# CLUSTERING MOBILE AD HOC NETWORKS USING GRAPH DOMINATION

by

Yuanzhu Peter Chen
B.Sc., Peking University, 1999

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in the School
of
Computing Science

© Yuanzhu Peter Chen 2004
SIMON FRASER UNIVERSITY
August 2004

All rights reserved. This work may not be
reproduced in whole or in part, by photocopy
or other means, without the permission of the author.

## APPROVAL

**Name:** Yuanzhu Peter Chen

**Degree:** Doctor of Philosophy

**Title of thesis:** Clustering Mobile Ad Hoc Networks Using Graph Domination

**Examining Committee:** Dr. Ramesh Krishnamurti
Chair

---

Dr. Arthur L. Liestman, Senior Supervisor

---

Dr. Tiko Kameda, Supervisor

---

Dr. Joseph G. Peters, Supervisor

---

Dr. Jiangchuan Liu, SFU Examiner

---

Dr. Ralf Klasing, External Examiner,
Project MASCOTTE,
INRIA

**Date Approved:** August 6. 2004

# SIMON FRASER UNIVERSITY

## Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author's written permission.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Bennett Library
Simon Fraser University
Burnaby, BC, Canada

# Abstract

A mobile ad hoc network (MANET) is an infrastructureless wireless communication network that supports mobile users. Scalability has been a critical issue for mobile ad hoc networks to become commercially successful. Building and maintaining hierarchies among user devices help to reduce control overhead and to address the scalability issue. Partitioning networks into smaller substructures, called *clustering*, is a fundamental building block for constructing and maintaining hierarchies.

In this thesis, we propose and test various algorithms to construct and maintain a clustered network structure based on graph domination. We start with two approximation algorithms for constructing small *weakly-connected dominating sets* with logarithmic performance ratios. These algorithms are implemented in a distributed setting with certain parallelism. To further limit the execution of the above distributed algorithms, we propose a zonal scheme to divide the execution into two levels, i.e. interzonal and intrazonal. The maintenance is also carried out according to the zonal scheme. We also define *gemini sets* to extend the results to directed graphs to model mobile ad hoc networks in the presence of unidirectional links.

*To my parents from whom I have come,*
*and my fiancee with whom I shall proceed.*

# Acknowledgments

I would like to thank Professor Arthur Liestman, my senior supervisor, for his constant unconditional support for my thesis work over the years. Being both a mentor and a true friend, Professor Liestman has made this journey unforgettable and invaluable. Your art of cultivating pupils has enlightened me the spirit of science – research and propagation.

I gratefully appreciate the School of Computing Science and Simon Fraser University who gave me the precious opportunity to bury this milestone of my life. Without the generous support from Professor Liestman, the School, and the University, this thesis would not have been possible.

Thanks to the Network Modeling Research Group and Professor Joseph Peters, those of us working on different topics using different methods get to join force in each other. The group meetings have been a major forum for us to obtain feedback from peers.

The members of the Network Lab and NLACF (Network Lab Association for Consumption of Food) have been a special source of joy and inspiration. Countless research ideas would have sparkled unseen without the perseverance in culinary search.

Last but not least, I would like to thank NSERC for the sponsorship during my thesis work that has shielded me from any financial distractions.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Telecommunication has been one of the most dynamic research fields in recent decades and shapes people's everyday life in many aspects. Wireless communication technology frees users from being tethered to the fixed communication infrastructure. These users ultimately desire communication networks with absolutely no constraints. Emerging technologies, such as cellular telephone systems, satellite networks, and wireless local area networks, give people much more freedom in communications. However, they are still constrained by ground or space based stations. Thus, one major goal of research in mobile communications is to further exploit the potential of mobile communication systems.

A mobile ad hoc network (MANET) [1] is a special type of wireless communication network, which consists solely of mobile hosts and dispenses with infrastructure. This new networking scheme presents new challenges such as medium access and sharing, routing, energy conservation, topology control, and security. Despite abundant research results produced in this area since the early 70's when the first ad hoc networking projects where launched, the ad hoc networking technology is still not commercially successful. One major reason is that the current protocols do not scale up well.

Building hierarchies among network nodes helps to abstract the network topology and hides unnecessary local details from distant parts of the network. From a graph theoretic point of view, by partitioning vertices into clusters, the network can be viewed as a graph on these clusters. Graph domination, a branch of graph theory, provides a good methodology for clustering ad hoc networks.

[1] Visit `http://www.ietf.org/html.charters/manet-charter.html` for the official charter

In this chapter, we first review the wireless communication schemes that preceded ad hoc networking. Then, we introduce ad hoc networking and its special challenges and research issues. In Section 1.3, we motivate the necessity of building hierarchies in ad hoc networks to solve the scaling problem that has been hindering ad hoc networking from achieving widespread use. We define the graph domination problem, the essential graph theoretic notion underlying this thesis, and its variants in Section 1.4. The computational complexity and approximability of these variants are reviewed. Ad hoc networks can be modeled for research purposes in different ways, as briefly discussed in Section 1.5, and we adopt the classic multi-port model in this work. In Section 1.6, we formally define the clustering problem in ad hoc networks and survey the major clustering algorithms based on graph domination along with a few algorithms based on other notions. In the last section (Section 1.7), we provide an overview of this thesis and highlight its contributions.

## 1.1 From existing peers to MANET's

Computers working together in a network have greater computational capacity than single isolated machines. As computer networks, such as the Internet, have grown in recent decades, the world has become "connected", allowing people around the world to work, study, and interact with each other in a much more efficient and economical fashion. A majority of the existing computer networks are connected using guided media, such as unshielded twisted pairs, coaxial cables, or optical fibers. Despite the fact that some media offer much higher capacity than only a few years ago, today's computer network applications are demanding even more capacity as well as mobility. This mobility requirement forces us to consider non-wired network technologies. No wires are needed if free space is used to propagate electromagnetic signals. The term "on-line" has been popular for several years, while "on-air" is emerging as the next buzzword.

### 1.1.1 Infrastructured wireless technologies

Several different technologies have been used to implement wireless networks. In each of these networks, individual users, or *subscribers*, have some physical device, or *subscriber unit*, with which they access the network. For example, a smart cell phone or a personal computer can be a subscriber unit in a specific wireless network. In addition, the network may include other physical devices, such as ground-based antennas and communications

satellites. These technologies are called *infrastructured*, because subscriber units cannot comprise a network by themselves without some supplementary devices.

The first public mobile telephone service was introduced in 1946 in twenty-five U.S. cities as a *centralized radio network*. This type of network utilizes a powerful antenna located on top of a very tall building. Signals from all senders must go through the central antenna before being relayed to the receiver. The transmission range of this antenna is at the magnitude of tens of kilometers. In fact, since the subscriber unit must also transmit to the central antenna, the transmission range of the subscriber unit also limits the range of communication using this system. Thus, for subscribers to be able to talk to each other their movements are restricted. Other flaws, besides the small coverage, were observed from centralized radio networks as follows. With all subscribers sharing the same antenna, the number of simultaneous users is limited by the total channel bandwidth available. It is power consuming to force the signal to travel a long distance, especially for portable subscriber units usually equipped with small batteries. Furthermore, the signals are vulnerable and error-prone when traveling a long distance. This is commonly referred to as *attenuation*.

In a newer technology, *personal communication system* (PCS), the geographical area covered by the network is divided into regions, called *cells*. Thus, telecommunication networks using this technology are also called *cellular networks*. Each cell contains a control basestation, with its own antennas. These basestations are usually connected with a dedicated fixed wired network, and are also connected to the Public Switched Telephone Network (PSTN). Before a call can be made between two subscribers, the basestation where the sender resides establishes a switched path to the receiver's basestation. Signals between the sender and receiver go through this wireless-wired-wireless path. After the call is completed, the path is released. Compared to the earlier centralized radio network, this technology requires less power for each unit, allows larger area coverage, and supports many more simultaneous users. Since basestations only have to control communications within their own cells, their transceiver range is not much larger than the radius of their resident cell. Therefore, non-neighboring cells can use identical sets of frequency bands without interfering with each other. This is called *spatial frequency reuse*. As the distance that signals must travel is much shorter, the subscriber units require much less power. This technology is more complex than that of a centralized radio network. For example, a sophisticated switching technique called *handoff* was proposed to enable users to continue phone calls uninterrupted when crossing cell boundaries.

The use of satellites gives rise to some additional possibilities. *Geostationary (GEO) satellites* are able to cover vast areas of the earth's surface. Although cellular networks are cost-efficient in urban/suburban areas and frequently traveled highways, it is cost-prohibitive to deploy them in sparsely populated areas where there is little demand for communications, such as in northern Canada, Siberia, and the South Pacific. At an altitude of 35,000 km above the equator, satellites are stationary relative to the earth's surface. Three equally spaced GEO satellites positioned over the equator are able to provide wireless communication capacity for most geographical areas except for those with high latitude, where the transceiving *elevation angle* is small. With GEO satellites, the round-trip communication delay is as much as 230 msec, which is too long for real-time applications such as teleoperation and video conferencing. As with the centralized radio networks, power consumption and signal attenuation are problems for the GEO satellite technology.

Yet another wireless technology, *low earth orbit (LEO) satellites* uses satellites deployed in low earth orbits at an altitude of less than 1,500 km above the earth. These satellites play the same role as basestations in cellular telephone systems, enabling ground-based subscriber units to communicate to each other. In practice, LEO offers a much larger area coverage than either land-based centralized radio networks or cellular networks. With orbits lower than GEO, LEO signals have a much smaller round-trip delay between ground units and satellites; a typical delay is 10 msec. The issue of spatial frequency reuse also arises here as in cellular networks. LEO has increased complexity due to the movement of the "basestations". Another drawback of the LEO approach is the high cost of designing, building, and maintaining it. Gvozdjac [36] presented a detailed case study on modeling information dissemination in LEO satellite networks.

*Wireless Local Area Network* (WLAN, or Wireless LAN), a more recent technology, is a flexible data communication system implemented as an extension to, or as an alternative for, a wired LAN within a building or neighborhood. In a typical WLAN configuration, a transceiver device, called an *access point*, connects to the wired network from a fixed location using standard LAN, such as Ethernet. A single access point can support a small group of wireless users within a range of tens to hundreds of meters. The access point can be located anywhere, as long as the desired radio coverage is obtained. Subscribers access the WLAN through wireless adapters, another example of subscriber units, and are allowed some movement without disconnecting from it. The IEEE 802.11 and HYPERLAN are the dominant wireless local area network specifications in North America and Europe,

respectively.

Bluetooth [37] is a short-range wireless communication solution initiated by Ericsson. In the Bluetooth specification, devices can form two types of master-slave structures: piconet and scatternet. A piconet has a star topology with a master device at the center and a set of slave devices around. Several piconets can be joined to form a scatternet. To do that, a slave device can have multiple masters and a master device can be a slave of another master.

### 1.1.2 Ad hoc networks

One goal of subscribers and of wireless networking scientists and engineers is *full mobility*. In other words, a subscriber should be able to use the network at any time and place, even if the subscriber is moving. One approach to achieve full mobility is the use of *mobile ad hoc networks*, in which only subscriber units are included. Such networks are *infrastructureless* as they do not require supplementary devices. A mobile ad hoc network is also called *MANET* or *ad hoc network*. The number of users in the network may vary from tens to tens of thousands, with all users in an area from hundreds of square meters to hundreds of square kilometers. The transmission range of a subscriber unit is significantly smaller than the network diameter, but we still wish to allow users in different parts of the network to be able to communicate. To achieve this, each subscriber unit must be responsible for relaying messages for others, that is, playing the role of a "basestation" though no actual basestations exist. Therefore, all subscriber units can talk to each other even though they may not be within mutual transmission range. In the future, ad hoc network subscriber units, such as phone handsets, personal computers, video cameras, PDA's, and other forms of embedded intelligence, will be deployed globally and a subscriber unit will never stand alone while turned on. As a first step, the deployment of ad hoc networks in smaller areas is appealing in scenarios such as disaster relief and battlefield communication, where no basestation is available or where the communication environment is hostile.

The idea of ad hoc networking has been around for over 30 years. As early as 1972, DARPA started the pioneering PRNet (Packet Radio Network) project [44]. Subsequently, various projects sponsored by the military, such as SURAN (Survivable Radio Networks), TI (Tactical Internet), and GloMo (Global Mobile Information Systems), were launched to implement the ad hoc networking paradigm [27]. This area was revitalized when many enabling technologies, such as wireless signal processing and encoding, distributed computing, VLSI circuit design and manufacturing, cryptography, positioning services, etc. have been

invented and developed.

## 1.2 Research issues in ad hoc networks

Research work on mobile ad hoc networks has mostly been focused on how to share the medium efficiently and fairly, how to relay messages between source and destination, how to control the network structure to make it scale, how to conserve battery power of the portable subscriber units, and how to prevent malicious intrusion and impersonation on multihop routes.

### 1.2.1 Medium access control

The free space as propagation media is subject to interference when multiple units send messages simultaneously. Medium access control (MAC) protocols define rules for orderly access to the shared medium and play a crucial role in efficient and fair sharing of the scarce wireless bandwidth.

The entire frequency band can be divided into independent sub-channels by some central frequency allocator so that signals in different sub-channels do not interfere with each other. This is called *frequency division multiple access* (FDMA). Similarly, time can be divided by a central scheduler into periodic time slots so that signals transmitted in different time slots are free from interfering with each other. This is called *time division multiple access* (TDMA).

However, TDMA and FDMA are not feasible for mobile ad hoc networks, where there is no such central authority. Instead, units in an ad hoc network must share the medium in asynchronously. To avoid disrupting an on-going transmission in its vicinity, a vertex that wants to send a packet must sense the medium until any on-going transmission is finished. This is complicated when not every vertex can hear everything. Suppose that vertex $B$ can communicate with vertices $A$ and $C$ but that $A$ and $C$ can not communicate with each other. If $B$ is receiving a message from $C$ when $A$ wants to send a packet, $A$ believes that no activity is going on within its vicinity because it can not detect $C$'s transmission. So $A$ may send a packet which interferes with the packet that $B$ is receiving from $C$. This is a classic example called the *hidden terminal problem* (Figure 1.1). In another example, called the *exposed terminal problem* (Figure 1.2), vertex $B$ refrains from sending packets to vertex

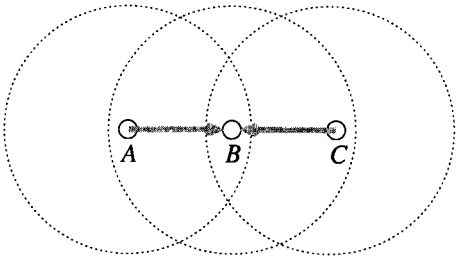



Figure 1.1: Hidden terminal problem.

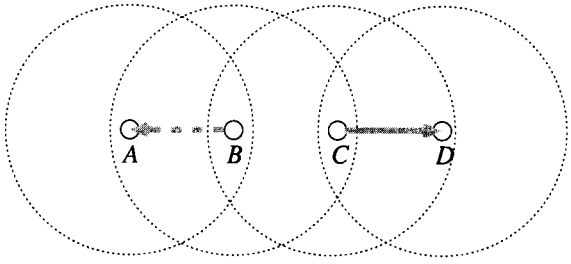



Figure 1.2: Exposed terminal problem.

$A$ even if no interference will occur at the receiver, simply because the sender $B$ hears some on-going transmission (say from $C$ to $D$) within its own vicinity.

These two examples shows that straightforward channel sensing protocols will fail in ad hoc networks. Fortunately, a plethora of medium access control protocols have been proposed to address the above problems. Among these are the MACA/MACAW [45, 17] protocols, which are the foundation of the MAC layer specification of IEEE 802.11. The MACAW protocol is essentially a four-way handshaking: RTS/CTS/DATA/ACK between the sender and receiver, where RTS is request-to-send, CTS is clear-to-send, and ACK is acknowledgment. Other MAC protocols that can be used to implement mobile ad hoc networks include the EY-NPMA protocol for HYPERLAN and the frequency hopping scheme for Bluetooth.

Research in this area includes work on improving the efficiency, fairness of bandwidth use, energy conservation, and the support of multicasting.

### 1.2.2 Routing

A network is called a *multihop* network if there is at least one pair of members that can not talk to each other directly. Generally, in a multihop network, messages travel along

several links forming a path from the source to the destination. The process of finding an appropriate path for the message to be sent along is called *routing*. The intermediate nodes on the path are responsible for deciding how to forward an incoming message and are called *routers*. The primary goal of a routing algorithm is to establish correct and efficient routes. Such an algorithm incurs costs in both time and channel bandwidth, so route construction should be done so as to minimize delay and bandwidth consumption.

A simple technique for routing is *flooding*. In this scheme, when a message arrives at a particular node for the first time, the node transmits it on all outgoing lines except the one to the node from which the message was received. With flooding, the message will quickly arrive at the destination, but the process unnecessarily consumes a large amount of channel bandwidth. Ni, Chen and Sheu [57] presented a more quantitative analysis, showing that flooding can be very costly and will result in redundancy, contention and collision.

In practice, wired networks do not utilize the flooding technique. Instead, they send messages via particular paths. Routers may exchange information about link status, calculate an overall view of the networks and construct routing tables. Note that it is not necessary to store the complete route for each pair of nodes. Instead, for each pair of nodes, it is sufficient to know the identity of the next link (or hop) on the route. Routing algorithms are capable of adapting to link failure and network congestion, by making changes to the routing tables. Routing algorithms vary in how link information is exchanged, how many routing tables are maintained, what these tables contain, and how network changes are reported.

Unlike the wired networks, in which the network structure is almost static and link failure is not frequent, ad hoc networks allow higher mobility, which permits rapid topology change. Thus, pre-calculated routing information can quickly become stale. Routing algorithms for ad hoc networks are also constrained by factors such as low bandwidth, limited power supply, and high error rates. Numerous routing algorithms for ad hoc networks have been proposed to handle these challenges. Royer and Toh [66] give a good review of current routing algorithms for ad hoc networks. The majority of the routing algorithms for ad hoc networks are classified into two categories: *table-driven* (or *proactive*) and *on-demand driven* (or *reactive*). In a table-driven routing algorithm, complete routing information is pre-calculated and ready to use at any time. Typical routing algorithms in this category include DSDV (Destination-Sequenced Distance-Vector) [60] and WRP (Wireless Routing Protocol) [55]. In an on-demand driven routing algorithm, a route is calculated as needed

although it may be partially based on some existing routing information. The representative protocols in this class are AODV (Ad-hoc On-demand Distance Vector) [62], DSR (Dynamic Source Routing) [43], and TORA (Temporally-Ordered Routing Algorithm) [58]. Other routing algorithms, such as ZRP (Zone Routing Protocol) [38], incorporate both proactivity and reactivity and are thus called *hybrid*.

### 1.2.3 Topology control

Currently, the biggest challenge in implementing mobile ad hoc networks is scalability. Given current MAC and routing protocol complexities, network performance will degrade as new units are added to the network, when units move faster, or when the density of units increases. Thus, we need to adjust the network structure to a scale in which the current MAC and routing protocols will work reasonably well. Topology control in ad hoc networks is the problem of determining an appropriate network structure or simplifying a given network structure. Readers are referred to Li [48] and Rajaraman [64] for more information on the topic of topology control.

Topology control schemes are divided into two categories. Schemes in the first category are based on controlling the number of wireless links in an ad hoc network by adjusting the power levels of the subscriber units. The quality of the output of a scheme can be evaluated according to criteria such as connectivity (the number of hops in routing paths) and capacity (the transportation capability of the network). In the other category, the network structure is simplified by clustering — the process of defining substructures of the network. Units that are close to each other form a cluster, and connections between units of different clusters form connections between clusters. The assumption of clustering is that units within a locus in a realistic ad hoc network are usually similar in nature and are stationary relative to each other. Clustering can be applied iteratively to a network topology to obtain a hierarchically clustered structure. More on clustering will be discussed in Section 1.6.

### 1.2.4 Others

The subscriber units are usually portable and powered by small batteries. To conserve battery power for these devices is a central issue. Simply reducing the activities of the subscriber units is not advisable. First, if nodes are sluggish in sending and receiving control messages, the ability of a network to respond to network changes is compromised,

which in turn may cause many other problems. Second, an ad hoc network is a cooperative network whose units must relay data for other units. A unit should not conserve its own power while degrading the performance of the entire network. Therefore, there is a subtle trade-off between power use and network performance.

Some research activities in ad hoc networking focus on building sensor networks. Micro-sensors are small modules with sensing, computing, and communication capabilities. These sensors can be deployed within a geographic area to provide continuous monitoring of environmental or weather information. Micro-sensors are generally not mobile and with a relatively small amount of battery power compared to other types of ad hoc networking devices. Research in the field of sensor networking focuses on device fabrication, energy conservation, information collection and dissemination, and data protection.

As in other wireless networks, security is an important issue due to the open nature of the transmission medium. Ad hoc networks must also confront potential threats from malicious impersonation. A message source has to trust intermediate units to relay its message to the destination, but the relayed message can be tampered by malicious units. What's more, routing information propagated through the network can be modified so that data packets are redirected to false destinations.

Other interesting research issues in ad hoc networking include using directional antennas to improve network capacity, providing quality-of-service guarantees, and incorporating location information in applications.

## 1.3 Motivation

The Internet, a large-scale multi-hop wireline communication network, has been a huge commercial success in the past decade. In contrast, despite the success of many enabling technologies related to ad hoc networking, one cannot help asking why there are no cost-effective off-the-shelf commercial ad hoc networking systems. Among the many challenges for ad hoc network designers and users, scalability is a critical issue. In particular, when a flat-topology network contains a large number of nodes, control overhead, such as routing packets, requires a large percentage of the limited wireless bandwidth.

Perkins [63] observed that "aggregating routing information is the key to Internet scalability". In particular, a node's IP address contains hierarchical information related to its location that can be used in routing. Due to the mobility of nodes in an ad hoc network,

this is not as simple to accomplish.

In a multihop packet-switched network, intermediate nodes are required to route packets between the source and destination if they (the source and the destination) are not directly connected. For example, in a distance-vector routing protocol, each node participating in the route calculation stores a routing table and shares it with all neighboring nodes. If the network has a flat topology (that is, all nodes are treated equally), the size of the routing table is proportional to the number of nodes in the entire network. Further, as network size increases, communication costs tend to consume a larger proportion of the bandwidth. Furthermore, as the rate of the network topology change increases, the exchange of routing tables between neighboring nodes must be more frequent to keep the routing information up to date. Other network parameters, such as network node density and traffic load, can also impair network scalability. Arpacioglu, Small, and Haas [8] have begun a study of the scalability issue of multihop networks and, in particular, ad hoc networks.

The Internet, a multihop packet-switched communication network, manages to function with approximately $10^9$ nodes. Each node in the Internet is given a 32-bit IP address that is assigned in a way such that all the nodes in the same subnet share the same address prefix. This very important property allows us to build a hierarchy in the Internet topology. Routing nodes do not need to store the IP addresses of all the nodes in the network; address prefixes are sufficient to direct packets to the proper subnets for further local routing.

Unfortunately, due to mobility, nodes in an ad hoc network can not be assigned such aggregate addresses. This is an obstacle for scaling up ad hoc networks.

However, we believe that many substructures in a large-scale, even global, ad hoc network are relatively stable. Users can indeed be mobile, but their movements are usually confined within a specific geographical area. For example, students may wander around a region during the day and commute within a metropolitan area on a daily basis. These movements cause local topology changes but do not drastically alter the overall structure of the network. Since many of these changes are confined to a relatively small region, one can abstract the network to obtain a simpler topology and avoid the need to inform the entire network of these topology changes. Local portions of the network are represented by super-vertices in the abstracted topology and connections between them are super-edges. Clustering is a process of defining such an abstracted structure of a network and can be applied recursively to obtain a multi-level hierarchy. We will give a more formal definition of clustering in Section 1.6.

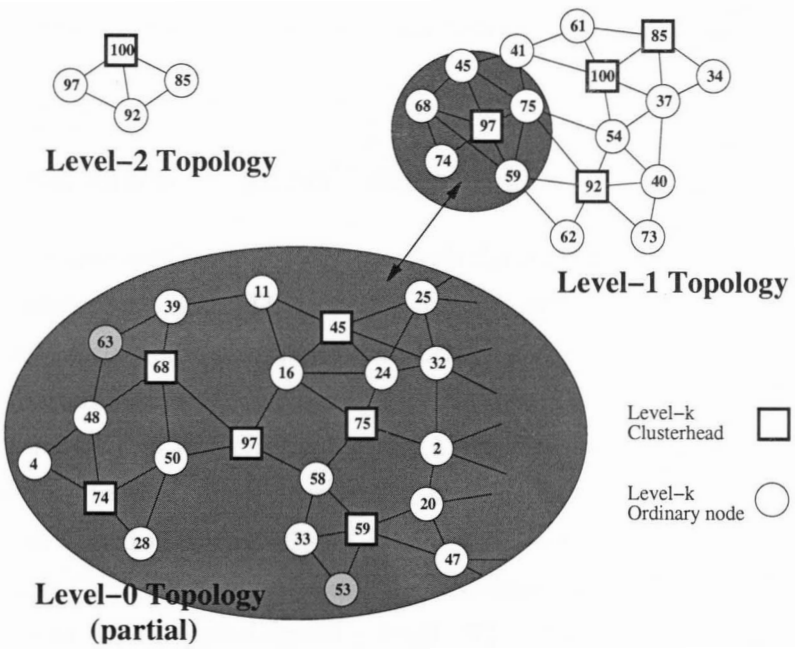



Figure 1.3: Hierarchical routing [70].

After clustering, each node in the hierarchy can be assigned a hierarchical address that indicates its position in each level of the hierarchy. Routing can easily be carried out using such addresses. We use an example from Sucec and Marsic [70] to explain this. Figure 1.3 depicts an $n$-node network with three hierarchy levels created by recursive clustering. We use the terms Level-0 to refer to the original network, Level-1 to refer to the structure obtained by clustering once, and Level-2 to refer to the structure obtained by clustering Level-1. Each node in the network can be assigned a 3-level hierarchical address. For example, in the figure, node 63 is a member of the level-1 cluster represented by node 68. Node 68, in turn, is a member of the level-2 cluster represented by node 97. Thus, node 63's hierarchical address is $(97_2, 68_1, 63_0)$, where the subscripts indicate address levels. In such a hierarchy, a node only needs to store a $3 \times c$ matrix to route packets, where $c$ is the number of sub-clusters within a cluster of the next higher level. Suppose source node 53 $(97_2, 59_1, 53_0)$ wants to send a packet to destination node 63 $(97_2, 68_1, 63_0)$. The packet is first routed to a node in node 63's level-1 cluster, say node 68 $(97_2, 68_1, 68_0)$, and then routed to node 53 within the cluster. In general, if $c$ is a constant order for each level, then the hierarchy level $L = O(\log n)$. Therefore, each node only needs to store a routing table of size $O(\log n)$ rather than of size $O(n)$. With this exponential savings from clustering, it is

possible for an ad hoc network to scale. (See Steenstrup [68] for more details on hierarchical routing.)

## 1.4 Preliminaries — graph domination

We will use graph domination, a subject in graph theory, as a major tool to attack the ad hoc network clustering problem. Thus, in this section, we introduce some notions related to graph theory and graph domination, and then briefly study the complexity and algorithmic aspects of graph domination.

We use the notation $G = (V, E)$ to denote a *graph* with vertex set $V$ and edge set $E$. Sometimes a vertex is also called a node and an edge is also called a link. We use $n$ to denote the size of $V$ and $m$ for the size of $E$ unless otherwise specified.

Given a graph $G = (V, E)$, the *closed neighborhood* $N[v]$ *of a vertex* $v$ in $G$ consists of the vertices adjacent to $v$ plus vertex $v$ itself. The *closed neighborhood* $N[S]$ *of the set* $S \subseteq V$ is the union $\bigcup_{v \in S} N[v]$. The *closed distance-k neighborhood*, $N_k[v]$, of a vertex $v \in V$ is the set of vertices that are within distance-$k$ of $v$. The *open neighborhood* $N(v)$ *of a vertex* $v$ in graph $G$ consists of the vertices adjacent to $v$ is $N(v) = N[v] \setminus v$ with other open neighborhood terms defined and denoted analogously. A vertex subset $S \subseteq V$ is called an *independent set* if no vertices in $S$ are adjacent. Given a vertex subset $S \subseteq V$, the subgraph *induced* by $S$, denoted $\langle S \rangle$, is $(S, E \cap (S \times S))$. Intuitively, the graph $\langle S \rangle$ has $S$ as the vertex set and includes all edges of $E$ that have both endpoints in $S$. Readers are referred to West [72] for other graph theoretic concepts and results.

