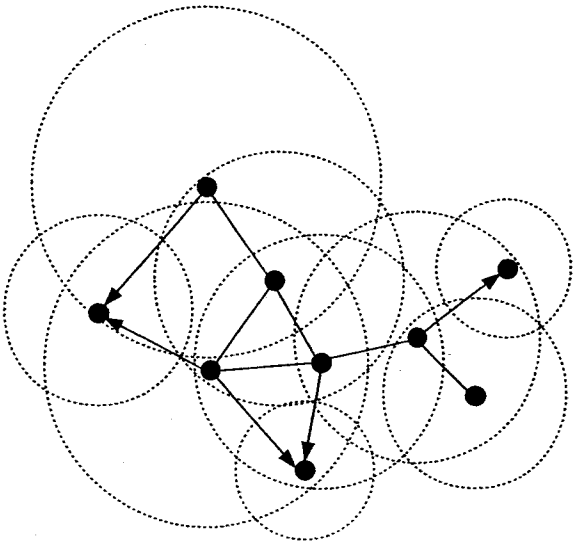


Figure 1: A disk graph representing a wireless network of nodes with different transmission ranges

Wu [46] extended the concept of dominating set in a UDG to a *Dominating and Absorbent Set (DAS)* in a DG for the first time. In a directed graph $G = (V, E)$, node u is a dominating neighbor of node w if $(u, w) \in E$, and node v is an absorbent neighbor of node w if $(w, v) \in E$. Node x both dominates and absorbs node v if (x, v) is a bidirectional edge in E. A set $V' \subset V$ is a *dominating* set of G if every vertex $v \in V - V'$ is dominated by at least a vertex $u \in V'$. Also, a set $V' \subset V$ is an *absorbent* set if every vertex $v \in V - V'$ is absorbed by at least a vertex $u \in V'$. A set is a DAS if it is both a dominating and an absorbent set. Figure 2 gives an example of the above definitions.

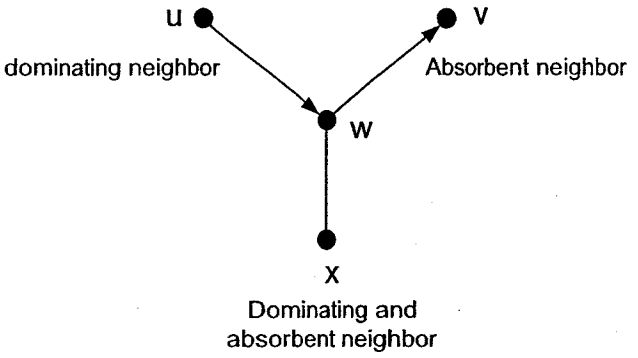


Figure 2: Dominating and absorbent neighbors of node w

Following the above definitions, backbone formation in DG involves finding a DAS which is *strongly connected.* A directed graph G is strongly connected if for any pair of vertices (u, v), there exists a path from u to v and also from v to u. Therefore, for directed graphs, we are interested in constructing a *Strongly Connected Dominating and Absorbent Set (SCDAS).* In other words, our goal is to generate a set $S \in V$ such that every node in the graph is either in S or has at least one dominator and one absorbent in S, and the subgraph induced by S is strongly connected. Also, just like in UDGs, we seek to minimize the size of this set.

Although minimizing the size of the backbone is our primary goal, it is not the only one. As mentioned earlier, ad hoc networks are known to have stringent requirements with respect to several protocol design criteria. First and foremost, the lack of centralized administration and the large scale deployments in many applications prohibit the use of centralized algorithms. Therefore, it is preferable to use a distributed

model in which nodes run a *distributed algorithm* autonomously and do not rely on a centralized coordinator. However, the decisions made by a node may depend on other nodes in the network. In addition, scalability requirements call for distributed algorithms with low time and message complexities. Furthermore, due to high frequency of topological changes, either induced by node mobility or node failure, algorithms that rely only on local information and are more robust to such changes are preferable. A *local algorithm* is a distributed algorithm in which a node makes decisions based on the information obtained through communication with nodes located no more than a constant (independent of the size of the network) number of hops away from it. This is different from a (non-local) distributed algorithm in that in the latter, the decisions made by a node may depend on the nodes that are arbitrarily far away from that node and therefore the size of the network may affect the way individual nodes make their decisions.

In summary, we are interested in designing efficient distributed/local algorithms that construct small-sized CDSs (SCDASs) in networks with symmetric (asymmetric) links and we want these algorithms to have low time and message complexities while exhibiting the required robustness in dealing with topological changes.

1.3 Summary of Contributions

Having discussed the problem statement in the previous section, we give a summary of our results in this section. They will be discussed in detail in chapters 3 and 4.

1. We give an efficient distributed algorithm with linear time and message complexity for the construction of CDS in wireless networks with symmetric links.

2. We give a local implementation of our algorithm for the construction of CDS in location-aware wireless networks with symmetric links.

3. We propose schemes to locally maintain the CDS in the face of topological changes, even in the absence of geographical location.

4. We extend our algorithm to construct SCDASs in wireless networks with asymmetric links.

Also, in each section, we present the main results of our extensive experimental work conducted to verify the efficiency of our algorithms in comparison with several classical algorithms in this area.

1.4 Outline of Thesis

In Chapter 2, we present a literature review on the connected dominating set (CDS) problem in networks with symmetric links and the strongly connected dominating and absorbent set (SCDAS) problem in networks with asymmetric links. In Chapter

3, we propose our algorithms for CDS construction in networks with symmetric links and analyze their performance and complexities. we extend our work to construct SCDASs in networks with asymmetric links in Chapter 4. Finally, Chapter 5 wraps up this thesis by some concluding remarks and pointing out possible directions for future work.

Chapter 2

Related Work

In most of the applications of CDS/SCDAS in ad hoc networks outlined in the introduction, it is desirable that the cardinality of the generated CDS/SCDAS be minimum. However, it has been proved in [23] that MDS and MCDS are NP-hard problems in general graphs. In [13], it has been proved that these problems remain NP-hard in UDGs. As a result, extensive research has been focused on designing approximation algorithms for these problems.

It has been proved in [21] that Chvátal's greedy algorithm's approximation ratio of $\ln \Delta$ [11] is a tight bound for the computation of DS in general graphs. However, in the case of UDGs, although MDS and MCDS problems remain NP-hard, the authors in [36], [1], and [7] showed that a constant approximation ratio is achievable.

For the MDS problem, the first algorithm running in polylogarithmic time with a non-trivial expected approximation ratio of $O(\log \Delta)$ and an approximation ratio of $O(\log n)$ with high probability was proposed by the authors of [26]. Nieberg and

Hurink [39] presented a *polynomial-time approximation scheme (PTAS)* for the MDS problem in UDGs. Their approach does not assume a geometric representation of the graph as the input. Given any graph as the input, their algorithm recognizes whether or not the input graph is a UDG. If so, it returns a dominating set with the approximation ratio of $1+\epsilon$. Otherwise, it returns a certificate indicating that the input graph is not a UDG. However, since the time complexity of their algorithm is $O(n^{c^2})$ with $c = O(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon})$, ϵ cannot be arbitrarily small in practice. Kuhn and Wattenhofer [29] gave a distributed algorithm using LP relaxation techniques to compute a dominating set of expected approximation ratio of $O(k\Delta^{\frac{2}{k}}\log\Delta)$ with time complexity $O(k^2)$, where k is an arbitrary constant and Δ is the maximum node degree.

A PTAS for computing MCDS in UDGs was proposed in [10]. One of the greatest contributions of this work is that it shows an MCDS in UDGs can be approximated to any degree given sufficient computing time. In the remainder of this chapter, we will study the general trends in the formation of CDS/SCDAS used by the state-of-the-art algorithms and will take a look at the algorithms and heuristics that are most relevant to our proposed algorithms. These algorithms, many of which have been simulated for performance comparison in this thesis, are among the most efficient algorithms in the literature.

2.1 Undirected Graphs

We have classified CDS formation algorithms into *centralized* and *distributed.* We first review the centralized algorithms in the following section and then discuss distributed algorithms in section 2.1.2.

2.1.1 Centralized Algorithms

Many algorithms designed in the literature are inspired by the seminal work of Guha and Khuller [24], which proposes two greedy heuristics with bounded approximation ratios for the construction of the CDS. The first algorithm grows a tree from the vertex with maximum degree using a greedy heuristic. The second algorithm grows separate components that form a dominating set and then connects them together using a Steiner tree. These two approaches later served as major techniques in CDS construction and were used in a number of distributed algorithms which will be explained in section 2.1.2.

The first algorithm initially marks all nodes white. It then starts to grow a tree T rooted at the node with the maximum degree (maximum number of white neighbors). The root is colored black and all its neighbors are colored gray. Then, the algorithm "scans" the gray nodes and their white neighbors iteratively, and selects the gray node or the pair of gray and white node with the maximum number of white neighbors. The selected node(s) are marked black and their neighbors are marked gray. The algorithm terminates when all the nodes have been marked either black or gray. At

termination, the set of black nodes forms a CDS with an approximation ratio of $2(1+H(\Delta))$, where H is the harmonic function, and Δ is the maximum node degree.

The second algorithm, which has two phases, adopts a completely different approach. In the first phase, all nodes are initially colored white. Whenever a node in included in the dominating set, it is colored black and all its *dominatees* (its 1-hop neighbors) are colored gray. A *piece* is defined to be either a connected black component or a white node. At each step of the first phase, a node which causes the maximum number of reduction in the number of pieces is selected to be colored black. This phase terminates when no white nodes are left. In the second phase, pairs of black components are recursively connected up by choosing a chain of two gray nodes until there is one connected black component. This black component forms a CDS of size at most $(ln(\Delta)+3).|OPT|$, where OPT is the minimum CDS.

The distributed implementations of both these algorithms, which we will review in section 2.1.2.1, were proposed in [16].

The authors in [37] proposed to use a *Steiner tree with minimum number of Steiner nodes (ST-MSN)* [9], [20], [32] to make the process of connecting MIS nodes more efficient. Their algorithm consists of two phases. In the first phase, an MIS is constructed which satisfies the following property: every subset of the maximal independent set is two hops away from its complement. All the nodes in the MIS are colored black and the rest of the nodes are colored gray. Since ST-MSN in the Euclidean plane is NP-hard [32], they use a 3-approximation algorithm to interconnect the MIS in the second phase. In this phase, a gray node that is adjacent to at least three black nodes

is colored black. If no such node exists, a gray node that is adjacent to at least two black components is colored black. At the end, the set of black nodes forms a CDS with an approximation ratio of 6.8 in UDGs.

While most of existing algorithms for the MCDS problem are based on selecting nodes to be part of the connected dominating set, the authors in [6] used the opposite method. Their algorithm assumes all the nodes are initially in the connected dominating set C. Also, all the nodes in C are initially marked as *non-fixed*. The *effective degree* of a node is defined to be the number of its non-fixed neighbors in C at any given time. At each step, a non-fixed node u with minimum effective degree is selected from C. If removing u from C makes the graph induced by C disconnected, then u is fixed; Otherwise, u is removed from C. Meanwhile, if u has no fixed neighbor in C, its neighbor with maximum effective degree is fixed. The algorithm terminates when there are no non-fixed nodes left in C. This algorithm does not provide a bound, but simulations show that the size of the generated CDS is desirably small.

2.1.2 Distributed Algorithms

In the context of ad hoc networks, centralized algorithms are usually not practical due to the lack of centralized administration and the large scale of these networks. Therefore, a lot of the research in this area has been focused on the development of distributed protocols, some of which are indeed the distributed implementations of centralized algorithms existing in the literature. In this section , we will briefly review some of the most relevant distributed algorithms that follow the typical trends

in CDS construction and attempt to do so as efficiently as possible.

In our review, we will focus on a number of key performance criteria such as time and message complexity, approximation ratio (if any), and the degree of locality in the algorithm; i.e. the size of the neighborhood of which a node need to be aware of in order to make its decisions. Some of the algorithms and heuristics discussed below have been proposed in the context of UDGs. Note that they work well for general graphs too, however, the performance analysis and bounds will not be applicable any more.

Topological changes in ad hoc networks, especially Mobile Ad Hoc Networks (MANETs) and WSNs, are quite frequent. These changes may occur as a result of nodes switching on or off, node mobility, node failure, and so on. Thus, a robust algorithm should accommodate such changes in network topology and provide the flexibility to deal with them as efficiently as possible. Among the algorithms that we will review, some have incorporated a maintenance phase to address such changes, while others have suggested the recalculation of the CDS. Therefore, in each case, we will also discuss how the algorithm deals with the issue of maintenance after the formation of CDS.

2.1.2.1 Greedy Algorithms

In a series of routing algorithms proposed by Das et al. in [16], [17], and [42], MCDS has been used as a virtual backbone or spine to provide for hierarchical routing. In order to form this virtual backbone, they propose the distributed implementations of

the two greedy algorithms of Guha and Khuller [24] discussed in section 2.1.1.

The first distributed algorithm finds a dominating set S in the first stage. In order to do so, an unmarked node u compares its effective degree $\delta(u)$, the number of unmarked neighbors, with the effective degree of all its two-hop neighbors ($N_2(u)$). Node u is added to S if $\delta(u) > \delta(u)$ for all $v \neq u$ in $N_2(u)$. Minimum node IDs are used to break ties. Once a node is added to S, it is marked. In the second stage, the algorithm connects all the components formed by $(u, dom(u))$ edges in a greedy manner. Before connecting components, all the edges that connect two nodes inside a component are discarded and the rest of the edges are assigned weights equal to the number of endpoints not in S. When connecting the components, edges of smaller weights are selected in order to minimize the number of connectors. At the end, the interior nodes in the resulting spanning tree form the CDS. This algorithm has a performance ratio of $2H(\triangle) + 1$ derived from its centralized version in [24]. Its time and message complexity are $O((n + |C|)\Delta)$ and $O(n|C| + m + n \log n)$ respectively, where C is the generated CDS and m is the cardinality of the edge set.

The second algorithm adopts a different approach by growing a connected dominating set C from a node with maximum degree. An *extension* to a fragment is defined to be either a one-edged or two-edged path consisting of one node in the fragment and one or two nodes, respectively, not in the fragment. The *effective combined degree* of an extension is defined to be the number of unmarked nodes adjacent to the non-fragment nodes in the extension. The extension with the highest effective combined degree is considered the best extension. The algorithm iteratively adds the

best extension in a greedy manner to form the final set C. This algorithm approximates C with a performance ratio of $2H(\Delta)$ in $O(|C|(\Delta + |C|))$ time, using $O(n|C|)$ messages.

In order to maintain the CDS in face of node mobility, Das et al. classify node movement into single-node movement and multiple-node movement. They propose a method to locally update the CDS in the case single-node movements. Multiple-node movements can also be treated as several single-node movement as long as the neighborhoods affected by those movements do not overlap. However, if the affected neighborhoods overlap, the CDS needs to be recalculated.

2.1.2.2 MIS-Based Algorithms

Maximal Independent Set (MIS) based CDS construction is perhaps the most popular method used in various forms in the literature. MIS-based algorithms, like the ones proposed in [1], [2], [14], [22], [25], and [57] might follow slightly or completely different approaches in how they build the MIS or later connect it up to form the CDS, but they all take advantage of the property that any MIS in UDGs has a constant approximation ratio. They use this property to guarantee constant approximation bounds for the generated CDSs.

Alzoubi et al. proposed a classical MIS-based algorithm in their seminal work in 2002 [1]. In this algorithm, an arbitrary rooted spanning tree T is first constructed. This spanning tree can be constructed through a leader election algorithm, such as the one presented in [12]. The tree T helps give rise to a global ordering through the

unique ranks assigned to nodes in the tree. The rank of a node is the ordered pair of its level (hop distance from the root of the tree) and its ID. Once the tree T has been constructed and the ranks assigned to nodes, all nodes are marked white and the root is marked black. Then, starting from the root and spreading out following the ranks, each node is marked black unless it already has black neighbors, in which case it will be marked gray. The marking process terminates when it reaches the leaf nodes. At this point in the algorithm, the set of black nodes forms an MIS, which is also a DS of the underlying graph. During the final phase, starting from the root, black nodes start joining the CDS and send an $INVITE$ message to their two-hop neighborhood. At each iteration, a black node and the gray node through which it received the $INVITE$ message for the first time will be added to the CDS until all black nodes have joined the CDS. The resulting CDS has a constant approximation ratio of 8, which is based on the following lemma bounding the size of an independent set in UDGs:

Lemma 1 *[1] The size of any independent set in a unit disk graph $G = (V, E)$ is at most $4 \times |OPT| + 1$, where OPT denotes the minimum CDS.*

The algorithm has a time complexity of $O(n)$, and a message complexity of $O(n \log n)$ dominated by the leader election phase. The creation of the required infrastructure (spanning tree) and the consequent formation of the dominating tree are serialized schemes which will not allow for locality of maintenance in this algorithm. As a result, a single change in the network topology necessitates the recalculation of the CDS. Its high message complexity is potential drawback of this algorithm.

To address the problems of the above algorithm, namely high message overhead, serial nature of the algorithm, and inability to provide local maintenance, Alzoubi et al. proposed a message optimal algorithm in [2]. Like the algorithm in [1], it first builds an MIS; however, in doing so, it does not use a spanning tree. Instead, the MIS is created through the following marking process. Any node with the smallest ID among its one-hop neighbors is marked black and sends a dominator message to its one-hop neighbors. Any node that receives a dominator message is marked gray and sends a dominatee message to all its one-hop neighbors. A node that receives a dominatee message form all of its neighbors is also marked black and sends a dominator message to its one-hop neighbors. Once all the nodes have changed their color, the set of black nodes forms a DS. In the second phase each dominator is connected to all dominators within three-hop distance. The nodes connecting any pair of dominators are referred to as connectors. The final CDS is the set of all dominators and connectors in the graph.

Alzoubi et al. presented another important lemma in [2]:

Lemma 2 *[2] Let S be any MIS of the UDG G and u be an arbitrary node in S.*

- *The number of nodes in S that are exactly two hops away from u is at most 23.*
- *The number of nodes in S that are exactly three hops away from u is at most 47.*

Using Lemmas 1 and 2, they prove that the above algorithm has an approximation

ratio of $192 \times |OPT| + 48$, which is much larger than their first algorithm. Experimental results from [5] and [25] also show that the average size of the generated CDS by the second algorithm is much larger than that of the first one as this algorithm generates a mesh-like CDS by connecting all two-hop and three-hop away dominators.

The algorithm in [2] provides local maintenance in face of topological changes. The key idea is to maintain the MIS and connectivity between all MIS nodes. Therefore, whenever a new dominator appears in a new vicinity, it must connect to all the dominators within three-hop distance. A connector maintains its status as long as it connects at least two dominators; otherwise, it will change its status to dominatee.

This algorithm has linear time and message complexity. It should be noted that the algorithm in [1] is an example of a class of distributed algorithms which are sometimes referred to as *single leader* algorithms in the literature, whereas the algorithm in [2] is an example of *multiple leader* algorithms. Typically, multiple leader algorithms exhibit higher degree of parallelism, have lower message overhead, and in the context of CDS formation, they can deal with topological changes locally.

Later, it was shown in [53] that the approximation factor of the algorithm in [1], can be improved to 7.8. In fact, Wu et al. [53], [54] showed that the bound on the size of the MIS in unit disk graphs can be improved from what was stated in lemma 1. Their main contribution is the following lemma:

Lemma 3 *[53] For any unit disk graph G, the size of a maximal independent set is at most* $3.8 \times |OPT| + 1.2$ *where OPT is the minimum CDS.*

This result was later used by several MIS based algorithms in the establishment

of bounds for the generated CDSs, as we will see in the rest of this section. However, Funke et al. [22] further improved this bound in 2006. Their main contribution was the following lemma:

Lemma 4 *[22] The size of any independent set in a unit disk graph G is at most* $3.453 \times |OPT| + 8.291$.

Therefore, all those algorithms whose approximation ratios are based on Lemma 3 can have improved approximation ratios using Lemma 4.

Basagni [4] proposed an algorithm that adopts the same MIS based approach in constructing a CDS. In this algorithm MIS nodes are called clusterheads. Their Distributed Clustering Algorithm (DCA) makes use of a generic weight assigned to nodes to give rise to a local ordering for the execution of the algorithm. The idea is that a node decides whether to become a clusterhead or not when all its neighbors with bigger weights have made their decisions. This weight can be adjusted to select clusterheads that have desirable properties based on a given application. Once an MIS is constructed using this cluster-based scheme, clusterheads that are at most three hops apart are connected up via intermediate nodes (gateways) to form a connected backbone. This algorithm has linear time and message complexity. Although no explicit approximation bound has been given for this algorithm, a constant approximation ratio like the one calculated in [2] is easily conceivable for this algorithm as well.

Extensive simulations were conducted in [5] to compare the performance of algorithms proposed in [2], [4], and [50] (explained later in section 2.1.2.1). It is shown

that DCA generates CDSs that are larger than those constructed by Wu and Li's algorithm [50] for relatively sparse networks and only a little smaller for more dense networks while it consistently generated larger CDSs that [1]. So they introduce sparsification rules in [5] to reduce the size of the backbone. The idea is to sparsify the CDS and generate a sparsified CDS that they call DCA-S, by breaking cycles of size 3 and 4, namely DCA-S(3), and DCA-S(4). The sparsification phase does not add much to the complexity of the algorithm, but does reduce the size of the CDS particularly as network density increases. DCA-S strikes a good compromise between [1] and [50].

In [25], Han presents a zone-based distributed algorithm that combines the zone and level concepts used by other algorithms in the literature to reduce the size of the CDS. In this algorithm, the network is first partitioned into different zones. Then a dominating tree is constructed in each zone. Finally adjacent zones are connected up by inserting bridges at zone borders to generate the final CDS.

In the following, the three phases of this zone-based algorithm are briefly explained. During the first phase, zones are formed and are assigned unique IDs. First, a node with the highest rank (rank could be ID or degree) among its one-hop neighbors becomes a dominator and sends a DOMINATOR message to all its neighbors. Any node receiving a DOMINATOR message for the first time marks itself as DOMINATEE and broadcasts a DOMINATEE message to all its neighbors. This procedure continues until all nodes become either dominator or dominatee. In this process, a node may become a dominator because it has the absolute highest rank among its

neighbors. Such a node is called a *seed dominator*. Or, it may become a dominator because its higher-rank neighbors have already become dominatees. Such a node is called a *non-seed dominator*. In the partitioning phase, the ID of a seed dominator automatically becomes the zone ID and is sent to all the nodes in the DOMINATOR message sent by the seed dominator. A dominatee also includes this information in the DOMINATEE message that it broadcasts. Using these messages, the non-seed dominator can also understand what zone they belong to. In case of receiving different zone IDs from their neighbors, non-seed dominators can select which zone to join. Once the zone partitioning phase is over, the second phase can begin. Note that by the end of this phase, the set of seed and non-seed dominators form an MIS.

Since the main goal of this algorithm is to reduce the size of the CDS, instead of building a dominating tree for the whole graph, it adopts a hierarchical approach in connecting the MIS. In the second phase, dominating trees rooted at the seed-dominators are formed inside zones. To build this tree inside each zone, a *level*, which is the hop distance from the root of the tree, is assigned to every node. Then every dominator broadcasts a ONE-HOP-DOMINATOR message containing the node IDs, zone IDs and levels of all its one-hop away dominators. By doing this, every dominator will know its two-hop away dominators and every dominator connects to exactly one such dominator in its zone by choosing the path that contains dominatees of the highest rank, thereby making them connectors.

In the third phase, zones need to be connected. A *border dominatee* is defined to be a dominatee which has a neighbor with a different zone ID. A *border dominator* is a

dominator that has a two-hop or three-hop away dominator with a different zone ID. In order for the dominators to decide if they are border dominator, border dominatees broadcast a TWO-HOP-DOMINATOR message including the node and zone IDs of their two-hop away dominators. Using this information, border dominators can connect to neighboring zones by connecting to their two or three-hp away dominators from a neighboring zone. The dominatees on the paths connecting two zones will become connectors. At the end of this final phase, the set of all dominators and connectors forms a CDS.

This algorithm has linear time and message complexity and produces a CDS of size at most $163.4 \times |OPT| + 43$. Also, extensive simulations were conducted that showed their algorithm outperforms the algorithms in [2] in terms of the size of the resulting CDS. For the maintenance of the CDS, Han suggests techniques similar to what is proposed in [2] to deal with topology changes.

In [57], an energy-aware MIS-based CDS construction algorithm is proposed. In this algorithm, every node is assigned a weight, which is a function of its degree and remaining battery power. As with all MIS-based algorithms, they first build an MIS through an iterative marking process which begins with an initiator. The marking process is basically similar to what is proposed in other algorithms in this category such as [1] with a slight modification. Unmarked nodes that receive a DOMINATEE message compete to become a dominator instead of having all nodes receiving a DOMINATEE message for the first time immediately becoming dominator. By doing so, they slightly reduce the size of the MIS and the resulting CDS compared to [1].

In the second phase, they use a greedy scheme in connecting up the MIS using the property that in the generated MIS, every MIS node has at least one two-hop away MIS node. The greedy scheme is to have every MIS node select a non-MIS neighbor of the highest weight as a connector. The set of the MIS nodes and connectors forms a CDS with an approximation ratio of 7.6. In establishing this bound, they argue that the number of connectors in the second phase does not exceed the size of the MIS and using Lemma 3, the resulting CDS has an approximation ratio of $7.6 \times |OPT| + 2.4$. They also conduct simulations which show their algorithm produces slightly smaller CDSs than [1] and is also more energy-efficient.

The authors claim that the time and message complexity of this algorithm is $O(n)$, however, this analysis holds only under the assumption that the initiator in the first phase is designated beforehand and the need for a leader election phase is obviated. Otherwise, the message complexity would be dominated by the complexity of the leader election phase which is $O(n \log n)$. As a serialized algorithm, it will not be able to handle topological changes locally and a recalculation of the solution should be performed.

The first local algorithms with constant approximation ratios for both the dominating set and connected dominating set were proposed in [14]. In their algorithms, they assume nodes are aware of their geographical coordinates. Using this information and a tiling scheme, they impose an ordering on the local execution of the algorithm that enforces a constant bound on the time complexity of the algorithm. In the tiling scheme, the plane is divided into tiles of twelve hexagons of diameter one as depicted

in Figure 3.

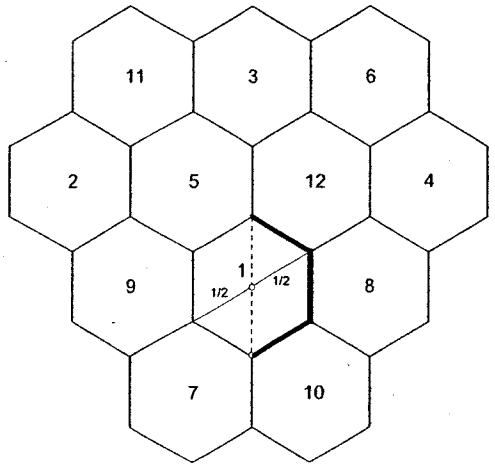


Figure 3: A tile divided into 12 hexagons of unit diameter. The bold edges belong to hexagon 1.

The arrangement of tiles in the plane is such that any two nodes of the same class number are either adjacent or at distance greater than two. This guarantees that the selection of a dominator in one hexagon does not influence the selection of a dominator in another hexagon of the same class number.

Initially, nodes calculate their class numbers using their coordinates and the tiling information and communicate this information with their neighbors. The selection of dominators starts with nodes of class number 1 and proceeds in an ascending order of class numbers until nodes of class number 12 finish executing the algorithm. The way in which this is implemented is that nodes can only start running the algorithm once all their lower class neighbors have terminated the algorithm and sent them the results. At each round, the set of nodes in a given hexagon that have not yet been dominated form a candidate set. The node which is closest to the center of the

hexagon among the candidates is designated as the dominator and the result is sent to all higher class neighbors waiting for it. Note that since a node in a hexagon dominates all other nodes in that hexagon, at most one node is selected in each hexagon for the dominating set. Clearly, a hexagon whose nodes have all been already dominated will not contain any dominator. At the end of the last round, the set of dominators forms a DS, which is also an independent set of the underlying graph, with a competitive ratio of 5.

It is important to note that the selection of a dominator in a given hexagon depends only on the information received from nodes of lower class number. This implies that the longest chain of dependency would be 11 (nodes of class number 12 depend on the information received from 11 hops away). In other words, the algorithm terminates in constant time, regardless of the number of nodes in the network.

Once the MIS is computed, the next step is to connect it using nodes called *bridges* in [14]. At the beginning of this step, a coordinator is elected in each non-empty hexagon by the nodes in that hexagon using some leader election algorithm. Then the coordinators find bridges in a greedy manner to connect the dominators. A bridge could be a vertex or an edge that connects two different connected components. Note that these components are separate from the standpoint of the coordinator (by looking at its k-hop neighborhood), and might not be necessarily two separate components from a global perspective. The greedy algorithm first tries to connect the components using vertices (bridges of size 1), and then by using bridges of size 2 if necessary. The order of execution follows the class numbers in an ascending order, just as in the DS

algorithm. Finally, the set of dominators and bridges forms a CDS with a competitive ratio of $7.453 + \epsilon$, where ϵ could be arbitrarily small. Clearly, as a local algorithm, it can maintain the CDS locally in face of topological changes as a result of node mobility or nodes switching on/off.

More recently, in [45], a local PTAS for the minimum dominating and the connected dominating set problems in location aware UDGs was presented. The locality distance of their algorithm for the connected dominating set is smaller than that of [14], but their dominating set algorithm has a much larger locality distance. For example, in order to achieve the same approximation ratio of 5 as in [14], they use a locality distance of almost 917 times larger. Furthermore, construction of the connected dominating set is entirely dependent on the dominating set in that it uses the latter as an input. In summary, they show that, theoretically, a $1 + \epsilon$ approximation ratio for the construction of DS and CDS is feasible. However, it is not a practical algorithm.

2.1.2.3 Pruning-Based Algorithms

Wu et al. proposed a simple localized algorithm in [50] for the construction of CDS in general graphs. Initially, all nodes are unmarked. Also, nodes exchange their neighborhood information with all their one-hop neighbors. As a result, each node knows all its two-hop neighbors. The algorithm is based on a marking rule: every node with two unconnected neighbors marks itself as a dominator. The set of dominators (marked nodes) forms a CDS which usually contains a large number of redundant

nodes. To address this problem, they propose two pruning rules to reduce the size of the set returned by their algorithm. These two rules are applied to the nodes in the CDS. The first rule removes a node u if it has a neighbor v with a higher ID in the CDS that covers all neighbors of u. The second rule states that a node u can be pruned from the CDS if it has two connected neighbors v and w with higher IDs such that v and w cover all of u's neighbors. Note that the role of IDs in these two rules is to avoid the simultaneous removal of neighboring nodes in the CDS. This scalable algorithm is very simple and has a low message complexity that gives it particular practical merits, but can generate a CDS that is quite large. Its approximation factor is $\frac{n}{2}$ as shown in [1]. The time and message complexity of this algorithm are $O(\Delta^3)$ and $\theta(m)$, respectively, where Δ is the maximum node degree and m is the number of edges in the graph. One major advantage of this algorithm is its locality of maintenance in which only the neighbors of a node need to update their status in case that node switches on/off or moves.

In [15], Dai and Wu proposed a generalization of the two existing rules referred to as *Rule K*, in which a node u unmarks itself if it has K connected neighbors with higher IDs that cover all of u's neighbors. Since Rule K needs global information, they restricted Rule K to only consider immediate neighbors.

Wu et al. extended the work in [15] to calculate a power-aware CDS that intends to prolong the average life span of a host while reducing the size of the resulting CDS in the pruning phase [49]. They achieve this goal by replacing *energy level* as a new parameter instead of ID in the pruning rules described above. IDs will only be used

to break ties between nodes that have the same energy level.

Butenko et al. proposed a purely pruning-based greedy heuristic in [6] for the construction of CDS in general graphs. The centralized description of the algorithm was explained in section 2.1.1. They also give a distributed version of that algorithm in [6]. Initially all the node are in the CDS. The algorithm starts from the node with the lowest degree in the graph. This node can be selected using a leader election algorithm such as [12] modified with the property that the leader should have the minimum number of neighbors. Let node u be the node currently running the algorithm. If the removal of node u results in disconnectivity in the graph (can be verified by running distributed BFS/DFS), the node u marks itself black and selects its neighbor with minimum *effective degree* (number of non-black neighbors) to run the algorithm. On the other hand, if node u is allowed to remove itself from the set, then its black neighbor v will select its neighbor with minimum effective degree to run the algorithm. If u does not have a black neighbor, then it selects a non-black neighbor with minimum effective degree to run the algorithm. Once all the last node terminated the algorithm, the set of black nodes forms the *final* CDS. Note that in this algorithm, there exists a CDS from the very beginning and at each round, it possibly becomes smaller until the final CDS is formed by the set of black nodes.This algorithm has a time complexity of $O(n \log^3 n)$ and a message complexity of $O(nm + n^2 \log^3 n)$. Clearly, its time and message complexities make it an unlikely option in the context of ad hoc networks. Moreover, it cannot handle topological changes locally as it needs to run a distributed BFS/DFS to determine the connectivity of the whole graph. Finally, they conduct

simulations that show that their algorithm outperforms the algorithm proposed in [1] in terms of the size of constructed CDS.

In [28], a new constant-approximation local algorithm for location-aware UDGs was proposed. The network model and the tiling used in this algorithm to enforce locality is similar to the algorithm proposed in [14], which was discussed in the category of MIS-based algorithms. However, the key difference between the two algorithms is in how they construct the CDS. While [14] follows the MIS based approach in constructing the CDS, the algorithm in [28] starts with a connected CDS and maintains connectivity throughout the execution. It then uses an efficient pruning test to prune away the redundant nodes in the CDS.

This algorithm uses a local approximation of *minimum spanning tree (MST)* as a guideline in constructing the CDS. Nodes run the algorithm in the exact same order as described in [14]. Initially, a coordinator is elected locally in each hexagon (using a leader election algorithm). The nodes in hexagon i proceed as described in the following. The set of nodes with an edge in the spanner to a higher class number hexagon form a *local CDS* candidate set. The coordinator selects node(s) from this set based on some heuristic. The four proposed heuristics all attempt to minimize the number of nodes selected from this set based on some criteria such as node degree, proximity to the center and so on. Once the nodes from the candidate set have been selected by the coordinator, they send a message to their neighbor(s) in the higher class number hexagon asking them to select a dominator among themselves as the other end of the edge. This is done to ensure connectivity between hexagons. Once

the nodes of class number 12 (hexagon 12) finish running the algorithm, the set of all selected nodes in all hexagons forms a CDS.

To reduce the size of the resulting CDS, a pruning test is performed at every CDS node: node u can remove itself from the CDS if (i) all its dominatees have at least one other dominator and (ii) the subgraph induced by those neighbors of u that are in the CDS is connected.

Extensive simulations conducted in [28] show it outperforms its MIS-based counterpart in [14] in terms of the size of the resulting CDS and it can handle the maintenance of the CDS locally.

2.2 Directed Graphs

In an ad hoc wireless networks, some links may be unidirectional for several reasons. Node in the network may have different powers and transmission ranges due to different functionalities or they may adjust their transmission range for topology control purposes and so on. The hidden terminal problem can be another reason for the (temporary) existence of unidirectional links in the network. In such cases, the network is modeled as a *disk graph (DG)* rather than a UDG.

2.2.1 Wu's distributed Algorithm

Wu [46] extended the concept of a dominating set in undirected graphs to a *dominating and absorbent set* in directed graphs. Wu gave a simple local algorithm for the

computation of *strongly connected dominating and absorbent set (SCDAS)* in directed graphs. Initially all the nodes are unmarked. A node u is marked if there exists a node v in its dominating set and a node w in its absorbent set, but v does not dominate w. In other words, a node is marked only if it lies on the shortest path from one neighbor to another. Just as in the marking algorithm of [50], this marking rule generates a large SCDAS with a lot of redundant nodes. Therefore, Wu proposed two rules to reduce the size of the constructed SCDAS. These two rules are indeed the extended forms of the rules proposed in [50] that are applied to the nodes in the SCDAS. The first rule allows a node u to be removed from the SCDAS if its dominating (absorbent) neighbor set is covered by the dominating (absorbent) neighbor set of node v with a higher ID. The second rule allows a node u to be removed from the SCDAS if its dominating (absorbent) neighbor set is covered by the union of the dominating (absorbent) sets of the two connected nodes v and w provided that the ID of node u is the smallest among the three nodes.

Furthermore, two implementations of the two pruning rules are proposed. The first implementation, called *restricted implementation* requires two-hop information. This implementation requires u and v to be bidirectionally connected in the first rule and v and w to be neighbors of node u in the second rule. However, a *general implementation* that does not require nodes u, v and w to be neighbors requires three-hop information. This algorithm does not provide a constant approximation ratio and its time and message complexities are $\Theta(m)$ and $O(\Delta^3)$ respectively. Finally, local maintenance procedures were proposed by Wu to deal with topological changes.

2.2.2 Park's centralized Algorithms

In [40], Park et al. propose a centralized constant approximation algorithm for the construction of a *Minimum SCDAS (MSCDAS)* in wireless ad hoc networks with different transmission ranges. In their work, it is assumed that the ratio of the maximum to the minimum transmission range is bounded. They also present two heuristics and evaluate their performance through simulations. The heuristics are indeed the counterpart of Guha and Khuller's algorithm [24] for undirected graphs.

In the constant approximation algorithm, an outgoing spanning tree and incoming spanning tree rooted at an arbitrary node u are constructed. The non-leaf nodes of the two trees form a SCDAS. The also give the following important lemma:

Lemma 5 *[40] In a directed graph $G = (V, E)$, the size of any Independent Subset (IS) is upperbounded by*

$$2.4(k + \frac{1}{2})^2 \times |OPT| + 3.7(k + \frac{1}{2})^2$$

where $k = \frac{r_{max}}{r_{min}}$ and OPT *is the minimum SCDAS.*

Using Lemma 5, they prove that the approximation ratio of the algorithm described above is $9.6(k + \frac{1}{2})^2 \times |OPT| + 14.8(k + \frac{1}{2})^2$. In other words, they show that the size of the SCDAS generated by this algorithm is at most four times the size of any IS in the directed graph.

Park et al. also present two centralized heuristics for the construction of SCDAS in directed graphs. Both of these heuristics rely on a DAS as the input. Therefore, we first discuss how they form the DAS. The algorithm that constructs the DAS consists

of two stages: construction of a DS and an AS. The union of the two sets then forms the final DAS. In finding the DAS, initially all nodes are marked as uncolored. Then, at each iteration, a node u is colored black and all its uncolored neighbors with an incoming edge from u (its dominatees) become gray. This process terminates when no uncolored node is left. The selection of the node to be marked black at each iteration can be based on two different criteria: (*i*) *random selection*, and (*ii*)*highest degree selection*, where the degree of a node is defined to be the sum of the number of its incoming and outgoing edges. At the end of this stage, the set of black nodes forms a DS. Before proceeding to the construction of AS, a preprocessing step is performed whose goal is to reduce the number of nodes selected as AS by checking if any gray node (a dominated node) is also absorbed by a black node.If there exist such node(s), they are colored white.

Once the preprocessing phase is finished, nodes in the graph are either black, gray or white. Since white nodes are already absorbed, the construction of AS is equivalent to finding an absorbent node for every gray node. In doing so, a greedy approach is adopted: at each iteration, a gray node that absorbs the highest number of gray nodes is marked black and its absorbed gray nodes are marked white. This stage terminates when no gray node is left, at which time the set of black nodes forms a DAS.

Then two heuristics are proposed to make the above DAS connected. The first one, called *greedy spider contraction algorithm (G-SCA)*, uses a greedy approach to find an approximation for the *directed Steiner tree with minimum Steiner nodes (DSMSN)*

problem to minimize the number of white nodes required to connect the black nodes. Let S denote the set of black nodes (DAS) returned by the above algorithm and r be an arbitrary node. The idea is to build an in-connected tree to r from every node in S and an out-connected tree the node r to every node in S. The union of the nodes in the two trees forms an SCDAS.

The second heuristic, called *greedy strongly connected component merging algorithm (G-CMA)*, iteratively finds two *Strongly connected Components (SCC)* which can be merged at minimum cost among all the pairs of SCCs, and merges them by coloring the white nodes on the two directed paths between the two SCCs black. The algorithm terminates when there is only one SCC left. The set of black nodes forms a SCDAS. They conduct simulations that show *G-CMA* consistently outperforms *G-SCA* in terms of the size of the resulting set. No time or message complexity analysis is given for the algorithms, but distributed implementations of these algorithms seem too expensive to be practical in the context of ad hoc wireless networks.

Chapter 3

Algorithms for Networks with Symmetric Links

In this chapter, we propose our algorithms for networks with symmetric links. In other words, it is assumed that if there exists an edge from node u to node v, then there also exists an edge from node v to node u. It should be noted that although in all the descriptions and explanations given in this section, it is assumed that the underlying graph is a UDG, the proposed algorithm is applicable to a larger class of graphs called *Disk Graph with Bidirectional links (DGB)*. In $DGBs$, nodes are not required to have the same transmission range. They may have different transmission ranges, but unidirectional (asymmetric) links are ignored.

We first describe our algorithm in a centralized manner in order to provide a better understanding of how it generates a connected dominating set. Then, we present both a distributed and a local implementation of the algorithm. Finally, we

present simulation results that compare the performance of our proposed algorithm with its competitors in each category. In all the simulations, it is assumed that the underlying graphs are $UDGs$.

3.1 Definitions and preliminaries

We consider a wireless network of homogeneous nodes where all nodes have the same transmission range. Node u is a neighbor of node v if and only if they are adjacent in the graph. We use N_u to denote the set of neighbors of node u, referred to as neighborhood of u. Every node u has a rank $(\delta(u), id(u))$ which is an ordered pair of its *effective degree* and *id*, where the effective degree of node u is the number of u's neighbors in CDS, i.e $\delta(u) = |\{v|v \in N_u \wedge v \in CDS\}|$. Since the membership of nodes in the CDS changes during the algorithm, so does the effective degree of a node. By assigning a unique *id* to every node, it is ensured that when comparing nodes' ranks, ties are broken.

It should be noted that the definition of a node's rank can be generalized to a generic weight function which is an ordered pair of a node's *weight* and its *id*. This weight can be defined based on the goal function. Since we intend to minimize the size of the generated set in this thesis, a node's weight is defined as its effective degree. As another example, if it is intended to generate an energy-aware CDS that prolongs the network lifetime, then the weight can be defined as the remaining energy level (battery power) of a node. Finally, note that the definition of the weight function

changes the order in which nodes run the algorithm and consequently the set of nodes in the resulting CDS.

3.2 Centralized Description

The CDS is the set of nodes with either *in* or *pending* status. Initially, all the nodes have a *pending* status. At each step, we select the node u with the lowest rank among the nodes with *pending* status. Node u, then determines whether or not it remains in the CDS, by running a local test. This local test consists of two sub-tests; the ***domination test***, and the ***connectivity test***.

Node u passes the domination test if all its neighbors have at least one other dominator. Node u passes the connectivity test if the subgraph induced by its neighbors that are marked as belonging to the CDS is connected. It is clear that both these conditions can be evaluated locally by node u using information obtained from its neighbors.

If a node passes both tests, its status changes to *out*; otherwise, its status changes to *in*. The algorithm terminates when there are no *pending* nodes left. The formal description of this algorithm is given in Algorithm 3.

It is easy to see that after the elimination of a node u that passes both the domination and connectivity tests, the set of nodes with *in* or *pending* status is still a CDS. Furthermore, since a node is removed from the set of *pending* nodes at every step of the algorithm, the algorithm terminates in n steps.

Algorithm 1 Centralized Connected Dominating Set (CDS) Algorithm

$CDS \leftarrow V$
$P \leftarrow V$
while $P \neq \emptyset$ **do**
  $u \leftarrow argmin\{(\delta(v), id(v)) | v \in P\}$
  $P \leftarrow P - \{u\}$
  $dominationTest \leftarrow true$
  for all $v \in N_u$ **do**
    if $((N_v \cap CDS) - \{u\} = \emptyset)$ **then**
      $dominationTest \leftarrow false$
    end if
  end for
  if $dominationTest$ **then**
    if $G[(N_v \cap CDS) - \{u\}]$ is connected **then**
      $CDS \leftarrow CDS - \{u\}$
      for all $(v \in N_u \wedge v \in CDS)$ **do**
        $\delta(v) \leftarrow \delta(v) - 1$
      end for
    end if
  end if
end while
Return CDS

Also note that since at any time during the execution of the algorithm, the set of nodes with *in* or *pending* status forms a CDS and all the nodes are initially in *pending* state, there always exists a feasible solution at any time during the computation of the CDS. This might be particularly useful in applications where protocol setup time is crucial.

3.3 Distributed Implementation

In order to convert the above algorithm into a distributed algorithm, every node needs to perform an initial setup which provides the essential information for the execution of the algorithm. First, every node u exchanges its rank with all its neighbors and stores the set of its neighbors in $N_{u,CDS}$, a variable holding the set of neighbors in CDS. Additionally, it maintains a list of its lower rank neighbors in $Lower\_Rank_u$.