A *dominating set* of graph $G$ is a subset $S \subseteq V$, such that every vertex $v \in V$ is either in $S$ or adjacent to an element in $S$. In other words, $\bigcup_{v \in S} N[v] = V$. If vertex $v$ is in a dominating set $S$ and vertex $u$ is a neighbor of $v$, then we say $v$ *dominates* $u$ and $u$ is *dominated* by $v$. An edge is said to be *dominated* if either of its endpoints is in the dominating set; otherwise it is *free*. The black vertices in the graph in Figure 1.4 form a dominating set of the graph.

Now we are ready to introduce some important variations of graph domination. An *independent dominating set* is a dominating set that is also an independent set. A *connected dominating set*, or CDS for short, of graph $G$ is a dominating set of $G$ whose induced subgraph is connected. In the graph in Figure 1.5, the black vertices form a connected dominating set and the subgraph induced by these vertices is indicated by black edges.

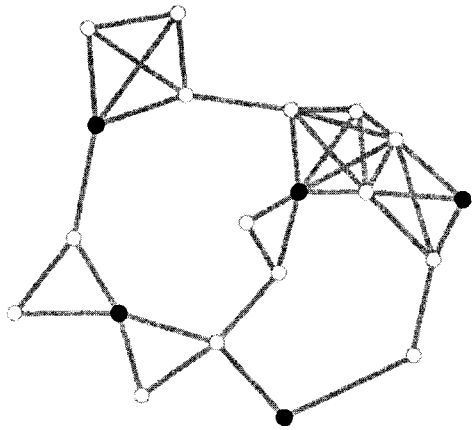


Figure 1.4: Dominating set.

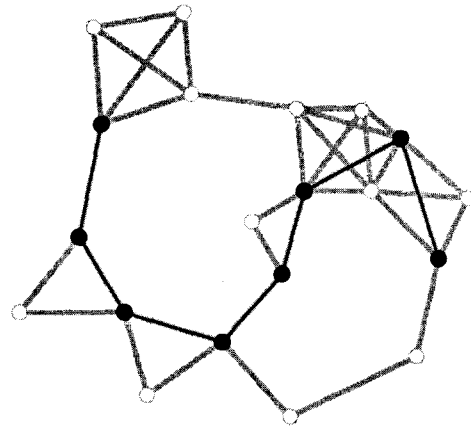


Figure 1.5: Connected dominating set.

The subgraph *weakly induced* by $S$ ($S \subseteq V$) is the graph $\langle S \rangle_w = (N[S], E \cap (N[S] \times S))$. Intuitively, $\langle S \rangle_w$ includes all the vertices in $S$ and their neighbors as the vertex set, and contains all edges which have at least one endpoint in $S$. Figure 1.6 illustrates a subgraph weakly induced by the solid black vertices and the subgraph is indicated by black edges. A set $S$ is a *weakly-connected dominating set*, or WCDS for short, of graph $G$ if $S$ is dominating and $\langle S \rangle_w$ is connected. The solid black vertices in Figure 1.7 are an example of a weakly-connected dominating set.

Haynes, Hedetniemi and Slater [40, 39] is a very good survey in the area of domination in graphs covering the graph theoretical, computational complexity and algorithmic aspects of this topic.

We can use the vertices in a dominating set as representatives to simplify the structure of an ad hoc network. The variations above provide us different levels of connectivity and independence. To simplify the network structure as much as possible, a minimum sized dominating set (or dominating set variant) is desired. The *domination number* $\gamma(G)$ is defined to be $\min_{S \subseteq V}\{|S| \mid S \text{ is a dominating set of } G\}$. A *minimum dominating set* of $G$ is a dominating set whose cardinality is $\gamma(G)$. The connected domination number $\gamma_c(G)$, weakly-connected domination number $\gamma_w(G)$, and independent domination number $\gamma_i(G)$ are defined analogously. When the graph $G$ is understood from context, it can also be omitted.

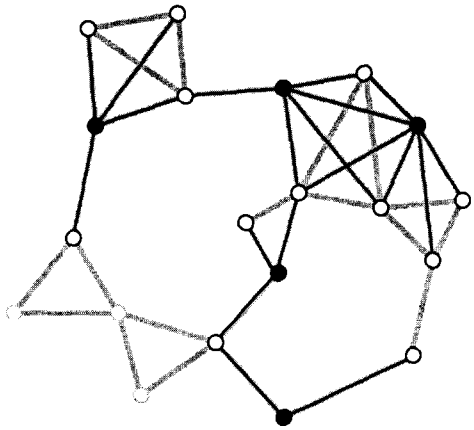


Figure 1.6: Weakly-induced subgraph.

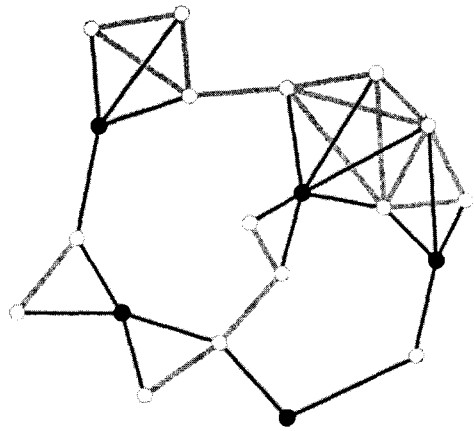


Figure 1.7: Weakly-connected dominating set.

Unfortunately, determining any of the values $\gamma(G)$, $\gamma_c(G)$, $\gamma_w(G)$, or $\gamma_i(G)$, or finding a minimum dominating set (or minimum connected dominating set, minimum weakly-connected dominating set, minimum independent dominating set) is NP-complete [29]. Therefore, researchers have proposed approximation algorithms to provide practical results. For example, Guha and Khuller [35] propose two approximation algorithms for finding small connected dominating sets. What is more, due to the close similarity between the minimum dominating set problem and the set cover problem, it is quite likely that no algorithms with approximation ratio asymptotically better than $O(\log n)$ can be found for any of the above variants of the dominating set problem [26, 51]. More algorithms for constructing small dominating sets (and variants) will be surveyed in Section 1.6.

## 1.5 Modeling Ad Hoc Networks

Modeling mobile ad hoc networks must deal with a challenging trade-off between complexity and fidelity. The traditional multi-port model used in distributed computing is simple enough but may lack precision in imitating ad hoc networks; while models that incorporate MAC issues can be too complicated to manipulate mathematically. By comparing the multi-port model with two newer models, we believe that it is justified to use the traditional distributed computing model. An additional benefit from using a classic model is the possibility of using existing fundamental algorithms in distributed computing.

The distributed computing community uses a classic *multi-port network model* as the basis for designing and analyzing distributed algorithms for various problems. In that model, a process (essentially a vertex of the graph) can send/receive a message to/from each of its neighbors in a single *round* (time unit) in a synchronous network. An asynchronous version of the model does not require that the message transmission and reception be contained in fixed time slots. This is not the most realistic model for ad hoc networks because interference between links, which is not as serious a problem in wired networks, is not taken into consideration. Although not a perfect model for ad hoc networks, this model is generally regarded as reasonable and has been adopted by a large number of researchers in the area.

Bar-Yehuda, Goldreich, and Atai [13] developed a model for ad hoc networks that incorporates interference, and this model is further utilized by Kowalski and Pelc [46]. This model uses undirected graphs with the following scheduling constraints to model ad hoc networks:

- An edge $e$ between two vertices indicates that the transmitter of each vertex can reach the receiver of the other;
- Vertices send messages in synchronous steps;
- In every step, each vertex may act either as a transmitter or a receiver, but not both;
- A transmitter sends a message that can potentially reach all of its neighbors;
- In a given step, a receiver gets a message if and only if exactly one of its neighbors transmits during the step. (If two or more neighbors transmit at the same time, a collision occurs.)

The synchronous scheduling scheme used in this model is shown to be very complicated. Furthermore, this model only allows a single protocol to be executed on an ad hoc network at once.

Researchers in probabilistic and randomized algorithms use their results in inter-connected networks and other wired networks for routing and message queuing in ad hoc networks. Because medium access control (MAC) is well-studied for such networks, these papers generally use randomized algorithms to solve the routing and queuing (which they call "scheduling") problems optimally or approximately. The work of Adler and Scheideler [2] is an example. After integrating MAC, the problem is shown to be much harder. Adler and Scheideler

show that to schedule and optimally route a permutation (a one-to-one mapping between sources and destinations) in their general ad hoc networking model is NP-complete. It is also NP-complete to approximate it within a ratio of $O(n^{1-\epsilon})$ for any $\epsilon > 0$. To solve the problem in a less general model, they divide the whole problem into MAC, routing, and scheduling problems. Using a special, yet sensible, MAC protocol (called LPC, local probabilistic communication), the routing/scheduling problem is much better solved. Intuitively, the LPC protocol suppresses the probability that each node attempts to access the channel so that whenever it does so, it succeeds in sending/receiving a packet with a probability of at least 1/2 (despite the interference from other nodes). Thus they can adapt their results from wired networks to ad hoc networks. However, the problem with the LPC model is that the probability that a node attempts to access the channel can be as low as $\Theta(1/n)$.

The multi-port model may seem too optimistic and unrealistic to use. However, it may be feasible when used for analyzing the time used by a particular protocol in a network that runs multiple protocols at the same time. That is, it takes about the same time for a vertex $v$ to send a message to one neighbor compared to the time for $v$ to send a different message to all of the neighbors. This is true because the MAC layer of a vertex schedules all the packets from the upper layer independently and the small increase in traffic caused by this particular protocol can be ignored compared to the significantly large amount of total traffic. Thus, we will adopt the classic (possibly asynchronous) multi-port model in this thesis.

## 1.6 Related work — clustering ad hoc networks

Here, we define the *clustering problem* as follows. We are given an undirected graph $G = (V, E)$ representing a communication network where the vertices are the nodes in the network and the edges are the communication links. The clustering process first divides $V$ into a collection of (not necessarily disjoint) subsets $\{V_1, V_2, \ldots, V_k\}$, where $V = \bigcup_{i=1}^{k} V_i$, such that each subset $V_i$ induces a connected subgraph of $G$. Note that these induced subgraphs can overlap. Each such vertex subset is a cluster. Ideally, the size of the clusters falls in a desired range and the induced subgraphs have small diameters. An abstracted graph $G' = (V', E')$ is constructed, where each vertex $v_i' \in V'$ corresponds to a subset $V_i$. There is an edge from $v_i'$ to $v_j'$ if and only if there is an edge of $E$ from some vertex $u_i \in V_i$ to some vertex $u_j \in V_j$. One vertex in each cluster is elected to represent the cluster. This vertex is commonly called

the *cluster-head* or *cluster-leader*. The abstracted network $G'$ can also be clustered leading to a multi-level hierarchy.

A natural way to cluster an ad hoc network is to use the notion of graph domination or one of its variants. The members of a dominating set are chosen as cluster-heads and the neighborhood of each cluster-head comprises a cluster. In this section, we focus on clustering algorithms based on graph domination. We briefly mention other notions used to define clusters.

### 1.6.1 Clustering with independent dominating sets

One can produce a relatively small number of clusters of a given graph by insisting that the dominating set is also an independent set.

Baker and Ephremides [10] devised one of the earliest clustering algorithms for ad hoc networks, the linked cluster algorithm. This algorithm is executed in a synchronous ad hoc network, where each node has a dedicated TDMA time slot to avoid collisions. It takes $|V|$ time slots for a node to learn the structure of its neighborhood. A vertex $v$ is chosen as a cluster-head by a neighbor $u$ if $v$ has the highest vertex ID within $N(u)$. The chosen vertices form an independent dominating set of $G$.

Gerla and Tsai [32] proposed two clustering algorithms based on vertex ID and vertex degree. In the lowest-ID algorithm, each vertex with the lowest ID within its closed neighborhood is selected as a cluster-head. In the highest degree algorithm, each vertex with the highest degree in its closed neighborhood is selected. The cluster-heads chosen by these two algorithms form an independent set. However, as noted by Chen, et al. [20], these algorithms do not work on all graphs. In particular, for some graphs the cluster-heads do not form a dominating set and, thus, not every vertex has a cluster-head. Lin and Gerla [50] corrected this flaw and proposed a modified lowest-ID algorithm that constructs independent dominating sets. Extending this result, Chen, et al. [20] presented an algorithm for constructing distance-$k$ dominating sets with the additional property that the members of the dominating set are at distance at least $k+1$ from each other. This algorithm selects vertices based on the highest-degree within distance-$k$ neighborhoods with the lowest-ID used to break ties. Basagni generalized this result to show that any meaningful measure can be used to determine cluster-heads [14].

For some other results on clustering with independent dominating sets, see the papers of An and Papavassiliou [7], Hou and Tsai [41], and Gerla, Kwon, and Pei [31].

### 1.6.2 Clustering with dominating sets

The use of independent dominating sets as cluster-heads is problematic when the network topology changes. In particular, when two cluster-heads move within transmission range of each other, one of them must defer to the other which can trigger cluster-head changes that may propagate throughout the network. Such an effect is called *chain reaction* [31]. By relaxing the independence condition on dominating sets, this chain reaction effect does not occur. Thus, it may be of interest to simply consider dominating sets.

Liang and Haas [49] presented a distributed greedy algorithm for dominating sets that mimics a centralized greedy algorithm. In the centralized algorithm, a dominating set is constructed by adding, in each iteration, the vertex with the largest number of free neighbors. This yields a dominating set with approximation ratio $O(\log \Delta)$, where $\Delta$ is the maximum vertex degree. The authors showed that this algorithm can be distributed so that, in each iteration, vertices only need to know about the structure of their distance-2 neighborhood. Consequently, both algorithms have the same logarithmic approximation ratio. Jia, Rajaraman and Suel [42] devised a randomized version of this algorithm that terminates in $O(\log |V| \log \Delta)$ rounds with high probability. The approximation ratio is expected to be $O(\log \Delta)$ and is $O(\log |V|)$ with high probability.

For some other results on clustering with dominating sets, see the papers of Amis, Prakash, Vuong, and Huynh [6], Belding-Royer [15], and Sivakumar, Sinha, and Bharghavan [67].

### 1.6.3 Clustering with connected dominating sets

Some researchers argue that better connectivity among the cluster-heads is an advantage for applications such as message broadcasting. The vertices of a connected dominating set induce a connected subgraph that can be used as a *virtual backbone* so that broadcast redundancy is reduced significantly [69].

As the minimum connected dominating set decision problem is NP-complete in general graphs, Guha and Khuller [34, 35] proposed two centralized greedy algorithms for finding suboptimal connected dominating sets in arbitrary connected graphs. In one algorithm, vertices are added to a connected set so as to maximize the number of newly dominated vertices. In the other, the connectivity of the subgraph induced by adding each candidate to the current set is also considered. Both algorithms have an approximation ratio of

$O(\log \Delta)$. Due to the close similarity between the connected dominating set problem and the set cover problem, it is unlikely that an approximation algorithm with performance ratio asymptotically better than $O(\log \Delta)$ can be found for the connected dominating set problem [26].

Das and Bharghavan [24] provided distributed implementations of the algorithms of Guha and Khuller [34, 35] for constructing connected dominating sets in ad hoc networks. These distributed algorithms generate the same connected dominating sets as their centralized counterparts and, thus, have exactly the same approximation ratio since they utilize central coordinators to oversee the entire execution.

To address the issue of non-localized computation in the distributed algorithms of Das and Bharghavan, Wu and Li [73, 74] presented a localized distributed algorithm for finding small connected dominating sets in which each node only needs to know its distance two neighborhood. The algorithm consists of two marking phases. Initially, each vertex is marked $F$ to indicate that it is not in the connected dominating set. In phase one, a vertex marks itself $T$ if any two of its neighbors are not directly connected. This process marks all vertices that can be potentially included in a connected dominating set. In phase two, a $T$ vertex $v$ changes its mark to $F$ if either of the following conditions is met:

1. $\exists u \in N(v)$ which is marked $T$ such that $N[v] \subseteq N[u]$ and $id(v) < id(u)$;
2. $\exists u, w \in N(v)$ which are both marked $T$ with $N[v] \subseteq N[u] \cup N[w]$ and $id(v) < \min\{id(u), id(w)\}$.

The left and right examples shown in Figure 1.8 illustrate conditions 1 and 2, respectively. The vertices colored black are $T$ vertices and those colored white are $F$ vertices. In both cases, the vertex $v$ will change its mark to $F$ provided the identity condition holds. This algorithm constructs a connected dominating set in a localized fashion. However, there is no known non-trivial upper bound on the size of the connected dominating set generated.

In a more recent paper, Dubhashi, et al. [25] presented a distributed algorithm for constructing small connected dominating sets and weakly-connected dominating sets with an $O(\log \Delta)$ approximation ratio. The connected dominating set algorithm first constructs a dominating set and then adds extra vertices in an economical way such that the resultant dominating set is connected and has a provable performance ratio. Specifically, the algorithm

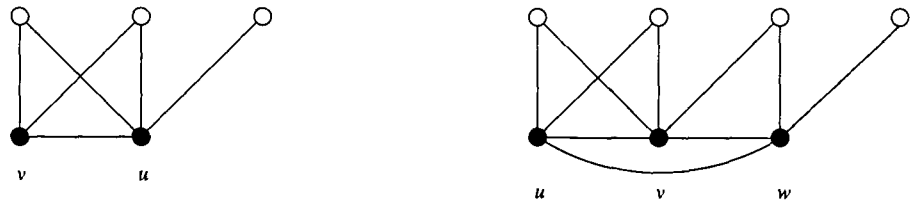



Figure 1.8: Unmarking conditions.

utilizes the randomized algorithm of Jia, Rajaraman and Suel [42] to construct a dominating set that is expected to be within a $O(\log \Delta)$ factor of the minimum size. The algorithm utilizes the observation that a simple undirected graph $G$ on $n$ vertices has at most $n^{1+\frac{2}{g-1}}+n$ edges (Lemma 15.3.2 in Matousek [54]), where $g$ is the girth (length of the shortest cycle) of $G$. Given a dominating set $S$ of $G$, an auxiliary graph $G'$ is constructed on $S$. By removing cycles of length less than $\lfloor 1 + 2\log|S| \rfloor$ in $G'$, $G'$ has no more than $2|S|$ edges left. An edge $e = (u, v)$ of $G'$ corresponds to a set of paths $\{P_i(u, v)\}$ in $G$ of length at most three. For each pair of such vertices $u$ and $v$, additional vertices are added to $S$ only if $u$ and $v$ are neither adjacent nor joined by a path comprised of only dominated vertices in $G$. At most two additional vertices on a path are added to $S$. As this is done for each edge in $G'$, the total number of vertices added to $S$ is at most $4|S|$. Therefore, $S$ becomes a connected dominating set of expected size at most $O(\log \Delta)$ times the minimum.

In an obstacle-free two-dimensional space where all vertices have the same transmission range, ad hoc networks can be modeled using unit disk graphs (UDG's) [23]. UDG's are the intersection graphs of equal sized circles in the plane, that is, there is an edge between two vertices if their corresponding circles intersect.

Alzoubi, Wan, and Frieder [3, 4] proposed a localized algorithm for finding small connected dominating sets in UDG's. Initially, a maximal independent set of the given UDG $G$ is chosen. This set is also a minimal dominating set of $G$. Other vertices are added to guarantee that the set is connected. The algorithm takes advantages of some particular geometric properties of UDG's that guarantees that the size of the chosen connected dominating set is within a constant factor of the minimum.

### 1.6.4 Clustering with weakly-connected dominating sets

Chen and Liestman [21] introduced the use of weakly-connected dominating sets for clustering ad hoc networks. This inherently sparser structure models the scatternet configuration

of Bluetooth [37]. In the Bluetooth specification, devices can form two types of master-slave structure: piconet and scatternet. A piconet has a star topology with a single master device at the center and a set of slave devices around. Several piconets can be joined to form a scatternet. A weakly-connected dominating set of a graph faithfully captures the scatternet topology with the vertices in the dominating set being the master devices.

A series of algorithms was presented by Chen and Liestman [21] with an approximation ratio of $\ln \Delta + O(1)$ based on the algorithms of Guha and Khuller [34, 35] for connected dominating sets. These greedy algorithms construct weakly-connected dominating sets incrementally by adding a vertex to the current set in each iteration. As in the connected dominating set case, a $\ln \Delta + O(1)$ approximation ratio upper bound can be proved for this algorithm and it is asymptotically optimal.

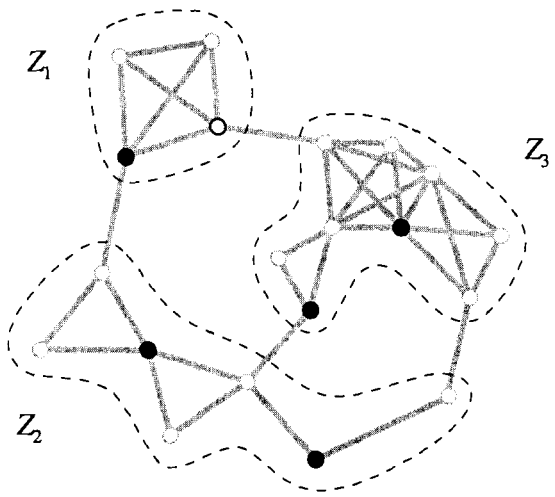



Figure 1.9: Zonal clustering scheme.

In order to decentralize these algorithms, Chen and Liestman [22] proposed a *zonal* version of these algorithms. In the zonal clustering algorithm, given a *zone size control parameter* $x$, each zone is a connected subgraph of the input network with no more than $2x$ vertices. A zone has a dedicated vertex known by all zone members as *root*. The zonal construction algorithm has two levels: intrazonal and interzonal. In the intrazonal level, a weakly-connected dominating set is independently constructed for each zone. In the interzonal level, the root of a zone adds additional vertices to its weakly-connected dominating set to guarantee that the union of the dominating sets for the individual zones is a weakly-connected dominating set for the entire network. As an illustration, the network

in Figure 1.9 is partitioned into three zones. The solid black vertices are the dominators of each zone. The hollow black vertex in zone $Z_1$ is added to guarantee the weak connectivity of the dominating set. The advantage of the zonal approach is that the zone size control parameter $x$ can be used to control the zone granularity, providing a trade-off between the extent of network structure simplification and the locality of algorithm execution.

An algorithm of Dubhashi, et al. [25] similar to the one presented in Section 1.6.3 constructs small weakly-connected dominating sets with an $O(\log \Delta)$ approximation ratio.

Alzoubi, Wan and Frieder [5] proposed an algorithm related to their algorithm for connected dominating sets in unit disk graphs (UDG's) [4] that generates weakly-connected dominating sets of size within a constant factor of the minimum for UDG's (see Section 1.6.3).

### 1.6.5 Clustering by methods other than graph domination

Other algorithms for clustering ad hoc networks have been proposed that are not based on graph domination.

Krishna, Vaidya, Chatterjee, and Pradham [47] presented a clustering algorithm where clusters are formed without cluster-heads. A *clique* in graph $G = (V, E)$ is a subset $S$ of $V$, whose induced subgraph is complete. The authors use maximal cliques as clusters as illustrated in the example of Figure 1.10. A node is called a *boundary node* if it belongs to more than one cluster. Nodes in the same cluster can communicate directly with each other, while nodes in different clusters must rely on boundary nodes to relay messages. This is similar to the concept of the Internet BGP routing.

Banerjee and Khuller [11] proposed a protocol based on a *spanning tree.* In their scheme, a cluster is a subset of vertices whose induced graph is connected. These subsets are chosen with consideration to cluster size and the maximum number of clusters to which a node can belong. The idea is to group branches of a spanning tree into clusters of an approximate target size. The resulting clusters can overlap and nodes in the same cluster may not be directly connected. Figure 1.11 shows an example of clusters obtained by this method. In this figure, the spanning tree is shown in black.

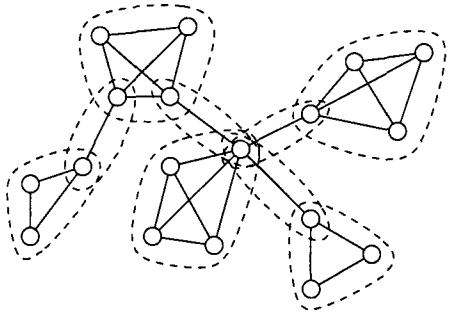


Figure 1.10: Clique-based clustering.

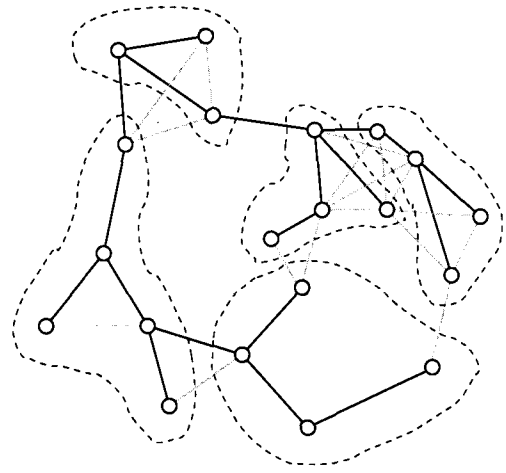


Figure 1.11: Spanning tree-based clustering.

## 1.7 About this thesis

In this section, we provide an outline of the thesis structure and summarize its major contributions.

### 1.7.1 Thesis overview

#### Chapter One — Introduction

An ad hoc network is usually represented by an undirected graph $G = (V, E)$. (We consider the problem in directed graphs in Chapter Five.) Clustering ad hoc networks partitions a given network into smaller connected substructures. Each partition or its representative (*clusterhead*) can be viewed as a vertex of an abstracted graph from a higher level and connections between partitions can be viewed as edges in the abstracted graph.

Graph domination provides a method of selecting clusterheads. A dominating set $S$ of a given graph is a vertex subset such that all other vertices are adjacent to some vertex in $S$. Neighborhoods of such vertices can be regarded as clusters. Connectivity among $S$ is important for technical reasons. Therefore, we are interested in a particular variant of graph domination, weakly-connected dominating sets. Here, in addition to being a dominating set, the edges incident to a vertex in $S$ are required to form a connected subgraph of $G$.

The decision version of the minimum weakly-connected dominating set problem is NP-complete even for bipartite or chordal graphs. Although it allows logarithmic approximation algorithms, it is conjectured that no asymptotically better performance ratios are possible. In this thesis, we focus on approximation algorithms and heuristics for constructing and maintaining small weakly-connected dominating sets.

### Chapter Two — Approximation Algorithms

We propose two centralized greedy algorithms to approximate the minimum weakly-connected dominating set problem. These algorithm are based on those for connected dominating sets. The approximation ratio of these algorithms is $\ln \Delta + O(1)$, where $\Delta$ is the maximum vertex degree of the graph. Two distributed versions of these algorithms are also presented. By using a special vertex in the network to coordinate the distributed algorithms, these distributed algorithms behave exactly as their centralized counterparts and, thus, inherit the same approximation ratio. To enhance the parallelism of these distributed implementations, we dispense with the central coordinator and propose algorithms that construct weakly-connected dominating sets from multiple starting vertices.

We are not able to show interesting bounds on the approximation ratios of these latter algorithms. Therefore, we resort to simulations and show that the size of the weakly-connected dominating set generated by the decentralized algorithms is almost the same as that generated by the centralized algorithms. We also compare the size of these weakly-connected dominating sets to the size of connected dominating sets generated by a greedy algorithm. We also consider other metrics, such as the changes in pairwise vertex distances and the number of edge-disjoint paths when comparing $\langle S \rangle_w$ to $G$.

### Chapter Three — Zonal Computation

In order to further decentralize the distributed algorithms in Chapter Two, we propose *zonal* versions of these algorithms. In a zonal clustering algorithm, given a *zone size control parameter* $x$, each zone is a connected subgraph of the input network with no more than $2x$ vertices. A zone has a dedicated vertex known by all zone members as *root*. The zonal construction algorithm has two levels: intrazonal and interzonal. In the intrazonal level, a weakly-connected dominating set is independently constructed for each zone. In the interzonal level, the root of a zone adds additional vertices to its weakly-connected

dominating set to guarantee that the union of the dominating sets for the individual zones is a weakly-connected dominating set for the entire network. The advantage of the zonal approach is that the zone size control parameter $x$ can be used to control the zone granularity, providing a trade-off between the extent of network structure simplification and the locality of algorithm execution.

In the experiments, we compare the same set of metrics (dominating set size and change in vertex distance and in the number of edge-disjoint paths) for different values of the zonal size control parameter and see how they change with different zone granularities.

### Chapter Four — Maintenance

The zonal algorithm for constructing weakly-connected dominating sets in Chapter Three works well for static ad hoc networks, such as sensor networks. In this chapter, we present an algorithm that updates the weakly-connected dominating set when the topology changes. The process of cluster maintenance is logically divided into two layers — intrazonal and interzonal. We first present a non-zonal update algorithm that can be used to maintain the weakly-connected dominating set structure resulting from the static algorithms in Chapter Two. Then we describe a zonal maintenance algorithm utilizing the above algorithm. In particular, we restrict the execution of the earlier algorithm to zones to accomplish intrazonal maintenance and add interzonal maintenance procedures to maintain the weakly-connected dominating set of the entire network.