During the algorithm execution, nodes exchange information about their dominators using *Dominator_Query* and *Dominator_Reply* messages which will be explained later. Initially, every node sets a local flag $Reply\_In\_Transit$ to $false$ indicating that no *Dominator_Reply* message that has been sent by it in response to a *Dominator_Query* message is currently in transit. Also, every node maintains a *Dominator Query queue* (*DQQ*) to store the incoming *Dominator_Query* messages to be able to reply to them in a first-in-first-out order. Initially, this queue is empty. Moreover, every node sets its status to *pending*. As in the centralized algorithm, each node has one of the three possible statuses, *in*, *out*, or *pending*; initially all nodes have *pending* status. When a node runs Algorithm 2, it changes its status to either *in* or *out* depending on whether it stays in the CDS or not. At any stage in the course of the execution of the algorithm, the set of nodes with *pending* or *in* status form a CDS. However, in the end, the set of nodes with *in* status form the final CDS since no node with *pending* status will be left when the algorithm terminates.

After the initial setup, every node u runs Algorithm 2. A node with the lowest rank

Algorithm 2 Distributed Connected Dominating Set Algorithm, executed by node u

when $Lower\_Rank_u = \emptyset$
$dominationTest \leftarrow true$
Send $Dominator\_Query$ to N_u
for all $(v \in N_u)$ **do**
  Wait for $Dominator\_Reply(v, D_v)$
  if $(D_v - \{u\} = \emptyset)$ **then**
    $dominationTest \leftarrow false$
  end if
end for
if $dominationTest$ **then**
  if $G[N_{u,CDS} - \{u\}]$ is connected **then**
    $Status_u \leftarrow out$
    Send $Finished\_Msg(out)$ to N_u
  else
    $Status_u \leftarrow in$
    Send $Finished\_Msg(in)$ to N_u
  end if
else
  $Status_u \leftarrow in$
  Send $Finished\_Msg(u, in)$ to N_u
end if

Upon receiving *Dominator_Query_Msg* from v:
if $!(Reply\_In\_Transit)$ **then**
  $Reply\_In\_Transit \leftarrow true$
  Send $Dominator\_Reply(u, D_u)$ to v
else Enqueue $Dominator\_Query(v)$ in DQQ
end if

Upon receiving *Finished_Msg(status)* from v:
if $status = out$ **then**
  $\delta(u) \leftarrow \delta(u) - 1$
  $N_{u,CDS} = N_{u,CDS} - \{v\}$
end if
if $(Rank(v) < Rank(u))$ **then**
  $Lower\_Rank_u = Lower\_Rank_u - \{v\}$
end if
if $DQQ \neq \emptyset$ **then**
  $v \leftarrow$ Dequeue DQQ
  Send $Dominator\_Reply(u, D_u)$ to v
else $Reply\_In\_Transit \leftarrow false$
end if

among its neighbors becomes an initiator and runs the domination and connectivity tests. The assignment of unique ranks to nodes ensures that there is at least one such node. In order to run the domination test, node u sends a *Dominator_Query* message to all its neighbors v and asks them to send back a list of their current dominators D_v included in the message *Dominator_Reply*(v, D_v). The set of dominators of node v, D_v, is its neighbors with either *in* or *pending* status.

A node receiving a *Dominator_Query* message only sends a *Dominator_Reply* message if it does not have a previous *Dominator_Reply* message in transit. For example, we assume, without loss of generality, that node u receives a *Dominator_Query* message from its neighbor v and then from another neighbor w which is running the algorithm at the same time. We also assume that node u has no *Dominator_Reply* message in transit. It first sends a *Dominator_Reply* message to node v and enqueues the incoming message from node w in DQQ. Once it receives the $Finished\_Msg$ from node v, it dequeues w's query and sends a reply to it. This order ensures that simultaneous dropout of two dominators in the neighborhood of a node is ruled out.

Once all the *Dominator_Reply* messages are received, node u proceeds to the connectivity test if all its neighbors have at least one other dominator. Otherwise, it changes its status to *in* and stays in the CDS ($Status_u \leftarrow in$).

The connectivity test at node u examines if the subgraph induced by $N_{u,CDS}$, its neighbors with either *in* or *pending* status, is connected. If so, node u drops out of the CDS ($Status_u \leftarrow out$). Otherwise, it stays in the CDS ($Status_u \leftarrow$

in). Note that the connectivity test does not require any additional message exchange because nodes send their list of dominators during the domination test in the $Dominator\_Reply(v, D_v)$ message. Therefore, connectivity test is performed locally by comparing the list of neighbors and their dominators.

Node u sends a $Finished\_Msg(status)$ to its neighbors when it changes its status to *in* or *out*. Upon receiving a $Finished\_Msg(status)$, any node u removes the sender from its lower rank neighbors if the sender has a lower rank. If the *status* is *out*, node u removes the sender from its set of CDS neighbors, $N_{u,CDS}$, and updates its effective degree. It also attends to the next pending $Dominator\_Query$ message in DQQ if there exists such a message; otherwise it resets the $Reply\_In\_Transit$ flag. The details of this algorithm are shown in Algorithm 2.

Another conceivable distributed implementation is to have nodes elect a leader which starts the algorithm (which we refer to as *single initiator* as opposed to *multiple initiators* in Algorithm 2). Despite exhibiting a lower degree of parallelism, this implementation is more appropriate in scenarios where the leader is pre-determined due to its special functionalities and there is no need to run an expensive leader election algorithm. An example of such a scenario is a *sink* node in an ad hoc sensor network that can be the initiator to run Algorithm 2 to form a backbone that is used for data gathring/dissemination.

3.3.1 k-Hop Extension of the Algorithm

In Algorithm 2, the connectivity test can be extended to check whether the subgraph induced by the k-hop neighbors in CDS is connected. In order to do this, a node u needs to exchange its ranks with its k-hop neighbors and adjusts its local variables $N_{u,CDS}$, and $Lower\_Rank_u$ accordingly. Furthermore, when a node finishes running the domination and connectivity tests, it sends a $Finished\_Msg$ to its k-hop neighbors instead of immediate neighbors only. Obviously, this extension will result in a smaller-sized CDS as k increases, but at the expense of message overhead. Experimental results investigating the effect of using a larger neighborhood show that $k = 4$ is a good compromise and significantly reduces the size of the final CDS while not imposing a considerable overhead on the algorithm.

Note that there are two possible ways to implement the k-hop extension. In the first one, all nodes initially set their k-hop parameter to the desired locality level and run the algorithm. The other approach is to have all nodes run the algorithm with regard to their immediate neighborhood ($k = 1$), and then the nodes selected in the first round as CDS nodes will run the algorithm for higher values of k. The latter is more efficient in terms of message overhead, thus we used the second implementation in our simulations.

3.3.2 Example

In this section we illustrate how our proposed algorithm constructs the CDS through an example. The example, shown in Figure 4, was generated by randomly scattering

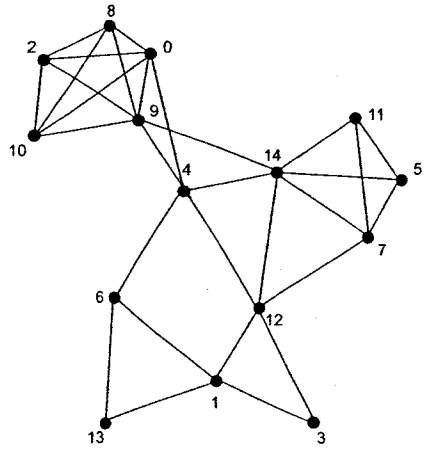
15 nodes with the transmission range of 20 meters in a square area whose side length is 60 meters. The nodes' IDs are labeled beside the nodes (IDs start from 0). Initially all the nodes have pending status and are marked black. Note that at each point during the computation of the final CDS, the set of black nodes yields a feasible solution. At each round, when a node runs the algorithm, it either stays in the CDS and remains black, or drops out and becomes white. The nodes run the algorithm according to the following execution scenario. Note that we have divided the execution into distinctly separate *rounds* to simplify the process of explaining the flow of execution, however, nodes in a given round may not run the tests in the exact same order as described here.

- Nodes exchange their information with all their one-neighbors. Nodes 2, 3, 5, and 13, which have the minimum rank among their neighbors, initiate the algorithm. Let's look at what happens when node 2 runs the algorithm. It passes the domination test because all its neighbors (nodes 0, 8, 9, and 10) are dominated by at least one other node. So node 2 proceeds to the connectivity test. Since its current neighbors in CDS are 0, 8, 9, and 10 and the subgraph induced by these nodes is connected, node 2 passes the connectivity test and therefore drops out of CDS. Nodes 3, 5, and 13 similarly drop out of the CDS when they run the tests. So at the end of the first round, all the four initiators leave the CDS as they are redundant nodes. These nodes send a $Finished\_Msg(out)$ to all their neighbors.

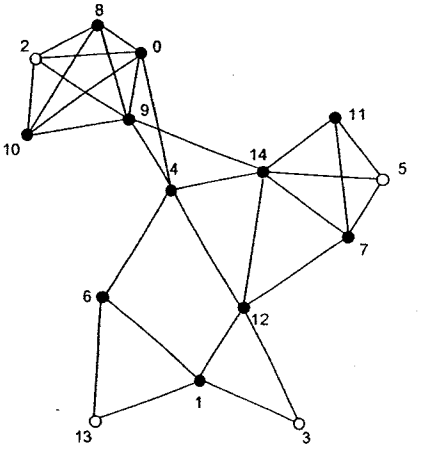
- When the one-hop neighbors of the initiators receive the $Finished\_Msg(out)$ messages from the initiators, they update their lower-rank neighbor lists and the second round begins. At this point, nodes 1, 7, and 8 have the lowest ranks among their neighbors with pending status. and run the tests depicted in Figure 4-c. Node 1 passes the domination test and proceeds to the connectivity test. It has two neighbors in the CDS (nodes 6 and 12). From the local view of node 1, these two nodes are not connected and leaving the CDS renders the resulting set disconnected, so node 1 stays in the CDS and sends a $Finished\_Msg(in)$ message to all its neighbors. Nodes 7 and 8, on the other hand, both drop out since they pass both tests.

- In the third round, nodes 6, 10, and 11 run the algorithm since they have no lower-rank neighbor with pending status at this point. Node 6 passes the domination test but fails the connectivity test. So node 6 changes its status to *in*. Nodes 10 and 11 both drop out since they pass both tests. This round is illustrated in Figure 4-d.

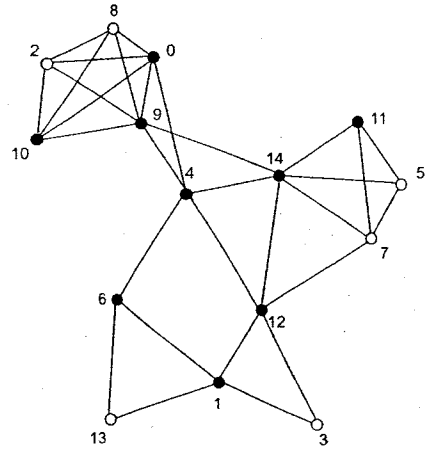
- As depicted in Figure 4-e, in the fourth round, nodes 0 and 12 run the tests. Node 0 drops out since all its neighbors are also dominated by node 9, which is its only neighbor in the CDS. Therefore it passes both domination and connectivity tests and drops out. Node 12, on the other hand, remains in the CDS because it fails the connectivity test. So at the end of this round, only three nodes (4, 9, and 14 are still pending.)



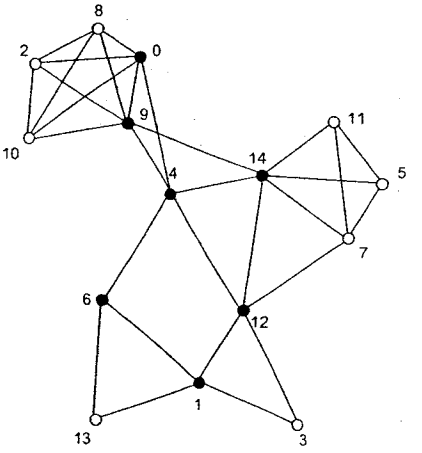
(a) All nodes are initially in the CDS



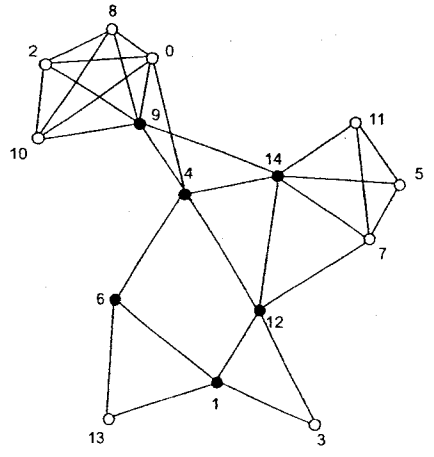
(b) Round 1: Initiators run the tests



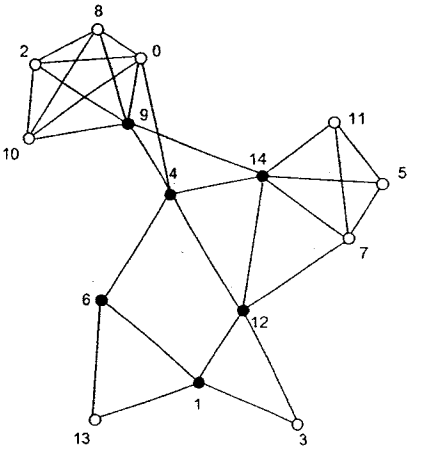
(c) Round 2: Nodes 1,7,8 run the tests



(d) Round 3: Nodes 6,10,11 run the tests



(e) Round 4: Nodes 0,12 run the tests



(f) Rounds 5,6,7: Nodes 9,14,4 run the tests
(Final CDS constructed)

Figure 4: CDS construction by our algorithm (the set of black nodes constitutes the CDS)

- Among nodes 4, 9, and 14, node 9 has the lowest rank. Thus, it runs the tests before the other two nodes and stays in because it fails the domination test. Then node 14 runs the tests and stays in the CDS for the same reason. Finally, the last node which is node 4 runs the tests. Node 4 passes the domination test because all its neighbors are dominated by at least one other node. However, the subgraph induced by its neighbors that are in the CDS (nodes 6, 9, 12, and 14) is not connected from its local view. Therefore, it stays in the CDS and the algorithm terminates as no node with pending status is left. The final CDS is formed by the set of black nodes in Figure 4-f (CDS = 1, 4, 6, 9, 12, 14).

In the above scenario, nodes ran the connectivity test with regard to their one-hop neighborhood. In other words, we assumed that $k = 1$. As discussed in section 3.3.1, nodes can make more informed decisions if they look at a larger neighborhood while running the connectivity test. Although choosing very large neighborhoods while running the test implies significant message overhead and is not viable in the context of energy-constrained ad hoc networks, considering small neighborhoods such as 2 or 3 hops away does not seem to be very expensive in many applications. In our algorithm, this extension can be very useful.

In the example depicted in Figure 4, if we set $k = 2$, and have the nodes, which were selected to remain in the CDS, run the connectivity test with regard to their two-hop neighborhood, the size of the set can be further reduced. As illustrated in Figure 5, by looking at two-hop neighborhood, node 1 is enabled to see from its local view that nodes 6 and 12 are indeed connected via node 4 and that it can safely opt

out of the CDS. This decision reduces the CDS to nodes 4, 6, 9, 12, 14. The other CDS nodes also run the connectivity test with regard to their two-hop neighborhood in the order imposed by the algorithm, but no further change results.

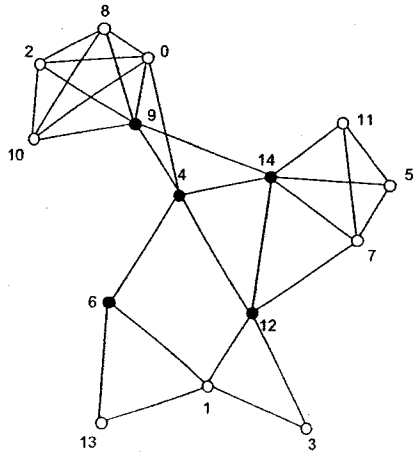


Figure 5: Considering larger neighborhood in the connectivity test

3.3.3 Performance Analysis

In order to calculate the message complexity, we should compute the number of messages sent by each node during the execution of Algorithm 2. In the setup phase, each node u sends a constant number of messages to its k-hop neighbors. Assuming Δ is the maximum degree in the network, the size of u's k-hop neighborhood is bounded by Δ^k. So, the message complexity of this phase is $O(n\Delta^k)$. While executing the algorithm, a node sends $O(\Delta)$ messages to its immediate neighbors during the domination test, and no message passing is required for the connectivity test if implemented as described in Algorithm 2. The connectivity test is done using the locally-stored information maintained and updated when a node receives $Finished\_Msg$ messages from

nodes in its k-hop neighborhood. Finally, every node sends a constant number of messages ($Finished\_Msg$) to its k-hop neighbors when it finishes running the algorithm, which amounts to $O(n\Delta^k)$ messages. Therefore, the overall message complexity of the Algorithm 2 is $O(n\Delta^k)$.

The execution time of Algorithm 2 is equal to the length of the longest dependency chain where each node in the chain has to wait for the next node to finish running the algorithm. This dependency chain can be linear in the number of nodes in the network as can be seen by the example of a line graph. In a line graph, the two end nodes become the initiators as they have the lowest effective degree and then their adjacent nodes run the tests and this continues until the two chains of execution that ripple inwards meet in the middle. Therefore the worst-case time complexity of the algorithm is $O(n)$.

3.3.4 Competitive Ratio

In spite of the fact that our algorithm exhibits a very good performance for networks with uniform random distribution and yields results that are desirably close to optimal, it is possible to conceive of certain examples in which it has a very poor performance. As an example, suppose the network consists of nodes with transmission range of r arranged in two concentric circles with radii r and $2r$ respectively. This setting is illustrated in Figure 6. Note that all the edges between the nodes that lie on the same circle have been eliminated to enhance clarity of the figure. In this example, regardless of the number of the nodes in the network, no node can drop out

of the CDS when it runs the connectivity test with $k = 1$, and the final CDS consists of all the nodes in the network, whereas the optimal CDS consists of a constant number of nodes. However, if k is increased to 2, then a large percentage of nodes drop out because they become able to see the connectivity of the neighboring nodes with regard to their two-hop neighborhood.

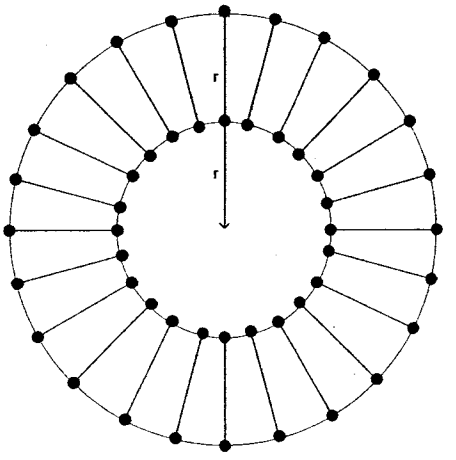


Figure 6: An example showing the worst-case performance of the algorithm

This example shows that there are certain configurations in which it is not possible to compute a constant ratio between the size of the CDS constructed by our algorithm and that of the optimal solution. More specifically, if the locality of the connectivity test is not well-adjusted based on the network, our algorithm may produce a CDS as large as the size network in the worst case. However, extensive simulations presented in section 3.6 show that such poor performance occurs in very special configurations, and that for random distributions, the performance is quite satisfactory.

3.4 Maintenance of The Backbone

Topological changes are frequent in ad hoc networks. In MANETs, nodes may switch on/off or move from one location to another location in the network. There might be also nodes that join or leave the network at different intervals. In the context of sensor networks, although there is typically no node mobility, topological changes may occur due to node depletion. Additionally, new sensors might be deployed to patch up those areas in which sufficient coverage is no longer available due to the depletion of a considerable number of nodes. All these scenarios imply topological changes in the network. Such unique characteristics of ad hoc networks bring about new challenges in the design of robust protocols. More specifically, a protocol must provide appropriate mechanisms to handle such changes as efficiently as possible.

In the context of CDS formation algorithms, there are generally two kinds of mechanisms to handle topological changes: *periodic reconstruction* and *on-demand update* [2], [50]. While periodic reconstruction forces all the nodes in the network to re-run the algorithm, on-demand update typically involves nodes in a small neighborhood affected by the change to cooperate to reshape the CDS locally. Clearly, the second approach is more desirable since it is more energy-efficient and incurs much less message overhead. Therefore, in order to best maintain the CDS, it is important to use local updates as much as possible and recalculate the CDS only as a last resort.

The distributed algorithms that have been proposed for CDS construction fall into

one of the two categories: *(i) serialized* and *(ii) parallel.* Serialized distributed algorithms need to perform a global function that is sequential in nature, such as building a tree, selecting a leader that initiates forming an MIS and so forth. Local updates for such algorithms need to be specified and analyzed in a completely different fashion than the original algorithm for CDS construction. In addition, the approximation bounds due to the original algorithm might be affected as a result of the additional update procedures. However, parallel distributed algorithms such as Algorithm 2, despite their distributed nature, have the potential to deal with local changes locally. This advantage makes parallel distributed algorithms better candidates than their serialized rivals for ad hoc networks in terms of maintenance.

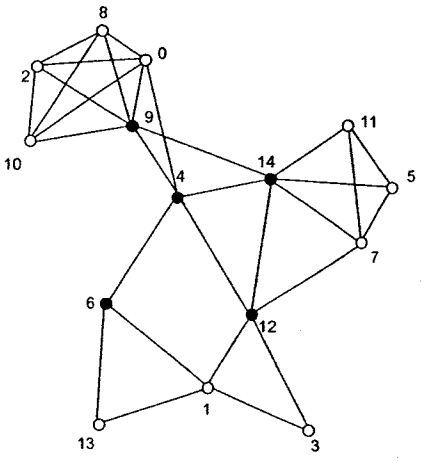
In this section, we discuss the maintenance strategies that we accommodated into our algorithm in order to use local updates to handle topological changes as much as possible. We will show that these techniques are often efficient in avoiding recalculation of the whole solution. However, there will be scenarios which cannot be addressed using these strategies and therefore recalculation becomes inevitable.

We generalize the topological changes to two types: *node addition* and *node removal.* Basically node addition is a generalization of scenarios such as when a node switches on, a mobile node joins a network, or a new sensor node is deployed. Node removal, on the other hand, may include scenarios such as when a node switches off, a mobile node leaves the network, or a sensor node dies.

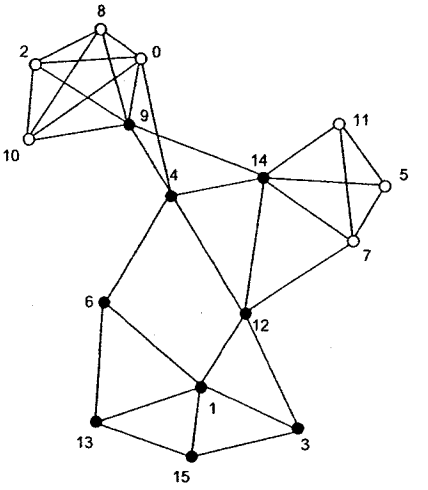
3.4.1 Node Addition

When a new node is added to the network, it sends a message to its neighbors and checks to see if there already exists a dominator among its neighbors. If so, it does not need to take any further step as it is dominated. However, if none of its neighbors is a dominator, it switches to *pending* status and sends a message to all its neighbors requesting them to *revert* to *pending* status. Then all the nodes with the *pending* status run the algorithm to determine the new node that needs to be added to the CDS.

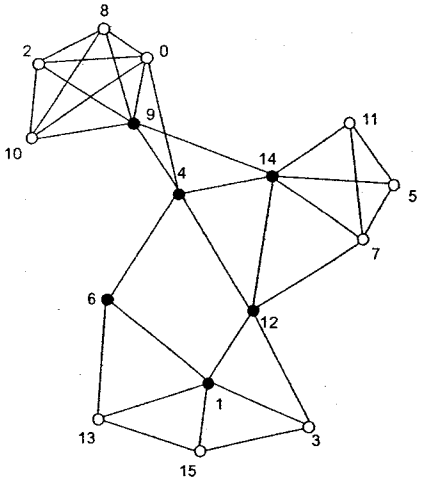
The selection of a new node as a result of the above procedure may make it possible to locally prune away some nodes. Therefore, as an optimization phase, when a new node is added to the CDS, its neighbors which are in the CDS run the connectivity test to check if they have become redundant as a result of this change in the formation of CDS. If so, they drop out of the CDS.



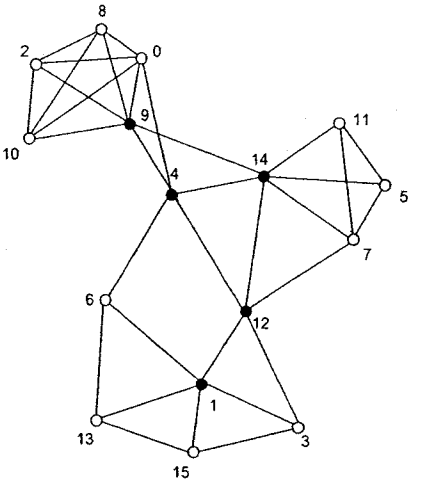
(a) Network topology before change



(b) Node 15 is added to the network – Nodes 1,3,13 revert to pending status



(c) Nodes 1,3,13, and 15 run the algorithm



(d) Nodes 6 and 12 run the algorithm (k=2)

Figure 7: CDS maintenance when a new node is added to the network

In the example illustrated in Figure 7, the initial topology shown in part (a) is the same topology discussed in Figure 5 in section 3.3.2. As depicted in Figure 7-b, node 15 is added to the network. When this node joins the network, it sends a message to its neighbors (nodes 1, 3, 13) to see if there already exists a dominator in its neighborhood. In this example, none of its neighboring nodes are in the CDS.

Therefore, it sends them a message asking them to switch to *pending* status. Then nodes 1, 3, 13, and 15 run the algorithm (with $k = 1$). When they finish, node 1 remains in the CDS and nodes 3, 13, and 15 drop out as depicted in Figure 7-c. Finally nodes 6 and 12 run the algorithm to decide whether they have become redundant as a result of the recent change. If they run the connectivity test with $k = 1$, there will be no change in the status of the CDS. However, if they run this test with $k = 2$, then node 6 drops out of the CDS, as illustrated in 7-d. To achieve consistency, the value of k can be agreed upon by all the nodes before running the algorithm and used later in the maintenance as well.

3.4.2 Node Removal

Before discussing the procedures that deal with node removal, note that we assume that the elimination of a node does not disconnect the underlying graph. If a non-CDS node v is removed, its dominator u updates its neighbor list. If node v was its only dominatee and node u is a leaf dominator, then node u drops out of the CDS and becomes a non-CDS node. However, if it has other dominatees or it is non-leaf CDS node, then it takes no further action after updating its neighbor list.

Indeed, it is the elimination of a CDS node that needs to be handled carefully because the CDS may need to be reconstructed. We will show that as long as the underlying graph remains connected, in the face of single node failures, we can recalculate the CDS locally. But before explaining the maintenance procedure, we need to discuss the following important properties of the CDS built by Algorithm 2.

For ease of exposition, we start with $k = 1$, where k is the size of the neighborhood used by a node to run the connectivity test in Algorithm 2. We also start by assuming that the removal of a CDS node u breaks the CDS into two components.

Lemma 6 *Consider the subgraph $G' = (V', E')$ induced by the nodes in the CDS constructed in the graph $G = (V, E)$ using Algorithm 2 ($k = 1$). Assume that the removal of node u splits G'_{-u} into two components C_1, C_2 such that G_{-u} is still connected. Then there must be a path in G_{-u} between C_1 and C_2 consisting of only neighbors of u.*

Proof. We denote the set of neighbors of node u in component C_1 by $\tilde{V}$ and the set of its neighbors in C_2 by $\tilde{W}$. Since the graph G_{-u} is connected, consider a shortest path P_1 between the two components of the CDS C_1 and C_2. Since it is a shortest path we can assume that only the endpoints say v_1 and w_1 are CDS nodes and all interior nodes are non-CDS nodes. All of these interior nodes were initially in the CDS, but eventually they all dropped out of the CDS. Each such node that drops out of the CDS, only does so because it passes the connectivity test; the subgraph induced by its neighbors is connected. In particular, when such a node drops out, it is assured of another path that is currently in the CDS between the two components C_1 and C_2. The same is true of any nodes that drop out of this new path, ad nauseam.

We claim that as these nodes drop out of the CDS, we must eventually come to a node that has u as its neighbor. More formally, let x_1 be the first node in P_1 to have dropped out of the CDS. When x_1 dropped out, it detected the presence in its

one-hop neighborhood of a CDS path between its two neighbors in P_1. This implies that at the moment x_1 dropped out, there was a CDS path between v_1 and w_1 that contained at least three consecutive neighbors of x_1. If this path includes the node u, we have found the node we are looking for. If not, consider the shortest sub-path of this path that connects C_1 and C_2; call this path P_2 and call its endpoints in C_1 and C_2 as v_2 and w_2 respectively. As illustrated in Figure 8, the nodes v_1 and v_2 may or may not be the same node. Likewise, nodes w_1 and w_2 may or may not be two distinct nodes.

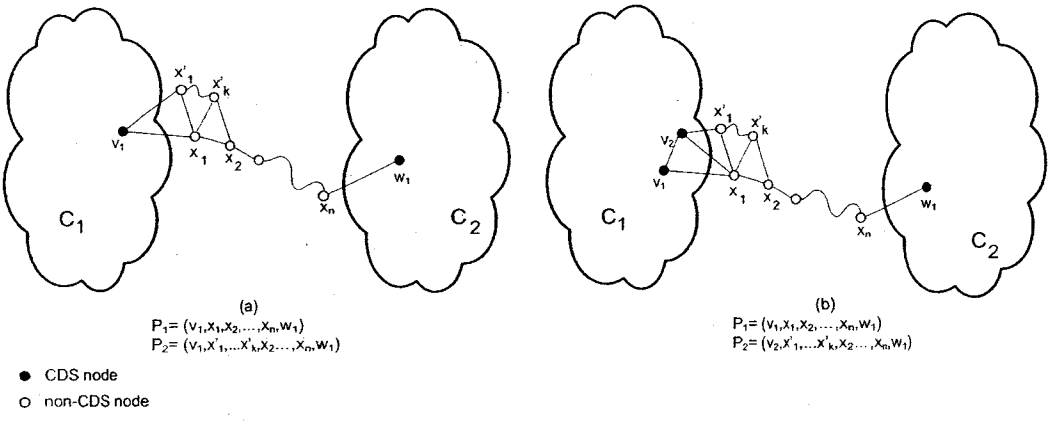


Figure 8: Paths P_1 and P_2 may have the same or different endpoints.

Now, consider P_2 and let the first node that dropped out be called x_2. Once again, if x_2 is a neighbor of u, we are done, otherwise, we consider the next path P_3 defined as above. We know that when the process ended, the only paths between the components C_1 and C_2 were of the form (v, u, w) with $v \in \tilde{V}$ and $w \in \tilde{W}$. Therefore, there has to have been a node x_i that dropped out because it detected in its one-hop neighborhood a path that went through u between its two neighbors on the path P_i

with endpoints in C_1 and C_2.

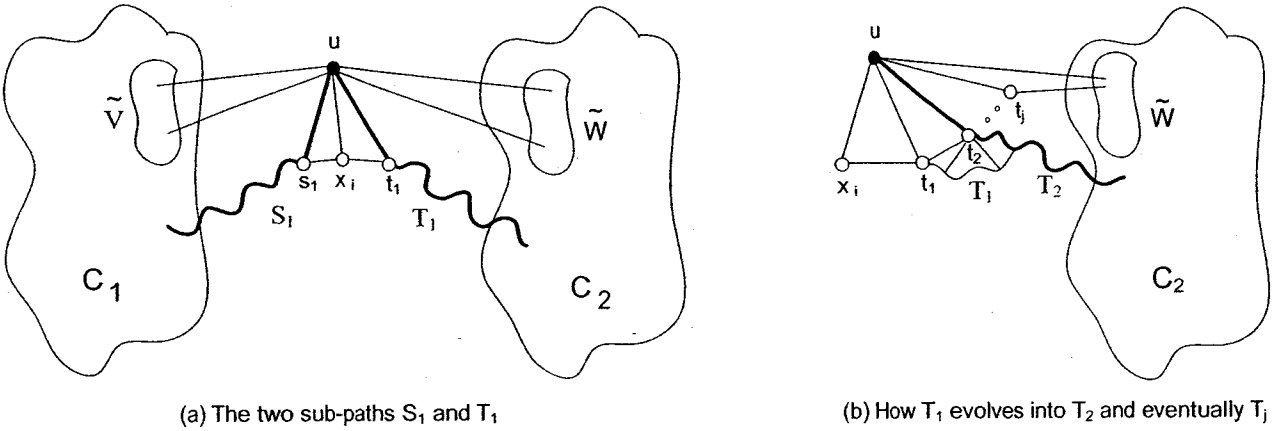


Figure 9: There exists a path between $\tilde{V}$ and $\tilde{W}$ consisting of all neighbors of u. Note that all the nodes in $\tilde{V}$ and $\tilde{W}$ are neighbors of u

As depicted in Figure 9-a, denote by S_1 (T_1) the sub-path from C_1 to u (u to C_2). If x_i is directly connected to both $\tilde{V}$ and $\tilde{W}$, we are done. If x_i is connected directly only to $\tilde{V}$ ($\tilde{W}$), we consider T_1 (S_1). Otherwise, we will consider both, but the arguments for both are exactly the same.

We consider what happens to the nodes on T_1. We claim that there is a path consisting of non-CDS nodes $T = (t_1, t_2, \ldots, t_j)$ such that each t_k (for $1 \leq k \leq j$) is connected to u. Further, t_j is also connected to a node in $\tilde{W}$.

Let t_1 be the neighbor of u in T_1. As illustrated in Figure 9-b, we show how to derive the rest of the path T. Observe that t_1 is connected to x_i and to u. If t_1 is connected to a node in $\tilde{W}$, we are done with $j = 1$. Otherwise, we consider the time that t_1 dropped out of the CDS. At this time, T_1 has possibly evolved to a different

path between t_1 and $\tilde{W}$, which we will call T_1'. Now t_1 decides to drop out because its two immediate neighbors on T_1' (one of which is u) are connected via a path in t_1's one-hop neighborhood. We therefore have a new path between u and w, called T_2. Call the neighbor of u on this path t_2. Clearly, t_2 is a neighbor of both t_1 and u as needed. Once again, if t_2 is a neighbor of a node in $\tilde{W}$ which is a neighbor of u, we are done, otherwise we continue. This process has to end with a node that is a neighbor of both u and a node in $\tilde{W}$ since the only paths in the final CDS between u and C_2 were the edges (u, w) with $w \in \tilde{W}$.

By using a symmetric argument for the sub-path S_1, we have a path between the two components that consists of only neighbors of u. □

We go on to consider the case where the removal of node u splits G'_{-u} into multiple components. Clearly, since the graph G_{-u} is connected, there is a path in G_u between every pair of components in G'_{-u}. We call C_i and C_j *adjacent* if there exists a path P in G_{-u} between C_i and C_j such that (a) P consists of only non-CDS nodes except for endpoints in C_i and C_j and (b) as P evolves into other paths, as described in the proof of Lemma 6, due to the dropping out of non-CDS nodes, no CDS node from some other component is ever encountered. We call such a path a *direct* path. Otherwise C_i and C_j are non-adjacent. It is straightforward to see the following extension of Lemma 6 to the case of adjacent components.

Lemma 7 *Consider the subgraph $G' = (V', E')$ induced by the nodes in the CDS constructed in the graph $G = (V, E)$ by using Algorithm 2. Assume that the removal of node u splits G'_{-u} into multiple components $C_1, C_2, \ldots C_i$ such that G_{-u} is still*

connected. If C_i and C_j are two adjacent components, then there must be a path in G_{-u} between C_i and C_j consisting of only neighbors of u.

Proof. Start with a direct path between components C_i and C_j and continue as in Lemma 6. □

We claim that non-adjacent components are always connected via adjacent components.

Lemma 8 *For every pair of components C and C', there is a path in G_{-u} which consists of juxtapositions of direct paths and CDS paths.*

Proof. If C and C' are adjacent, there is a direct path between them by definition, and we are done. If C and C' are not adjacent, consider any path between C and C', and say it goes through components $C_{i_1}, C_{i_2}, C_{i_3}, \ldots, C_{i_k}$. It suffices to argue that the claim holds true for a pair of consecutive (not necessarily adjacent) components C_i and C_j. If C_i and C_j are adjacent, we are done. Otherwise, take the sub-path between C_i and C_j that is free of CDS nodes. Obviously it is not a direct path. Therefore, as this path evolves, it must go through some other component $\hat{C}$. We now recursively look at the sub-path between C_i and $\hat{C}$ and the sub-path between $\hat{C}$ and C_j. This process can only end with a direct path between two components. □

This implies that if adjacent components can be reconnected, then the entire CDS will be reconnected. In the following, we describe a maintenance procedure that reconnects adjacent components using the neighbors of u.

Theorem 1 *Consider the subgraph $G' = (V', E')$ induced by the nodes in the CDS constructed in the graph $G = (V, E)$. If $k = 1$ in the connectivity test in Algorithm 2, and the elimination of a node u results in G'_{-u} not being a CDS any more, G'_{-u} can be restored using a simple local procedure as long as G remains connected.*

Proof. The elimination of a CDS node can have three possible consequences:

1. One or more nodes will be left un-dominated.

2. CDS will become disconnected.

3. The combination of the above two conditions.

We use two important properties to propose our local maintenance scheme. The first one holds for any CDS: if the underlying graph remains connected, a non-CDS node x_i that is left un-dominated as a result of the elimination of its only dominator u, is either two hops away from some other CDS node v or is adjacent to another non-CDS node which has been left un-dominated by the elimination of u. Figure 10 gives an example of this property of a CDS.

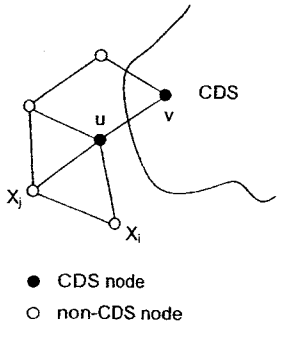


Figure 10: A non-CDS node x_i which is left un-dominated when its only dominator u is removed, is either two hops away from another dominator v or is adjacent to another such node x_j

The second important property is that of the path(s) that connect any two adjacent components in G_{-u} in Algorithm 2, as proved in lemma 7: there is at least one such path that only consists of neighbors of u. We use these two key properties to propose the following procedure to locally fix the CDS.

Maintenance procedure A non-CDS node that notices the failure of its only dominator sends a *DOMINATION WARNING* message to its 1-hop neighbors and reverts to pending status. A CDS node that notices the failure of a neighboring CDS node sends a *CONNECTIVITY WARNING* message to its 1-hop neighbors. Any node that receives a *DOMINATION/CONNECTIVITY WARNING* message reverts to pending status. All the nodes with the pending status run Algorithm 2.

If the failure of a CDS node u does not disconnect G'_{-u} and only leaves some nodes un-dominated, sending the *DOMINATION WARNING* message to one-hop neighbors guarantees that at least a node with a neighbor in the CDS re-runs the algorithm. Thus, not only do the un-dominated nodes become dominated again, they also get reconnected to the CDS.

If the failure of node u disconnects G'_{-u} into multiple components while the underlying graph is connected, as proved earlier, there exists a path between any two adjacent components consisting of only neighbors of u. These neighbors include at least two CDS nodes as endpoints. As shown in Figure 11, nodes x_1 and x_n revert to pending status due to the *CONNECTIVITY WARNING* message they receive from v and w and any intermediate node x_i reverts to pending status because it loses its

only dominator (u). When all the nodes with pending status re-run the algorithm, the two adjacent components get reconnected. It follows from Lemma 8 that when all the adjacent components are reconnected, the CDS will be restored.

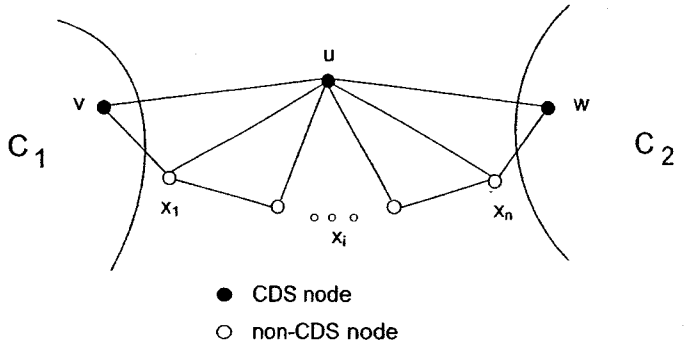


Figure 11: The maintenance procedure restores the CDS by reconnecting C_1 and C_2.

□

The discussion above assumed that nodes run the connectivity test in Algorithm 2 with regard to their one-hop neighborhood ($k = 1$). However, the following corollary of Lemma 7 is straightforward to see.

Corollary 1 *Consider the subgraph $G' = (V', E')$ induced by the nodes in the CDS constructed in the graph $G = (V, E)$ when the connectivity test is run with regard to k-hop neighborhood in Algorithm 2. Assume that the removal of node u splits G'_{-u} into multiple components C_1, C_2,...,C_i such that G_{-u} is still connected. Then there must be a path in G_{-u} between any two adjacent components which only consists of nodes in u's k-hop neighborhood.*

The procedure given in the proof of Theorem 1 can be modified by sending the *CONNECTIVITY WARNING* message to k-hop neighbors. Thus, we showed that

the in the face of single node failures, the CDS can be restored locally for any arbitrary k as long as the underlying graph remains connected.

3.5 Local Implementation

A local algorithm is a distributed algorithm in which a node makes decisions based on the information obtained through communication with nodes located no more than a constant (independent of the size of the network) number of hops away from it. It has been shown in [33] that any local algorithm has a constant time complexity.

Unlike the non-local distributed algorithm presented in Section 3.3, we assume that nodes are aware of their geographic locations in the local implementation. This information is used in a tiling scheme first proposed in [14] to break potential symmetries and prevent a message to propagate beyond a constant neighborhood and thus bring about the desired locality of the algorithm.

In the tiling scheme proposed in [14], the plane is divided into tiles of twelve hexagons of diameter one and each hexagon is assigned a class number from 1 to 12. Since every hexagon has diameter one, any two nodes within one hexagon are adjacent. A node is assigned a class number corresponding to the class number of the hexagon containing it. This approach guarantees that two nodes of the same class number are either adjacent or at Euclidean distance greater than two, which is used to ensure the locality of our algorithm.

We redefine the rank of node u to be an ordered 3-tuple $(Class\_number(u), \delta(u), id(u))$ where $Class\_number(u)$ is the class number of the hexagon containing u and $\delta(u)$ is its effective degree (number of neighbors in the CDS). Using this new rank, each node runs Algorithm 2. This 3-tuple rank ensures that the execution of the algorithm by

any node depends on the nodes with lower class numbers and thus information can propagate up to eleven hops away, given that there are a maximum of twelve class numbers in the tiling used. This implies constant time complexity. The analysis of the message complexity is similar to what was discussed for the distributed implementation and thus the local implementation has also the same message complexity of $O(n\Delta^k)$.

Extending this local algorithm to k-hop neighborhood is restricted by the minimum distance between two distinct hexagons of same class number. Since in the tiling scheme described above, the distance between any two distinct hexagons of same class number is greater than two, we can increase k up to two without violating the locality of the algorithm. Further increasing k can result in smaller CDS provided that the tiling scheme is modified such that any two nodes with the same class number are either adjacent or have distance greater than k.

3.6 Experimental Results

We conducted extensive simulations to compare the performance of our distributed algorithm, both in its non-local and local form with their state-of-the-art competitors in each category. We evaluated the CDS size generated by the algorithm as the main criterion. We also considered average route length on the CDS generated by the algorithm as a measure of its quality. In the category of distributed algorithms, we compared our distributed implementation, hereafter referred to as PInOut_Dk

(after the name of the three statuses that the nodes can have) with BCOP [6], Zone-Based [25],WAF [1], and ECDS [57] since these algorithms produce the smallest-sized CDSs in the literature. In the category of local algorithms, we compared our local implementation, hereafter referred to as PInOut_Lk with TBC [14], WuLi_KR [15], TBLS_MD [28], and WuLi [50]. To the best of our knowledge, these are the only *practical* local CDS construction algorithms in the literature.

We used Java Platform (JDK 6 update 10) in all our simulations. Given that the transmission range of nodes and the area of the network are fixed, we varied the density of the network by assigning different values to n. In our simulations, we assigned the values 50, 100, 150, 200, 250, and 300 to n to start with a sparse network of average node degree of 3.53 and end with a dense network of average node degree of 21.2. The nodes are randomly distributed in a geographic area of 200 m by 200 m. Since the transmission range of nodes in our network model is 30 meters, two nodes are adjacent if and only if their Euclidean distance is less than or equal to 30 meters. For each value of n, we generated as many random graphs as required until we had 1000 connected graphs. The connected graphs were stored in a file and used across all simulations for the same value of n. We categorized the simulation results into two groups based on the type of algorithms (distributed and local) being compared. The results are presented in the following two sections.