We present experimental settings and results to test the performance and efficiency of the maintenance algorithm. The results show that both the size of the weakly-connected dominating set and the connectivity of the abstracted network stabilize after a short simulation time and these values remain approximately the same as initially.

### Chapter Five — Clustering in the Presence of Unidirectional Links

In some ad hoc networks, due to asymmetric device capabilities and interference conditions at different vertices in the network, wireless links can be simplex (or *unidirectional*). We use a digraph $G = (V, A)$ to represent such networks. In this chapter, we extend our graph-domination based clustering algorithms to digraphs.

First, we generalize the weakly-connected dominating set notion to digraphs. Here, the weakly-connected dominating set must dominate all other vertices of the graph in both (in

and out) directions and the arcs incident on vertices in the set must form a strong subgraph of $G$. We use the term *gemini set* to denote this generalization. We propose centralized approximation algorithms for constructing small gemini sets. Second, we present a set of distributed algorithms for constructing small gemini sets of digraphs.

In simulations, we compare the size of the gemini set generated by the centralized approximation algorithm and that generated by the distributed algorithm, respectively. We also consider the connectivity change.

**Chapter Six — Future Work**

We discuss some problems that are left open by this work and some more strategic problems in the ad hoc networking research.

### 1.7.2 Contributions

The primary contributions of the thesis are as follows.

- We introduce the use of weakly-connected dominating sets to cluster ad hoc networks. This particular variant of graph domination has a lower connectivity requirement than connected dominating sets. Furthermore, it captures the scatternet configuration of Bluetooth faithfully.

- Our zonal approach divides the clustering process into two levels: interzonal and intrazonal. In the intrazonal level, a weakly-connected dominating set is independently constructed for each zone. In the interzonal level, the root of each zone adds additional vertices to its weakly-connected dominating set as needed to guarantee that the union of the dominating sets for the individual zones is a weakly-connected dominating set for the entire network. The advantage of the zonal approach is that the zone size control parameter $x$ can be used to control the zone granularity, providing a trade-off between the extent of network structure simplification and the locality of algorithm execution. In general, the zonal scheme can be used in solving other distributed computing problems, where result quality and computation locality need to be balanced.

- We define gemini sets to generalize the notion of weakly-connected dominating sets to directed graphs. We provide approximation algorithms to find small gemini sets.

# Chapter 2

# Approximation Algorithms

We present a series of approximation algorithms for finding a small weakly-connected dominating set (WCDS) in a given graph to be used in clustering mobile ad hoc networks. The structure of a graph can be simplified using weakly-connected dominating sets and made more succinct for routing in ad hoc networks. The $GW_M$ and $GW_S$ algorithms are based on the centralized greedy approximation algorithms of Guha and Khuller [35] for finding small connected dominating sets (CDS's). The theoretical performance ratio of these algorithms is $\ln \Delta + O(1)$ compared to the minimum size weakly-connected dominating set, where $\Delta$ is the maximum degree of the input graph. The $DGW_M$ and $DGW_S$ algorithms are the distributed implementations of the $GW_M$ and $GW_S$ algorithms, respectively, which have identical performance ratio as their centralized counterparts. The $DW_{SYNC}$ and $DW_{ASYNC}$ algorithms are distributed heuristics which have been shown to generate weakly-connected dominating sets of sizes similar to those generated by the $GW_M$ algorithm. In the analysis of the synchronous distributed algorithms, i.e. $DGW_M$, $DGW_S$, and $DW_{SYNC}$, the classic multi-port distributed computing model [52] is used (See Chapter 1 for more details on different model options). Comparisons between our work and some previous work (CDS-based) are also given in terms of the size of resultant dominating sets and graph connectivity degradation.

## 2.1 Centralized greedy algorithms

In this section, we present two centralized approximation algorithms for finding small weakly connected dominating sets in an arbitrary graph. For each algorithm we analyze its performance by determining the size of the resultant weakly connected dominating sets compared to a minimum weakly connected dominating set. Both algorithms have a $\ln \Delta + O(1)$ approximation ratio. These algorithms form the basis for the upcoming distributed algorithms.

### 2.1.1 The $\mathrm{GW_M}$ algorithm

This algorithm is based on Algorithm II of Guha and Khuller [35]. It is called $\mathrm{GW_M}$ (greedy WCDS algorithm growing multiple black pieces) for reasons that we will see shortly. Given a graph $G = (V, E)$, we associate a color (*white*, *gray*, or *black*) with each vertex. All vertices are initially white and change color as the algorithm progresses. The algorithm is essentially an iteration of the process of choosing a white or gray vertex to dye black. When any vertex is dyed black, any neighboring white vertices are changed to gray. At the end of the algorithm, the black vertices constitute a weakly-connected dominating set.

The term *piece* is used to refer to a particular substructure of the graph. A *white piece* is simply a white vertex. A *black piece* contains a maximal set of black vertices whose weakly induced subgraph is connected plus any gray vertices that are adjacent to at least one of the black vertices of the piece. Figure 2.1 illustrates the definitions. The pieces are indicated by the dotted regions. Vertices 4, 5, 6, and 7 are each white pieces. The other vertices are divided among the two black pieces, one containing the black vertex 1 and the other containing the black vertices 2 and 3.

We define the *improvement* value of a (non-black) vertex $u$ to be the number of distinct pieces within the closed neighborhood of $u$. That is, the improvement value of $u$ is the number of pieces that would be merged into a single black piece if $u$ were to be dyed black.

In each iteration, the algorithm chooses a single white or gray vertex to dye black. The vertex is chosen greedily so as to reduce the number of pieces as much as possible and the process is repeated until there is only one piece left. In particular, a vertex with maximum improvement value is chosen (with ties broken arbitrarily). When the algorithm terminates, the black vertices are the required weakly-connected dominating set $S$.

Figure 2.1 illustrates the situation after the third iteration of the algorithm for the given graph. Dyeing vertex 5 black would merge four pieces, reducing the number of pieces by

three. Dyeing any of the other vertices black would merge at most 3 pieces. Thus, we choose vertex 5 to dye black in the next iteration.

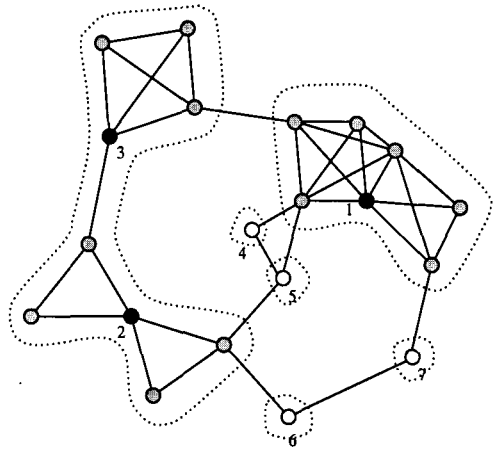



Figure 2.1: A snap shot of pieces.

Let $OPT$ denote a minimum size weakly-connected dominating set for $G$ and let $\Delta$ denote the maximum degree of $G$. We can bound the size of the weakly-connected dominating set found by the $\mathrm{GW_M}$ algorithm as follows:

**Theorem 2.1.1** *The size of the weakly-connected dominating set found by the $GW_M$ algorithm is at most* $(\ln \Delta + 1) \cdot |OPT|$.

**Proof:** Let $u_1$ be an arbitrary vertex of $OPT$. As $u_1$ is of degree at most $\Delta$, at most $\Delta + 1$ distinct vertices can be dominated by $u_1$ (including $u_1$ itself). As $OPT$ is a weakly-connected dominating set of $G$, there must be another element of $OPT$ at distance at most 2 from $u_1$. Let $u_2$ be such a vertex. As at least one vertex in $u_2$'s closed neighborhood is also in the closed neighborhood of $u_1$, at most $\Delta$ new distinct vertices are dominated by $u_2$. Again, as $OPT$ is a weakly-connected dominating set of $G$, there must be another element of $OPT$ at distance at most 2 from either $u_1$ or $u_2$. This vertex, called $u_3$ dominates at most $\Delta$ new distinct vertices. We repeat this argument until we have included all $|OPT|$ elements of $OPT$. Thus, $G$ can contain at most $n \leq (\Delta + 1) + \Delta(|OPT| - 1)$ vertices. It follows that $|OPT| \geq \frac{n-1}{\Delta}$.

In each iteration of the algorithm, we dye a vertex black and put it in set $S$. Observe that the improvement value of any vertex is monotonically non-increasing over time. At the beginning of the algorithm, the improvement value of every vertex $u$ is equal to its degree.

When a neighboring vertex is colored black, $u$ becomes gray and its improvement value decreases by at least one. When a neighboring vertex is colored gray, $u$'s improvement value may decrease but will not increase. When $u$ is dyed black, it no longer has an improvement value.

Let $a_i$ be the number of pieces left after the $i^{\text{th}}$ iteration and let $a_0 = n$. Consider the $i+1^{\text{st}}$ iteration. Since the addition of the (non-black) vertices of $OPT$ would join all of the remaining $a_i$ pieces, decreasing the number of pieces by $a_i - 1$, there is at least one non-black vertex of $OPT$ which would decrease the number of pieces by at least $\left\lceil \frac{a_i - 1}{|OPT|} \right\rceil$.

Figure 2.2 depicts the situation after 2 iterations on the given graph. In the figure, the five circled vertices are a minimum weakly-connected dominating set ($OPT$). At this point, one vertex $u$ in $OPT$ has already been dyed black. Picking the remaining 4 vertices of $OPT$ would join the remaining $a_2 = 10$ pieces. Therefore, our greedy algorithm is guaranteed to decrease the number of pieces at least by $\left\lceil \frac{a_2 - 1}{|OPT|} \right\rceil = 2$ in the $3^{\text{rd}}$ iteration.

This gives us the recurrence relation,

$$a_{i+1} \leq a_i - \left\lceil \frac{a_i - 1}{|OPT|} \right\rceil \leq a_i \left(1 - \frac{1}{|OPT|}\right) + \frac{1}{|OPT|}.$$

Solving it, we get the following bound:

$$\begin{aligned} a_{i+1} &\leq a_0 \left(1 - \frac{1}{|OPT|}\right)^{i+1} + \frac{1}{|OPT|} \sum_{j=0}^{i} \left(1 - \frac{1}{|OPT|}\right)^{j} \\ &= (a_0 - 1)\left(1 - \frac{1}{|OPT|}\right)^{i+1} + 1 \end{aligned}$$

Letting $i + 1 = |OPT| \cdot \ln \frac{a_0 - 1}{|OPT|}$, we have:

$$\begin{aligned} a_{i+1} &\leq (a_0 - 1)\left(1 - \frac{1}{|OPT|}\right)^{i+1} + 1 \\ &= (a_0 - 1)\left(1 - \frac{1}{|OPT|}\right)^{|OPT| \cdot \ln \frac{a_0 - 1}{|OPT|}} + 1 \\ &\leq (a_0 - 1)\left(\frac{1}{e}\right)^{\ln \frac{a_0 - 1}{|OPT|}} + 1 \\ &= (a_0 - 1) \cdot \frac{|OPT|}{a_0 - 1} + 1 \\ &= |OPT| + 1 \end{aligned}$$

That is, after $|OPT| \cdot \ln \frac{a_0 - 1}{|OPT|}$ iterations, the number of pieces remaining is at most $|OPT| + 1$. For each additional vertex we choose, we will decrease the number of pieces

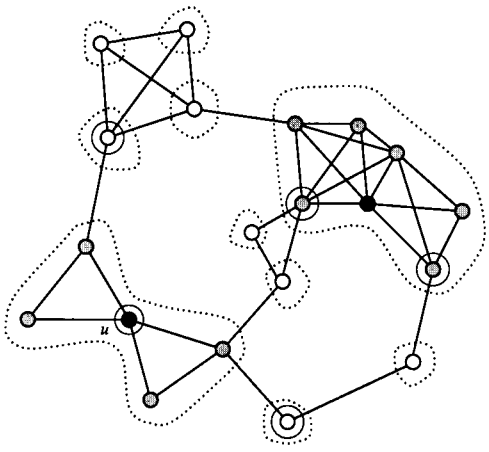



Figure 2.2: Greedy scenario of $GW_M$.

by at least one. Thus, we need only pick at most $|OPT|$ additional vertices to reduce the number of pieces to one. The total number of vertices that we choose is no more than $|OPT| \cdot \ln \frac{a_0-1}{|OPT|} + |OPT|$. Since $|OPT| \geq \frac{n-1}{\Delta}$, the solution found by our algorithm has at most $|OPT| \cdot (\ln \Delta + 1)$ vertices. □

### 2.1.2 The $GW_S$ algorithm

$GW_S$ is a greedy algorithm for constructing small weakly-connected dominating sets by growing a single black piece, called $T$. As with the previous algorithm, the vertices of $G$ are colored *white*, *gray*, or *black*. Initially, all vertices are white and $T$ is empty. When a white or gray vertex $u$ is dyed black, all white vertices adjacent to $u$ are colored gray. When a vertex is dyed black, it is placed into $T$ along with all of its newly gray neighbors. During the execution, a vertex is called a *candidate* if it is either a gray vertex, or a white vertex adjacent to some gray vertex.

The algorithm starts by choosing an arbitrary vertex in $G$ to dye black. In each subsequent iteration, either a gray vertex of $T$ or a white vertex adjacent to a gray vertex of $T$ is chosen to dye black. In the latter iterations, the algorithm considers all of the candidate vertices. For each vertex $u$ it counts the number of white vertices that are in $u$'s closed neighborhood. By coloring $u$ black, this number of vertices will be added to $T$. The candidate vertex with the maximum such value is chosen to be dyed.

Figure 2.3 illustrates the situation after the 3rd iteration in the given graph. Black vertices 1, 2, and 3 are the vertices dyed in the first 3 iterations, respectively. Vertex 4,

which is at distance two from the black vertex 2, is chosen in the 4th iteration, adding three vertices to $T$.

This iterative process continues until all vertices in $G$ are non-white. The set of black vertices at the termination of the algorithm constitute the desired weakly-connected dominating set.

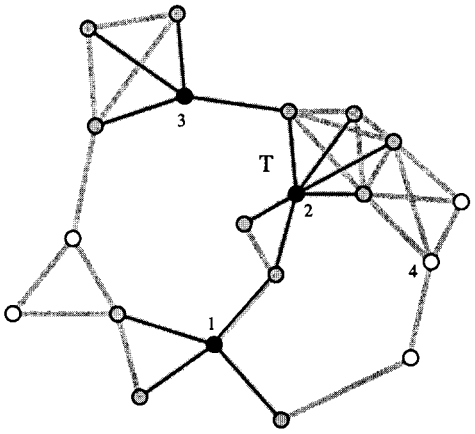



Figure 2.3: Growing the single black piece of $\mathrm{GW_S}$.

To analyze the approximation performance, let $S$ denote the set of black vertices at some point during the execution of the algorithm. As $\langle S \rangle_w$ is always connected and the algorithm terminates when there are no white vertices remaining, $S$ is a weakly-connected dominating set of $G$.

**Lemma 2.1.2** *The size of the weakly-connected dominating set $S$ found by the $GW_S$ algorithm is at most $(\ln \Delta + 2)\gamma$, where $\gamma$ is the domination number of $G$.*

**Proof:** Let $OPT_{DS} = \{v_1, v_2, \cdots, v_\gamma\}$ be a minimum dominating set of $G$. Partition the vertices of $G$ into sets $P_i$, for $1 \leq i \leq \gamma$, such that $v_i \in P_i$ and every vertex $w$ of $V - OPT_{DS}$ is placed into $P_i$ for some $i$ such that $v_i$ dominates $w$.

The proof is based on a charging analysis. Each time we dye a vertex black, we add one vertex to $S$ and incur a charge of one unit. This charge is equally distributed among all of the white vertices that are colored in that step. The total charge for the entire process is $|S|$, the size of the weakly-connected dominating set. We will show that the total charge among the vertices of $P_i$ (for any $i$) is at most $\ln \Delta + 2$. Since there are $OPT_{DS}$ such sets in $G$, the theorem follows.

Assume that when we choose a vertex to dye black, we color $x$ white vertices and charge such vertex $\frac{1}{x}$. We now consider the number of charges among the vertices of a single set $P_i$. Let $u_j$ denote the number of white vertices after iteration $j$. For the sake of simplicity, we assume that $u_0 = |P_i|$ and that some vertices of $P_i$ are colored in each iteration, so the number of white vertices in $P_i$ decreases in each iteration.

The number of vertices of $P_i$ that are colored in the first iteration is $u_0 - u_1$. Each of these vertices is charged at most $\frac{1}{u_0 - u_1}$. (The actual charge may be smaller since some white vertices outside of $P_i$ may also be colored in the same iteration.) Once any vertex in $P_i$ is colored, vertex $v_i$ itself is eligible to be chosen. After the $j-1^{\text{st}}$ iteration, choosing $v_i$ would add $u_{j-1}$ vertices to $S$, so any vertex chosen must add at least that many to $S$ and, thus, any vertices of $P_i$ colored in the $j$th iteration incur a charge of at most $\frac{1}{u_{j-1}}$. Therefore, in the $j^{\text{th}}$ iteration, at most $\frac{u_{j-1}-u_j}{u_{j-1}}$ charges are incurred among the vertices of $P_i$. Eventually, $u_k = 0$ for some $k$.

Summing the charges within $P_i$, we get at most

$$\begin{aligned}
& \frac{1}{u_0 - u_1}(u_0 - u_1) + \sum_{j=2}^{k} \frac{1}{u_{j-1}}(u_{j-1} - u_j) \\
= \; & 1 + \sum_{j=2}^{k} \frac{u_{j-1} - u_j}{u_{j-1}} \\
\leq \; & 1 + \sum_{j=2}^{k} (H(u_{j-1}) - H(u_j)) \\
= \; & 1 + (H(u_1) - H(u_k)) \\
\leq \; & 1 + (H(\Delta) - H(u_k)) \\
= \; & 1 + H(\Delta) \\
\leq \; & 2 + \ln \Delta
\end{aligned}$$

Therefore, the lemma holds. □

Knowing $\gamma \leq \gamma_w$, we have the following bound,

**Theorem 2.1.3** *The performance ratio of the $GW_S$ algorithm is* $\ln \Delta + 2$.

## 2.2 $DGW_M$ and $DGW_S$ — the distributed implementations

In an ad hoc network, a subscriber unit is not assumed to be able to know the structure of the network beyond its own neighborhood before global knowledge is established. In that sense, a distributed algorithm for weakly-connected dominating sets is more feasible in the real world. In this section, we present distributed versions of $GW_M$ and $GW_S$, denoted $DGW_M$ and $DGW_S$, with the same performance ratios of $\ln \Delta + 1$ and $\ln \Delta + 2$, respectively, for finding a small weakly-connected dominating set $S$ of the input graph $G = (V, E)$.

The centralized versions of both $DGW_M$ and $DGW_S$ construct their weakly-connected dominating set $S$ by including a globally best vertex for each iteration. In order to achieve the same execution results in the distributed scenario, we use a special vertex to be a global arbitrator to decide which vertex should be included in each iteration.

Both algorithms have three stages:

1. Leader election — to find the global arbitrator of the network.
2. BFS (Breadth-First Search) tree construction — to construct a sparse subgraph of $G$ with small diameter for information dissemination and collection.
3. Main algorithm — the major distributed greedy procedure for constructing a weakly-connected dominating set.

The first two stages are standard problems in distributed computing with well-established solutions. We use the asynchronous multi-port network model and assume a constant upper bound on the time that a message is delivered. The leader election algorithm we choose for both $DGW_M$ and $DGW_S$ is from Awerbuch [9]. The time and communication complexities are $O(n)$ and $O(m + n \log n)$, respectively. We use the LayeredBFS algorithm [52], a simple asynchronous BFS tree construction algorithm, to construct the BFS trees. The algorithm takes $O(diam_G^2)$ time units to finish using $O(m + n \times diam_G)$ messages.

The major procedure of the $DGW_M$ algorithm runs on top of the leader election and BSF tree algorithms as above. In particular, the root $r$ of the BSF tree is generated by the leader election algorithm. The notions of piece and improvement value are the same as in $GW_M$. We assume that every vertex knows which vertices are its neighbors, the colors and piece ID's of these neighbors, and, consequently, its own current improvement value.

An iteration starts when the root $r$ of the BFS tree broadcasts an `improvement_inquiry` message down to all other vertices via the tree edges. This message is issued to determine the

maximum improvement value within each subtree. When a leaf vertex receives this message, it reports its improvement value to its parent in the BFS tree. When an internal vertex has collected the replies from all of its children, it reports the maximum improvement value within its subtree back to its own parent. The replies are convergecast to $r$ along the tree edges, so that the root $r$ can determine the globally maximum improvement value and its location. Then $r$ sends a message to a vertex with maximum improvement value to start the coloring process. Once a vertex is colored black, it broadcasts the change to its neighbors, and the white neighbors color themselves gray. Similarly, every time a vertex is colored gray, it informs its neighbors so that they can update their information. Then the new black vertex broadcasts its vertex ID along the dominated edges to update the piece ID of all vertices within its piece. When this is done, a vertex can determine its improvement value by counting the number of distinct piece ID's within its neighborhood. After the coloring and piece ID broadcasting process, the new black vertex sends an acknowledgment message to $r$. When $r$ receives this acknowledgment, it is ready to send an `improvement_inquiry` message to start the next iteration.

The $\text{DGW}_\text{M}$ algorithm proceeds for $|S|$ iterations. The improvement inquiry and collection in each iteration takes at most $O(diam_G)$ time units using $O(n)$ messages, where $diam_G$ denotes the diameter of graph $G$. Broadcasting the piece ID along the dominated edges takes at most $O(|S|)$ time units using at most $O(m)$ messages. Since $diam_G = O(|S|)$, each iteration is finished in $O(|S|)$ time units using $O(m)$ messages. Therefore, including leader election and BFS tree construction, the time and message complexities of $\text{DGW}_\text{M}$ are $O(|S|^2)$ and $O(m \times |S| + n(\log n + diam_G))$, respectively. This is summarized in Table 2.1.

| | Time$\times d$ | Message |
|---|---|---|
| Leader election | $O(n)$ | $O(m + n \log n)$ |
| BFS tree construction | $O(diam_G^2)$ | $O(m + n \times diam_G)$ |
| Main procedure | $O(\|S\|^2)$ | $O(m \times \|S\|)$ |
| Total | $O(\|S\|^2)$ | $O(m \times \|S\| + n(\log n + diamg_G))$ |

Table 2.1: Breakdown of $\text{DGW}_\text{M}$.

The $\text{DGW}_\text{S}$ algorithm, the distributed implementation of $\text{GW}_\text{S}$, also runs on top of a BFS tree rooted at the arbitrary vertex $r$ chosen by a leader election algorithm. As in

$GW_S$, a vertex is called a candidate if it is either gray, or white and adjacent to a gray vertex. $DGW_S$ grows a single black piece by coloring black a candidate vertex with the maximum number of white neighbors in each iteration. To locate such a vertex, root $r$ sends a `best_candidate_inq` inquiry message along the tree edges.

As the main procedure of the algorithm involves $|S|$ iterations and $O(n)$ messages are needed for each iteration, the message complexity is $O(n \times |S|)$. The time complexity is $O(diam_G)$ for each iteration and $O(diam_G \times |S|)$ for the main procedure. Table 2.2 shows the time and message complexities of the entire algorithm — $O(|S| \times diam_G)$ and $O(n \times |S| + m + n \log n)$.

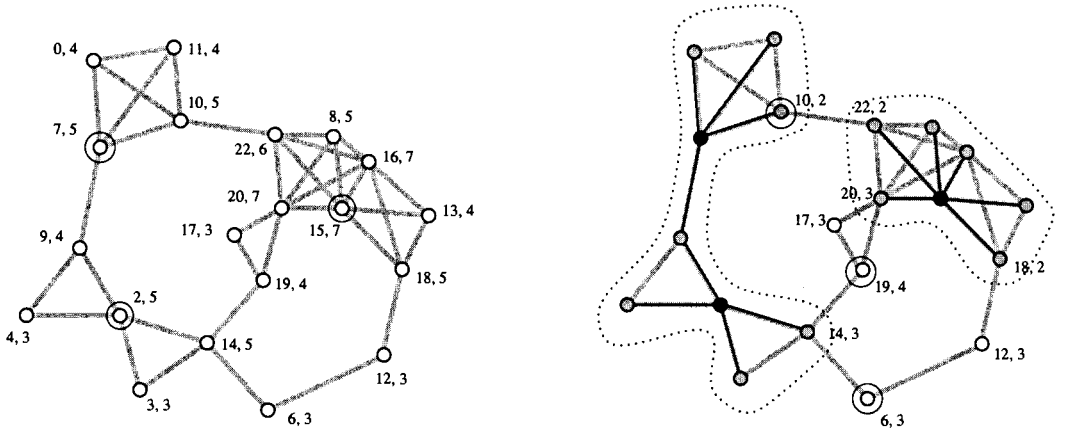
| | Time$\times d$ | Message |
|---|---|---|
| Leader election | $O(n)$ | $O(m + n \log n)$ |
| BFS tree construction | $O(diam_G^2)$ | $O(m + n \times diam_G)$ |
| Main procedure | $O(\|S\| \times diam_G)$ | $O(n \times \|S\|)$ |
| Total | $O(\|S\| \times diam_G)$ | $O(n \times \|S\| + m + n \log n)$ |

Table 2.2: Breakdown of $DGW_S$.

## 2.3 Distributed heuristics for small weakly-connected dominating sets

Although the $DGW_M$ and $DGW_S$ algorithms have eliminated the requirement of an omniscient view using a central arbitrator and decision message passing, they are still sequential inherently in process since only one vertex can be colored black in each iteration. It is desirable to be able to color multiple vertices black in each iteration in order to speed up the execution. In other words, different parts of $G$ could have vertices colored black in parallel.

In a fully distributed approach, we grow multiple black pieces in parallel, each of which using its own internal decision mechanism to determine its own best candidate. Here, any gray or white vertex may be a *candidate*. (Recall that in the $GW_S$ algorithm a candidate vertex was either a gray vertex or a white vertex adjacent to some gray vertex.) To be considered as a candidate, a gray or white vertex must also have the largest improvement value among any vertex in its closed neighborhood. In each iteration, each piece determine its own candidate(s). A black piece may have more than one candidate, while a white piece




(a) Iteration 1, three best candidates. (b) Iteration 2, three best candidates.

Figure 2.4: First 2 iterations.

may have at most one candidate (itself). Each piece selects from its own candidate vertices the candidate with maximum improvement value, called the *best candidate*. The chosen candidate vertex is then colored black, causing neighboring white vertices to be colored gray and tangent pieces to be merged into a larger one. A tie is broken arbitrarily, say by using the vertex ID. The *piece ID*, unique to each piece, of a new piece is broadcast to all vertices in the new piece and the new piece is ready for next iteration. We assume every node knows the color and piece ID information of all of its neighbors at any time. The algorithm terminates when no piece has a candidate with improvement value greater than one.

Figure 2.4 gives an example of how these best candidates are found in each iteration. The numbers beside each vertex are vertex ID, improvement pairs. We omit the vertex labels for those vertices with improvement value of 1. Figure 2.4(a) depicts the scenario of the first iteration when 3 white vertices (vertices 2, 7, and 15) find themselves to be candidates by comparing their improvement values with those of their neighbors. When two neighboring vertices, such as vertices 7 and 10 in our example, have the same improvement value, the vertex with the lower ID is chosen. The 3 candidates are automatically the best candidates as they are the only vertices in their (white) pieces, and they decide to color themselves black. At the end of the first iteration, two larger black pieces are formed, as shown in Figure 2.4(b). During the second iteration, another two white vertices, vertex 6 and 19, and one gray vertex, vertex 10, are the best candidates. Note that in Figure 2.4(b), vertex 14 is not a candidate since it does not have the largest improvement value in its closed

neighborhood.

A spanning tree is maintained for each piece. The root of the spanning tree is responsible for finding the best candidate according to the messages convergecast to it. The root sends a message to the best candidate notifying it to color itself black. Coloring a vertex black causes neighboring white vertices to become gray and the neighboring pieces are merged. At this moment, the vertices of the piece do not all have the same piece ID, but the piece consists of the set of vertices reachable from each other using only dominated edges. A leader election process is executed in the new piece and the ID of the new leader is used as the piece ID. The leader then initiates the construction of a spanning tree within the piece by broadcasting a message along the dominated edges. This spanning tree is used to find the best candidate of the piece in the next iteration.

In the following subsections, we present two distributed algorithms for finding small weakly-connected dominating sets in the synchronous and asynchronous settings, respectively. These algorithms choose multiple vertices to color black in each iteration.