3.6.1 Performance comparison of distributed algorithms

As mentioned earlier, nodes can look farther while running the connectivity test when k increases, thereby increasing the possibility of breaking longer loops that would be otherwise undetectable when looking at the immediate neighborhood. As a result, the CDS size decreases by breaking these loops. Our experiments showed that when k is increased to 2, 3, and 4, the CDS size is reduced by up to 12.7%, 19.7%, and 22.8% respectively. The gain by increasing k beyond 4 is not significant. Thus, we chose $k = 1$ and $k = 4$ for PInOut_D in the comparison of our algorithm with those distributed CDS formation algorithms in the literature that generate the smallest-sized sets. These algorithms, which were discussed in chapter 2, are BCOP [6], Zone_based [25], WAF [1], and ECDS [57]. Although BCOP is not a practical algorithm due to its significantly high complexity, it was merely selected as a benchmark since it generates the smallest CDS size prior to our work. WAF, ECDS, and Zone_based are cluster-based algorithms that all construct an MIS and evolve it into a CDS by adding connectors. While WAF has a high complexity, ECDS and Zone_based have linear message complexities.

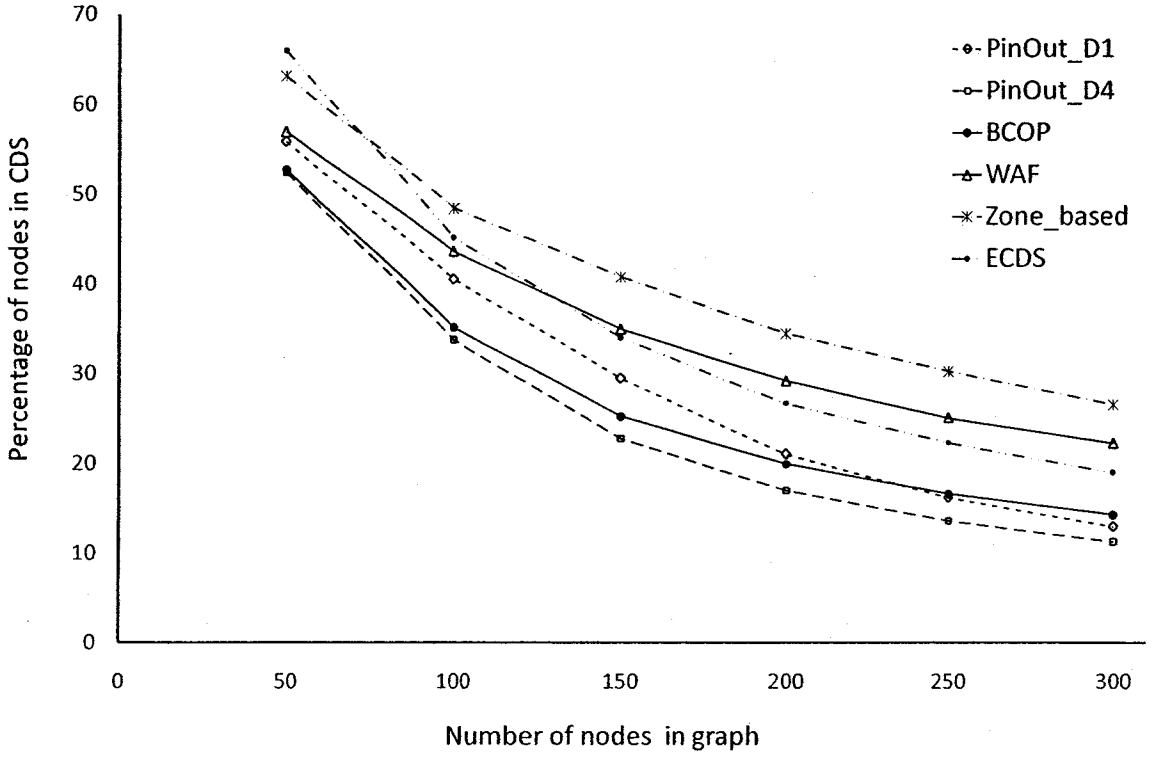


Figure 12: Percentage of nodes in the CDS for different distributed algorithms.

As shown in Figure 12, PInOut_D4 consistently generates the smallest-sized CDS and PInOut_D1 is the next to the best in dense networks. For sparse networks ($n = 50$), all the algorithms generate CDSs larger than 50% of the nodes in the network with PInOut_D4 being the best at 52.72%. As the density increases, the difference between the group of algorithms that generate CDSs directly (PInOut_D1, PInOut_D4, and BCOP) and those that start with an MIS (WAF, ECDS, Zone_based) become more noticeable. For average densities ($n = 150$), the CDS generated by BCOP, ECDS, WAF, and Zone_based is 11.08%, 49.23%, 53.67%, and 78.86% larger than that constructed by PInOut_D4, respectively. This difference is increased to

26.68%, 68.17%, 97.34%, and 135.19%, respectively for dense networks ($n = 300$). The results clearly confirm the efficiency of PInOut_D in terms of CDS size when compared with its competitors.

We also compared the *average shortest path length* (ASPL) on the backbone generated by PInOut_D1 and PInOut_D4 against all the other algorithms mentioned above. It is among the most significant qualitative metrics and shows how well a CDS performs as a backbone for routing or data gathering/dissemination protocols. It also affects the network's lifetime; a higher value of ASPL implies more nodes are involved in forwarding messages. As illustrated in Figure 13, ASPL on the CDS produced by Zone_based is constantly the closest to that of the original graph. This is due to its relatively large size compared to the other five algorithms. While PInOut_D1 produces reasonably small-sized CDS, especially as the graph grows denser, it always gives the next best ASPL compared to that of the original graph. ECDS and PInOut_D4 produce a moderate backbone in terms of ASPL, but BCOP has an ASPL of up to 2.25 times larger than that of the original graph.

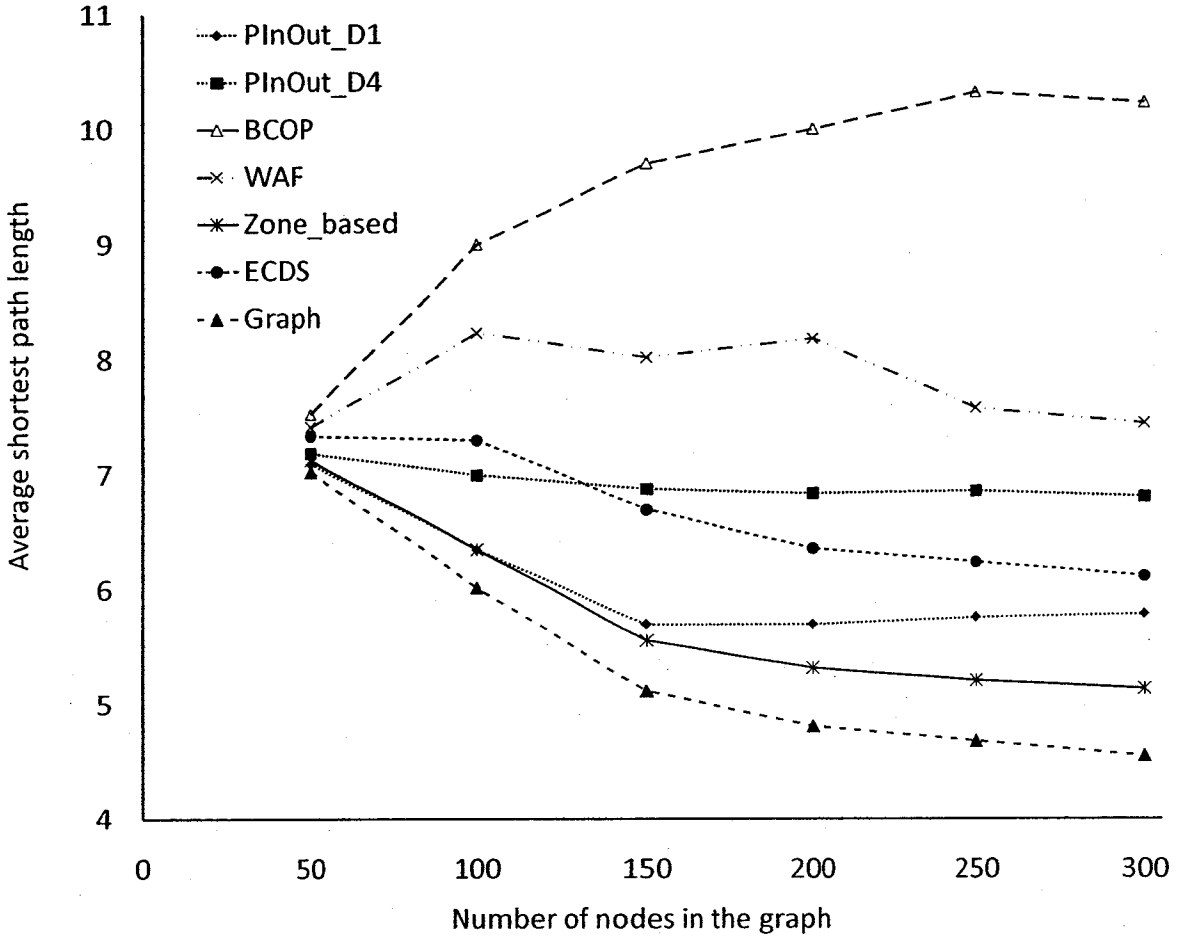
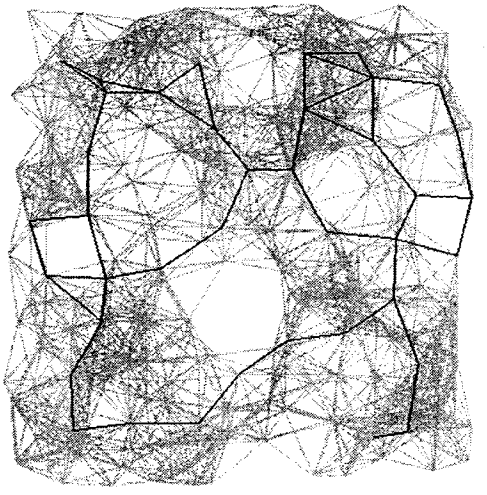


Figure 13: Average shortest path in the CDS for different distributed algorithms

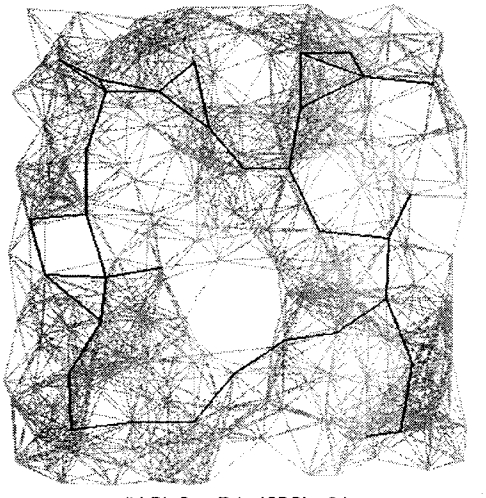
In order to provide some insight into some of the reasons why our algorithm outperforms all its MIS-based competitors in terms of the size of the constructed CDS, we provided a graphical sample of our simulation results in Figure 14. In this sample network, 300 nodes of equal transmission range of 30m are scattered in an area of 200m by 200m. The CDSs constructed by the distributed algorithms that we used in our simulations are depicted in parts (a) through (f).

As illustrated in the figure, the most conspicuous differences are the existence

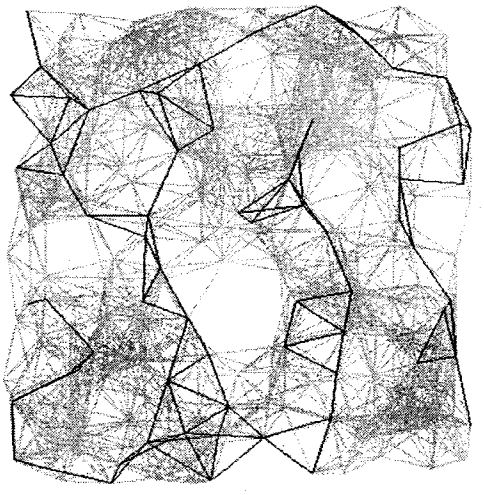
of short cycles and selection of quite a few nodes (with typically low degrees) along the borders of the network in the MIS-based algorithms (Figure 14-(c)[1],(d)[25] and (e)[57]), both of which can be attributed to the construction of MIS. These algorithms first build a DS by constructing an MIS which involves pushing the nodes in the set as far away as possible to ensure no two nodes are adjacent. This approach leads to the selection of low-degree nodes in the graph, especially along the borders, which would have otherwise been unnecessary. And what makes the CDS even larger is that many of these nodes introduce additional connectors into the CDS. Figure 14-a shows that a lot of these nodes are not selected by our algorithm. The zone-based algorithm [25] breaks the network into zones and uses the MIS strategy to build a dominating tree in each zone and then connects the zones. Clearly, this approach further increases the size of the set.



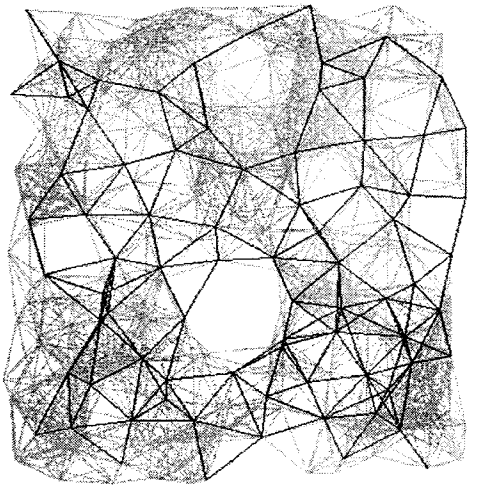
(a) PInOut_D1 - |CDS| = 39



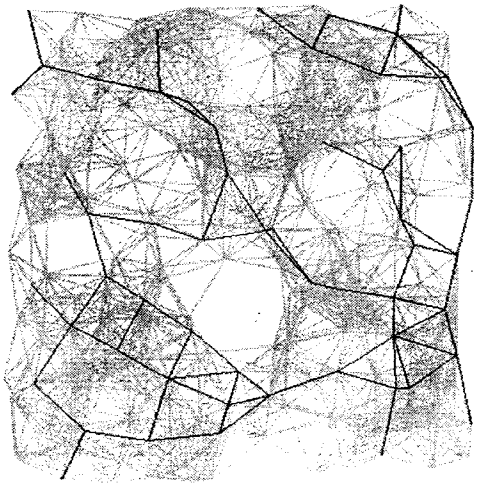
(b) PInOut_D4 - |CDS| = 34



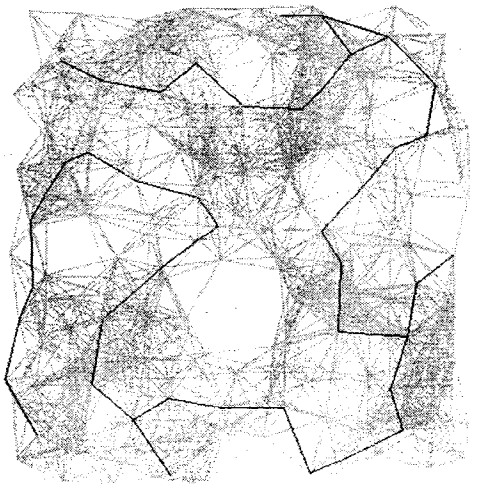
(c) WAF - |CDS| = 69



(d) Zone-based - |CDS| = 78



(e) ECDS - |CDS| = 57



(f) BCOP - |CDS| = 44

Figure 14: Comparison of the CDSs constructed by different distributed algorithms

3.6.2 Performance comparison of local algorithms

As discussed in Section 3.5, given the tiling scheme we used in the implementation of PInOut_L, the increase of k is restricted to 2. Our simulation results show that even the CDS generated by PInOut_L2 ($k = 2$), for moderate and dense networks, is comparable to TBSL_MD which generates the smallest CDS among all the other local algorithms.

As shown in Figure 15, while for sparse networks TBLS_MD outperforms all other algorithms, as the density increases PInOut_L$(1, 2)$ quickly catch up and outperform WuLi, WuLi_KR and TBC. For networks of moderate density ($n = 150, 200$) and dense networks ($n = 250, 300$), PInOut_L$(1, 2)$ has a very good performance which is comparable with that of TBLS_MD. Although WuLi_KR is a very good improvement over WuLi, they both generate the largest CDSs among all competitors. However, we should keep in mind that, unlike TBC, TBLS_MD, and PInOut_L$(1, 2)$, WuLi and WuLi_KR do not require information about the location of the nodes which is a big advantage when nodes are not equipped with positioning systems. Figure 16 depicts the ASPL metric for all the local algorithms compared in this section. As expected, ASPL has an inverse proportional relation with the size of the CDS.

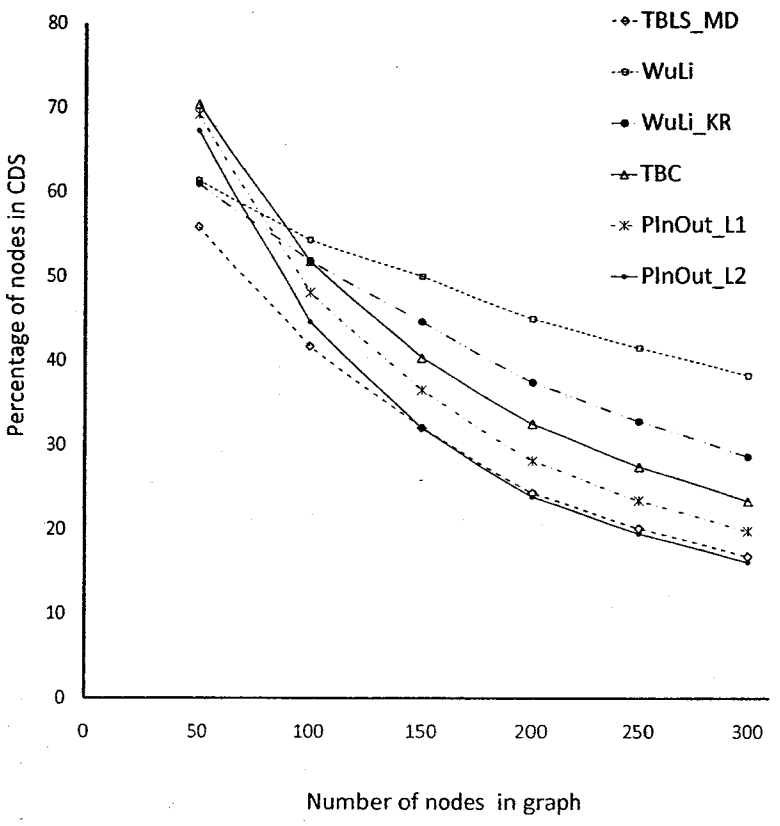


Figure 15: Percentage of nodes in the CDS for different local algorithms

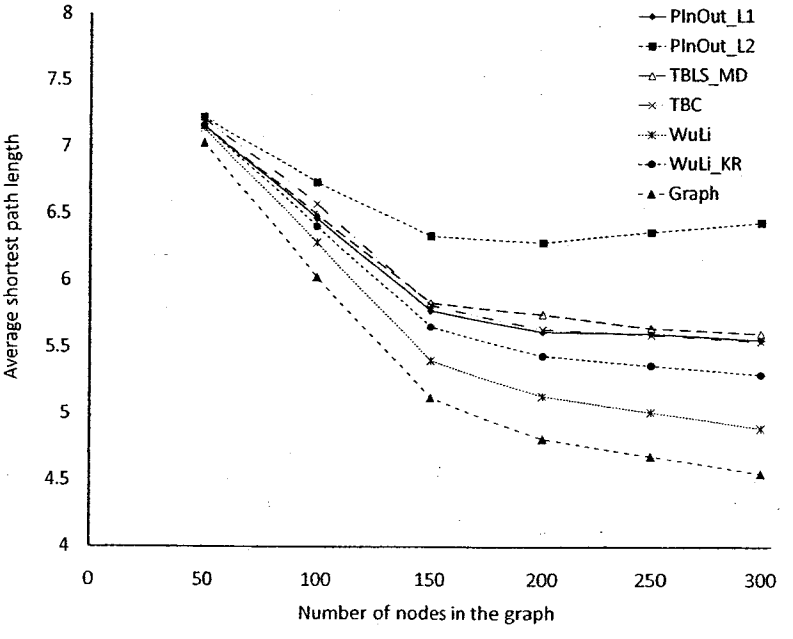


Figure 16: Average shortest path in the CDS for different local algorithms

Chapter 4

Algorithms for Networks with Asymmetric Links

In Chapter 3, we modeled the wireless network as a UDG and proposed our algorithm based on this model. However, as discussed in Chapter 1, nodes in the network may not necessarily have the same transmission range; therefore, in this chapter, we extend our algorithm to construct SCDASs in networks with asymmetric links, modeled as a *Disk Graph (DG).*

In a disk graph $G = (V, E)$, a node $v_i \in V$ has a transmission range $r_i \in [r_{min}, r_{max}]$. If $d(v_i, v_j)$ denotes the Euclidean distance between the two nodes v_i and v_j, then there exists a directed edge $(v_i, v_j) \in E$ iff $d(v_i, v_j) < r_i$. In other words, there is a directed link from v_i to v_j only if v_j lies in the disk centered at v_i. An edge (v_i, v_j) is unidirectional if $(v_i, v_j) \in E$, but $(v_j, v_i) \notin E$. If $((v_i, v_j) \in E$ and also $(v_j, v_i)) \in E$, then the edge (v_i, v_j) is bidirectional.

Note that a disk graph in which the transmission range of each node is selected at random from $[r_{min}, r_{max}]$ is not the same as the well-known *Quasi Unit Disk Graph (QUDG)* model with parameters r and R, introduced in [3] for the first time. In the *QUDG* model, for two nodes u and v with Euclidian distance $|uv|$: if $|uv| \leq r$, then u and v have an edge in the graph; if $|uv| > R$, then u and v do not have an edge in the graph and if $r < |uv| \leq R$ then u and v may or may not have an edge. To further clarify their difference, it should be noted that in the *DG* model, once a node u has selected its transmission radius r_i, it has a link to every node whose distance to u is smaller than r_i. However, in the *QUDG* model, a link exists with a certain 'probability' between two nodes whose Euclidian distance is greater than r but smaller than R.

4.1 Definitions and preliminaries

In our algorithm, we use $N_d(u)$ to denote the dominating neighbor set of node u, i.e. $N_d(u) = \{v|(v,u) \in E\}$. A node $v \in N_d(u)$ is also referred to as an *incoming* or *ingress* neighbor of node u in the literature. Likewise, $N_a(u)$ is used to denote the absorbent neighbor set of node u; i.e $N_a(u) = \{v|(u,v) \in E\}$. A node $v \in N_a(u)$ is also referred to as an *outgoing* or *egress* neighbor of u in the literature. Figure 17 illustrates the dominating and absorbent neighbor sets of a node. Note that these two sets may overlap. In other words, a bidirectional neighbor of node u is both a dominator and an absorbent of node u. We define the degree of a node as the sum of

the number of its dominators and absorbents.