### 2.3.1 $\mathrm{DW}_{\mathrm{SYNC}}$ — synchronous distributed approach

The $\mathrm{DW}_{\mathrm{SYNC}}$ algorithm assumes a synchronous network, where there is a master clock that all vertices share. Therefore, all computation and message passing can be divided into rounds, and each round of the execution (computation and message passing) is coordinated among all vertices. We use the classic multi-port distributed computing model (See Chapter 1 for more details). To be formal, in each step, a vertex receives zero or one message from each incident link, carries out internal computation based on its current state and the input messages, and sends out zero or one message on each incident link. All messages sent at the end of a round are received at the beginning of the next round successfully assuming the communication channels are reliable. The assumption that at most one message is sent in each direction along any communication link is not restrictive since we can always combine multiple messages into a single message. All internal computation is simple and can be finished within a round, otherwise we can always extend the duration of a round so that the longest computation can be accommodated. In addition, we assume that each vertex is aware of the vertex ID's, piece ID's and colors of all of its neighbors. The piece ID can be anything that is unique to each piece. In our algorithm, we use the vertex ID of a representative vertex in that piece, called the *root*, to be the piece ID. To enforce the synchrony of the network, we further assume that each vertex has a reasonable upper bound

on the number of vertices in the network, denoted $n$. This bound is used to ensure that the sub-procedures of the algorithm are synchronized among different pieces. If we restrict the execution of the algorithm to a certain area of the network, then the upper bound on the vertices involved can also be determined (see Chapter 3 for more details on this). In Section 2.1.1, we propose an asynchronous algorithm that does not assume knowledge of $n$.

Recall that a piece is a substructure of the network and can be either white or black. A white piece is a white vertex by itself and a black piece is a connected component weakly induced by the black vertices. Here, a candidate of a piece in the distributed algorithm is a vertex with the highest improvement value within its neighborhood and the best candidate of a piece is the candidate with the highest improvement value within that piece. The entire process of finding a weakly-connected dominating set is divided into iterations, and in each iteration, the best candidates of all pieces are colored black, forcing their white neighbors gray, and tangent pieces are merged if they are in the same connected component of the subgraph weakly induced by the black vertices. See Figure 2.4 as a simple example. Details of execution and timing of each iteration are described below.

Initially, all vertices are activated and they exchange vertex ID's and piece ID's with their neighbors. At the beginning, the piece ID of a vertex is the same as its own vertex ID. By counting the number of different piece ID's within its neighborhood, a vertex is able to calculate its initial improvement value. Then each vertex broadcasts its improvement value to all neighbors so that it learns if it is a candidate. The initialization process takes a constant number of rounds. After this, the algorithm enters its first iteration and this continues until the pieces of the entire network merge and become a single piece.

At the beginning of each iteration, pieces are stable and can be distinguished by the piece ID of each vertex. In other words, at this point of an iteration, all vertices in a piece, a connected component of the subgraph weakly induced by the black vertices, have the same piece ID. Plus, within each piece, there is a spanning tree rooted at the root vertex. Such a tree, called a *broadcast tree*, is used to broadcast and convergecast messages as will be seen later.

From the first iteration on, each piece root determines its best candidate, the candidate with the largest improvement value in its piece, by broadcasting a `best_candidate_inq` and convergecasting the comparison results via the broadcast tree. Once a root has determined its best candidate if it has one, it unicasts a `please_color_black` message to the best candidate. When the best candidate has received the `please_color_black` message, it

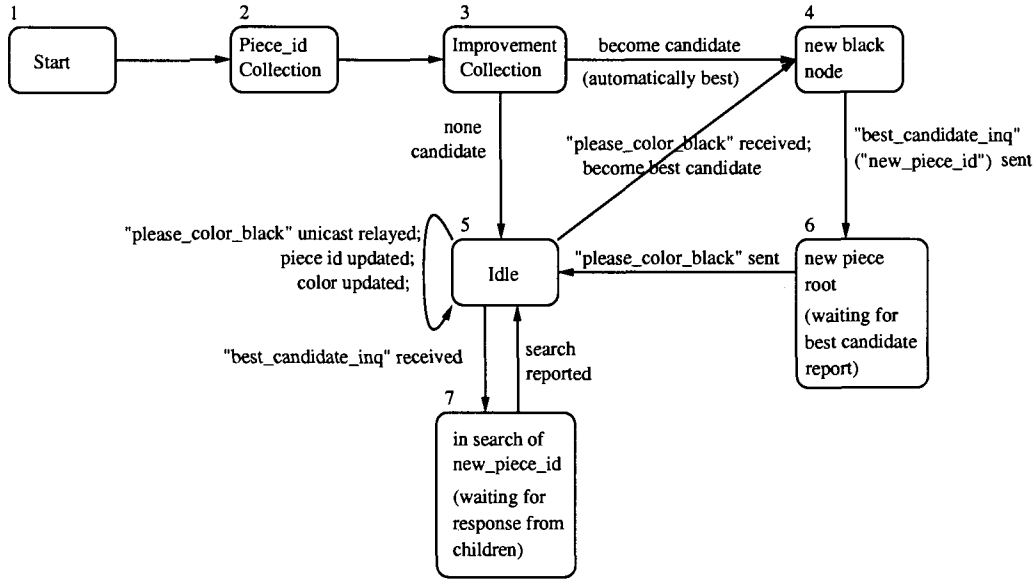



Figure 2.5: Flow chart for a single vertex in $\mathrm{DW}_{\mathrm{SYNC}}$.

colors itself black and broadcasts a `colored_black` message to all neighbors. In the next round, all white vertices receiving the `colored_black` message color themselves gray and broadcast a `colored_gray` message within its neighborhood so that its neighbors can update their color information. This can be done in at most $3n + O(1)$ rounds.

After $3n + O(1)$ rounds, the best candidate selection and color updating are finished. In the following rounds, tangent pieces are merged into a single piece, a new root is selected, a broadcast tree rooted at a new leader is constructed and the new piece ID is distributed in each new piece. To be more specific, each new black vertex generated in this iteration broadcasts a `my_piece_ID` message within its new piece. To make sure these messages travel only within the new piece, they only take dominated edges. When a message reaches a free edge, it knows it is at the border of a piece and does not go beyond it. After another $n$ rounds, the broadcast process is completed, and each new black vertex has received a collection of piece ID's from all of the new black vertices within the new piece. Of the new black vertices of the new piece, the one with the lowest ID declares itself root of the new piece. In the following rounds, the new root broadcasts a `new_piece_ID` message within its new piece and forms a new broadcast tree. After yet another $n$ rounds, this iteration is over and all vertices are ready for the next iteration.

Figure 2.5 shows a flow chart of a vertex's process. The initialization finishes at state

3. A vertex may or may not become black in the first iteration. After the first iteration, moving between states 5 and 7 is the basic activity of a vertex unless it becomes a black vertex. States 4 and 6 depicts the behavior of a new black vertex. Once a new black vertex finishes locating its own best candidate it goes back to state 5 again and resumes circulating between states 5 and 7, until is informed by its piece root that the algorithm is completed. Note that each movement between states 5 and 7 corresponds to an iteration.

The total time of the algorithm is at most $|S| \times (5n + O(1))$, since it takes at most $|S|$ iterations to finish. The total message cost is $O(m \times |S|)$ since at most $|S|$ leader elections are held, each requiring at most $O(m)$ messages, and this is the most costly process in terms of messages.

$\mathrm{DW_{SYNC}}$ does not, in general, generate the same weakly-connected dominating set as $\mathrm{GW_M}$.

### 2.3.2 $\mathrm{DW_{ASYNC}}$ — asynchronous distributed approach

Here, we use an asynchronous model in which all vertices in the network are autonomous machines using their own clocks and there is no common clock. We assume that there is a constant bound on how much time a link can delay message delivery and how much time a vertex can spend on a single step of execution.

The basic process of our algorithm is as described above. Whenever a vertex is colored black (from white or gray), it causes all of its white neighbors to be colored gray by sending an update message. Whenever a vertex changes its color or its piece ID, it informs all of its neighbors about the change.

The piece ID is used to help calculate the improvement value of vertices. However, due to propagation delay, vertices in the same piece may appear to have different piece ID's. By calculating an improvement value without waiting for the piece ID's to be fully propagated, the improvement value may be calculated incorrectly. In our experiments, this situation does not occur very often and the size of the weakly-connected dominating set generated by the $\mathrm{DW_{SYNC}}$ algorithm is very close to that generated by $\mathrm{GW_M}$ (see Section 2.4). Recall that a root vertex of a piece is the vertex whose vertex ID is used as the piece ID. The algorithm proceeds as follows: a root vertex broadcasts a `best_candidate_inq` message within its piece. When other vertices forward this inquiry, they only use the dominated edges to make sure that the message does not propagate beyond the piece border. These out-going messages form a broadcast tree in the piece. Leaves of this tree reply with their

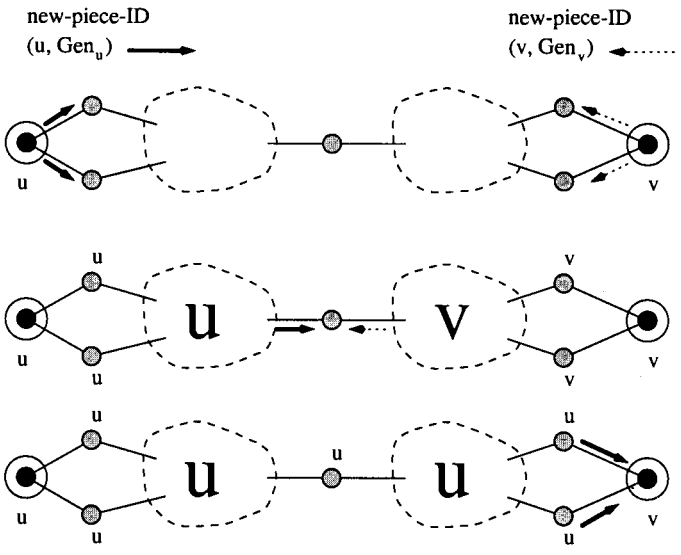



Figure 2.6: When two front lines meet.

improvement values and internal vertices of the tree collect results from their children and forward the best value back up the tree. When the root knows that it has a best candidate with improvement value of more than 1, it sends a `please_color_black` message to the corresponding vertex. When that vertex receives the `please_color_black` message, it colors itself black and tells all neighbors about this message. The white neighbors color themselves gray, causing some tangent pieces to merge. The newly colored black vertex becomes the root of the new piece and sends a `new_piece_ID` message to all vertices within the larger piece. The root also sends a `best_candidate_inq` along with the `new_piece_ID` message, starting the next iteration.

Due to the asynchrony and the possibility of multiple simultaneous changes to the structure of the pieces, it is possible that multiple vertices will declare themselves to be roots of a single piece and initiate a new search process. We need a mechanism to choose among the roots as their `new_piece_ID` messages encounter each other. We define the notion of the *generation* of a black vertex. A black vertex that has just been colored from white is a first generation black vertex. When a root (black) vertex chooses the best candidate of its piece, the old root assigns the new root to be of generation one greater than its own. The `new_piece_ID` messages carry the generation number of the root. When two `new_piece_ID` messages meet each other, the later generation message (with the larger generation number) continues to broadcast while the earlier generation message is dropped. When two

`new_piece_ID` messages with the same generation number meet, we favor the smaller piece ID to break the tie.

In Figure 2.6, we assume that two new black vertices, $u$ and $v$, broadcast their `new_piece_ID` messages within the new black piece and $u$ has a larger generation number. Before the messages meet, the left side portion of the piece accepts $u$ as the new piece ID and the right side takes $v$. Once the two messages meet, the message of $u$ overtakes that of $v$ and pushes toward $v$ so that the entire piece eventually accepts $u$ as the piece ID.

The algorithm ends when a new black vertex finds that its best candidate has an improvement value of one.

## 2.4 Comparisons

Using the members of a weakly connected dominating set as clusterheads and the closed neighborhood of each clusterhead as a cluster, we can represent the network as a graph of clusters from a higher level. Clusterheads can be directly joined by a link or may share a neighbor that can relay communications between them. Note that links between non-clusterheads are not included in the higher level graph. Using a weakly-connected dominating set to define clusters helps to simplify the network structure. However, it may compromise the network structure to some degree as some edges are not included.

To measure the performance of our clustering algorithm, we consider three parameters. First we consider the size of the dominating set produced. Since our goal is to find a small weakly-connected dominating set in order to abstract the network structure as much as possible, we prefer smaller values for this parameter. The second parameter is the average distance between pairs of vertices in the *clustered network* (the subgraph weakly induced by the dominating set). Clustering using weakly-connected dominating sets removes edges not incident on vertices of the dominating set resulting in a sparser structure. This sparser structure dilates the network so that the distance between pairs of vertices in the clustered network may be longer than the corresponding distance in the original network. We do not wish the dilation to be too large, so smaller average distance is preferred. The third parameter is the average number of edge-disjoint paths between pairs of vertices in the clustered network. This parameter measures, in some sense, the capacity of the network. As we are simplifying the structure and deleting both edges and vertices, we can expect that this value will be less than in the original graph. However, we would prefer that it does not

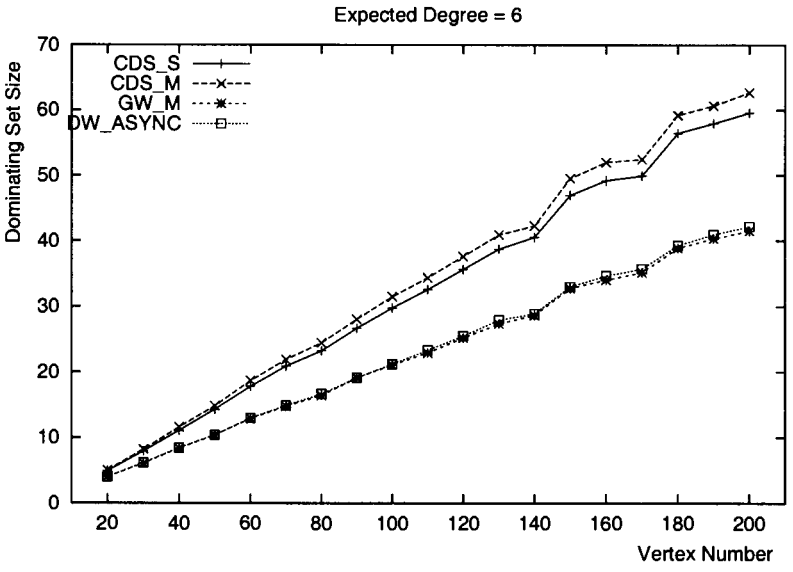



Figure 2.7: Dominating set size — average degree 6.

drop significantly.

In our experiments, we generate random graphs repeatedly and run our algorithms and those of Das and Bharghavan [24], measuring the above parameters. The random graphs have expected average degree of 6 and 12, providing two levels of density. The size of the graphs ranges from 20 to 200 vertices and 40 to 200 vertices, respectively. For each graph size and density level, we repeat the algorithms 100 times. To simulate the structure of ad hoc networks, we place vertices (subscriber units) randomly in a rectangular area in a 2D-plane. The coordinates of the vertices are chosen uniformly in each dimension. We assign each subscriber unit a random transmission range with a predefined expected value. When two subscriber units are placed within range of each other, an edge is added between the vertices simulating a reliable link between them. By changing the number of vertices in the plane and the expected transmission range, one can adjust the network size and density.

For each randomly generated network, we measure the dominating set size resulting from the two algorithms of Das and Bharghavan [24] and our $\mathrm{GW_M}$ and $\mathrm{DW_{ASYNC}}$ algorithms. Figures 2.7 and 2.8 show the results obtained by the four algorithms when the graph is sparse and dense, respectively. In the figures, `CDS_S` is the "single-black-piece" connected dominating set algorithm (Algorithm II of Das and Bharghavan) `CDS_M` is the "multiple-black-piece" connected dominating set algorithm (Algorithm I of Das and Bharghavan),

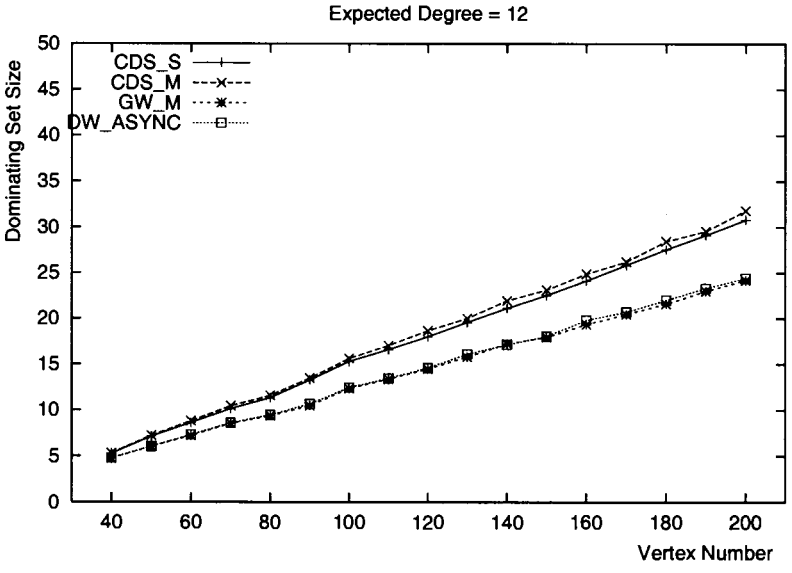



Figure 2.8: Dominating set size — average degree 12.

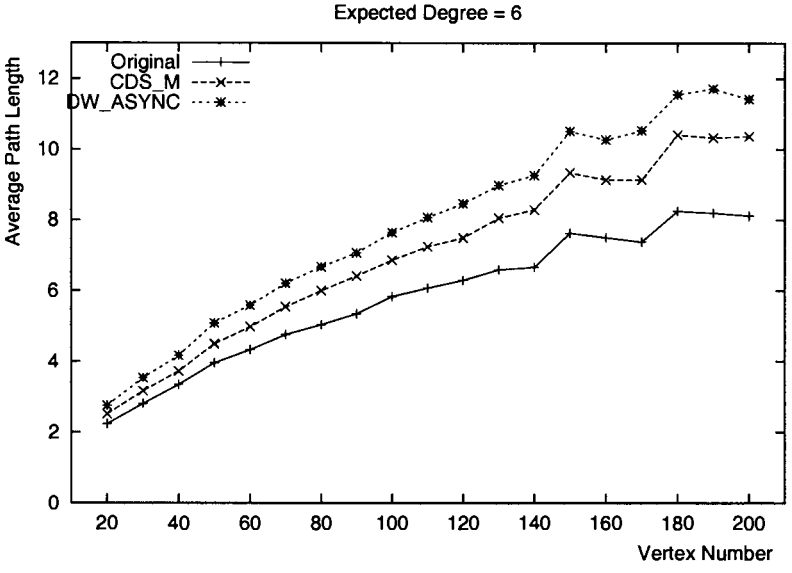



Figure 2.9: Average vertex distance — average degree 6.

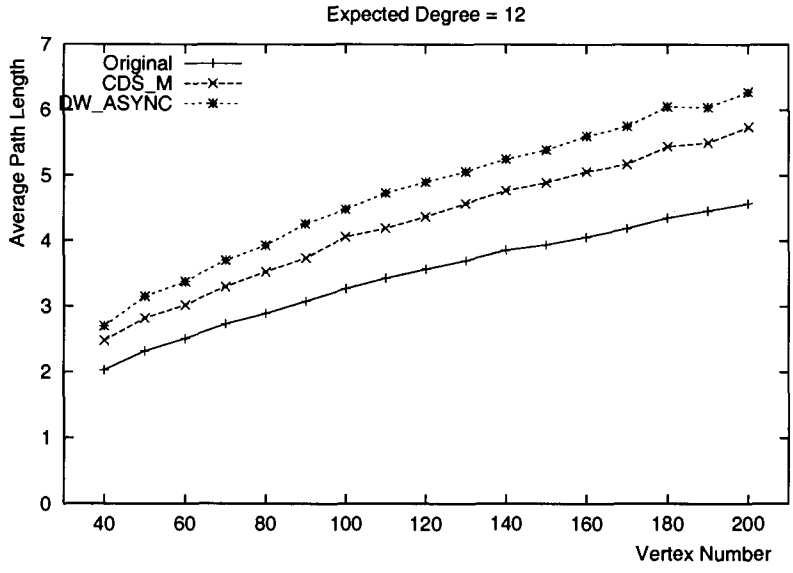



Figure 2.10: Average vertex distance — average degree 12.

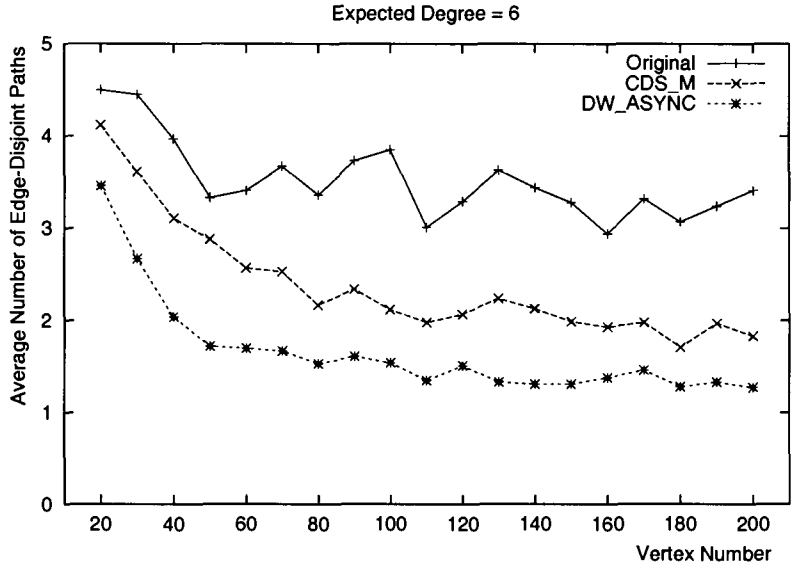



Figure 2.11: Average number of edge-disjoint paths — average degree 6.

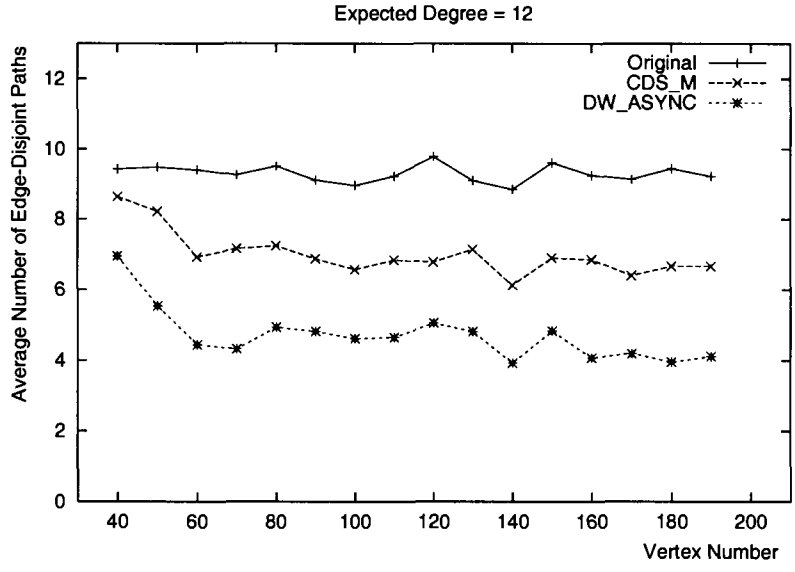



Figure 2.12: Average number of edge-disjoint paths — average degree 12.

`GW_M` is our centralized greedy algorithm $GW_M$, and `DW_ASYNC` is our distributed algorithm $DW_{ASYNC}$. It is apparent that our algorithms generate smaller dominating sets than those of Das and Bharghavan.

The average distance and number of edge-disjoint paths between vertex pairs are compared between the original graph (`Original`), Algorithm I of Das and Bharghavan (`CDS_M`) and our $DW_{ASYNC}$ algorithm (`DW_ASYNC`). Both sparse and dense graphs were considered as shown in Figures 2.9 to 2.12. These results indicate that the average distance between vertex pairs increases by about 20% when connected dominating sets are used for clustering and increases a further 10% when weakly connected dominating sets are used. These increases are not unreasonable. The average number of edge-disjoint paths is reduced by about 25% when connected dominating sets are used for clustering and by a further 20% when using weakly connected dominating sets. Again, these decreases are as expected.

# Chapter 3

# Zonal Computation

Distributed algorithms for solving ad hoc networking problems are generally expected to be localized, i.e. individual processes only need to know the information within vicinity. The localized requirement can make distributed algorithms more efficient and reliable. Interested readers are referred to the book of Peleg [59] for a detailed elaboration of this topic. A zonal distributed algorithm is a compromise between execution locality and result quality. In such algorithms, a network is first divided into overlapping or non-overlapping connected substructures, called *zones*. A best-effort non-localized procedure can be executed within each zone and a localized procedure can be executed in the super-graph on the zones. The size of the zones in a zonal distributed algorithm is usually controllable. In this chapter, we present a distributed zonal distributed algorithm for constructing small weakly-connected dominating sets, denoted ZW, with a single parameter to control the zone granularity.

## 3.1 Introduction

The greedy algorithms for finding small weakly-connected dominating sets in Chapter 2 have asymptotically optimal approximation ratios. Their distributed implementations have been shown to have similar performance by experiments. However, these distributed implementations are not localized, in the sense that some nodes need to know the distant information and the execution time becomes increasingly longer as the network gets larger. This is less desirable especially when the clustering process needs to be executed frequently. In order to make the clustering process more viable in real distributed environments, such as mobile ad hoc networks, algorithms that only use local information (that is, information from

nodes and links within constant distance) are desired. These algorithms are generally called localized distributed algorithms. In the context of clustering ad hoc networks, the work of Wu and Li [73] is a typical localized distributed algorithm for finding small connected dominating sets.

Wu and Li present a clever localized distributed algorithm for finding small connected dominating sets to cluster ad hoc networks, in which a node only needs to know information from within its distance-2 neighborhood. What is more, the execution time is bounded by $O(\Delta^3)$ from above. The penalty, however, for this near-sightedness is that the algorithm does not guarantee a good approximation ratio. In fact, it was shown by Alzoubi et al. [71] that the approximation ratio can be as bad as $\frac{n}{2}$, where $n$ is the number of nodes in the network.

In this chapter, we present ZW, a zonal weakly-connected dominating set construction algorithm. In ZW, a zone is a connected subgraph of the input network that has no more than $2x$ vertices, where $x$ is a preset *zone size control parameter*. A zone has a dedicated vertex known by all zone members as the *root*. The zonal construction algorithm has two levels:

- Intrazonal — a weakly-connected dominating set is independently constructed for each zone;
- Interzonal — the root of each zone determines a set of vertices to add to its weakly-connected dominating set to guarantee that the union of all the zones' dominating sets is a weakly-connected dominating set for the entire network.

The ZW algorithm consists of three phases. In the first phase, the graph is partitioned into non-overlapping zones by constructing a spanning forest. At the end of this phase, the root of each tree in the spanning forest is the *zone root* and its ID is known by all vertices of the zone as the *zone ID*.

In the second phase, the $\text{DGW}_\text{S}$ of Chapter 2 is independently executed in each zone to find a small weakly-connected dominating set within the zone. The zone root coordinates this process by collecting zonal progress information and then making and disseminating decisions throughout the zone.

In the third phase, a weakly-connected dominating set of the entire graph is formed by adding more vertices (if required) to the union of the weakly-connected dominating sets of

all the zones. In particular, when two zones are adjacent but not connected by a dominated edge, a dominated vertex is added to the dominating set.

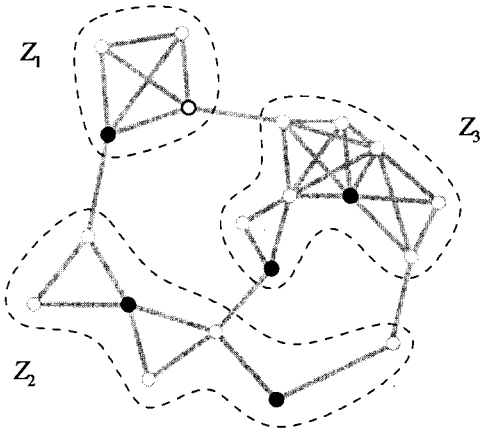



Figure 3.1: Zonal clustering scheme.

Figure 3.1 depicts an example of the zonal clustering scheme. It is a snapshot when the clustering is initially completed in the given network. The network is partitioned into three zones, $Z_1$, $Z_2$, and $Z_3$, as indicated by the dashed contours. The solid black vertices belong to their zone's weakly-connected dominating set after the intrazonal phase. The hollow black vertex can be added to the weakly-connected dominating set of zone $Z_1$ to "fix" the border to $Z_3$. Note that the border between zones $Z_1$ and $Z_2$ and that between zones $Z_2$ and $Z_3$ need not be fixed because these borders contain dominated edges.