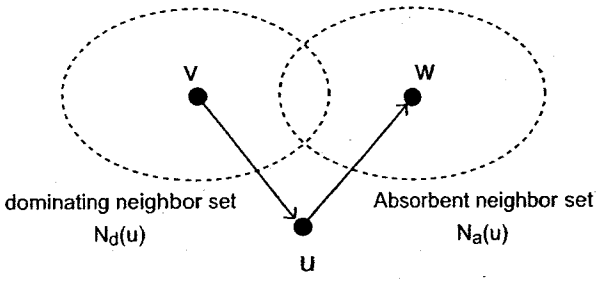


Figure 17: Dominating and absorbent neighbor sets of node u

Every node u has a rank $(\delta(u), id(u))$ which is an ordered pair of its *effective degree* and id, where the effective degree of node u is the number of u's neighbors in SCDAS, i.e $\delta(u) = |\{v|v \in N_a(u) \wedge v \in SCDAS\}| + |\{v|v \in N_d(u) \wedge v \in SCDAS\}|$. Since the membership of nodes in the SCDAS changes during the algorithm, so does the effective degree of a node. Assigning a unique id to every node, provides a mechanism to break ties.

As explained in Chapter 3, the rank of a node in our algorithm is defined based on the goal function. Since we intend to minimize the size of the constructed SCDAS, we use a node's effective degree in the definition of its rank.

4.2 Centralized Description

We extend the centralized description of our algorithm described in chapter 3 as follows. Node u passes the **Domination and Absorbency Test (DAT)** if:

(a) all the nodes that are dominated by u have at least one other dominator.

(b) all the nodes that are absorbed by u have at least one other absorbent.

Node u passes the **connectivity test** if the subgraph induced by its dominators and absorbents is strongly connected. The formal description of this extended algorithm is given in Algorithm 3.

4.3 Distributed Implementation

In order to present the distributed implementation of the above centralized description of our algorithm, we use most of the same details described in Chapter 3. Therefore, we will not explain all those details again. Instead, we just focus on the modifications and extensions that we need in order to adapt that implementation for directed graphs.

Initially, every node u exchanges its rank with all its neighbors (incoming and outgoing) and stores the set of its neighbors in $N_{u,SCDAS}$, a variable holding the set of neighbors in SCDAS. Also, it maintains the list of its lower rank neighbors in $Lower\_Rank_u$. Again note that in this section, whenever we talk about neighbors in general, we mean both incoming and outgoing neighbors. Four messages are used in conjunction with the domination test. *Dominator_Query* and *Dominator_Reply*

messages are used to verify if the nodes dominated by u have other dominators or not. Likewise, *Absorbent_Query* and *Absorbent_Reply* message are used to verify if the nodes absorbed by u have other absorbents. An outgoing neighboring node v includes the list of its dominators, D_v, in *Dominator_Reply*(v, D_v) and an incoming neighbor v includes the list of its absorbents, A_v, in *Absorbent_Reply*(v, A_v). The same mechanism as described in chapter 3 is used to avoid simultaneous drop-out of nodes while performing this test.

The connectivity test at node u examines if the subgraph induced by $N_{u,SCDAS}$ is strongly connected. If so, node u drops out of the SCDAS. The rest of the actions taken by nodes, such as sending the *Finished_Msg(status)* when they finish running the algorithm or how they update their effective degree when they receive a *Finished_Msg(status)* from a neighbor are the same as before. The formal description of this algorithm is given in Algorithm 4.

Algorithm 3 Centralized Strongly Connected Dominating and Absorbent Set (SCDAS) Algorithm

$SCDAS \leftarrow V$
$P \leftarrow V$
while $P \neq \emptyset$ **do**
  $u \leftarrow argmin\{(\delta(v), id(v)) | v \in P\}$
  $P \leftarrow P - \{u\}$
  $DAT \leftarrow true$
  //For every node v dominated by u, check if there is some other node
  //that dominates v
  for all $v \in N_a(u)$ **do**
    if $((N_d(v) \cap SCDAS) - \{u\} = \emptyset)$ **then**
      $DAT \leftarrow false$
    end if
  end for
  //For every node v absorbed by u, check if there is some other node
  //that absorbs v
  for all $v \in N_d(u)$ **do**
    if $((N_a(v) \cap SCDAS) - \{u\} = \emptyset)$ **then**
      $DAT \leftarrow false$
    end if
  end for
  if DAT **then**
    if $G[((N_a(v) \cup N_d(v)) \cap SCDAS) - \{u\}]$ is strongly connected **then**
      $SCDAS \leftarrow SCDAS - \{u\}$
      for all $(v \in N_a(u) \wedge v \in SCDAS)$ **do**
        $\delta(v) \leftarrow \delta(v) - 1$
      end for
      for all $(v \in N_d(u) \wedge v \in SCDAS)$ **do**
        $\delta(v) \leftarrow \delta(v) - 1$
      end for
    end if
  end if
end while
Return SCDAS

Algorithm 4 Distributed Connected Dominating and Absorbent Set Algorithm, executed by node u

when $Lower\_Rank_u = \emptyset$
$DAT \leftarrow true$
Send $Dominator\_Query$ to $N_a(u)$ and $Absorbent\_Query$ to $N_d(u)$
for all $(v \in (N_a(u) \cup N_d(u)))$ **do**
  Wait for $Dominator\_Reply(v, D_v)$ and $Absorbent\_Reply(v, A_v)$
  if $((D_v - \{u\} = \emptyset) or (A_v - \{u\} = \emptyset))$ **then**
    $DAT \leftarrow false$
  end if
end for
if DAT **then**
  if $G[N_{u,SCDAS} - \{u\}]$ is strongly connected **then**
    $Status_u \leftarrow out$
    Send $Finished\_Msg(out)$ to $(N_a(u) \cup N_d(u))$
  else
    $Status_u \leftarrow in$
    Send $Finished\_Msg(in)$ to $(N_a(u) \cup N_d(u))$
  end if
else
  $Status_u \leftarrow in$
  Send $Finished\_Msg(in)$ to $(N_a(u) \cup N_d(u))$
end if

Upon receiving ***(Dominator/Absorbent)_Query_Msg*** **from** v:

if $!(Reply\_In\_Transit)$ **then**
  $Reply\_In\_Transit \leftarrow true$
  Send $(Dominator/Absorbent)\_Reply(u, D_u/A_u)$ to v
else Enqueue $(Dominator/Absorbent)\_Query(v)$ in DQQ
end if

Upon receiving *Finished_Msg(status)* from v:

if $status = out$ **then**
  if $((v \in N_a(u)) \& (v \in N_d(u)))$ **then**
    $\delta(u) \leftarrow \delta(u) - 2$
  else
    $\delta(u) \leftarrow \delta(u) - 1$
  end if
  $N_{u,SCDAS} = N_{u,SCDAS} - \{v\}$
end if
if $(Rank(v) < Rank(u))$ **then**
  $Lower\_Rank_u = Lower\_Rank_u - \{v\}$
end if
if $DQQ \neq \emptyset$ **then**
  $v \leftarrow$ Dequeue DQQ
  Send $Dominator/Absorbent\_Reply(u, D_u/A_u)$ to v
else $Reply\_In\_Transit \leftarrow false$
end if

Before proceeding to the experimental results, we explain how a node's effective degree is computed and updated in the above algorithm. Initially, every node u sets it effective degree to 0 ($\delta(u) = 0$). Then it increments $\delta(u)$ for each incoming or outgoing neighbor v. If node u has a bidirectional link to node v; i.e. node v is both in $N_a(u)$ and $N_d(u)$, then node v causes $\delta(u)$ to be incremented by 2. That is why we decrease $\delta(u)$ by 2 during the execution of the algorithm if a bidirectional neighbor v drops out of SCDAS in Algorithm 4.

4.3.1 Performance Analysis

The analysis of time and message complexities of Algorithm 4 for the computation of an SCDAS is not as straightforward as that of Algorithm 2 in Chapter 3 for the

computation of a CDS. The existence of unidirectional links in the network causes a special challenge: If node u has a unidirectional link to node v, then v can directly receive packets from u and is therefore aware of the existence of its incoming neighbor (dominator); however, node u cannot directly hear from node v and thus is not aware of its existence. In other words, the main issue is that a node cannot identify its outgoing (absorbent) neighbor(s). One solution is to have each node in the network emit a beacon, with its ID appended to it, at regular intervals. Any node that receives a beacon appends its own ID and forwards it. Since we assume the underlying graph is strongly connected, every node will eventually hear from its absorbent neighbors and can detect them using the chain of IDs appended by forwarding nodes. Using this solution, the message complexity incurred when a node wants to identify the set of its absorbents, or when it inquires them about their other dominators during the domination and absorbency test, is not necessarily restricted to a certain neighborhood. The reply messages from an absorbent node v to a dominator u may be forwarded along a path containing $O(n)$ nodes. Therefore, using the same analysis as the one explained for Algorithm 2 in Chapter 3, the time and message complexities of the Algorithm 4 are $O(n^2)$ and $O(n^2\Delta^k)$, respectively.

However, for practical reasons, we are interested in networks in which there is a bound on the maximum length of the directed reverse path between any pair of nodes with a directed edge. We call a directed graph *α-reciprocal* if for every directed edge $(u, v) \in E$, there exists a directed path from v to u of length at most α. Under this assumption, the bound for the time and message complexities of Algorithm 4 would

be $O(\alpha n)$ and $O(\alpha n \Delta^k)$, respectively.

4.4 Experimental Results

We conducted extensive simulations to evaluate the performance of our algorithm in networks with asymmetric links. To study the impact of various *percentages of unidirectional links (PUL)* as well as different node densities on the size of the constructed SCDAS, we extended our simulations described in chapter 3 as explained in the following. The nodes are randomly distributed in a geographic area of $200m$ by $200m$. Each node is assigned a transmission range randomly selected from the range $[r\_min, r\_max]$.

We present the results in the four following sections. In Section 4.4.1, we look at the relationship between the ratio of the maximum to minimum transmission range on the percentage of unidirectional links in the input graphs. In Section 4.4.2, we investigate the impact of the degree of locality with which nodes run the connectivity test in Algorithm 4 on the size of the constructed SCDASs. Then, we will compare the performance of our algorithm with that of its competitors in Sections 4.4.3 and 4.4.4 under varying node densities and percentages of unidirectional links.

4.4.1 Impact of Transmission Range on The Percentage of Unidirectional Links

Clearly, the $[r\_min, r\_max]$ range affects the percentage of unidirectional links in the network in that the latter is proportional to the relative difference between r_min and r_max. Therefore, we created five different scenarios in which we experimented with different ranges to generate different PULs. The transmission ranges in the first four scenarios were selected from $[10m, 50m]$, $[20m, 50m]$, $[30m, 50m]$, and $[40m, 50m]$ respectively. In the last set, all the nodes were assigned the fixed transmission range of $50m$ to make it possible to also compare differences between UDGs and DGs as input graphs. In each of the above five scenarios, we also varied the number of nodes n in the network form 50 to 300 with increments of 50 to investigate the impact of node density. For each value of n in each scenario, we generated as many random graphs as required until we had 1000 *strongly connected* graphs. The graphs were stored in files and used across different simulations using different algorithms. Before proceeding to present the results, it is useful to first have a look at the input graphs to investigate the relationship between the ratio of the maximum to minimum transmission range and the percentage of unidirectional links in the graphs.

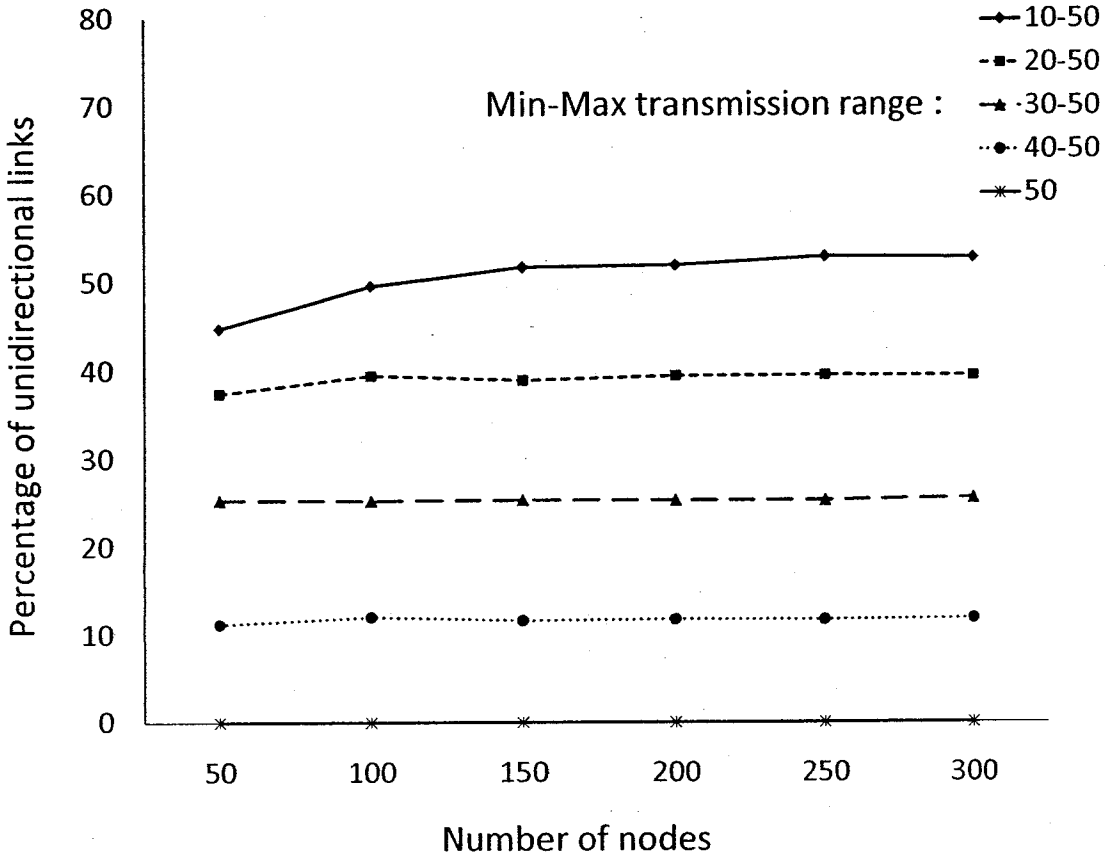


Figure 18: Relationship between maximum-to-minimum transmission range ratio and percentage of unidirectional links

As illustrated in Figure 18, it can be generally seen that the percentage of unidirectional links in the network is a function of the ratio of maximum to minimum transmission range and is almost independent of the node density in the network. For example, in our simulations, when transmission range varies between $40m$ and $50m$; i.e. $\frac{r\_max}{r\_min} = 1.25$, the percentage of unidirectional links varies between 11.2% to 12% for different values of n. When the $\frac{r\_max}{r\_min}$ ratio is increased to 1.67, 2.5 and 5, the percentage of unidirectional links rises to 25%, 40% and 52%, respectively. The

only cases in which graphs exhibit slightly unpredictable behavior is when the ratio of maximum to minimum transmission range is high ($\frac{r\_max}{r\_min} = 5$) and the network is very sparse ($n = 50, 100$). Finally, as expected, when all the nodes have the identical transmission range of $50m$, there are no unidirectional links in the network.

4.4.2 Impact of Locality on The Size of The SCDAS Constructed by Our Algorithm

We also investigated the effect of the degree of locality k with which the connectivity test is run on the size of the generated SCDAS. Our goal was to experimentally determine the best tradeoff between the degree of locality in this test and the number of nodes that can be pruned. As depicted in Figures 19 and 20 , the curves gradually flatten out beyond $k = 5$ in relatively sparse ($n = 50, 100$) networks and in networks of moderate density ($n = 150$) and there is no considerable reduction in the number of SCDAS nodes by further increasing k. However, in relatively dense ($n = 200$) to very dense networks ($n = 200, 250$), the curves typically become flat faster, at $k = 3$. As the ratio of maximum to minimum transmission range decreases, increasing k becomes less helpful.

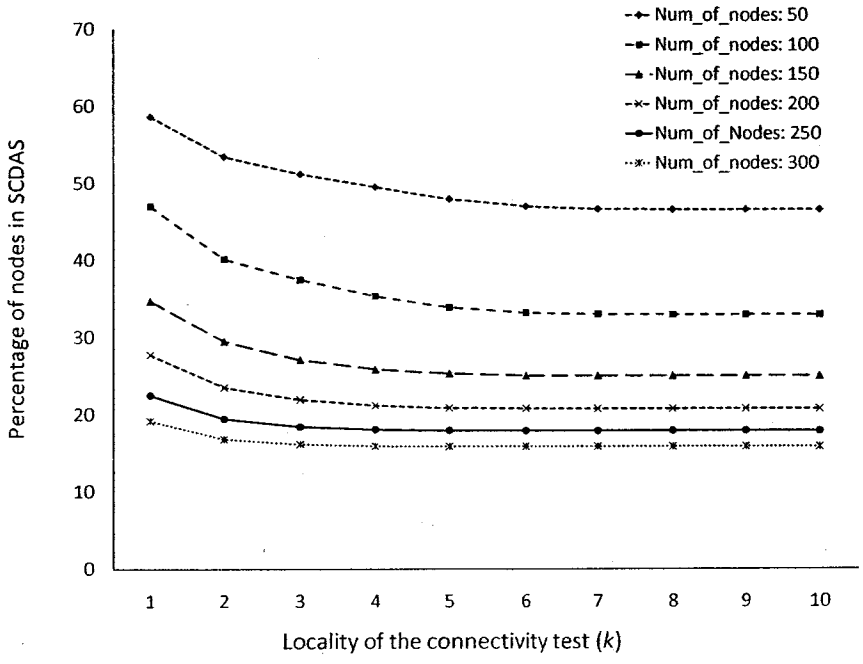


Figure 19: Impact of the locality of connectivity test on the size of the SCDAS when transmission ranges vary between 10 m and 50 m

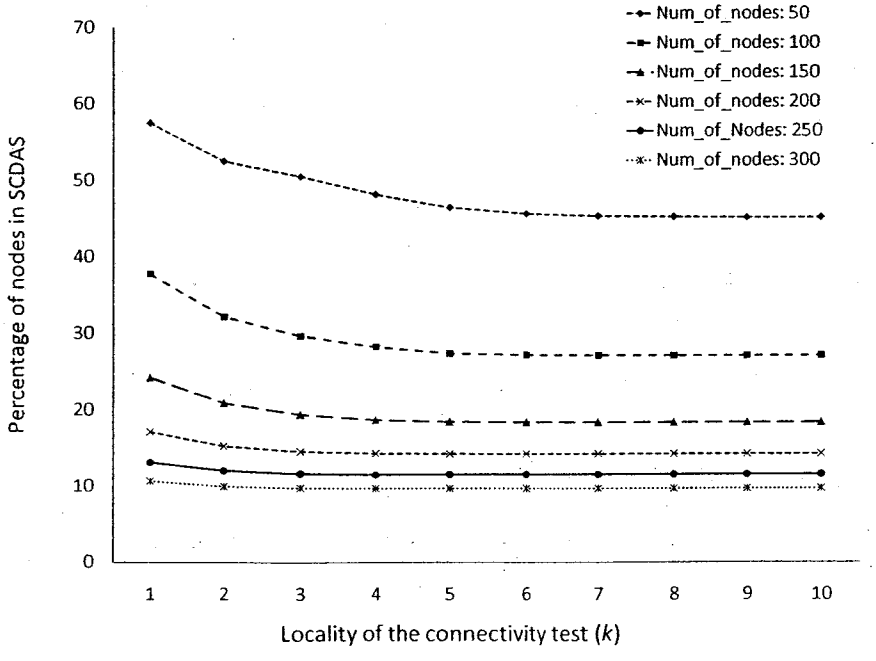


Figure 20: Impact of the locality of connectivity test on the size of the SCDAS when transmission ranges vary between 20 m and 50 m

Another interesting observation is that in very dense networks, especially when the ratio of maximum to minimum transmission range is not more than 2.5, running the connectivity test in the immediate neighborhood yields results that are almost as good as running it for very large values of k, possibly even the diameter of the network. This can be seen in Figures 20 and 21, when the number of nodes is more than 200.

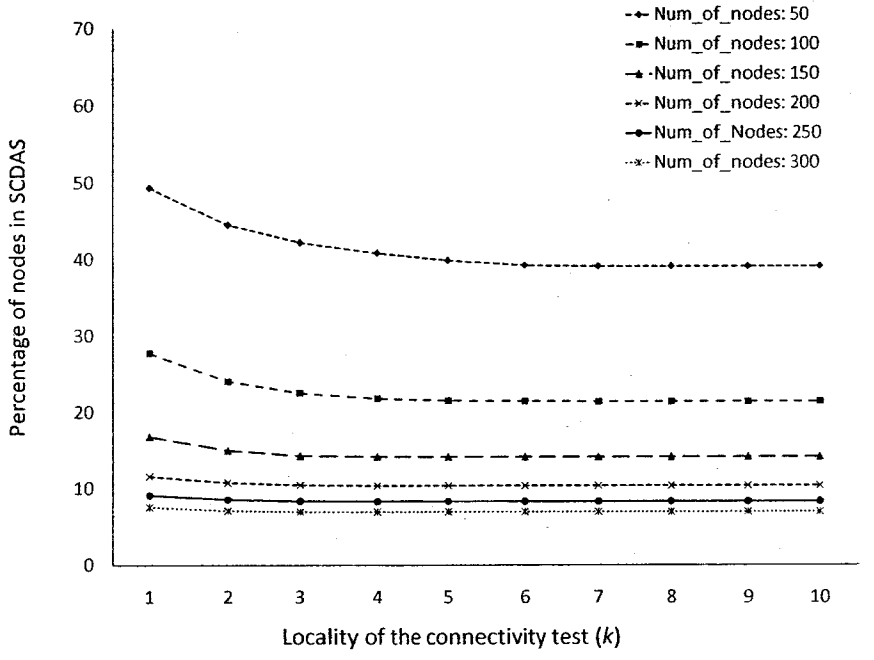


Figure 21: Impact of the locality of connectivity test on the size of the SCDAS when transmission ranges vary between 30 m and 50 m

Since the general trends exhibit more or less the same behavior for scenarios in which the transmission range varies between $40m$ and $50m$, or when all the nodes have the same transmission range of $50m$, the corresponding graphs are not shown.

From the above observations, it seems that $k = 4$ can strike a good compromise between the locality of the algorithm and the size of SCDAS for average networks.