The advantage of the zonal approach is that the zone size control parameter $x$ can be used to control the zone granularity, providing a trade-off between the extent of network structure simplification and the locality of algorithm execution.

Section 3.2 discusses the first phase of clustering — partitioning graphs into zones. Section 3.3 revisits the approximation algorithms for finding small weakly-connected dominating sets. The border fix phase is investigated in detail in Section 3.4. Experiments and results are studied in Section 3.5, followed by the conclusion.

## 3.2 Graph partitioning

Now we concentrate on the first phase of our localized distributed clustering algorithm — the process of partitioning a given graph $G = (V, E)$ into non-overlapping zones. Let $n = |V|$

and $m = |E|$. We discuss and compare three different approaches and proceed with the third one. To analyze the time and communication complexities, we use the synchronous multi-port network model (Section 1.5).

### 3.2.1 Approaches to graph partitioning

One approach is to generalize the independent dominating set algorithm of Lin and Gerla [50]. Here we generate a distance-$k$ dominating set $S$ of $G$, such that every vertex that is not in $S$ has a vertex in $S$ within its distance-$k$ neighborhood and, furthermore, each pair of vertices in $S$ is separated by a distance of at least $k+1$. $S$ is generated by performing *distance-k neighborhood broadcasts*, in which the broadcast message travels at most $k$ hops away from the source by setting the TTL (time to live) value of a message to $k$. At the beginning, each vertex in $V$ makes a distance-$k$ neighborhood broadcast to advertise its vertex ID. After this, every vertex knows the vertex ID's of every other vertex in its distance-$k$ neighborhood. Each vertex that has the lowest vertex ID within its distance-$k$ neighborhood proclaims itself a distance-$k$ dominator and broadcasts its status change within its distance-$k$ neighborhood. Every vertex that has a distance-$k$ dominator within the distance-$k$ neighborhood changes to the status of distance-$k$ dominatee and broadcasts this status change within its distance-$k$ neighborhood. Note that a vertex that does not have the lowest ID within its distance-$k$ neighborhood still has a chance to be a dominator. When all vertices with lower ID's within its distance-$k$ neighborhood are known to be dominated, it becomes a dominator. In the end, a distance-$k$ dominating set $S$ is formed, consisting of the vertices with distance-$k$ dominator status. Furthermore, every pair of vertices in $S$ is separated by a distance of at least $k+1$. Every dominatee accepts the vertex that first becomes distance-$k$ dominator within its distance-$k$ neighborhood as its own distance-$k$ dominator.

The time complexity of this partitioning process is bounded by $O(diam_G)$, which is in turn bounded by $O(n)$. This follows from the fact that $diam_G$ is the maximum amount of time for any status change information to propagate through the graph. Practically, this time can be much lower since the distance-$k$ broadcast processes can be running in parallel throughout graph $G$. The communication cost of the flooding process at a particular relaying vertex $v$ is $d(v)-1$, where $d(v)$ is the vertex degree of $v$. Thus, for any vertex to complete a distance-$k$ broadcast, it can generate as many as $O((\Delta-1)^k)$ messages, with $\Delta$ being the maximum vertex degree in graph $G$. Recall that the number of messages that a single

distance-$k$ broadcast can generate is also trivially bounded by $O(m)$. Every vertex in $G$ does exactly two distance-$k$ broadcasts, one for the vertex ID advertisement and the other for the status change announcement. Therefore, the total communication cost of the partitioning algorithm is bounded by $O(n(\Delta-1)^k)$.

A second method to partition a network is to use vertex $r$ as a central coordinator of the network to grow a "hyper spanning tree" (HST) in $G$. Every node in the HST represents a partition in $G$, and $r$ resides in the root of the HST. An HST can be constructed as follows. The root $r$ first forms a partition of $G$ consisting if all vertices within distance $k$ of $r$ itself and adds the partition to the tree node set. To do that, $r$ starts a distance-$k$ broadcast and labels all vertices in its distance-$k$ neighborhood with its ID. Then $r$ chooses a vertex $v_1$ at distance $k+1$ and instructs it to form a second partition by labeling the unlabeled vertices within its distance-$k$ neighborhood with $v_1$'s ID. When this is completed, $v_1$ chooses yet another unlabeled vertex $v_2$ at distance $k+1$, and this process is repeated. If, for example, $v_2$ can not extend the partitioning, it reports to its parent, $v_1$ in this case, and $v_1$ will choose another unlabeled vertex $v_2$ at distance $k+1$ from $v_1$ to form a partition. The construction of the HST continues in this fashion and is similar to depth-first search. Backtracking is performed only when necessary. This approach is can take $O(n)$ time to form the tree. Its communication cost is at most $O(\frac{n}{k}(\Delta-1)^k)$ message since only $|V_{HST}|$ distance-$k$ broadcasts are needed. Leader election is used to choose $r$ and, if we wish to finish any leader election process in $O(diam_G)$ time units [1], it requires $O(m \times \log diam_G)$ messages.

The third approach to partitioning $G$ into zones is by growing a spanning forest in $G$. Each tree in the spanning forest corresponds to a partition. The size of each partition is roughly controlled by a parameter $x$. The GHS algorithm [28] is an asynchronous distributed algorithm for finding an MST (minimum spanning tree). The GHS algorithm grows components of the MST in parallel until all the components connect to form the MST. Initially, each vertex is a singleton tree by itself. In this partitioning approach, every vertex runs the GHS algorithm but terminates immediately after the root of each tree finds that its size exceeds a given threshold $x$. By doing this, we guarantee that the size of each tree is at most $2x$. The time and message costs for the GHS algorithm for finding an MST are $O(n \log n)$ and $O(m + n \log n)$ respectively. Thus, the time complexity of this approach, which is that used to construct a tree, is $O(x \log x)$. The message cost for each tree is no more than $O(m_i + x \log x)$, where $m_i$ is the number of edges incident to this tree, so the

| | Distance-$k$ DS | HST | | Spanning Forest |
|---|---|---|---|---|
| | | Partition | Leader election | |
| Time cost | $O(diam_G)$ | $O(n)$ | $O(diam_G)$ | $O(x \log x)$ |
| Message cost | $O(n(\Delta-1)^k)$ or $O(nm)$ | $O(\frac{n}{k}(\Delta-1)^k)$ or $O(\frac{n}{k}m)$ | $O(m \log diam_G)$ | $O(m + n \log x)$ |

Table 3.1: Costs of different graph partitioning methods.

total message cost is $O(m + n \log x)$. Note that $x$ is a constant indicating the approximate size of the partitions.

Table 3.1 shows a comparison of the three different approaches. We can see that the time costs of the first two approaches have global parameters but the third approach can be finished in a short time with the controllable parameter $x$ set to small values. Therefore, we utilize the third approach to partition a graph into non-overlapping zones in a non-global distributed fashion.

### 3.2.2 Graph partitioning using minimum spanning forests

In this section, we concentrate on the first phase of our zonal distributed clustering algorithm — the process of partitioning a given graph $G = (V, E)$ into non-overlapping zones by growing a spanning forest of the graph. At the end of this phase, the subgraph induced by each tree defines a zone.

Our algorithm for this phase is based on an algorithm of Gallager, Humblet, and Spira [28] that is based on Kruskal's classic centralized algorithm for Minimum Spanning Tree (MST), an iterative greedy algorithm. We assume that all edge weights are distinct, breaking ties using the vertex ID's of the endpoints. The MST is unique for a given graph with distinct edge weights. The algorithm maintains a spanning forest. Initially, the spanning forest is a collection of trees of single vertices. At each step the algorithm merges two trees by including an edge in the spanning forest. During the process of the algorithm, an edge can be in any of the three states: tree edge, rejected edge, or candidate edge. All edges are candidate edges at the beginning of the algorithm. When an edge is included in the spanning forest, it becomes a tree edge. If the addition of a particular edge would create a cycle in the spanning forest, the edge is called a rejected edge and will not be considered further. In each iteration, the algorithm looks for the candidate edge with minimum weight, and changes it to a tree edge merging two trees into one. During the algorithm, the tree

edges and all the vertices form a spanning forest. The algorithm terminates when the forest becomes a single spanning tree. This greedy algorithm finds the optimal solution, i.e., the MST.

Gallager, Humblet, and Spira [28] presented an asynchronous distributed algorithm, called the GHS algorithm, for finding the MST in a graph. In their algorithm, vertices in the graph grow their own trees in the spanning forest in parallel. Each tree in the spanning forest is called a *component.* Initially, every vertex is a component by itself. The central idea of the GHS algorithm is that each component locates its minimum weight outgoing edge, an edge of minimum weight which connects the component to some other component. Each vertex PROBEs each of its neighbors to see if the edge connecting them is an outgoing edge. The root of a component collects the weights of all outgoing edges to locate the minimum weight outgoing edge, and attempts to merge using the minimum weight outgoing edge. Some restrictions are applied during the merge process to guarantee balanced growth among components. The algorithm terminates when no outgoing edge can be found for any component, at which point the MST has been found.

Our partitioning process consists of a partial execution of the GHS algorithm which terminates before the MST is fully formed. We control the size of components by picking a value $x$. Once a component has exceeded size $x$, it no longer participates. In particular, it will not send or accept PROBEs. We call such a component *full.* Since every merge joins two components of size of no more than $x$, the size of a full component is at most $2x$. Note that a component that has not exceeded the threshold size by the time that all of the surrounding components become full may be of size $x$ or less. To implement this size control, the root of each component keeps track of the number of vertices in its component. When the size $x$ is exceeded, the root broadcasts a FULL message to all vertices of the component. Having received the FULL message, a vertex refuses the PROBEs from any neighbor.

The threshold size $x$ provides a mechanism to control the trade-off between the locality of the algorithm and size of the output weakly-connected dominating sets. When $x$ is small, the algorithm is fairly localized and its behavior is similar to that of the localized algorithms for connected dominating sets of Wu and Li [73] and of Alzoubi, et al. [4]. When $x$ is large, the algorithm is not as localized but tends to produce smaller weakly-connected dominating sets, as we will see in Section 5. The selection of the value of $x$ is a problem of interest. The threshold size $x$ can be preset to a particular value before the network is setup, if the

scale and behavior of the network is relatively predictable. In particular, the value of $x$ may be determined by the size of the network, the density of the nodes, and the locality requirements of the administrator. Experiments can be used to fine tune the value given the above parameters.

In Chapter 4, we develop algorithms for maintaining the cluster structure as the network changes. In this case, the value of $x$ will also need to be changed dynamically. However, this is beyond the scope of this chapter.

Note that in the centralized greedy algorithm and the original GHS algorithm for MST, the spanning forest maintained at any time is part of some MST. This condition does not necessarily hold after adding the size control, since a component only considers edges leading to non-full components when locating the minimum weight outgoing edge. But this modification is acceptable as our goal is simply to partition a graph into zones using some spanning forest with the size control. We do not strictly require a minimum spanning tree.

To calculate the costs of the partitioning phase, let $m_i$ denote the number of edges incident on the vertices within component $i$. The time complexity to form this component using the modified GHS algorithm is $O(x \log x)$ and the message cost is $O(m_i + x \log x)$. (See the analysis of Gallager, Humblet and Spira [28].) Therefore, the total time required is the same as that of each individual component, $O(x \log x)$, and the message cost is the sum over all the components, $O(m + n \log x)$. The threshold number $x$ is a variable used to control the approximate size of each partition. As $x$ increases, the partitions become larger which, intuitively, allows a smaller weakly-connected dominating set, as we can run the asymptotically optimal approximation algorithm (as we will see in next section) within larger partitions of the graph.

In principal, any distributed algorithm for constructing spanning trees can be modified for use here. Among algorithms for minimum spanning trees, the GHS algorithm does not have optimal time or message costs. However, the GHS algorithm is localized during the early stages of its execution and thus is easy to extend to our problem. Further, using a minimum spanning tree algorithm with edge weights obtained from subscriber unit distance (rather than an arbitrary spanning tree algorithm) tends to give us spanning trees of smaller heights and thus zones of smaller diameters.

## 3.3 Computing weakly-connected dominating sets of the zones

Once the graph $G$ is partitioned into zones and a spanning tree has been determined for each zone, we run the $DGW_S$ algorithm (Chapter 2) within each zone. This algorithm is the distributed implementation of $GW_S$, a centralized greedy algorithm for finding small weakly-connected dominating sets in graphs.

The $DGW_S$ algorithm generates the same weakly-connected dominating set as its centralized counterpart on any input graph. Without leader election, it can be done in $O(diam_G \times |S|)$ time with $O(n \times |S| + m)$ messages in a given graph $G = (V, E)$ (Table 2.2), where $|V| = n$ and $|E| = m$.

This algorithm can be run in any zone $Z_i = (V_i, E_i)$ in time $O(diam_{Z_i} \times |S_i|)$, using $O(x \times |S_i| + m_i)$ messages, where $|V_i| = n_i$, $|E_i| = m_i$, and $S_i$ is the weakly-connected dominating set produced by $Z_i$. Therefore, within the entire graph the total time is $O(x \times |S_{max}|)$ and the message cost is $O(n \times |S_{max}| + m)$, where $S_{max}$ is the largest weakly-connected dominating set produced by all zones.

## 3.4 Fixing the Borders

After we have calculated a small weakly-connected dominating set $S_i$ for each zone $Z_i$ of $G$, combining these solutions does not necessarily give us a weakly-connected dominating set of $G$. We will likely need to include some additional vertices from the borders of the zones in order to obtain a weakly-connected dominating set of $G$.

The edges of $G$ are either dominated (that is, they have either endpoint in some dominating set $S_i$) or free (in which case neither endpoint is in a dominating set). Two zones $Z_i$ and $Z_j$ adjacent along a dominated edge can comprise a single zone with dominating set $S_i \cup S_j$, and do not need to have their shared border fixed.

The root of zone $Z$ can learn, by polling all the vertices in its zone, which zones are adjacent and can determine which neighboring zones are not adjacent along a dominated edge. For each such pair of adjacent zones, one of the zones must "fix the border". To break ties, we assume that the zone with lower zone ID takes control of this process, where the zone ID is the vertex ID of the zone root. In other words, if neighboring zones $Z_i$ and $Z_j$ are not adjacent along a shared dominated edge, the zone with the lower subscript adds a new vertex from the $Z_i/Z_j$ border into the dominating set.

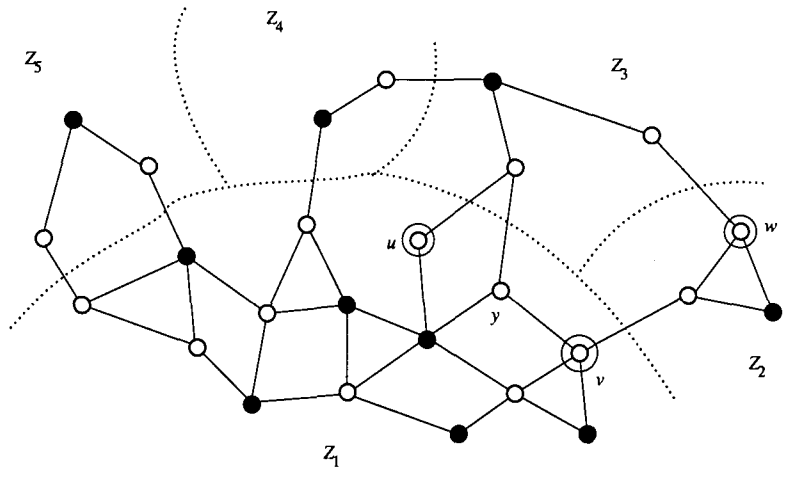



Figure 3.2: Fixing the borders.

For example, in Figure 3.2, the five zones have weakly-connected dominating sets indicated by the solid black vertices. Zone $Z_1$ is adjacent to zones $Z_2$, $Z_3$, $Z_4$, and $Z_5$. Among these, zones $Z_2$ and $Z_3$ do not share dominated edges with $Z_1$. As $Z_1$ has a lower zone ID than either $Z_2$ or $Z_3$, $Z_1$ is responsible for fixing these borders. The root of $Z_1$ adds $u$ and $v$ into the dominating set. $Z_2$ is adjacent to two zones, $Z_1$ and $Z_3$, but it is only responsible for fixing the $Z_2/Z_3$ border, due to the zone ID's. The root of $Z_2$ adds $w$ to the dominating set.

A detailed description of this process for a given zone $Z$ follows. The goal is for the root $r$ to find a small number of dominated vertices within $Z$ to add to the dominating set.

Assume that every vertex knows the vertex ID, color, and zone ID of all of its neighbors. (This can be done with a single round of information exchange.) Root $r$ collects the above neighborhood information from all of the border vertices of $Z$. We define a *problem zone* with regard to $Z$ to be any zone $Z'$ that is adjacent to $Z$, is not joined along a dominated edge, and has a higher zone ID than $Z$. Zone $Z$ is responsible for fixing its border with each problem zone.

We construct a bipartite graph $B(X, Y, E)$ using the collected information for root $r$. Vertex set $X$ contains a vertex for each problem zone with regard to $Z$, and vertex set $Y$ contains a vertex for every border vertex (a vertex adjacent to vertices of other zones) in $Z$. There is an edge between vertices $y_i$ and $x_j$ if and only if $y_i$ is adjacent to a vertex in problem

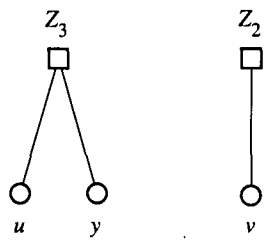



Figure 3.3: Bipartite graph for the problem of fixing a border.

zone $x_i$ in the original graph. Figure 3.3 shows the bipartite graph constructed by zone $Z_1$ in the example of Figure 3.2. In this bipartite graph, $X = \{Z_2, Z_3\}$ and $Y = \{u, y, v\}$. In this case, $\{u, v\}$ is a possible solution for $Z_1$ to add to the weakly-connected dominating set in order to fix its borders with $Z_2$ and $Z_3$. To find the smallest possible set of vertices to add to the dominating set, $r$ must find a minimum size subset of $Y$ to dominate $X$.

The decision version of this problem is equivalent to the dominating set problem in split graphs which is NP-complete (see Golumbic [33] for definition of split graph and Bertossi [16] for the NP-completeness result). As an optimal solution is not likely, we use a greedy procedure to find an approximate solution. In each iteration of this greedy procedure, we remove a maximum degree vertex from $Y$. Since the calculation for each zone is done by the zone root, the time and message costs are those incurred by the polling process, i.e. a broadcast and a convergecast along the tree edges of the spanning tree of each zone. Both costs can be bounded by $O(x)$ since the size of zones are within $O(x)$. Therefore, the time and message costs for the entire graph are $O(x)$ and $O(n)$, respectively.

After the root of each zone has executed the above algorithm, the resulting dominating set is a weakly-connected dominating set of the entire graph $G$.

## 3.5 Simulation

At the point, we have not been able to obtain a theoretical bound on the size of the weakly-connected dominating set produced by the ZW algorithm. Therefore we have performed simulations to determine the quality of this zonal distributed algorithm for finding small weakly-connected dominating sets, measuring the dominating set size, the path length, and the network capacity. We compare this algorithm to the $DGW_S$ algorithm of Chapter 2 and

to Das and Bharghavan's algorithm for connected dominating sets [24].

To measure the performance of our zonal algorithm, we consider three parameters. First we consider the size of the dominating set produced. Since our goal is to find a small weakly-connected dominating set, we prefer smaller values for this parameter. The second parameter that we are interested in is the average path length, that is, the average pairwise vertex distance. In each of the clustered graphs, we compute these distances using only the edges in the subgraph weakly induced by the dominating set. Small increases of this parameter are expected, as some edges in the clustered graph are removed. The third parameter is the average number of edge-disjoint paths between pairs of vertices. This parameter measures, in some sense, the capacity of the network. As we delete edges, we expect that this value will decrease, but we do not wish it to decrease too much.

Originally we planned to include the localized algorithms of Wu and Li [73] and of Alzoubi, et al. [4] in our comparison, but our preliminary experiments showed that for our data set these localized algorithms produced much larger dominating sets than the non-zonal distributed algorithm by Das and Bharghavan [24]. Therefore, in this chapter we focus on comparing the ZW algorithm with the non-localized distributed $DGW_S$ algorithm presented in Chapter 2 and that by Das and Bharghavan [24].

In our experiment we generate random graphs repeatedly and run the ZW algorithm, the $DGW_S$ algorithm, and the connected dominating set algorithm [24], measuring the above three parameters. The random graphs have expected average degree of 6 and 12, providing two levels of density. The size of the graphs ranges from 20 to 200 vertices and 40 to 200 vertices for the two densities, respectively. To simulate the structure of ad hoc networks, we place vertices (subscriber units) randomly in a rectangular area in a 2D-plane. The coordinates of the vertices are chosen uniformly in each dimension. We assign each subscriber unit a random transmission range with a pre-calculated expected value. When two subscriber units are placed within range of each other, an edge is added between the vertices simulating a reliable link between them. By changing the number of vertices in the plane and the expected transmission range, one can adjust the network size and density.

For each pair of number of vertices and expected degree, we generated 100 random graphs to test the algorithms. In order to investigate the influence of the size of the zones on the performance of the algorithm, we configured the experiment to run with parameter $x$ set to the values 10, 20, 40, 80 and 160. Given the sizes of the graphs in our simulations (up to 200), values 20 and 80 represent two typical locality levels, and these are the results

presented in this chapter. Intuitively, when $x = 20$, the algorithm behaves more locally. In particular, with average degrees either 6 or 12, we could expect the zone diameter to be 4 or less. Raising the threshold to 80, we would expect the zone diameter to be closer to 6. As the zone diameter increases, the locality of the algorithm decreases and it begins to behave more like the $DGW_S$ algorithm.

Figures 3.4 and 3.5 show the size of the weakly-connected dominating sets produced by the ZW algorithm with the zone size control parameter $x$ set to 20 and 80 (`ZW_20` and `ZW_80`, respectively) and by the $DGW_S$ algorithm (`DGW_S`), and the size of the connected dominating sets produced by the connected dominating set algorithm, Algorithm II of Das and Bharghavan [24] (`CDS`). Connected dominating sets are generally denser than weakly-connected dominating sets for the same graph, so we expect the size of connected dominating sets produced by the CDS algorithm to be larger than the others. In Figure 3.4, where the expected degree is 6, we can see that when the zones contains approximately 80 vertices (`ZW_80`) the size of the weakly-connected dominating set produced by the ZW algorithm is just slightly larger than that produced by the $DGW_S$ algorithm (`DGW_S`). For smaller zones (`ZW_20`), the size of the dominating set is again larger, but still smaller than the size of the connected dominating sets (`CDS`). In Figure 3.5 the expected degree is 12. The difference from Figure 3.4 is that when using small zones (`ZW_20`), the size of the weakly-connected dominating sets is larger than that of the connected dominating sets (`CDS`), though `ZW_80` is still smaller than `CDS`. In this case, we believe that the graph is fragmented by small zones because the threshold size is not much larger than average degree.

Figures 3.6 and 3.7 display the average path lengths in the original graph (`Original`), those in the graph clustered by weakly-connected dominating sets produced by the ZW algorithm (`ZW_20` and `ZW_80`), those in the graph clustered by weakly-connected dominating sets produced by the non-localized algorithm (`DGW_S`), and those in the graph clustered by the connected dominating sets produced by the CDS algorithm (`CDS`). In both Figures 3.6 and 3.7, we can see that the zonal algorithm (`ZW_20` and `ZW_80`) yields a smaller average path length than the distributed greedy algorithm (`DGW_S`). In addition, the zonal algorithm with small zones (`ZW_20`) yields smaller average path length than the connected dominating set algorithm of Das and Bharghavan (`CDS`).

Figures 3.8 and 3.9 show the average number of edge-disjoint paths in the original graph (`Original`), those in the graph clustered by weakly-connected dominating sets produced by the ZW algorithm (`ZW_20` and `ZW_80`), those in the graph clustered by weakly-connected

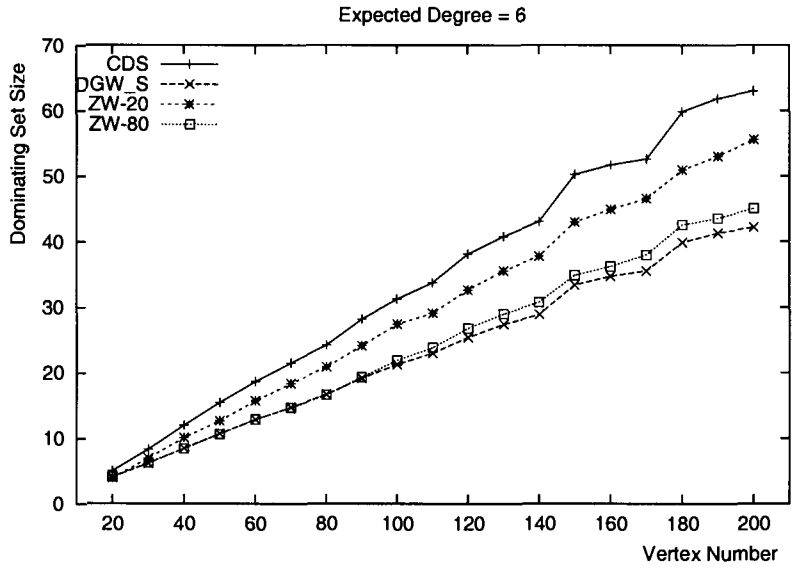



Figure 3.4: Dominating set size (a).

dominating sets produced by the non-localized algorithm (DGW_S), and those in the graph clustered by the connected dominating sets produced by the CDS algorithm (CDS). In both Figures 3.8 and 3.9, we can see that the zonal algorithm (ZW_20 and ZW_80) generates a larger average number of edge-disjoint paths than the distributed greedy algorithm (DGW_S). Again counter-intuitively, for small zones (ZW_20), the zonal algorithm generates a larger number of edge-disjoint paths than the connected dominating set algorithm (CDS) does.

## 3.6 Conclusion

We presented a zonal distributed algorithm, ZW, for finding small weakly-connected dominating sets for use in clustering MANETs. The algorithm consists of three phases: partitioning, executing $\text{DGW}_\text{S}$ within zones, and border fixing. We first partition the input graph into zones of sizes approximately $x$ using a modified GHS algorithm for constructing spanning forests. Then the distributed algorithm for weakly-connected dominating sets is executed in each zone. In the last phase, some additional vertices from zone borders are added in order to obtain a weakly-connected dominating set of the graph. The execution time of this algorithm is $O(x \times (\log x + |S_{max}|))$ and it uses $O(m + n \times (\log x + |S_{max}|))$ messages, where

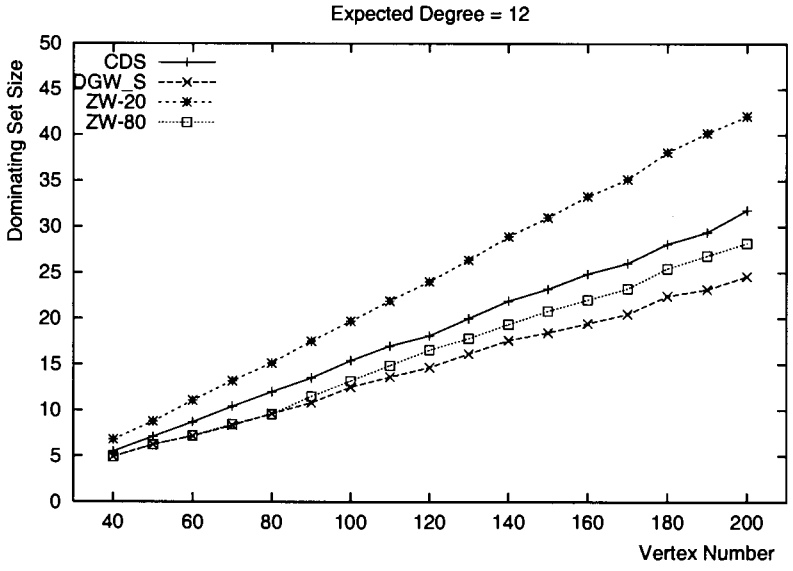



Figure 3.5: Dominating set size (b).

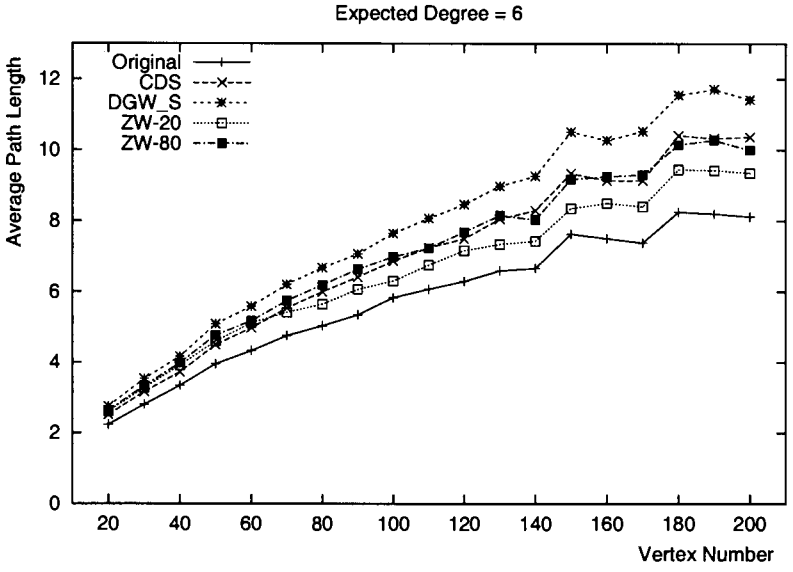



Figure 3.6: Average path length (a).

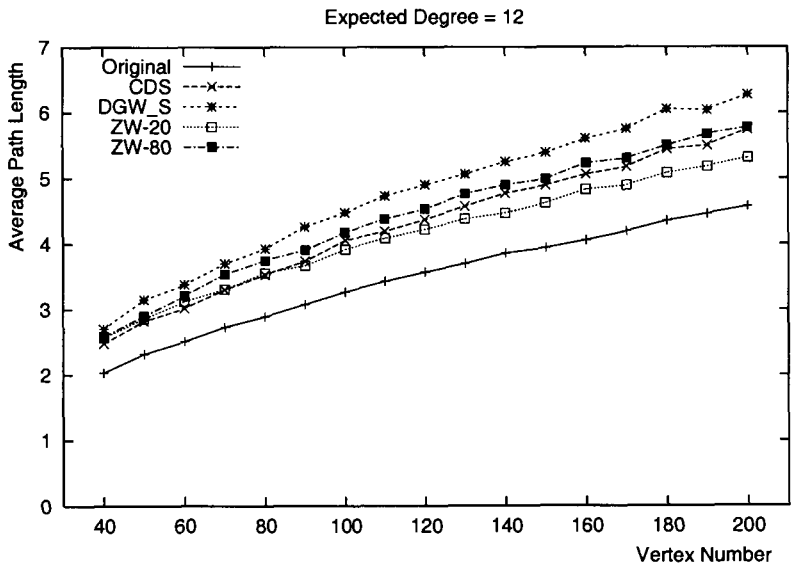



Figure 3.7: Average path length (b).

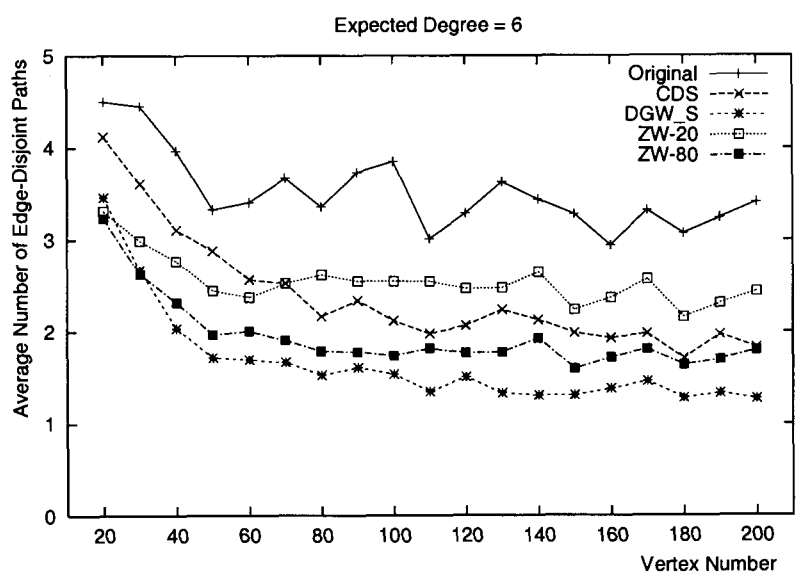



Figure 3.8: Average number of edge-disjoint paths (a).

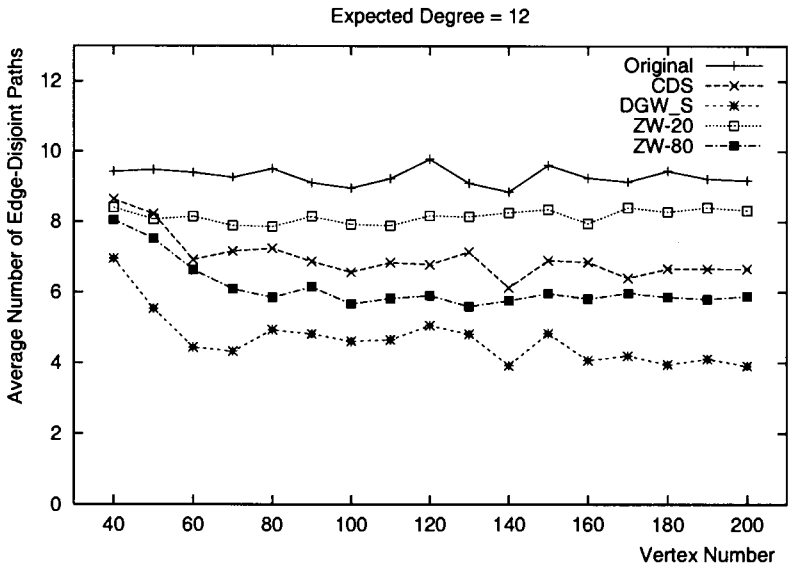



Figure 3.9: Average number of edge-disjoint paths (b).

$S_{max}$ is the largest weakly-connected dominating set generated by all zones and is of size at most $O(x)$. This zonal algorithm is regulated by a single parameter $x$, which controls the size of zones. When $x$ is small, the algorithm finishes quickly with a large weakly-connected dominating set. When it is large, it behaves more like the $\mathrm{DGW_S}$ algorithm and generates smaller weakly-connected dominating sets. Experiments were designed to test the quality of this zonal algorithm. Comparisons are made to the $\mathrm{DGW_S}$ algorithm for weakly-connected dominating sets and the algorithm for finding small connected dominating sets.

# Chapter 4

# Maintenance

The algorithms for constructing weakly-connected dominating sets presented in the previous chapters can only be used in static ad hoc networks, such as sensor networks. In this chapter, we present an algorithm that updates the WCDS structure in presence of topology changes.

Using the zonal idea, cluster maintenance can be logically divided into two layers — intrazonal (within zones) and interzonal (along zone borders). As usual, we are given an undirected graph $G = (V, E)$ to represent the network and $S \subseteq V$ is the weakly-connected dominating set to be maintained. In the figures, solid black vertices represent dominating vertices and white vertices represent dominated ones.

We assume that the underlying graph always remains connected despite the topology changes. We also assume that each vertex has a unique ID, such as the hardware address of the device. For the sake of simplicity, we assume that all the changes occur sequentially and that we have sufficient time to fix one change before the next begins. In the actual implementation of the protocol, various timeout mechanisms are used to avoid infinite waiting from simultaneous topology changes (see Subsection 4.2.1 for an example).

In Section 4.1, we present a non-zonal update algorithm that can be used to maintain the weakly-connected dominating set structure resulting from our static algorithms in Chapter 2. We classify the possible topological changes into four primitives: edge-down, vertex-down, edge-up, and vertex-up. In Section 4.2, we describe a zonal maintenance algorithm utilizing the algorithm of Section 4.1. Here, we restrict the execution of the earlier algorithm to zones to accomplish intrazonal maintenance and add interzonal maintenance procedures to maintain the weakly-connected dominating set of the entire network. We present experimental settings and results to test the performance and efficiency of the maintenance algorithm in

Section 4.3. The results show that both the size of the weakly-connected dominating set maintained and the connectivity of the abstracted network stabilize after a short simulation time and these values remain approximately the same as initially. Furthermore, the different zone sizes also show different emphases between algorithm locality and dominating set size.

## 4.1 Non-Zonal Maintenance

In this section, we present a non-zonal algorithm to maintain existing weakly-connected dominating sets in arbitrary graphs. The algorithm can easily be adapted to a zonal algorithm for intrazonal maintenance by restricting the execution to a single zone. The algorithm uses message flooding along dominated edges to propagate the control messages. We classify the possible topological changes into four primitives: edge-down (the loss of an edge), vertex-down (the loss of a vertex), edge-up (the addition of an edge), and vertex-up (the addition of a vertex).

### 4.1.1 Edge-down

We first deal with the loss of an edge. We have assumed that the root is known to every other vertex. The root is responsible for *fixing* the weakly-connected dominating set if the loss of an edge breaks the piece into *fragments*. Before an edge-down event, the graph contains a single black piece because it has a weakly-connected dominating set. The loss of a free edge (such as edge $(w, v)$ in Figure 4.2) does not change the subgraph weakly induced by the dominating set and the maintenance procedure need not be triggered. The loss of a dominated edge, however, must be reported by the vertices incident upon it, because the piece may have been broken into two fragments. In particular, when a vertex loses an incident dominated edge, the root vertex will be notified and will coordinate the procedure to add a dominated vertex to the dominating set if necessary. The response to an edge-down event consists of two logical parts. In the first part, *piece integrity test*, the two vertices that have detected the edge-down event each broadcast an `edge_loss` message to determine if the piece is broken. If so, the root initiates *breach suturing* to add a dominated vertex to the dominating set of the zone.

To determine if the piece has been broken, both endpoints broadcast an `edge_loss` message along the dominated edges. The `edge_loss` message must be acknowledged by all vertices that receive it. If the piece is not broken, each of the endpoints is able to detect the

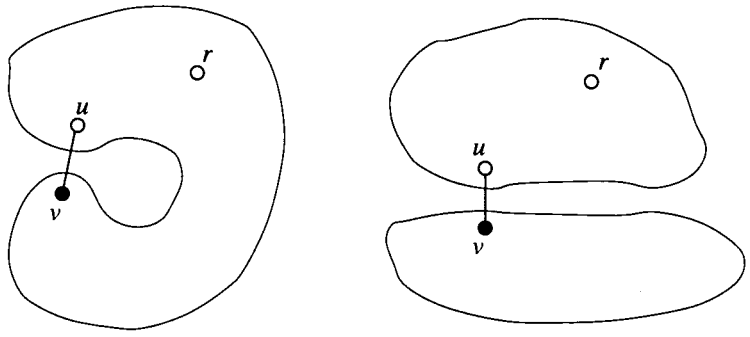



Figure 4.1: Edge down.

existence of the other. In this case, the endpoint with lower vertex ID sends a message to the root to indicate that the piece is still intact. If the endpoints do not detect the existence of the other, one of them is in the same fragment as the root and can notify the root that fixing is needed.

After the root determines that the piece needs to be rejoined, it broadcasts an `improvement_inquiry` message along the dominated edges to find a vertex that has the greatest *improvement* value. (Here, each vertex knows its improvement value, that is, the number of distinct fragments within its closed neighborhood.) The `improvement_inquiry` message carries the vertex ID of the message originator (the root, in this case) so that reachable vertices (those in the same fragment as the originator) will use that ID as the fragment ID. For an edge-down event, the maximum improvement value will be two. When such a vertex is located, the root instructs it to change to black. This results in a new weakly-connected dominating set for the entire graph.

### 4.1.2 Vertex-down

The vertex-down event handling can be regarded as a generalized edge-down event. In the implementation of the distributed algorithm, they are treated identically since a detecting vertex cannot really distinguish the loss of an edge and the loss of a neighboring vertex. The vertex-down event triggers a response consisting of a piece integrity test and breach suturing, if needed.

We define the *clustered degree* of a vertex $v$, denoted $c_{v,S}$, to be the number of neighbors of $v$ in the subgraph weakly induced by a dominating set $S$. In essence, this is the number of incident dominated edges of a vertex. If $v \in S$, $c_{v,S}$ is simply $v$'s degree in the original

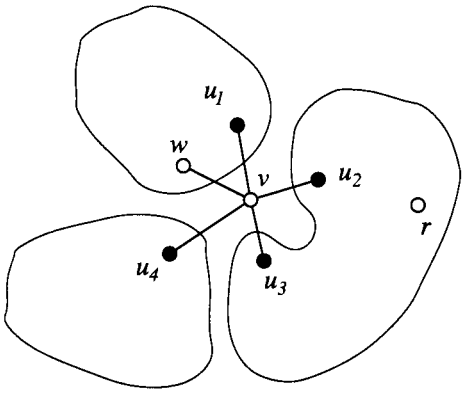



Figure 4.2: Vertex-down.

graph. When $S$ is understood from context, we write $c_v$ without confusion. For example, in Figure 4.2, $c_v = 4$. As a contrast to the edge-down event, the loss of vertex $v$ can break the piece as many as $c_{v,S}$ fragments. When detecting the loss of a neighbor $v$, each *detecting* vertex broadcasts an `edge_loss` message along the dominated edges. Again, each vertex receiving this message sends an acknowledgment. The way that a detecting vertex decides if the original piece has been broken or not is slightly different here. Each detecting vertex checks to see if there are $c_{v,S}$ detecting vertices in the fragments from the acknowledgments (rather than two as in the edge-down event). If so, the piece has not been broken and the detecting vertex with the lowest ID sends a message to the root vertex that the piece is still intact. Otherwise, the detecting vertex with the lowest ID that has been acknowledged by the root vertex sends a message to the root along the dominated edges that fixing is needed.

If the piece has been broken into two or more fragments due to the loss of vertex $v$, the root decides that more dominated vertices will be added to the dominating set. Unlike the case of the edge-down event, the root may need multiple iterations to locate these vertices. In each iteration, the root broadcasts an `improvement_inquiry` message along the dominated edges to find a vertex that has the greatest improvement value. This process stops when the maximum vertex improvement value is one, and the dominating set is weakly-connected.

The above assumes that the lost vertex is not the root. For the special case that the root is down, another vertex take on the role of the root. To achieve that, every root has a neighbor as a backup. If the loss is discovered by the backup, the backup assumes the role of the root. If the root was not lost but just the edge between the root and its backup is lost, both "roots" will try to add a vertex to the dominating set. For example,

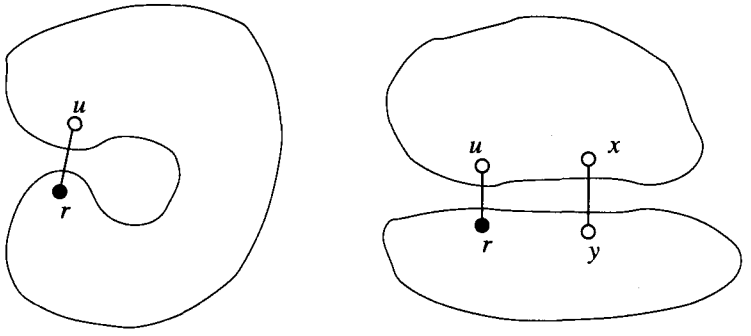



Figure 4.3: Possible scenarios for the presumed loss of the root.

the graphs in Figure 4.3 illustrate these situations, where vertex $r$ is the root and $u$ is its backup in each graph. If the root and its backup are in the same fragment, as in the left graph of Figure 4.3, vertex $u$ will learn that the original root $r$ is still alive during the piece integrity test, and thus, will stop acting as a root. If the two vertices in different fragments, as in the right graph of Figure 4.3, either of vertices $u$ and $r$ will broadcast the `improvement_inquiry` message in its fragment. However, when a vertex $x$ in $u$'s fragment that is also adjacent to a vertex $y$ in $r$'s fragment detects that $r$'s ID is used as fragment ID of $y$, then $x$ learns that the old root $r$ is still alive. Vertex $x$ will then inform $u$ using the returning `improvement_inquiry` message so that $u$ will stop acting as a root. Root $r$ will continue to join these two fragments.

### 4.1.3 Edge-up

When a new edge is inserted due to, for example, the endpoints moving closer, the weak connectivity of the dominating set will not be affected. However, adding an edge may cause some vertices in the dominating set to be redundant. That is, we can remove these vertices from the dominating set and still have a weakly-connected dominating set. Thus, we focus on localized procedures for eliminating redundancies caused by the edge-up event.

In event of edge-up, we only consider the case of adding a dominated edge. The addition of a free edge does not change the neighborhood of any dominating vertex and thus does not cause any redundancy. For example, in both graphs of Figure 4.4, when a dominated edge (indicated as dashed lines) is added, vertex 1 becomes redundant because its closed neighborhood is a subset of that of the neighboring dominating vertex 2. Thus, vertex 1 can be removed from the dominating set. A vertex $v$ in the dominating set may also become

redundant if its closed neighborhood is a subset of the union of the closed neighborhoods of $v$'s dominating neighbors. The addition of an edge can also cause this to occur. For example, in each of the two graphs in Figure 4.5, vertex 1 becomes redundant when the respective indicated edge is added.

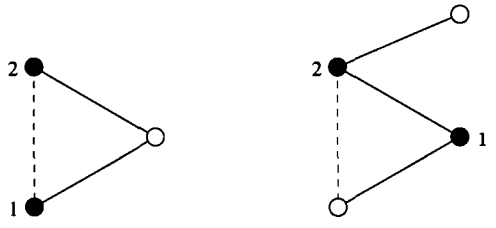



Figure 4.4: Edge-up — single coverage.

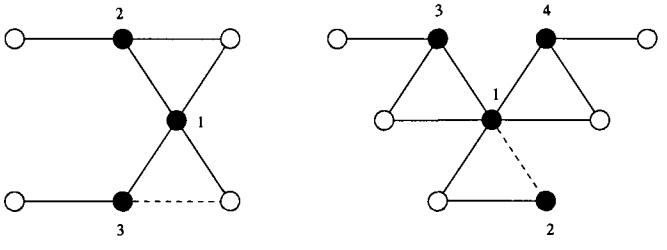



Figure 4.5: Edge-up — combined coverage.

We say that a dominating vertex $v$ is *redundant* if it has a set $T$ of dominating neighbors, such that:

1. $N[v] \subseteq N[T]$, and
2. $id(v) < id(u)$, for $\forall u \in T$.

Note that the weak connectivity of the dominating set is not violated with the removal of redundant vertices. The second requirement is used to avoid simultaneous abdications of multiple dominating vertices when they have identical neighborhoods.

The edge-up message is sent to "nearby" dominating vertices to determine if there is redundancy. Only the endpoints of the new edge and their immediate neighbors need to consider redundancy. The necessity can be observed from the previous examples. For sufficiency, we see that a vertex at distance two or more from the new edge does not have any neighbor whose neighborhood is changed and thus there is no need to calculate redundancies. Therefore, when a vertex has observes an edge-up event, it only needs to notify its immediate

neighbors. Assuming that every vertex knows the members of its neighborhood, a single round of information exchange among neighbors will suffice to determine redundancy.

### 4.1.4 Vertex-up

The handling of the vertex-up event is very simple. When a new vertex is added, if it has a neighbor in the dominating set, it is dominated; otherwise, it changes its status to be dominating. Note that no dominating vertices will become redundant in the latter case as the newly added dominating vertex is at least two hops away from any other dominating vertex. Therefore, no redundancy calculation is needed in handling the vertex-up event.

## 4.2 Zonal Maintenance

In order to maintain the weakly-connected dominating set structure in a more localized fashion, we divide the maintenance process into two levels, i.e. intra-zonal and inter-zonal. Intra-zonal maintenance can be accomplished by restricting the execution of the above algorithm within each zone, assuming that each zone's algorithm only takes care of vertices within the zone.

### 4.2.1 Intra-zonal Maintenance

Occasionally, vertices with the same zone ID may form a disconnected induced subgraph even though the whole network is connected. Thus, we modify the non-zonal maintenance process so that connected components of the induced subgraph that do not contain the zone root can automatically generate a root and form a new zone. To do that, the detecting vertex with the lowest ID of each fragment waits to be annexed after sending the `piece_ready` message. If nothing happens within a *time-out interval* chosen to be proportional to the zone granularity threshold $x$, these timed-out vertices of the connected component elect a root simply by broadcasting their ID's within the connected component. On the other hand, when two neighboring zones are both smaller than the threshold size, they merge with each other to maintain relatively stable zone sizes.

### 4.2.2 Inter-zonal Maintenance

Recall that in the zonal construction algorithm a zone deals only with some neighboring zones, called *problem zones*, and this suffices to generate a weakly-connected dominating set of the entire network. As in the non-zonal scenario, when a border vertex (a vertex adjacent to vertices of other zone) $v$ loses an edge to a foreign vertex (a vertex of other zones) $u$, it can not distinguish whether it was an edge-down event or vertex-down event. Thus, we treat them identically.

From $v$'s perspective, if it has a zone ID higher than $u$ does, it does nothing. Otherwise, $v$ will first check to see if it still has another dominated edge incident to $u$'s zone. If so, it does nothing. Otherwise, it sends a `zone_loss` message to its zone root $r$. Upon receiving this message, $r$ will modify its bipartite graph for the border fix procedure and will add a dominated vertex to the dominating set of the zone if needed.

Note that if a border vertex is lost, its zone root $r$ must take special care in addition to the usual intrazonal maintenance. In particular, if the lost border vertex was in the bipartite graph for border fix maintained by $r$, then it is removed from the bipartite graph. As a consequence, some other vertices may be added to the dominating set by $r$ to fix the broken border. See Figure 4.6 for an example of updating the bipartite graph for a zone rooted at $r$. Before vertex $y$ is lost, $y$ and $v$ were added to fix the borders with zones $Z_1$ and $Z_2$. After $y$ is removed, $u$ is added to the dominating set.

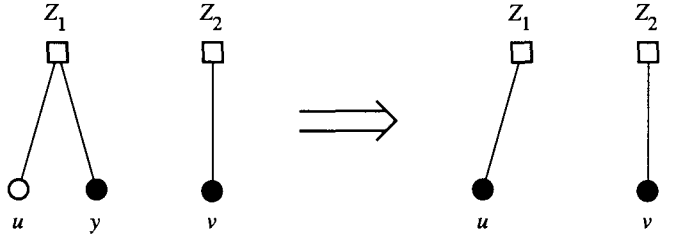



Figure 4.6: Updating the bipartite graph.

In the edge-up or vertex-up event, when a vertex $v$ detects a new edge to a problem zone, $v$ does nothing except to notify its zone root so that it can update the bipartite graph for future border maintenance.

### 4.2.3 Complexity Analysis

We assume that zone $Z$ has $O(x)$ vertices and maintains a weakly-connected dominating set $S_Z$. Also, we assume that the maximum vertex degree (number of neighbors a vertex has) is $\Delta$. All complexity analysis is done from the perspective of zone $Z$.

We first analyze the time and communication complexities of handling the edge-down event. In either part of the process, each broadcast message takes $O(|S_Z|)$ time and $O(|S_Z|\Delta)$ messages to finish, since it always propagates through dominated edges. As there are only constant number of such broadcasts involved, the time complexity of this process is $O(|S_Z|)$ and the communication complexity is $O(|S_Z|\Delta)$. To handle the vertex-down event, both parts can be more complicated than the edge-down event. In the piece integrity test, there can be as many as $\Delta$ detecting vertices, while breach suturing may take $\Delta - 1$ iterations in the worst case. Therefore, the process can take up to $O(|S_Z|\Delta)$ time and use $O(|S_Z|\Delta^2)$ messages to complete.

The most costly operation in the edge-up event handling is the redundancy calculation that is done when a dominated edge is inserted. In this case, at most $2\Delta$ vertices need to calculate redundancies, each of which exchanges the neighbor list with all its neighbors using at most $2\Delta$ messages in constant time. Thus, the total communication cost for each (dominated) edge-up event is bounded above by $4\Delta^2$ and it can be done in $O(1)$ time. The vertex-up event involves only incurs $O(\Delta)$ messages and constant time.

The most costly event at the interzonal level occurs if a border vertex in a problem zone of zone $Z$ is down. In that case, at most $\Delta - 1$ vertices in $Z$ may send the `zone_loss` message to the root of $Z$. Thus, the communication cost is $O(x\Delta)$ and time cost is $O(x\Delta)$.

Note that $|S_Z| = O(x)$, so these costs indeed increase as $x$ does. However, when $x$ is small, the algorithm is essentially localized. If, on the other hand, the network topology is expected to be less dynamic, larger $x$ can be used to obtain smaller weakly-connected dominating sets.

## 4.3 Simulation

The goals of simulating the proposed maintenance algorithm are to verify its stability and to determine how the zone size control parameter $x$ affects the size of the abstracted network and the locality of the algorithm execution. Super-vertices of the abstracted graph represent the neighborhoods of the vertices in the weakly-connected dominating set. Dominated edges

of the graph correspond to virtual edges between the super-vertices. The abstracted graph may have lower connectivity than the original network. Thus, we design simulations to show that, under some typical mobility assumptions, the algorithm keeps the size of the weakly-connected dominating set approximately the same as the beginning, does not reduce the network connectivity, and does not cause frequent cluster changes. (Note that the tests and results for static weakly-connected dominating sets and other domination variants can be found in earlier papers [21, 22].)

### 4.3.1 Settings

Our simulation is done in an 800m×600m rectangular area. The simulated network initially has 200 vertices with average transmission ranges of 100m and 200m to represent two network density levels. There is an edge between two vertices if and only if their distance is no more than the smaller of the transmission ranges. We set the zone size threshold $x$ (see Chapter 3) to be 10, 20 and 40 for different zone localities.

Vertices move according to certain mobility models. (Readers are referred to Camp et al. [18] for a survey on mobility models used for ad hoc networking research.) In an entity mobility model, mobile vertices move independently from each other. One entity mobility model is the Random Waypoint model. In this model, each vertex chooses a random destination, moves there at a randomly chosen speed, and then pauses for a random length of time before choosing the next destination. By contrast, in a group mobility model, vertices are divided into groups, the vertices within a group remain close to each other, but the groups move independently. One group mobility model is the Reference Point Group model. Each group has a moving logical center, called the *reference point*. Each vertex within the group may wander within a certain range of its reference point.

We use the Random Waypoint and the Reference Point Group mobility models, as representative entity and group mobility models, respectively. We add random vertex-down and vertex-up events to these mobility models in order to generate the four event primitives. To do that, we independently delete and/or insert a vertex at an expected period of 6 seconds. Thus, the expected vertex life span is 20 minutes and the network size remains at approximately 200 despite the dynamic nature of the network.

In our experiments, we set the pause time of the Random Waypoint model to zero. When the Reference Point Group model is used, mobile vertices are initially divided into groups of five. Group reference points move within the rectangular simulation area as entities in

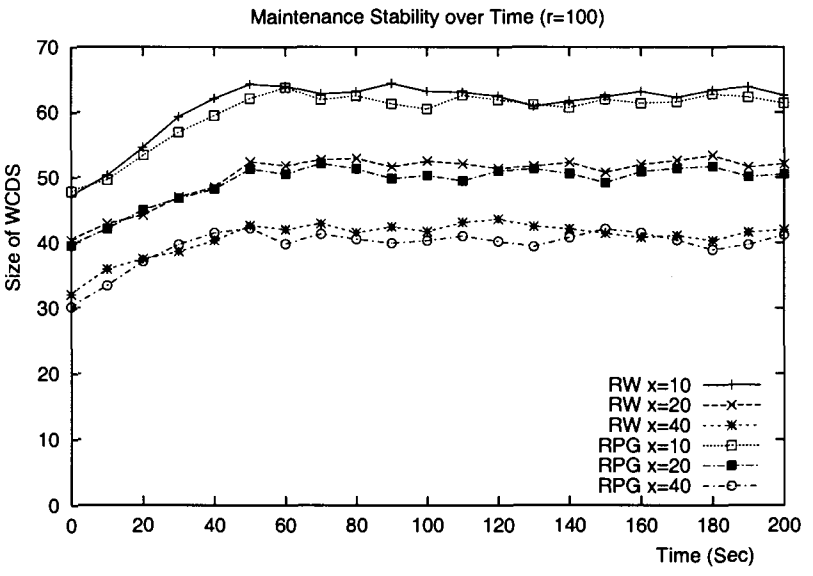



Figure 4.7: Maintenance stability (range = 100).

the Random Waypoint model. The vertices of a group are confined within a circular region of radius 100m centered at the group reference point. Their movements also follow the Random Waypoint model relative to the group reference point.

In our first experiments, we fixed the average vertex speed at 5m/s. With this setting, we determine how several measures (weakly-connected dominating set size, pairwise average vertex distance, and pairwise number of edge-disjoint paths) vary over time. We observed that these values stabilize after about 50 seconds of simulated time. In our second experiments, we considered the same measures under speed settings 1m/s, 5m/s, 10m/s, and 20m/s, and let each simulation run until the measured value stabilized. In addition, when the variable speed settings were considered, we noted the rates at which vertices change their roles as clusterheads and their cluster membership.