Therefore, in comparing our algorithm with its competitors, we selected two instances of our distributed implementation; namely *PInOut_Unidirectional Distributed ($k = 1$) (PInout_UD1)* and *PInOut_Unidirectional Distributed ($k = 4$) (PInOut_UD4).*

The only algorithms proposed in the literature to construct an SCDAS in networks with different transmission ranges are the ones in [40] and [46], which were discussed in detail in chapter 2. Thus, we compared the performance of our proposed algorithm with the localized marking algorithm in [46], hereafter referred to as *Wu* after the name the author and the two centralized algorithms in [40], namely *Dominating-Absorbent Spanning Tree (DAST)* and *Greedy Strongly Connected Component Merging Algorithm (G_CMA).* In our performance comparison, we focused on the impact of node density and the percentage of unidirectional links on the size of the SCDAS constructed by the four algorithms.

4.4.3 Impact of Node Density

In this section we will investigate the impact of node density on the performance of the four selected SCDAS construction algorithms. Figures 22 through 27 show six different node densities, in ascending order, in the presence of different PULs in the network. As it can be seen in Figure 22, Wu performs better than DAST in sparse networks, especially when PUL is higher. However, as PUL drops below 12% and the underlying graph tends to a UDG, DAST catches up and outperforms Wu. PInOut_UD1 and PInOut_UD4 consistently outperform the other three algorithms while G_CMA is closer to PInOut_UD1, standing in the middle. As the number of

nodes increases to 100 and 150 (moderate densities), DAST consistently outperforms Wu, and the gap between DAST and Wu as one group and G_CMA, PInOut_UD1 and PInOut_UD4 as the other group widens, especially in the presence of PULs of 12% and higher. Another noticeable trend is that DAST almost maintains the same distance from the algorithms in the second group up to PLUs of around 12%, but then considerably narrows down the gap as PUL tends to zero. As the number of nodes increases to 200, 250, and 300 (extremely dense networks), PInOut_UD1 and PInOut_UD4 take the lead more conspicuously and PInOut_UD1 increases its distance from G_CMA. On average, PInOut_UD1 and PInOut_UD4 construct SCDASs which are 24% and 35% smaller than those constructed by G_CMA respectively. The efficiency of our schemes become even more noticeable when we take into account the very high complexity of G_CMA and the fact that it is a centralized algorithm. Finally, the last consistent trend that can be seen from the figures is that as the node density increases, the difference between PInOut_UD1 and PInOut_UD4 decreases and in very dense networks, they perform almost equally well, especially for PULs of 40% and lower.

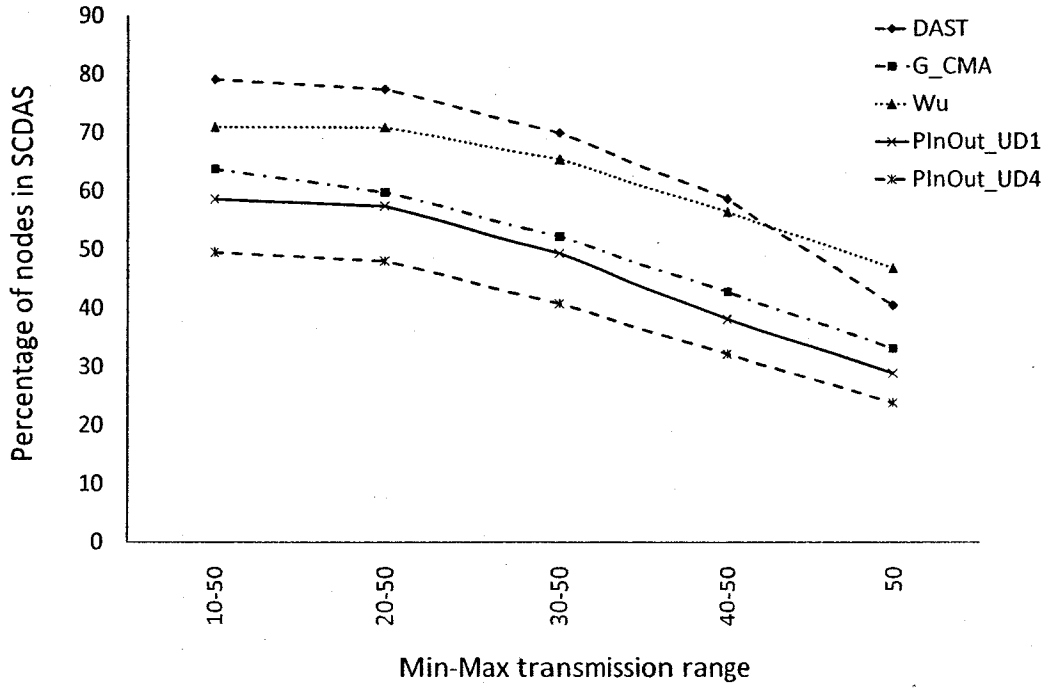


Figure 22: Impact of the node density – number of nodes =50

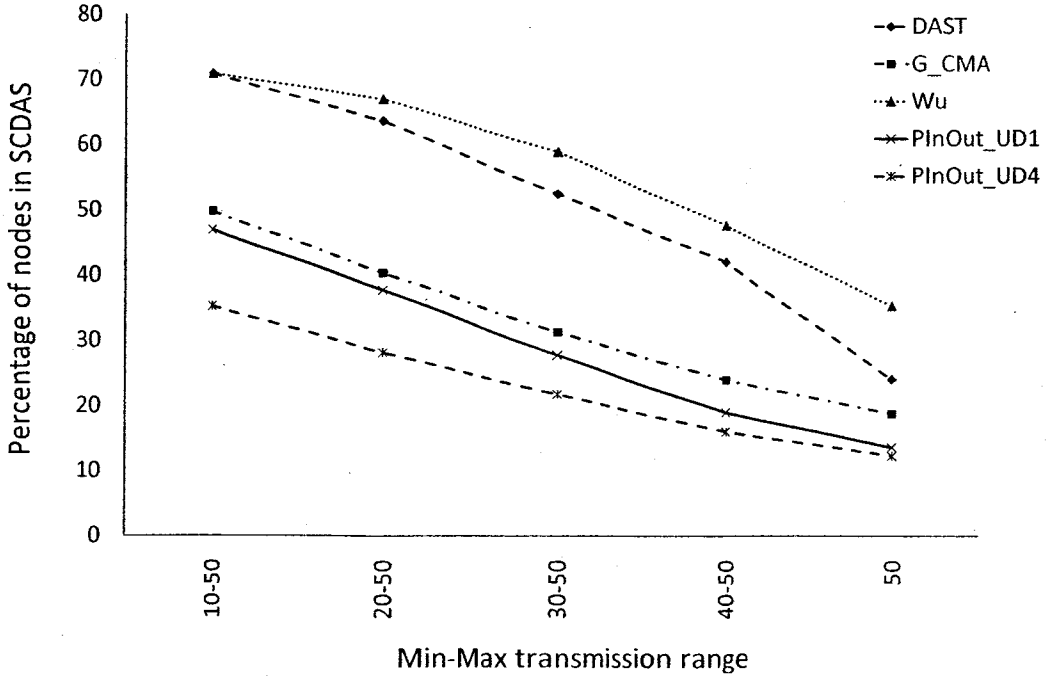


Figure 23: Impact of the node density – number of nodes =100

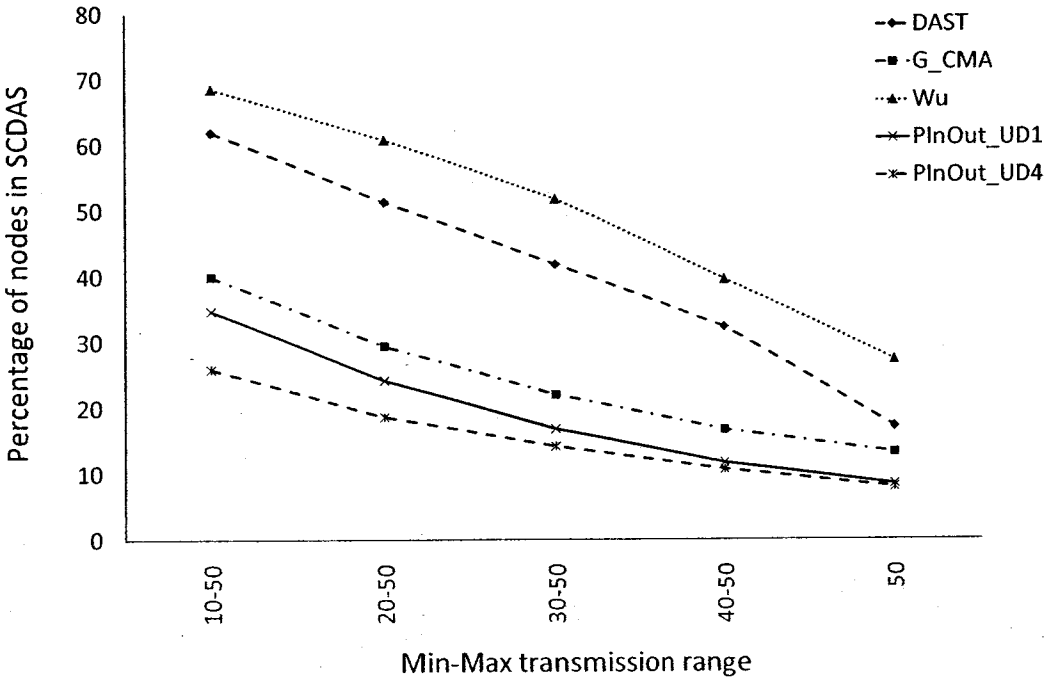


Figure 24: Impact of the node density – number of nodes =150

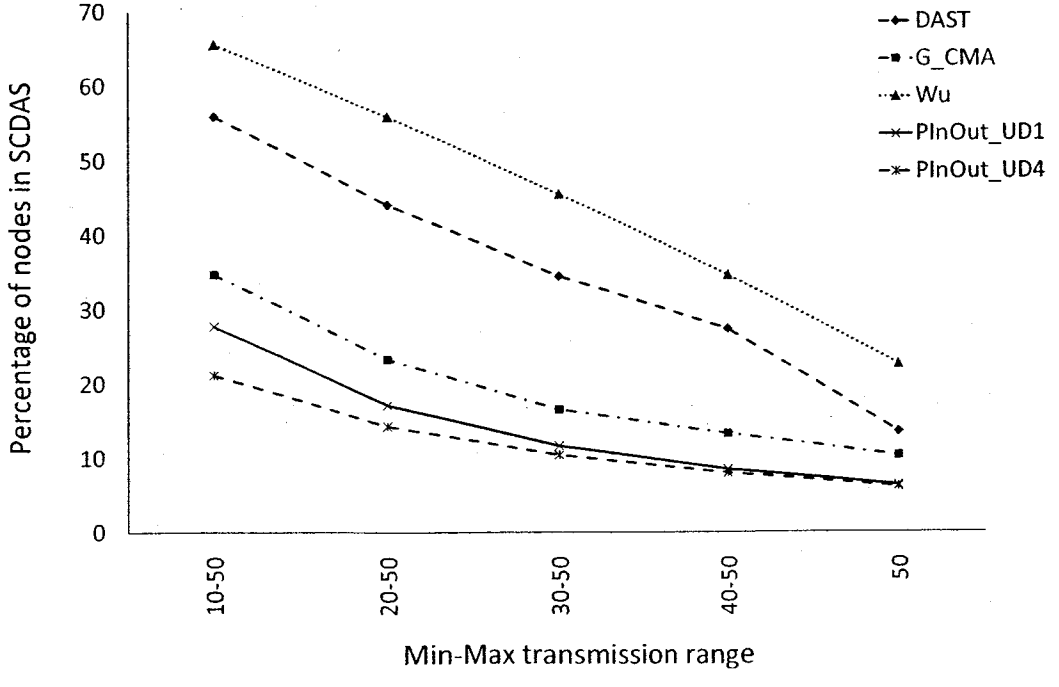


Figure 25: Impact of the node density – number of nodes =200

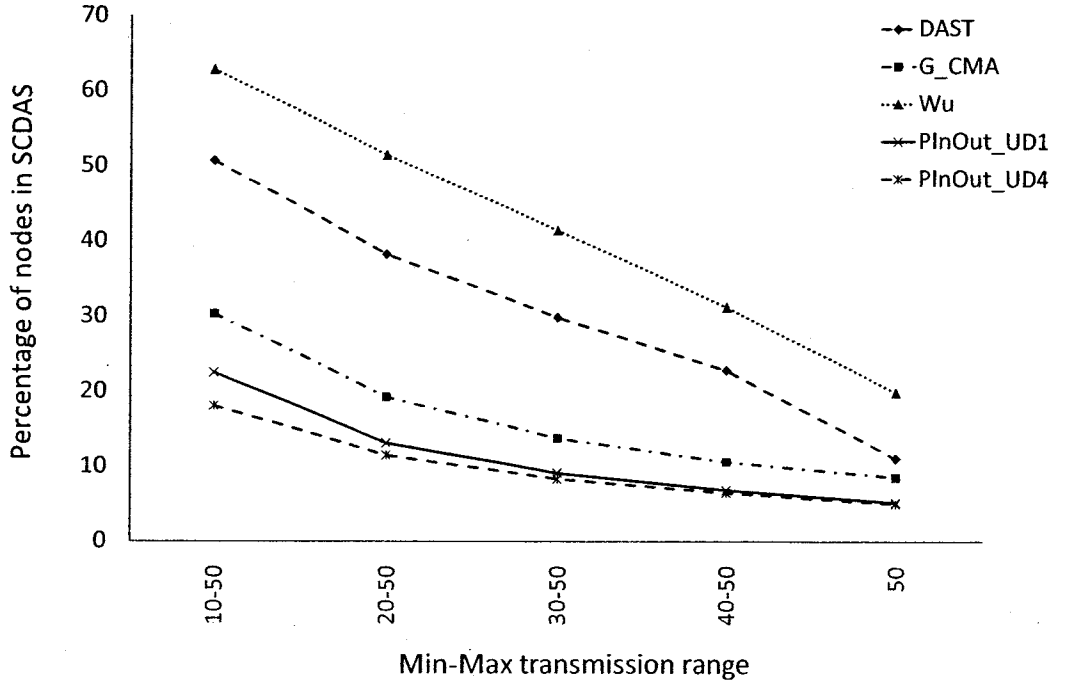


Figure 26: Impact of the node density – number of nodes =250

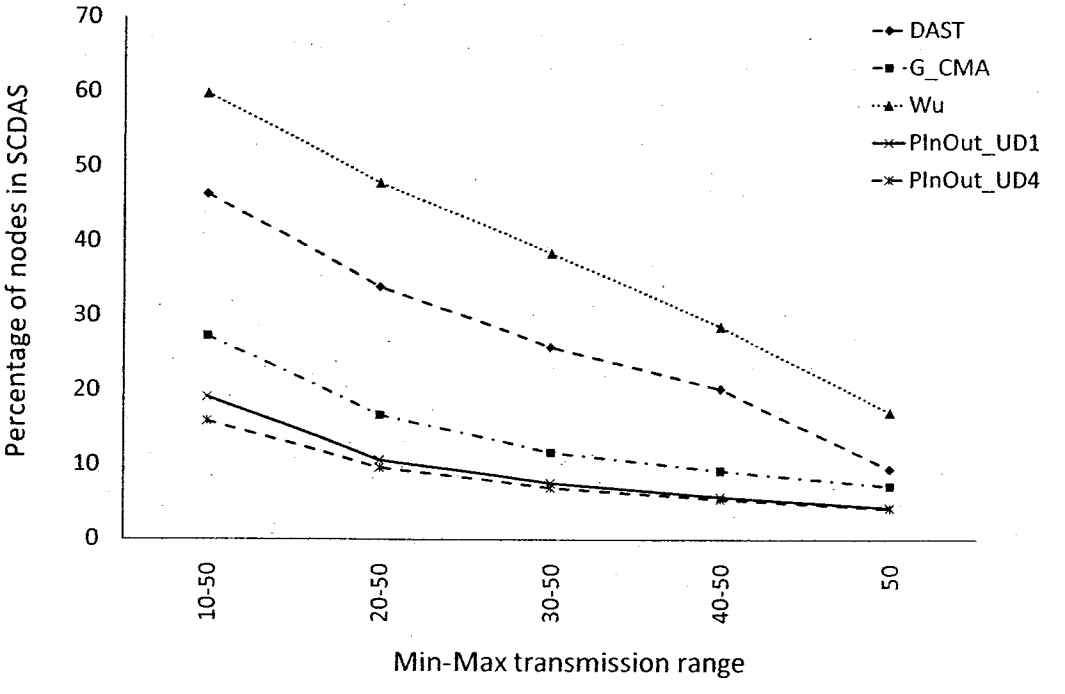


Figure 27: Impact of the node density – number of nodes =300

4.4.4 Impact of Unidirectional Links

In order to better analyze the impact of PUL on the performance of algorithms which we simulated in our experiments, we give a different presentation of our results in this section. We rearranged our results such that studying the behavior of each algorithm for a given range of $[r_{min}, r_{max}]$ becomes easier. Figures 28 through 31 show four different $[r_{min}, r_{max}]$ ranges and the size of the SCDAS constructed by each algorithm under varying number of nodes. As we discussed earlier in this chapter, each $[r_{min}, r_{max}]$ range corresponds to an almost fixed PUL. More specifically, if we ignore the slightly different trends in very sparse networks in the presence of high PULs and round up the average PULs for different $[r_{min}, r_{max}]$ transmission ranges, the transmission ranges [10,50], [20,50], [30,50], and [40,50] correspond to PULs of 52%, 40%, 25%, and 12% respectively. As depicted in Figure 32, we also considered the scenario in which PUL is zero to make it possible to more accurately predict the trends as PUL drops below 12% and tends to zero.

As illustrated in Figure 28, although Wu initially outperforms DAST when $n = 50$, it does not improve much as the node density grows in the presence of high PULs. In other words, when the ratio of maximum to minimum transmission range is very high in the network, Wu cannot take advantage of the increase in node density whereas the other algorithms all benefit from increased node density and reduce the relative size of the SCDAS which they construct. The reason for Wu's inability to use increased density in its favor is that its pruning rules (Rules 1 & 2) are not efficient when PUL is high.

By comparing Figures 28 through 31 with Figure 32, it can be seen that the existence of unidirectional links in the network adversely affects the performance of DAST. As can be seen in Figure 32, when there are no unidirectional links, DAST's performance is closest to G_CMA; it constructs SCDASs which are 128% larger than those generated by G_CMA on average. However, in the presence of unidirectional links, DAST's performance degrades as link density increases. The size of the SCDAS built by DAST is 153%, 175%, 189% and 191% larger than that of G_CMA when the minimum transmission range is 10, 20, 30, and 40 respectively. This shows that DAST is more sensitive to link density compared to G_CMA, PInOut_UD1, and PInOut_UD4.

The last interesting observation is about the relationship between the locality of the connectivity test (k) in our algorithm and the PUL. As shown in the figures, as the PUL decreases, so does the improvement in the SCDAS size as a result of increasing k. The reason is that detecting strong connectivity is more difficult in smaller neighborhoods (e.g. $k = 1, 2$) when a large percentage of links are unidirectional. In other words, the lower the PUL, the smaller the neighborhood required to detect strong connectivity in the graph.

Our simulations show that when PUL is around 52% (transmission range = [10,50]), there is an average reduction of 21% in the size of the SCDAS as k is increased from 1 to 4. However, this gain is reduced to 17%, 13%, and 9% when PUL is 40%, 25% and 12% respectively. As seen in Figure 32, when there are no unidirectional links in the network, and the network is extremely dense, this gain is

only around 3%. In summary, using a larger neighborhood in the connectivity test is helpful when PUL is relatively high. For lower PULs, it helps when the network is not very dense.