### 4.3.2 Results

We first consider the stability of the maintenance algorithm, that is, whether the weakly-connected dominating set remains roughly the same size over time. We record the set size under both mobility models, setting the average vertex speed and reference point speed to 5m/s, for the first 200 seconds of the simulation. The test is done with graphs of initial size of 200 vertices and average device transmission ranges of 100 and 200, respectively

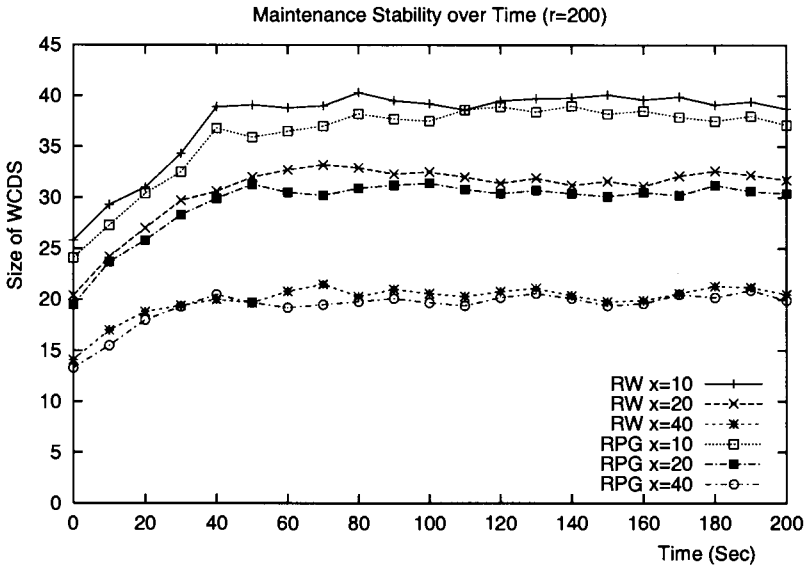



Figure 4.8: Maintenance stability (range = 200).

(Figures 4.7 and 4.8). Three different zone size control parameters, $x$ = 10, 20 and 40, are considered. As shown in the figures, during approximately the first 50 seconds of the simulation, the size of the weakly-connected dominating set slightly increases. Then the size of the dominating set stabilizes for the rest of the time. Note that the smaller the zone size control parameter $x$ is, the larger the dominating set. This shows a trade-off between locality of the algorithm execution and succinctness of the abstracted network. The idea of controllable zone sizes is a generalization of pure centralized greedy algorithms and pure localized algorithms. When larger zones and less locality are affordable, smaller abstracted networks can be generated. It is known that the network density increases to a higher level when vertices are moving than in the initial distribution [18]. We have observed this in our data. and, thus, the number of clusterheads needed is smaller. But the maintenance algorithm pays the price of larger dominating sets to avoid constructing a weakly-connected dominating set from scratch. The benefit of having a larger dominating set, though, is to have better connectivity properties as we will see. One other observation is that the maintenance algorithm behaves similarly under both mobility models. Therefore, from this point on, we only show the data generated by the Random Waypoint mobility model.

We also consider how the network connectivity of the abstracted graph (using only dominated edges) changes over time. We test this by measuring the average distance and

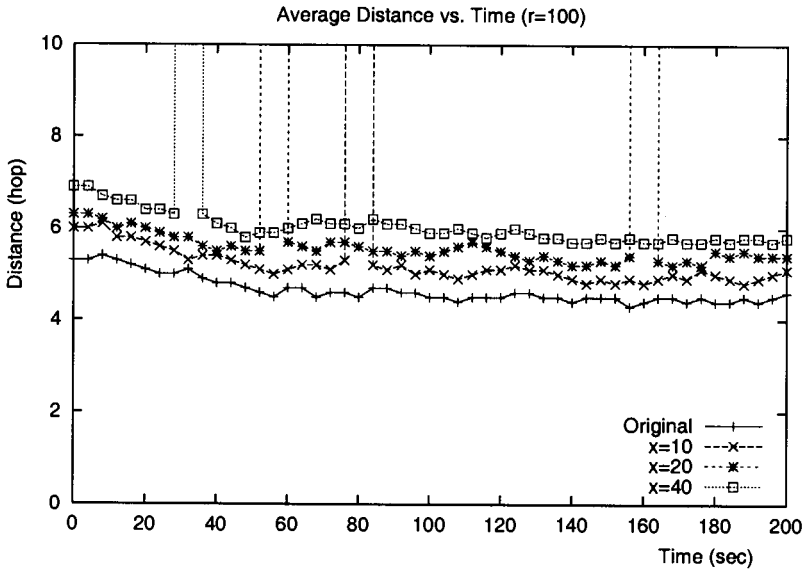



Figure 4.9: Pairwise vertex distance vs. time (range = 100).

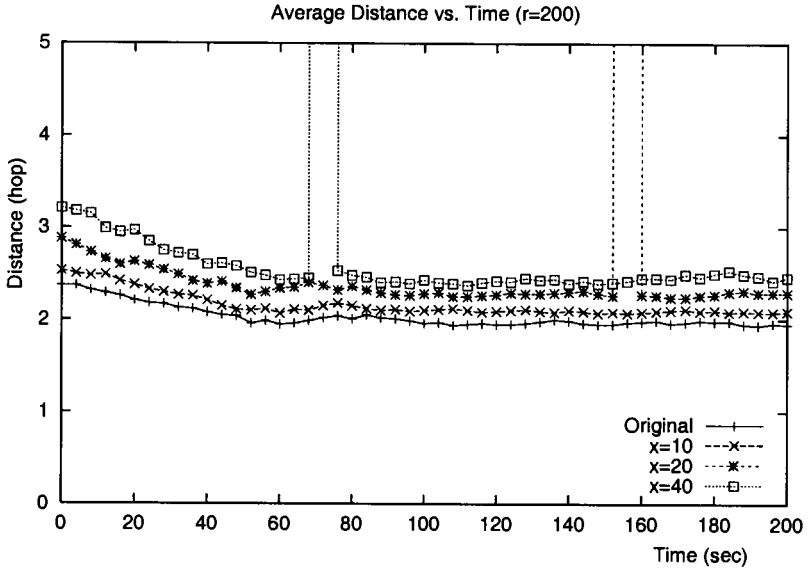



Figure 4.10: Pairwise vertex distance vs. time (range = 200).

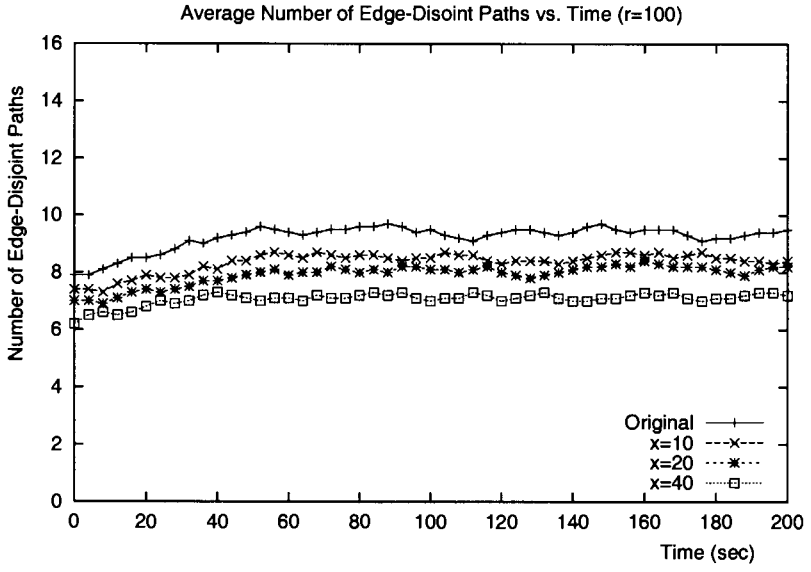



Figure 4.11: Number of edge-disjoint paths vs. time (range = 100).

average number of edge-disjoint paths between all pairs of vertices in the network under both network density levels (transmission ranges of 100 and 200). In order to have better network connectivity, it is desirable to have small average distances and large numbers of edge-disjoint paths. We compare these values to the same values for the original network. Figures 4.9 and 4.10 show that, for both network density levels and all zone size control parameters, the average pairwise vertex distance decreases first and then stabilizes as the network density does. The spikes on the curves in Figures 4.9 and 4.10 are due to the delay of the algorithm in rejoining the weakly-connected components after the abstracted graph has become disconnected. Because our algorithm reacts to topology changes in the network, these rare disconnections are inevitable. Figures 4.11 and 4.12 show that the average number of edge-disjoint paths increases slightly at the beginning and then remains roughly stable as the network density stabilizes.

To determine how the average vertex speed affects the weakly-connected dominating set size and the connectivity of the abstracted network, we record the same measures after the network stabilizes, for both network density levels with average vertex moving speeds of 1, 5, 10, and 20m/s. From Figures 4.13, 4.14 and 4.15, we can see that the weakly-connected dominating set size increases slightly and the network connectivity decreases slightly as the average vertex speed increases as was expected.

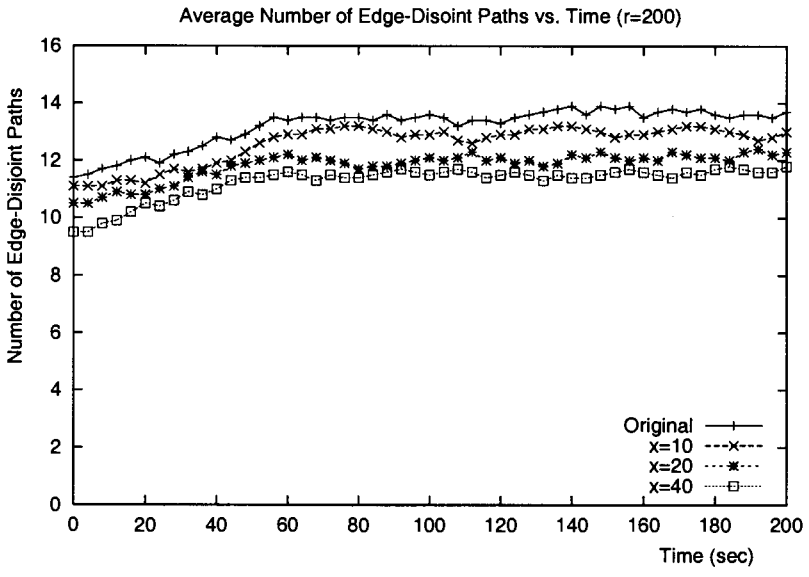



Figure 4.12: Number of edge-disjoint paths vs. time (range = 200).

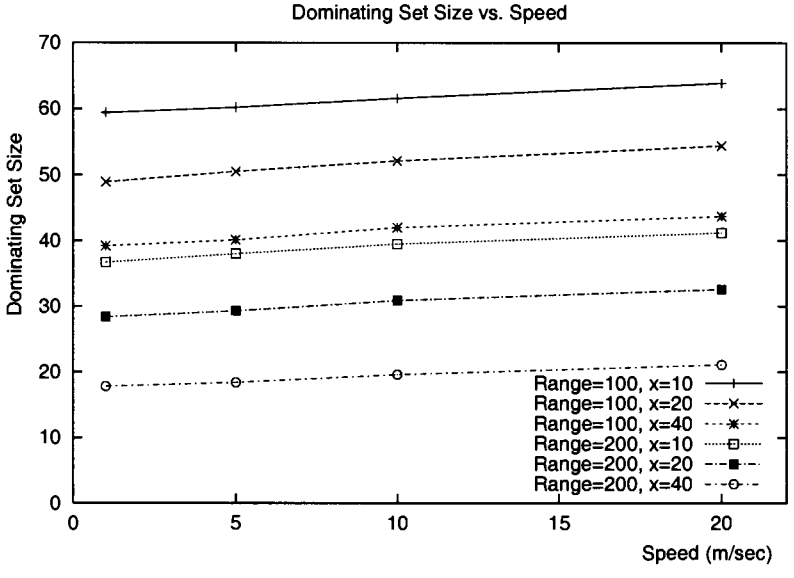



Figure 4.13: Dominating set size vs speed.

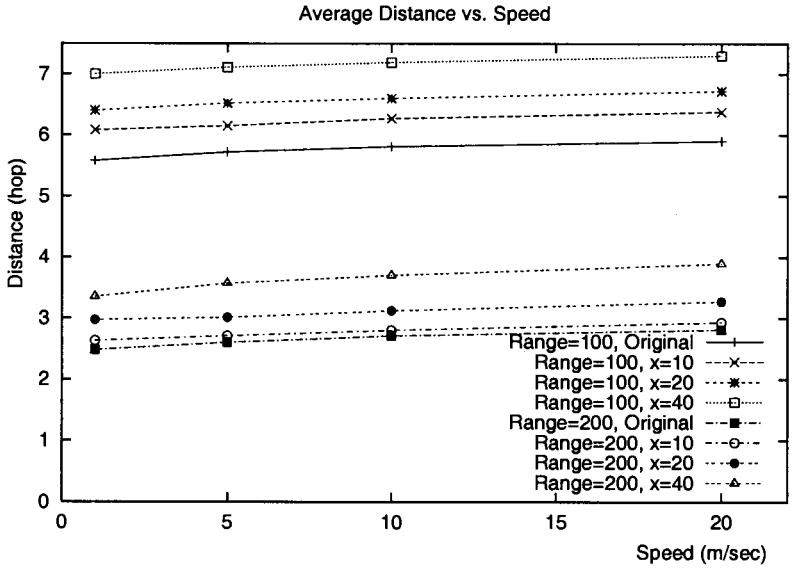



Figure 4.14: Pairwise vertex distance vs speed.

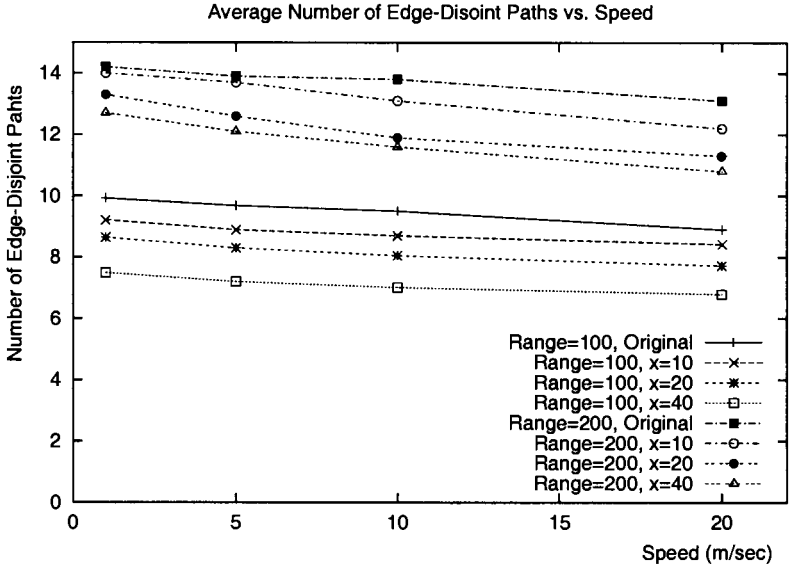



Figure 4.15: Number of edge-disjoint paths vs speed.

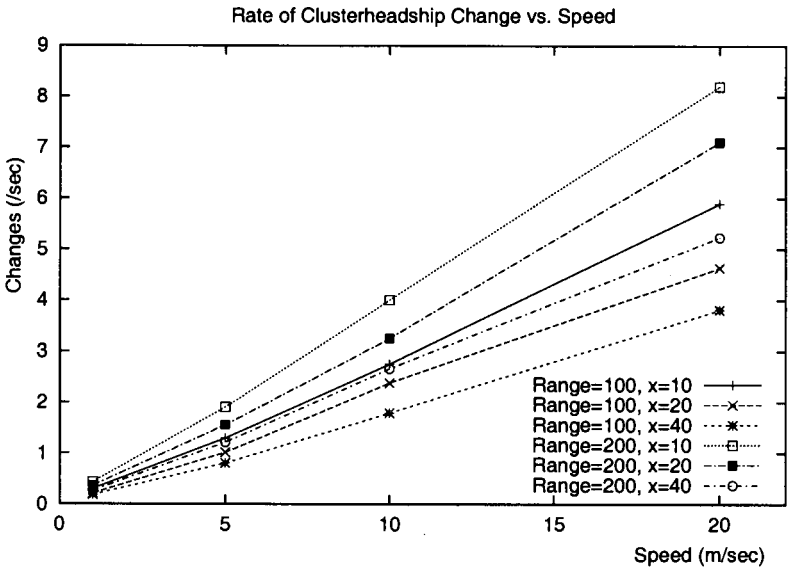



Figure 4.16: Rate of clusterhead change.

As the network topology changes, some vertices change status from dominating to dominated, or vice versa (clusterhead changes). Other vertices may move from one cluster to another (cluster membership changes). These changes affect routing and addressing in a hierarchical routing scheme. We consider the rates of these changes in the weakly-connected dominating set as the vertex speed varies. We record all the above changes occurring in each second with different average vertex speeds of 1, 5, 10, and 20m/s. The results are shown in Figures 4.16 and 4.17. We find that the change rates increase when vertices move faster as might be expected and that denser networks generate more such changes than sparser networks.

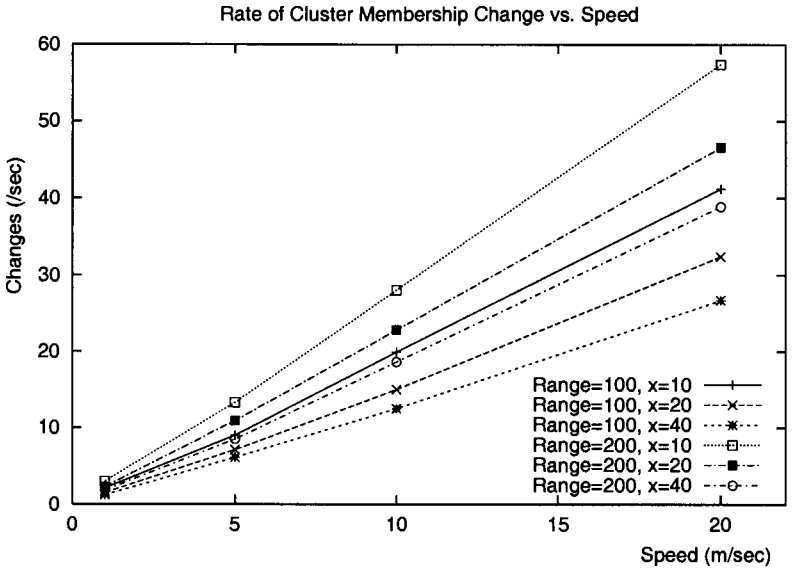



Figure 4.17: Rate of cluster membership change.

# Chapter 5

# Clustering in the Presence of Unidirectional Links

Due to the asymmetry of device capabilities and to interference, wireless links can be simplex. That is, it is possible that one vertex can send another but not vice versa. Such simplex wireless connections are referred to as *unidirectional links*. Directed graphs can be used to represent ad hoc networks with unidirectional links. Although some existing ad hoc network protocols do not allow unidirectional links, they may be worth considering. In graph theoretic terms, a pair of vertices in a digraph may not have a single bidirectional path connecting them, but they may still be connected by a pair of oppositely directed paths and, thus, able to communicate with each other. Consider, for example, a directed cycle, as shown in Figure 5.1. Vertices $u$ and $v$ are not connected by a bidirectional path, but they are connected by a pair of unidirectional paths.

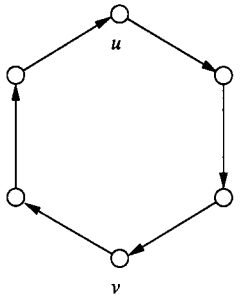



Figure 5.1: Vertices $u, v$ can communicate by a pair of directed paths.

## 5.1 Introduction

Unidirectional links are not uncommon in wireless communication networks. For example, in mixed terrestrial-satellite communication networks, satellites can transmit signals to all of the ground stations, but only a few of the ground stations can send signals back to the satellites.

Ad hoc networks may also result in unidirectional links although most of the existing protocols assume only bidirectional links. As in mixed terrestrial-satellite communication networks, there can be different types of devices in an ad hoc network. Even with identical devices, factors such as battery power can lead to differing transmission ranges. Also, interference may cause a device not to be able to properly receive packets although it may still be able to transmit.

Although utilizing unidirectional links in ad hoc networks is considered costly, they may be able to provide enhanced connectivity,

The use of unidirectional links in ad hoc networks results in some new protocol issues. In the data link layer, efficient medium sharing is made difficult. For example, handshaking in the distributed coordination function in the IEEE 802.11b/g MAC specification fails. Further, some mechanisms for error control and flow control, such as sliding window, are not applicable. In the network layer, most existing routing protocols for ad hoc networks do not work correctly in the presence of unidirectional links. For example, the generic Ad hoc On-demand Distance Vector (AODV) routing protocol [62, 61] may fail to discover a route even if there is a bidirectional path between the source and the destination.

To address the problems in the network layer, either new routing protocols can be devised or existing routing protocols can be modified. Gerla, Kleinrock, and Afek [30] present a distributed routing algorithm for unidirectional networks. In their proposal, routing tables are broadcast within the network to compute shortest paths between any strongly connected vertex pairs. Marina and Das [53] proposed a technique, called ReversePathSearch, to solve the route discovery problem in the AODV protocol. Others aim to abstract the network topology so that the unidirectional links appear bidirectional to the upper layers [65, 56]. The basic approach taken by these protocols is to find a *reverse path* for each unidirectional link such that the link layer control packets can be "tunneled" back from the downstream device to the upstream device. The work of Ramasubramanian et al. [65] is based on an improvement of that of Gerla, Kleinrock and Afek. Broadcasts are limited to distance-$r$

neighborhoods, such that the communication overhead is controlled and the length of the reverse paths is bounded. The other improvement made in this work is the utilization of "looped around" distance vectors by the reverse distributed Bellman-Ford algorithm. The next-hop information is obtained after propagating along a cycle that contains the unidirectional link.

In this chapter, we investigate the problem of clustering ad hoc networks in the presence of unidirectional links. We first introduce some related terminology, define domination variants for digraphs, and briefly discuss the computational complexity of these variants. In Section 5.3, we propose a series of greedy algorithms for these digraph domination variants. Our goal is to construct small dominating sets in a distributed setting. In Section 5.4, we present a set of distributed algorithms that can be executed in ad hoc networks with the presence of unidirectional links. These algorithms make fewer assumptions but we have not been able to show non-trivial approximation ratios for them. In Section 5.5, we compare the sets generated by the two types of algorithms.

## 5.2 Domination in directed graphs

### 5.2.1 Preliminaries

A digraph $G = (V, A)$ consists of a vertex set $V$ and an arc set $A$ containing ordered pairs of distinct vertices of set $V$. That is, $G$ has no loops or multiple (parallel) arcs, but pairs of oppositely directed arcs are allowed. In the figures, a thick line without an arrowhead indicates a pair of oppositely directly parallel arcs, i.e. a duplex link. A thin line with an arrowhead indicates a simplex link. Figure 5.2 shows an example of how we depict a graph in this chapter. We define the *in-set* (*in-neighborhood*) and *out-set* (*out-neighborhood*) of a vertex $v$ to be $I(v) = \{u : (u, v) \in A\}$ and $O(v) = \{u : (v, u) \in A\}$, respectively. $|I(v)|$ is called *in-degree* of $v$ and $|O(v)|$ is called *out-degree* of $v$. The *closed in-set* and *closed out-set* of a vertex $v$, denoted $I[v]$ and $O[v]$, respectively, are formed by adding the vertex $v$ itself to the in-set and out-set.

Given a digraph $G$, we define the distance function $d_G(u, v)$ from vertex $u$ to vertex $v$ to be the length of a shortest path from $u$ to $v$ in $G$. We define the distance function $d_G(u, S)$ from vertex $u$ to vertex set $S$ to be $\min_{v \in S} d_G(u, v)$. Similarly, we define $d_G(S, v)$ to be $\min_{u \in S} d_G(u, v)$. The diameter, $diam_G$, of a digraph $G$ is $\max_{u,v \in V} d_G(u, v)$. When the digraph is understood from the context without ambiguity, we can also ignore the subscript.

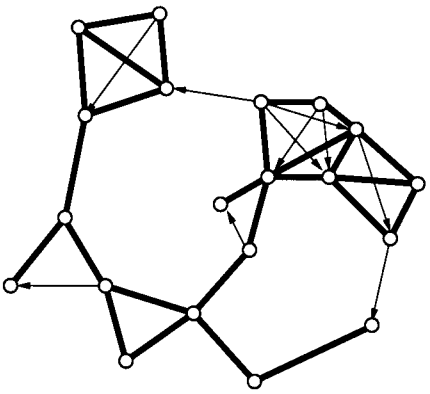


Figure 5.2: Depicting digraphs.

A digraph is called *strongly connected* or *strong* if for each pair of vertices $u$ and $v$, there is a directed path from $u$ to $v$ and a directed path from $v$ to $u$. Given a subset $S \subseteq V$, $\langle S \rangle_w$ is the subgraph *weakly induced* by $S$, where $\langle S \rangle_w = (O[S] \cup I[S], A \cap (O[S] \cup I[S]) \times S)$.

A subset $S \subseteq V$ is a *dominating set* if for all $v \notin S$, $v$ is an out-neighbor of some vertex $u \in S$. A subset $S$ of $V$ is called an *absorbent set* if for every vertex $v \notin S$, $v$ is an in-neighbor of some vertex $u \in S$. A subset $S \subseteq V$ is a *twin-dominating set* if it is both dominating and absorbent [19].

An arc $(u, v)$ is *dominated* by a vertex subset $S$ if $u \in S$. An arc $(u, v)$ is *absorbed* by a vertex subset $S$ if $v \in S$. Otherwise, it is *free*. A vertex $v$ is called *dominated* if $I[v] \cap S \neq \emptyset$. A vertex $v$ is called *absorbed* if $O[v] \cap S \neq \emptyset$. A vertex is called *twin-dominated* if $O[v] \cap I[v] \cap S \neq \emptyset$. Otherwise, it is *free*.

### 5.2.2 The gemini set

In order to cluster ad hoc networks where unidirectional links are present, we wish to choose a small set of subscriber units to be clusterheads, such that the subgraph weakly induced by the clusterheads spans the entire graph and is strongly connected. Given a strongly connected digraph $G = (V, A)$, our goal is to find a small subset $S \subseteq V$ such that $S$ is a twin-dominating set and $\langle S \rangle_w$ is a strongly connected spanning subgraph of $G$. Such a set $S$ is called a *gemini set*.

### 5.2.3 Notes on complexity

The decision versions of the dominating set, absorbent set, twin-dominating set, and gemini set problem in digraphs are all NP-complete because they are polynomially verifiable and they are at least as hard as the generic dominating set problem in the underlying undirected graphs. If we denote the sizes of a minimum dominating set, a minimum absorbent set, a twin-dominating set, and a gemini set by $\gamma_{out}$, $\gamma_{in}$, $\gamma_{twin}$, and $\gamma_{gemini}$, respectively, we immediately have the following inequality:

$$\max(\gamma_{out}, \gamma_{in}) \leq \gamma_{twin} \leq \gamma_{out} + \gamma_{in} \leq 2\max(\gamma_{out}, \gamma_{in}).$$

As we will see in Lemma 5.3.6, we also have:

$$\gamma_{twin} \leq \gamma_{gemini} \leq \left\lfloor \frac{diam + 3}{2} \right\rfloor \gamma_{twin} - 1.$$

## 5.3 Approximation algorithms

We present a series of approximation algorithms for finding small dominating sets, absorbent sets, twin-dominating sets, and gemini sets. We first present a greedy algorithm for constructing small dominating sets of $G$. This algorithm implies a dual algorithm for constructing small absorbent sets of $G$. Running both of these algorithms in $G$ and combining the dominating set and absorbent set gives us a twin-dominating set. Alternatively, we can first construct a dominating set of $G$ and then expand the set so that it is also absorbent. By connecting the vertices of a twin-dominating set economically, we provide an approximation algorithm that constructs a small gemini set of $G$. At the end of the section, we discuss the difficulties in transforming these algorithms to distributed algorithms.

### 5.3.1 Greedy directed dominating set (GDD) algorithm

This algorithm for finding small dominating sets in digraphs is a centralized greedy algorithm. It has a provable approximation ratio of $\ln \Delta_{out} + 1$, where $\Delta_{out}$ is the maximum vertex out-degree in the given digraph $G$.

This iterative algorithm is based on coloring vertices in $G$ using three colors: white, gray, and black. All vertices in $G$ are white initially. In each iteration, a white or gray vertex $v$ is colored black and all white vertices in $O(v)$ are colored gray. We define the *out-improvement* of a vertex $v$ to be the number of white vertices in $O[v]$. In each iteration, we chose a white

or gray vertex with the largest out-improvement among all such vertices. Ties are broken unambiguously, say, using vertex ID's. The iterative process continues until there are no white vertices left.

This simple centralized greedy algorithm has a logarithmic approximation ratio as shown in the following lemma: The proof is similar to that of Lemma 2.1.2.