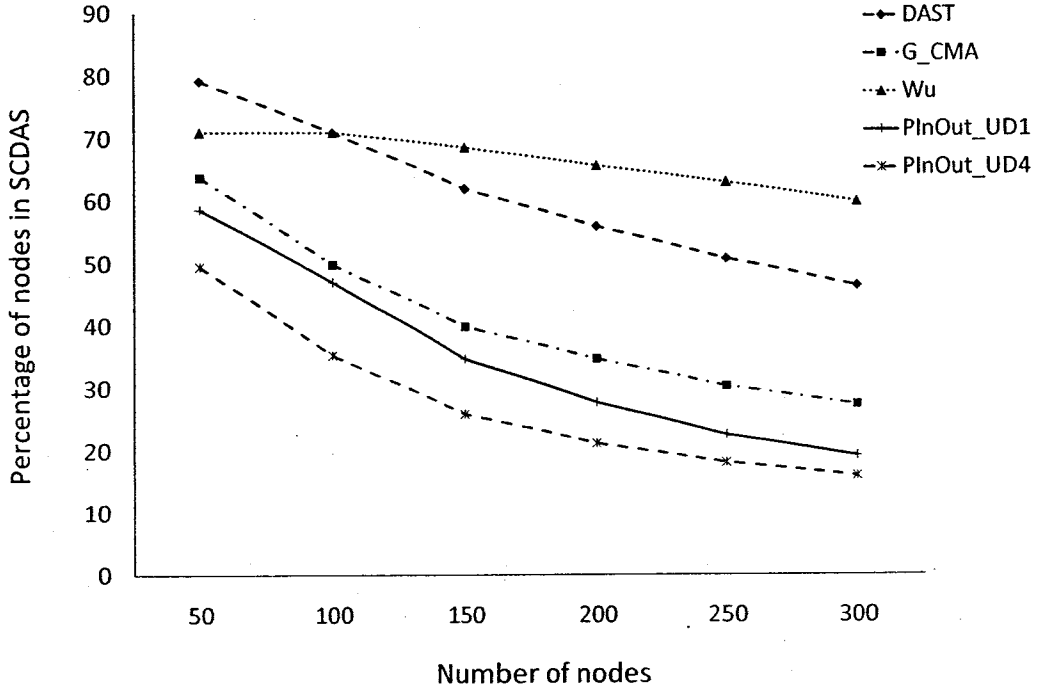


Figure 28: Impact of the percentage of unidirectional links – $[r_{min}, r_{max}] = [10, 50]$

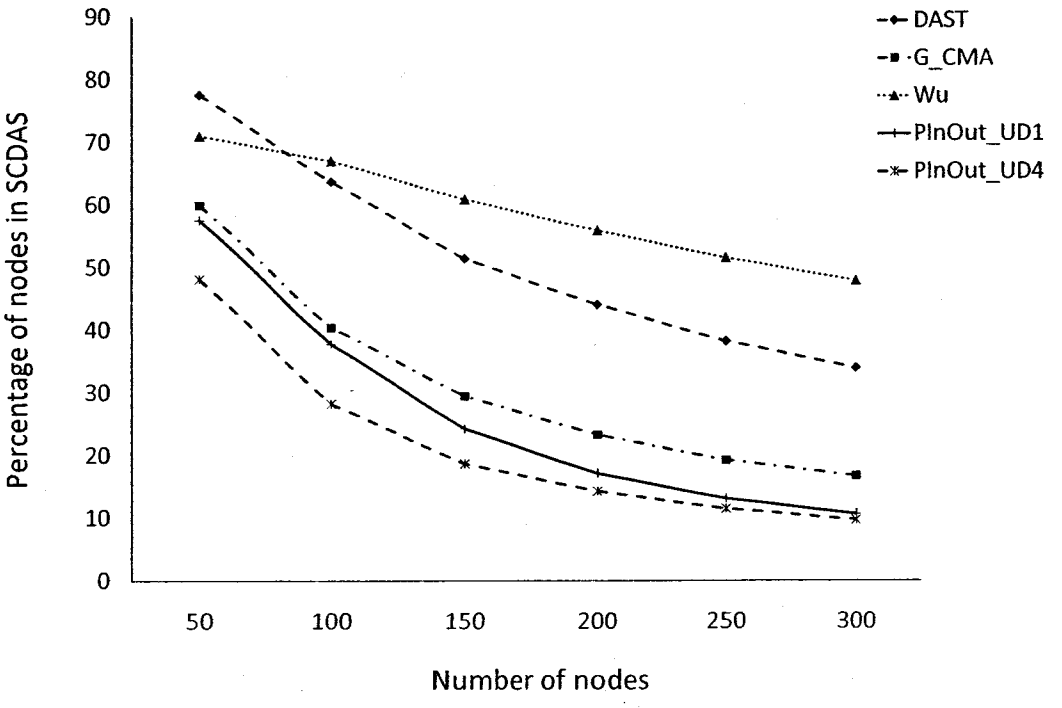


Figure 29: Impact of the percentage of unidirectional links – $[r_{min}, r_{max}] = [20, 50]$

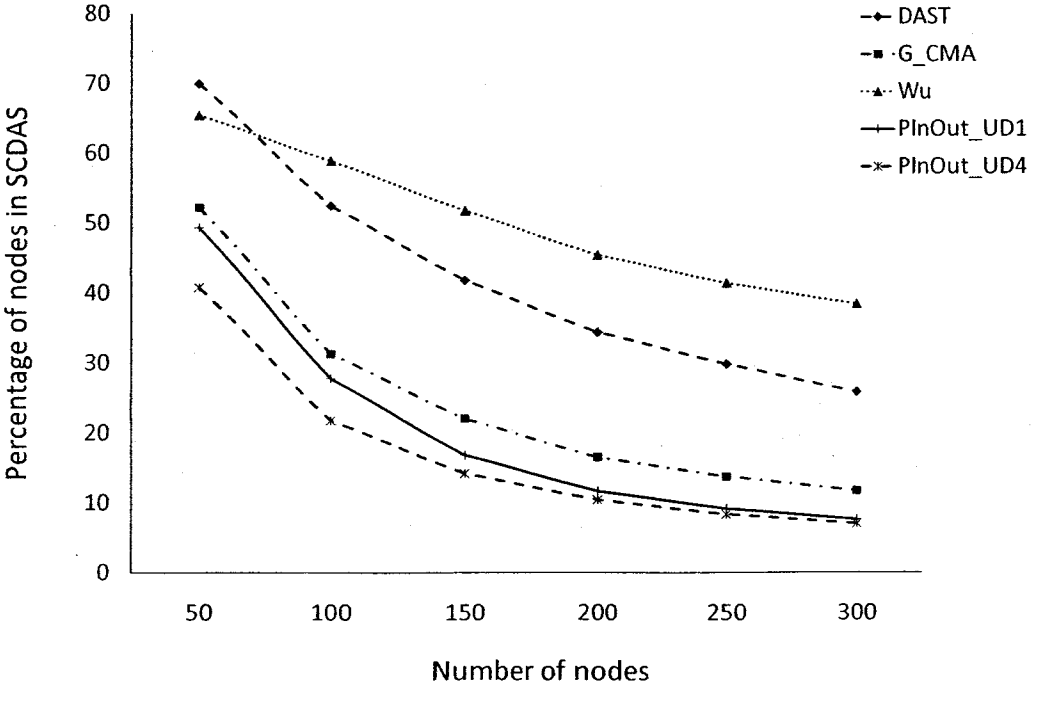


Figure 30: Impact of the percentage of unidirectional links – $[r_{min}, r_{max}] = [30, 50]$

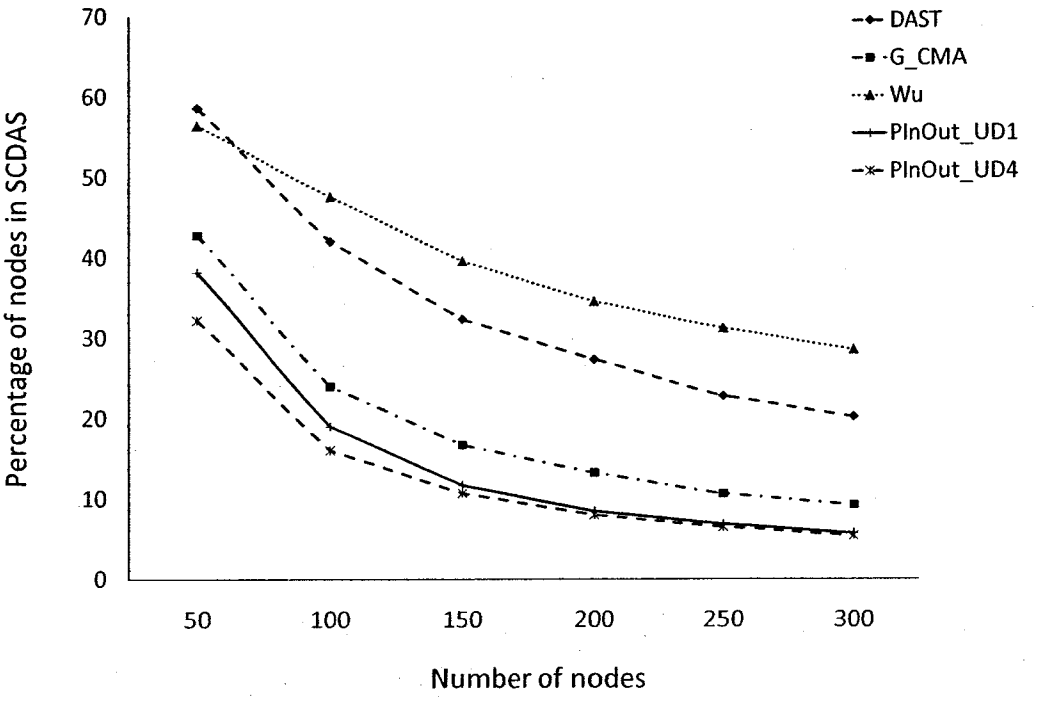


Figure 31: Impact of the percentage of unidirectional links – $[r_{min}, r_{max}] = [40, 50]$

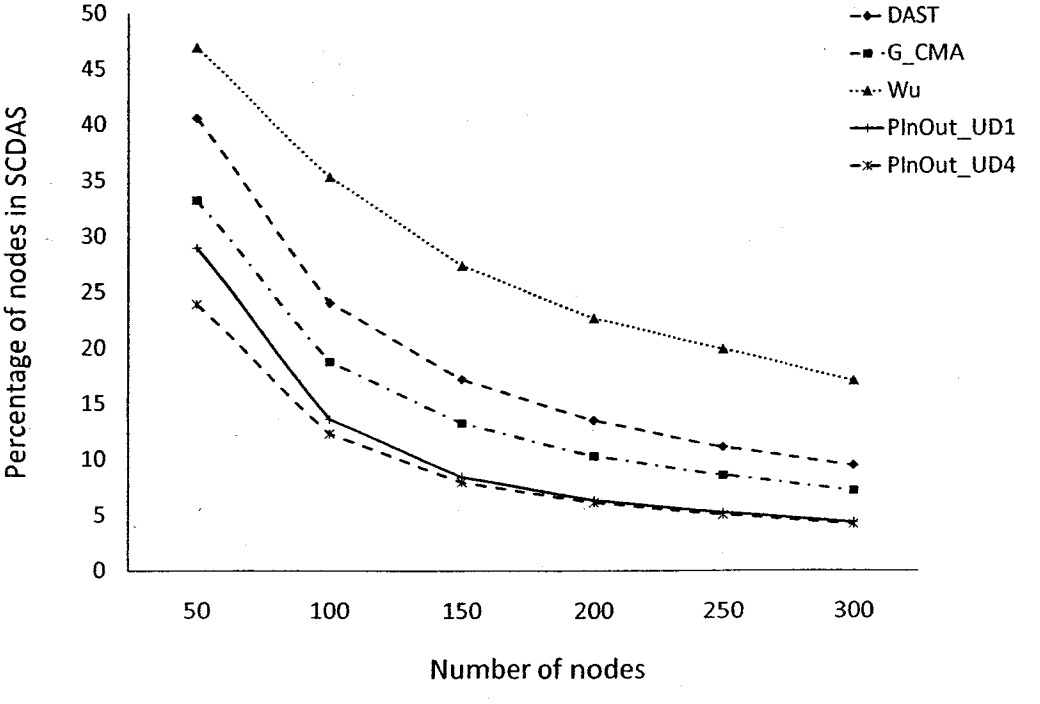


Figure 32: Impact of the percentage of unidirectional links – $[r_{min}, r_{max}] = [50, 50]$

Chapter 5

Conclusions and Future Work

In this thesis, we discussed the significance of providing a hierarchal infrastructure, also referred to as a virtual backbone, in wireless ad hoc networks in order to perform several key functions such as routing, activity scheduling, and topology control. Indeed, many of the defined objectives for ad hoc networks are not easily achievable without first addressing the problem of constructing such an infrastructure in an efficient manner. We then proposed efficient distributed algorithms with linear time and message complexities for the construction of such a backbone in both networks with symmetric and asymmetric links. We also gave a local implementation of our algorithm in location-aware UDGs. Extensive simulations show that our proposed algorithms outperform several classical CDS (SCDAS) construction algorithms in terms of the size of the generated sets. The general ranking scheme defined in our algorithm makes it possible to construct CDSs (SCDASs) based on other goal functions such as energy efficiency. Also, by choosing an appropriate degree of locality when

running the connectivity test, our algorithms provide the desired flexibility to adjust the trade-off between the size and the cost of the constructed set.

The virtual backbones (CDSs/SCDASs) constructed in this thesis only guarantee 1-domination and 1-connectivity. Although in practice, a node may be dominated by more than one CDS (SCDAS) node or there might be more than one path on the CDS (SCDAS) between two nodes in the graph, this is not guaranteed. In fact, our algorithm attempts to eliminate such redundancies to reduce the size of the constructed sets as much as possible. In order to deal with the lack of such redundancy which proves to be desirable when nodes fail, we proposed schemes to locally update the CDS (SCDAS) in case of node failures. Recently, another approach for achieving such robustness was proposed in [55]. In their approach, the authors use the concept of km-CDS in which the constructed CDS guarantees m-domination and k-connectivity in the underlying graph provided such a CDS exists. By ensuring that every node is dominated by at least m neighboring nodes in the CDS and that there exists at least k different paths between any pair of nodes in the CDS, the desired level of robustness and fault tolerance can be achieved as a built-in feature of the resulting CDS. Indeed, our approach in dealing with node failures has a reactive nature whereas constructing a km-CDS is a proactive scheme in providing such fault tolerance.

Two algorithms were proposed in [55] to construct a km-CDS. The first one is a centralized algorithm which first constructs an m-dominating set and then augments it to ensure k-connectivity. The second one is a distributed algorithm which constructs a 1-dominating set m times followed by making this m-dominating set k-connected.

Based on the observations provided in chapters 3 and 4, algorithms that construct a CDS in different phases tend to perform rather poorly in terms of the size of the resulting CDS since they often add unnecessary nodes to the set. The fact that such a scheme is used iteratively in the algorithms proposed in [55] may aggravate the situation. Therefore, one promising direction for future work is to adapt our algorithm for the construction of km-CDS in networks with symmetrical links. We predict that our adapted algorithm will be able to generate km-CDSs of much smaller sizes than those generated by algorithms in [55]. To the best of our knowledge, no work has investigated the use of such a scheme in networks with asymmetrical links. Thus, adapting our algorithm in Chapter 4 to construct a km-SCDAS can be of special interest.

Additionally, well-defined and extensive experiments should be conducted to evaluate and compare the efficiency of the two reactive and proactive approaches described above to verify which one provides the desired level of fault tolerance more efficiently while minimizing the control overhead and maximizing network lifetime. Due to the wide variety of applications and flavors conceived for ad hoc networks and the subsequent need to address issues of possibly completely different nature, it will not be surprising if no "one-size-fits-all" solution can be proposed. Therefore, it is of great importance to investigate the relevance of each approach to a given application.

Finally, another important direction for future work is the design and development of full-fledged protocols which use the algorithms proposed in this thesis at their core. Although some critical implementation issues were discussed in Chapter 3, many other

significant implementation details that are beyond the scope of this thesis remain to be worked out. These include message synchronization, size and format among others. Once all the protocol specifications are carefully defined, it will be possible to conduct experiments in order to investigate other important metrics such as protocol duration, byte overhead, energy consumption and resilience to node failures.

Bibliography

[1] K. Alzoubi, P.-J. Wan, and O. Frieder. New distributed algorithm for connected dominating set in wireless ad hoc networks. In *Proceedings of the 35th Hawaii International Conference on System Sciences*, pages 3849–3855, Jan. 2002.

[2] K. M. Alzoubi, P.-J. Wan, and O. Frieder. Message-optimal connected dominating sets in mobile ad hoc networks. In *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 157–164, 2002.

[3] L. Barrire, P. Fraigniaud, L. Narayanan, and J. Opatrny. Robust position-based routing in wireless ad hoc networks with unstable transmission ranges, 2001.

[4] S. Basagni. Distributed clustering for ad hoc networks. In *ISPAN '99: Proceedings of the 1999 IEEE International Symposium on Parallel Architectures, Algorithms and Networks*, pages 310–315, 1999.

[5] S. Basagni, M. Mastrogiovanni, and C. Petrioli. A performance comparison of protocols for clustering and backbone formation in large scale ad hoc network. In *Proceedings of The 1st IEEE International Conference on Mobile Ad Hoc and Sensor Systems, MASS 2004*, pages 70–79, October 25–27 2004.

[6] S. Butenko, X. Cheng, C. A. Oliviera, and P. Pardalos. A new heuristic for the minimum connected dominating set problem on ad hoc wireless networks. In *Recent Developments in Cooperative control and optimization*, pages 61–73, 2004.

[7] M. Cardei, X. Cheng, X. Cheng, and D. zhu Du. Connected domination in multihop ad hoc wireless networks. In *Proceedings of the 6th International Conference on Computer Science and Informatics*, 2002.

[8] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *ACM Wireless Networks*, pages 85–96, 2002.

[9] D. Chen, D.-Z. Du, X.-D. Hu, G.-H. Lin, L. Wang, and G. Xue. Approximations for steiner trees with minimum number of steiner points. *Journal of Global Optimization*, 18(1):17–33, 2000.

[10] X. Cheng, X. Huang, D. Li, and D. zhu Du. Polynomial-time approximation scheme for minimum connected dominating set in ad hoc wireless networks. *Networks*, 42(4):202–208, 2003.

[11] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[12] I. Cidon and O. Mokryn. Propagation and leader election in a multihop broadcast environment. In *12th International Symposium on Distributed Computing (DISC98)*, volume LNCS 1499, pages 104–118, 1998.

[13] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Math.*, 86(1-3):165–177, 1990.

[14] J. Czyzowicz, S. Dobrev, T. Fevens, H. González-Aguilar, E. Kranakis, J. Opatrny, and J. Urrutia. Local algorithms for dominating and connected dominating sets of unit disk graphs with location aware nodes. In *Proc. of the 8th Latin American Symposium on Theoretical Informatics (LATIN'08)*, volume LNCS 4957, pages 158–169, 2008.

[15] F. Dai and J. Wu. An extended localized algorithm for connected dominating set formation in ad hoc wireless networks. *IEEE Trans. Parallel Distrib. Syst.*, 15(10):908–920, 2004.

[16] B. Das and V. Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *International Conference on Communications*, pages 376–380, 1997.

[17] B. Das, R. Sivakumar, and V. Bharghavan. Routing in ad hoc networks using a spine. In *International Conference on Computers and Communication Networks*, pages 34–39, 1997.

[18] S. Datta, I. Stojmenovic, and J. Wu. Internal node and shortcut based routing with guaranteed delivery in wireless networks. In *Cluster Computing*, pages 169–178, 2002.

[19] M. Ding, X. Cheng, and G. Xue. Aggregation tree construction in sensor networks. In *Vehicular Technology Conference (VTC 2003-Fall)*, pages 2168–2172, 2003.

[20] D.-Z. Du, L. Wang, and B. Xu. The euclidean bottleneck steiner tree and steiner tree with minimum number of steiner points. In *COCOON'01*, pages 509–518, 2001.

[21] U. Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, July 1998.

[22] S. Funke, A. Kesselman, U. Meyer, and M. Segal. A simple improved distributed algorithm for minimum cds in unit disk graphs. *ACM Trans. Sen. Netw.*, 2(3):444–453, 2006.

[23] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1978.

[24] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4):374–387, 1998.

[25] B. Han. Zone-based virtual backbone formation in wireless ad hoc networks. *Ad Hoc Netw.*, 7(1):183–200, 2009.

[26] L. Jia, R. Rajaraman, and T. Suel. An efficient distributed algorithm for constructing small dominating sets. *Distrib. Comput.*, 15(4):193–205, December 2002.

[27] H. Kassaei, M. Mehrandish, L. Narayanan, and J. Opatrny. Efficient algorithms for connected dominating sets in ad hoc networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference(WCNC), to appear. 2010.*

[28] H. Kassaei, M. Mehrandish, L. Narayanan, and J. Opatrny. A new local algorithm for backbone formation in ad hoc networks. In *PE-WASUN '09: Proceedings of the 6th ACM symposium on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*, pages 49–57, 2009.

[29] F. Kuhn and R. Wattenhofer. Constant-time distributed dominating set approximation. *Distrib. Comput.*, 17(4):303–310, 2005.

[30] H. Lim and C. Kim. Multicast tree construction and flooding in wireless ad hoc networks. In *MSWIM '00: Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 61–68, 2000.

[31] H. Lim and C. Kim. Flooding in wireless ad hoc netwroks. *Computer Communications Journal*, 24:353–363, 2001.

[32] G.-H. Lin and G. Xue. Steiner tree problem with minimum number of steiner points and bounded edge-length. *Information Processing Letters*, 69(2):53–57, 1999.

[33] N. Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

[34] W. Lou and J. Wu. On reducing broadcast redundancy in ad hoc wireless networks. *IEEE Transactions on Mobile Computing*, 1:111–123, 2002.

[35] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Operating Systems Review*, 36(Special Issue):131–146, 2002.

[36] M. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25:59–68, 1995.

[37] M. Min, H. Du, X. Jia, C. X. Huang, S. C.-H. Huang, and W. Wu. Improving construction for connected dominating set with steiner tree in wireless sensor networks. *Journal of Global Optimization*, 35(1):111–119, 2006.

[38] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *MobiCom '99: Proceedings of the 5th annual*

ACM/IEEE international conference on Mobile computing and networking, pages 151–162, 1999.

[39] T. Nieberg and J. Hurink. A PTAS for the minimum dominating set problem in unit disk graphs. *Approximation and Online Algorithms*, pages 296–306, 2006.

[40] M. A. Park, J. Willson, C. Wang, M. Thai, W. Wu, and A. Farago. A dominating and absorbent set in a wireless ad-hoc network with different transmission ranges. In *Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc '07)*, pages 22–31, 2007.

[41] C. E. Perkins. *Ad Hoc Networking.* Addison-Wesley, 2001.

[42] R. Sivakumar, B. Das, and V. Bharghavan. An improved spine-based infrastructure for routing in ad hoc networks. In *International Symposium on Computers and Communications (ISCC'98)*, 1998.

[43] R. Sivakumar, P. Sinha, and V. Bharghavan. Cedar: a core-extraction distributed ad hoc routing algorithm. *IEEE Journal on Selected Areas in Communications*, 17:1454–1465, 1999.

[44] I. Stojmenovic, M. Seddigh, and J. Zunic. Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 13:14–25, 2001.

[45] A. Wiese and E. Kranakis. Local PTAS for dominating and connected dominating set in location aware unit disk graphs. pages 227–240. Springer-Verlag, 2009.

[46] J. Wu. Extended dominating-set-based routing in ad hoc wireless networks with unidirectional links. *IEEE Trans. Parallel Distrib. Syst.*, 13(9):866–881, 2002.

[47] J. Wu and F. Dai. Broadcasting in ad hoc networks based on self-pruning. In *IEEE INFOCOM*, pages 29–39, 2003.

[48] J. Wu, F. Dai, M. Gao, and I. Stojmenovic. On calculating power-aware connected dominating sets for efficient routing in ad hoc wireless networks. *IEEE/KICS Journal of Communications and Networks*, 4:59–70, 2002.

[49] J. Wu, M. Gao, and I. Stojmenovic. On calculating power-aware connected dominating sets for efficient routing in ad hoc wireless networks. In *International Conference on Parallel Processing (ICPP)*, pages 346–356, 2001.

[50] J. Wu and H. Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *DIALM '99: Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 7–14, 1999.

[51] J. Wu and B. Wu. A transmission range reduction scheme for power-aware broadcasting in ad hoc networks using connected dominating sets. In *Vehicular Technology Conference (VTC2003-Fall)*, volume 5, pages 2906– 2909, 2003.

[52] J. Wu, B. Wu, and I. Stojmenovic. Power-aware broadcasting and activity scheduling in ad hoc wireless networks using connected dominating sets. *Wireless Communications and Mobile Computing, a special issue on Research in Ad Hoc Networking, Smart Sensing and Pervasive Computing*, 3(4):425–438, 2003.

[53] W. Wu, H. Du, X. Jia, Y. Li, and D. D.-z. Huang, Scott C.-H. Maximal independent set minimum connected dominating set in unit disk graphs. *Technical Report TR04-047, Department of Computer Science and Engineering, University of Minnesota*, December 2004.

[54] W. Wu, H. Du, X. Jia, Y. Li, and S. C.-H. Huang. Minimum connected dominating sets and maximal independent sets in unit disk graphs. *Theoretical Computer Science*, 352(1):1–7, 2006.

[55] Y. Wu, F. Wang, M. T. Thai, and Y. Li. Constructing k-connected m-dominating sets in wireless sensor networks. In *Military Communications Conference, 2007. MILCOM 2007. IEEE*, pages 1–7, 2007.

[56] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 70–84, 2001.

[57] Z. Yuanyuan, X. Jia, and H. Yanxiang. Energy efficient distributed connected dominating sets construction in wireless sensor networks. In *IWCMC '06: Proceedings of the 2006 ACM international conference on Wireless communications and mobile computing*, pages 797–802, 2006.