**Lemma 5.3.1** *Given a digraph $G$ with maximum vertex out-degree $\Delta_{out}$, the size of the dominating set $S$ found by the GDD algorithm is at most $(\ln \Delta_{out} + 1)\gamma_{out}$, where $\gamma_{out}$ is the domination number of $G$.*

**Proof:** Let $OPT_{DS} = \{v_1, v_2, \cdots, v_{\gamma_{out}}\}$ be a minimum dominating set of $G$. Partition the vertices of $G$ into sets $P_i$, for $1 \leq i \leq \gamma_{out}$, such that $v_i \in P_i$ and every vertex $w$ of $V - OPT_{DS}$ is placed into $P_i$ for some $i$ such that $v_i$ dominates $w$.

The proof is based on a charging analysis. Each time we color a vertex black, we add one vertex to $S$ and incur a charge of one unit. This charge is equally distributed among all of the white vertices that are colored in that step. The total charge for the entire process is $|S|$, the size of the dominating set. We will show that the total charge among the vertices of $P_i$ (for any $i$) is at most $\ln \Delta_{out} + 1$. Since there are $OPT_{DS}$ such sets in $G$, the theorem follows.

Assume that when we choose a vertex to color black, we color $x$ white vertices and charge such vertex $\frac{1}{x}$. We now consider the number of charges among the vertices of a single set $P_i$. Let $u_j$ denote the number of white vertices in $P_i$ after iteration $j$. For the sake of simplicity, we assume that $u_0 = |P_i|$ and that some vertices of $P_i$ are colored in each iteration, so the number of white vertices in $P_i$ decreases in each iteration.

Consider the $j$th iteration (for $j = 1, 2, \ldots$). Choosing $v_i$ would add $u_{j-1}$ vertices to $S$, so any vertex chosen must add at least that many to $S$ and thus any vertices of $P_i$ colored in the $j$th iteration incur a charge of at most $\frac{1}{u_{j-1}}$. Thus, in the $j$th iteration, at most $\frac{u_{j-1}-u_j}{u_{j-1}}$ charges are incurred among the vertices of $P_i$. Eventually, $u_k = 0$ for some $k$.

Summing the charges within $P_i$, we get at most

$$\begin{aligned} & \sum_{j=1}^{k} \frac{1}{u_{j-1}}(u_{j-1} - u_j) \\ = & \sum_{j=2}^{k} \frac{u_{j-1} - u_j}{u_{j-1}} \end{aligned}$$

$$
\begin{aligned}
&\leq \sum_{j=2}^{k}(H(u_{j-1}) - H(u_j)) \\
&= (H(u_1) - H(u_k)) \\
&\leq H(\Delta_{out}) - H(u_k)) \\
&= H(\Delta_{out}) \\
&\leq 1 + \ln \Delta_{out}
\end{aligned}
$$

Therefore, the lemma holds. □

### 5.3.2 Greedy directed absorbent set (GDA)

The GDA algorithm constructs a small absorbent set for a given digraph. It is identical to the GDD algorithm except that it considers in-neighbors rather than out-neighbors and yields the same logarithmic approximation ratio as the GDD algorithm.

**Lemma 5.3.2** *Given a digraph $G$ with maximum vertex in-degree $\Delta_{in}$, the size of the absorbent set $S$ found by the GDA algorithm is at most $(\ln \Delta_{in} + 1)\gamma_{in}$, where $\gamma_{in}$ is the absorbency number of $G$.*

### 5.3.3 Greedy directed twin-dominating set (GDT) algorithms

A twin-dominating set can be obtained by taking the union of the dominating set and the absorbent set generated by the GDD and GDA algorithms, respectively. We denote this algorithm $\text{GDT}_{\text{UNION}}$. The following lemma is obtained from the above lemma and the observation that $\gamma_{twin} \geq \max(\gamma_{in}, \gamma_{out})$.

**Lemma 5.3.3** *Given a digraph $G$ with $\Delta = \max(\Delta_{out}, \Delta_{in})$, the size of the twin-dominating set $S$ found by the $GDT_{UNION}$ algorithm is at most $2(\ln \Delta + 1)\gamma_{twin}$, where $\gamma_{twin}$ is the twin-domination number of $G$.*

Another algorithm, denoted $\text{GDT}_{\text{2PH}}$, constructs a twin-dominating set in two phases. We first find a small dominating set $S$ of $G$ using the GDD algorithm. Since the set $S$ already absorbs the vertices of $I[S]$, we color the vertices in $S$ black and those in $I[S] - S$ gray. Then we execute the GDA algorithm beginning with the resulting coloring of $G$.

One might conjecture that the $\text{GDT}_{\text{2PH}}$ algorithm always generates smaller twin-dominating sets than the $\text{GDT}_{\text{UNION}}$ algorithm does. Although our simulation shows that this is true

for most cases, for some graphs the $\text{GDT}_{\text{UNION}}$ algorithm may actually yield a smaller twin-dominating set than the $\text{GDT}_{\text{2PH}}$ algorithm.

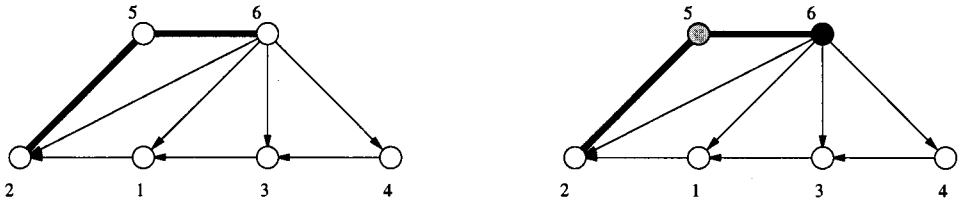



Figure 5.3: When $\text{GDT}_{\text{UNION}}$ beats $\text{GDT}_{\text{2PH}}$.

In the example in Figure 5.3, we are given a digraph on six vertices with ID's as indicated. If ties are broken by favoring vertices with lower ID's, the GDD algorithm generates the dominating set $\{v_6\}$ and the GDA algorithm generates the absorbent set $\{v_2, v_3\}$. Thus, the $\text{GDT}_{\text{UNION}}$ algorithm yields $S_{\text{UNION}} = \{v_2, v_3, v_6\}$. On the other hand, after phase one of the $\text{GDT}_{\text{2PH}}$ algorithm, vertex $v_6$ is black and vertex $v_5$ is gray. Phase two will select vertices $v_1$, $v_2$, and $v_3$ to absorb all the other vertices, so $\text{GDT}_{\text{2PH}}$ yields $S_{\text{2PH}} = \{v_1, v_2, v_3, v_6\}$. $|S_{\text{UNION}}| < |S_{\text{2PH}}|$ in this example.

The GDD algorithm can be implemented such that each vertex is require to have the knowledge within a constant radius of itself and the algorithm still constructs a dominating set with the approximation ratio $\ln \Delta_{out} + 1$. The GDA and GDT algorithms can also be implemented in this fashion. We describe the modified GDD algorithm and the other two modifications are similar.

In the modified implementation of the GDD algorithm, in each iteration, any white or gray vertex $v$ with positive out-improvement value colors itself black if it has the greatest such value within $N_2[v]$, where $N_2[v]$ denotes the closed distance-2 neighborhood of $v$ in the underlying undirected graph of digraph $G$. Surprisingly, this modified algorithm has the logarithmic approximation ratio as before.

**Corollary 5.3.4** *Given a digraph $G$ with maximum vertex out-degree $\Delta_{out}$, the size of the dominating set $S$ found by the modified implementation of the GDD algorithm is at most $(\ln \Delta_{out} + 1)\gamma_{out}$, where $\gamma_{out}$ is the out-domination number of $G$.*

**Proof sketch.** The same counting argument as in the proof of Lemma 5.3.1 can be used here. Consider vertex $v_i \in OPT_{DS}$ and its partition $P_i$ as before. During the $j$th iteration, $u_{j-1} - u_j$ white vertices in $P_i$ change colors due to, possibly multiple, new black vertices

within $N_2[v_i]$. Because each vertex that changes color to black is at distance at least three from any other such vertex in the underlying undirected graph, its set of white out-neighbors does not overlap with other such sets. For example, vertices $u$ and $w$ in Figure 5.4 can be colored black in the same iteration. Therefore, each of the $u_{j-1} - u_j$ white vertices in $P_i$ that change colors is charged once, and the charge is still no more than $\frac{1}{u_{j-1}}$. Consequently, the same charging analysis is valid here. □

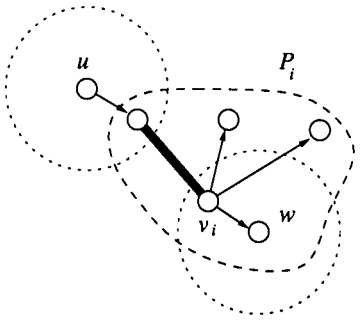



Figure 5.4: Proof illustration for Corollary 5.3.4.

### 5.3.4 GDG algorithm — approximating minimum gemini sets

The greedy directed gemini set (GDG) algorithm is a two-phase algorithm that constructs a gemini set for a given digraph with an approximation ratio of $\left\lfloor \frac{diam+3}{2} \right\rfloor \times \ln \Delta$. In the first phase, an instance of the GDT algorithm is executed to generate a twin-dominating set $S_t$. In the second phase, at most $\left\lfloor \frac{diam+1}{2} \right\rfloor \times |S_t| - 1$ more vertices are added to $S_t$ so that it becomes a gemini set, denoted $S_g$. The central idea of the second phase of the algorithm is that the vertices of a twin-dominating set $|S_t|$ are relatively close together so we need only to add a small number of additional vertices to it to turn it into a gemini set. The correctness of the algorithm is based on Lemmas 5.3.5 and 5.3.6 and the construction follows from the mathematical induction in the proof of Lemma 5.3.6.

**Lemma 5.3.5** *Given a twin-dominating set $S_t$ of a strong digraph $G$ of size $|S_t| > 1$, then for any vertex $v \in S_t$ either $d(v, S_t \setminus v) \leq 3$ or $d(S_t \setminus v, v) \leq 3$.*

**Proof.** Suppose $d(v, S_t \setminus v) = k$ $(k \geq 4)$. We will show that this forces $d(S_t \setminus v, v)$ to be at most 3. Let vertex $u \in S_t \setminus v$ be a vertex such that $d(v, u) = k$ and let $P = v_0 v_1 \ldots v_k$ be a shortest path from $v$ to $u$, where $v = v_0$ and $u = v_k$. Clearly, none of the intermediate

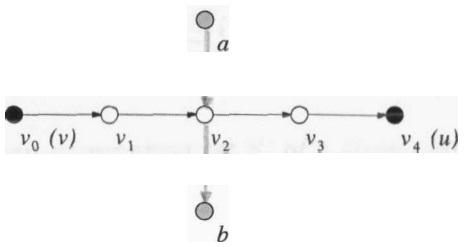



Figure 5.5: Proof illustration for Lemma 5.3.5.

vertices of $P$ is in $S_t \backslash v$; or, $d(v, S_t \backslash v)$ would be less than $k$. Consider vertex $v_2$, as depicted in Figure 5.5 with $k = 4$. Since $S_t$ is a twin-dominating set, $v_2$ is both dominated and absorbed. Let vertex $a$ be a vertex dominating $v_2$ and a vertex $b$ be a vertex absorbing $v_2$. If $v \neq b$, then there is a path from vertex $v$ to $b$ of length 3, which implies that $d(v, S_t \setminus v) = 3$, a contradiction. So, we have $v = b$. We also know that $v \neq a$ because otherwise $d(v, u) = k-1$, a contradiction. Therefore, the path $av_2v$ shows that $d(S_t \setminus v, v) \leq 3$ and the lemma holds. □

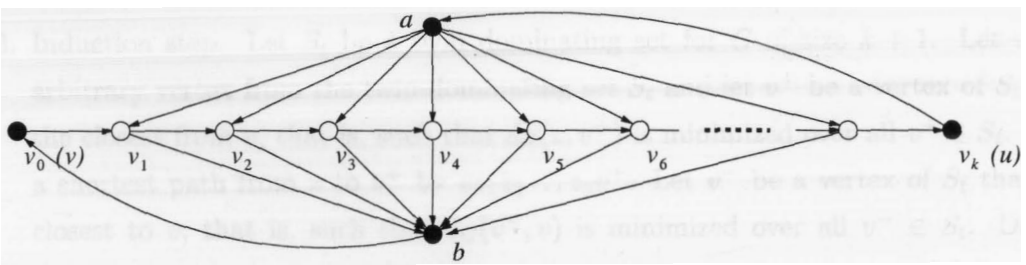



Figure 5.6: A gemini set can be much larger than a twin-dominating set.

The analogous result for undirected graphs is: Given a dominating set $S$ of $G$ of size $|S| > 1$, the for any vertex $v \in S$, $d(v, S \setminus v) = d(S \setminus v, v) \leq 3$. That is, the two bounds of the directed case hold simultaneously in the undirected case. Unfortunately, in the directed case, we only know that at least one of the bounds holds.

Consider the example of Figure 5.6, in which the black vertices $a$, $b$, $u$, and $v$ form a twin-dominating set. However, these four vertices do not form a gemini set for $G$. Note, in particular, that for any path from a vertex $v_i$ to a vertex $v_j$ where $1 \leq i < j \leq k-1$ to exist in any subgraph of $G$, all of the edges $(v_i, v_{i+1}), (v_{i+1}, v_{i+2}), \ldots, (v_{j-1}, v_j)$ must be included in the subgraph. That is, to be strongly connected, the subgraph must include all of the horizontal arcs in the figure. Thus, a gemini set must include at least one of each

consecutive pair of vertices $v_i, v_{i+1}$ for $1 \leq i \leq k-2$. Thus, at least $\left\lceil \frac{k-2}{2} \right\rceil = \left\lfloor \frac{k-1}{2} \right\rfloor$ vertices must be added to form a gemini set for this example.

**Lemma 5.3.6** *Given a twin-dominating set $S_t$ of a strong digraph $G$, $S_t$ can be augmented to become a gemini set $S_g$ such that $|S_g| \leq \left\lfloor \frac{diam+3}{2} \right\rfloor \times |S_t| - 1$.*

**Proof.** The central idea of the proof is to show that we only need to introduce $\left\lfloor \frac{diam+1}{2} \right\rfloor$ more vertices to connect a vertex of $S_t$ to the rest. To do that, we use mathematical induction on $|S_t|$.

1. Basic step. When $|S_t| = 1$, we know that there is a vertex $u$ that is adjacent to every other vertex and that is adjacent from every other vertex. Thus, $S_t = \{u\}$ and $S_g = S_t$ is also a gemini set of the graph $G$. Since $diam_G \geq 1$, we have $|S_g| \leq \left\lfloor \frac{diam_G+3}{2} \right\rfloor \times |S_t| - 1$.

2. Induction hypothesis. Given a digraph $G$ and a twin-dominating set of size $k$, the twin-dominating set can be expanded to a gemini set of size at most $\left\lfloor \frac{diam_G+3}{2} \right\rfloor \times k - 1$.

3. Induction step. Let $S_t$ be a twin dominating set for $G$ of size $k+1$. Let $v$ be an arbitrary vertex from the twin-dominating set $S_t$ and let $v^+$ be a vertex of $S_t$ that is the closest from $v$, that is, such that $d_G(v, v^+)$ is minimized over all $v^+ \in S_t$. Denote a shortest path from $v$ to $v^+$ by $vz_1z_2 \ldots z_q v^+$. Let $v^-$ be a vertex of $S_t$ that is the closest to $v$, that is, such that $d_G(v^-, v)$ is minimized over all $v^- \in S_t$. Denote a shortest path from $v^-$ to $v$ by $v^-y_1y_2 \ldots y_p v$. Note that from Lemma 5.3.5, we know that at least one of these two distances is $\leq 3$ and the other must be $\leq diam_G$.

   We now construct a graph $G'$ from $G$ by adding arcs from every element of $I(v)$ to $v^+$, from $v^-$ to every element of $O(v)$, and from every element of $I(v)$ to every element of $O(v)$. (In all cases, we do not add an arc if it already exists in $G$.) Then remove $v$ and any incident arcs to obtain $G'$.

   Observe that $G'$ is strongly connected since $G$ was strongly connected and any path in $G$ containing $v$ as an intermediate vertex can be replaced by including a "shortcut" from an element of $I(v)$ to an element of $O(v)$. Observe also that $diam_{G'} \leq diam_G$. Finally, observe that $S_t \setminus v$ is a twin-dominating set for $G'$ since vertices that were dominated by $v$ in $G$ are dominated by $v^-$ in $G'$ and vertices that were absorbed by $v$ in $G$ are absorbed by $v^+$ in $G'$.

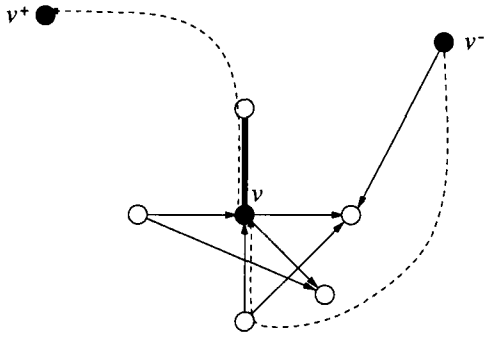



Figure 5.7: Portion of graph $G$.

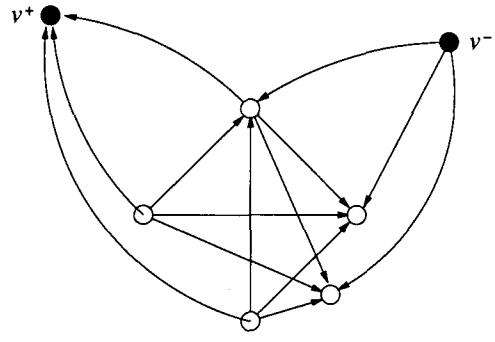



Figure 5.8: Portion of graph $G'$.

By the induction hypothesis, since $|S_t \setminus v| = k$, $S_t \setminus v$ can be augmented to form a gemini set $S'_g$ of $G'$ of size at most $\left\lfloor \frac{diam_{G'}+3}{2} \right\rfloor \times k - 1$. Now consider the set of vertex $S'_g \cup \{v\}$ in $G$. Since $S_t \subseteq S'_g \cup \{v\}$, $S'_g \cup \{v\}$ is a twin-dominating set of $G$, but it is not necessarily a gemini set of $G$. We now construct a gemini set $S_g$ of $G$ by adding vertices to $S'_g \cup \{v\}$.

Consider an arc $(a, b)$ added during the construction of $G'$. This arc connects two vertices which are on the path $v^- y_1 y_2 \ldots y_p v z_1 z_2 \ldots z_q v^+$ and, in fact, $a$ precedes $b$ on the path. To ensure that any such arc in $G'$ can be replaced by a path in the subgraph weakly induced by $S_g$, we need only add the even subscripted vertices $y_2$, $y_4$, ... and the even subscripted vertices $z_2$, $z_4$, .... We know that either $p \leq 2$ or $q \leq 2$ and that both $p \leq diam_G - 1$ and $q \leq diam_G - 1$, so we add at most $1 + \left\lfloor \frac{diam_G - 1}{2} \right\rfloor$ vertices to $S'_g \cup \{v\}$ to obtain $S_g$.

Thus, we have

$$\begin{aligned} |S_g| &\leq (|S'_g| + 1) + (1 + \left\lfloor \frac{diam_G - 1}{2} \right\rfloor) \\ &\leq \left( \left( \left\lfloor \frac{diam'_G + 3}{2} \right\rfloor \times k - 1 \right) + 1 \right) + \left( 1 + \left\lfloor \frac{diam_G - 1}{2} \right\rfloor \right) \\ &= \left\lfloor \frac{diam_{G'} + 3}{2} \right\rfloor \times k + \left\lfloor \frac{diam_G + 3}{2} \right\rfloor - 1 \end{aligned}$$

Since $diam_{G'} \leq diam_G$, we have

$$|S_g| \leq \left\lfloor \frac{diam_G + 3}{2} \right\rfloor \times (k+1) - 1.$$

Therefore, the lemma holds. □

Now we are ready to present the approximation ratio result of the GDG algorithm.

**Theorem 5.3.7** *Given a digraph $G$, the size of the gemini set generated by the GDG algorithm is at most $2\left\lfloor\frac{diam_G+3}{2}\right\rfloor(\ln\Delta+1)$ times the optimum, where $\Delta=\max(\Delta_{in},\Delta_{out})$.*

**Proof.** According to Lemma 5.3.3, the size of the twin-dominating set $|S_t|$ constructed in the first phase is at most $(2\ln\Delta+2)\gamma_{twin}$. In the second phase, the twin-dominating set $S_t$ is expanded into a gemini set $S_g$ and $|S_g|\leq\left\lfloor\frac{diam_G+3}{2}\right\rfloor\times|S_t|-1$. Since $\gamma_{twin}\leq\gamma_{gemini}$, we have $S_g\leq\left\lfloor\frac{diam_G+3}{2}\right\rfloor\times(2\ln\Delta+2)\gamma_{gemini}-1$ Therefore, the approximation ratio of $2\left\lfloor\frac{diam_G+3}{2}\right\rfloor(\ln\Delta+1)$ holds. □

## 5.4 Distributed heuristics for small gemini sets

The clustering algorithms presented in the previous section are intended to be implemented in ad hoc networks with unidirectional links. However, due to unidirectional links, the neighborhood information exchange in the GDD, GDD, GDT, and GDG algorithms can not be carried out in a straightforward way. In particular, if there is only a unidirectional link from vertex $u$ to $v$, $u$ will not be able to know the existence of $v$ or of $v$'s updated status. In this section, we address this problem by proposing another set of algorithms for constructing small gemini sets.

The degree of localization possible in such algorithms is limited. For example, for a vertex of a directed cycle (depicted earlier in Figure 5.1) to be aware of its down-stream neighbor, global information is needed. Therefore, in this section, we focus on algorithms for constructing small gemini sets which may require global operations.

**Definition 5.4.1** *[12] An* ear decomposition *of a digraph $G=(V,A)$ is a sequence $\mathcal{E}=\{P_0,P_1,\ldots,P_t\}$, where $P_0$ is a cycle and each $P_i$ is a path, or a cycle with the following properties:*

1. *$P_i$ and $P_j$ are arc-disjoint when $i\neq j$.*
2. *For each $i=1,\ldots,t$: If $P_i$ is a cycle, then it has precisely one vertex in common with $V(G_{i-1})$. Otherwise the end-vertices of $P_i$ are distinct vertices of $V(G_{i-1})$ and no other vertex of $P_i$ belongs to $V(G_{i-1})$. Here, $G_i$ denotes the digraph with vertices $\bigcup_{j=0}^{i}V(P_j)$ and arcs $\bigcup_{j=0}^{i}A(P_j)$.*

3. $\bigcup_{j=0}^{t} A(P_j) = A$.

*Each* $P_i$, $0 \leq i \leq t$, *is called an* ear *of* $\mathcal{E}$. *The* number of ears *in* $\mathcal{E}$ *is* $t+1$. *An ear* $P_i$ *is* trivial *if* $|A(P_i)| = 1$. *All other ears are* non-trivial.

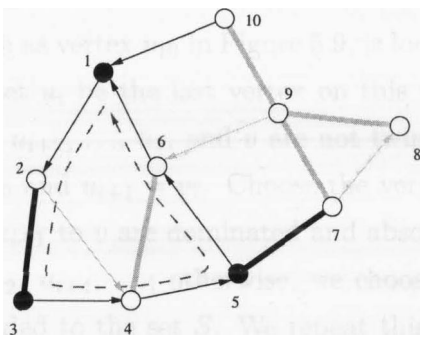



Figure 5.9: Initial cycle.

Three algorithms are presented in this section. The central idea is to construct a subset $S \subseteq V$ iteratively which will become a gemini set. After each iteration, the vertices in $S$ are strongly connected in $\langle S \rangle_w$. More vertices are added to $S$ in each iteration until all vertices in $V$ are twin-dominated and, since $\langle S \rangle_w$ is strongly connected, $S$ is a gemini set. Vertices are added to $S$ in the order they appear in an ear-decomposition, that is, beginning with $P_0$, then $P_1$, etc. When there is still a vertex $v$ that is not twin-dominated at the end of an iteration, a simple path originating from and ended at a twin-dominated vertex containing $v$ is located. More vertices on this path are added to $S$ as needed. For example, the current set $S$ for the digraph in Figure 5.9 consists of three vertices, $v_1$, $v_3$, and $v_5$. The arcs of $\langle S \rangle_w$ are black and free arcs are gray. Vertices $v_8$, $v_9$, and $v_{10}$ are not twin-dominated in the example. A simple path that can add more vertices to $S$ is $v_7 v_9 v_{10} v_1$.

The first algorithm, $\text{EAR}_\text{C}$, is a centralized algorithm. The second algorithm, $\text{EAR}_\text{D}$, is distributed and grows $S$ sequentially. The third algorithm, $\text{EAR}_\text{P}$, is distributed and adds parallelism with constructing multiple strong components of $\langle S \rangle_w$ in each iteration. In these algorithms, we also call choosing a vertex to be included to $S$ "alternate coloring".

### 5.4.1 $\text{EAR}_\text{C}$

Assume that we are given a strong digraph $G = (V, A)$ with a unique root vertex $r \in V$. The $\text{EAR}_\text{C}$ algorithm first finds a simple cycle containing $r$, say $C = v_1 v_2 \ldots v_k$, where

$v_1 = v_k = r$. It adds vertices $v_{2i-1}$ $(i = 1, 2, \ldots, \lceil \frac{k}{2} \rceil)$ to $S$, i.e., it alternately chooses the vertices on $C$ such that all of the cycle arcs are both dominated and absorbed. In the example of Figure 5.9, $v_1$ is the root, $v_1v_2v_3v_4v_5v_6v_1$ is the cycle, and vertices $v_1$, $v_3$, and $v_5$ are added to $S$.

In each of the following iterations, a vertex $v$ that is not twin-dominated but has a twin-dominated out-neighbor, such as vertex $v_{10}$ in Figure 5.9, is located. We find a shortest path $ru_1u_2 \ldots u_kv$ from $r$ to $v$. Let $u_i$ be the last vertex on this paths that is twin-dominated — that is, the vertices $u_{i+1}$, $u_{i+2}$, ..., $u_k$, and $v$ are not twin-dominated. In our example, the path is $v_1v_2v_3v_4v_5v_7v_9v_{10}$ and $u_{i+1} = v_7$. Choose the vertices alternately so that all of the edges on the path from $u_{i+1}$ to $v$ are dominated and absorbed. That is, if $u_i$ is already in $S$, we choose vertices $u_{i+2}$, $u_{i+4}$, ...; otherwise, we choose vertices $u_{i+1}$, $u_{i+3}$, .... In our example, vertex $v_9$ is added to the set $S$. We repeat this process until all vertices are twin-dominated.

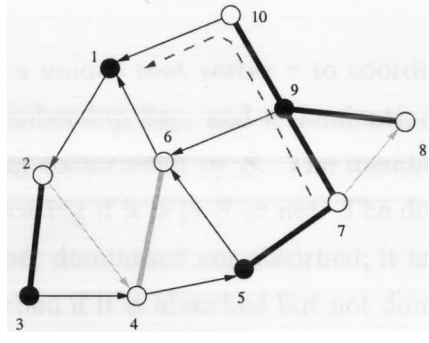



Figure 5.10: Path in iteration two.

Each iteration of this process is also called an *exploration*. A cycle $C$ is identified in the first exploration and a path $P$ is identified in each following exploration. A vertex on $C$ or a vertex on $P$ that is not twin-dominated is *explored* when the cycle or path is identified. All vertices explored in an iteration are strongly connected to all the previously explored vertices in the subgraph $\langle S \rangle_w$ weakly induced by the current set $S$. Note that, after an iteration, some other vertices may become strongly connected to the previously explored vertices in $\langle S \rangle_w$ though they are not themselves on any path or cycle of the exploration. These vertices are also said to be explored. Thus, the set of explored vertices after an iteration are those newly twin-dominated by $S$. For example, in Figure 5.10, vertex $v_8$ becomes twin-dominated and strongly connected to the previously explored vertices in $\langle S \rangle_w$ when path $v_7v_9v_{10}v_1$ is

identified. We will explain how this can affect the distributed versions of this algorithm in the upcoming subsection.

### 5.4.2 $\text{EAR}_{\text{D}}$

The $\text{EAR}_{\text{D}}$ algorithm is a distributed implementation of the $\text{EAR}_{\text{C}}$ algorithm. The input is a strong digraph $G = (V, A)$ representing an asynchronous network. Each vertex of $G$ is a network node and each arc is a reliable unidirectional first-in-first-out (FIFO) channel. In this construction algorithm, the topology of the network is assumed to be static. From the viewpoint of an individual vertex, the set of in-neighbors that a vertex can directly hear from is always the same. A vertex $v$ sends to its out-neighbors a list of vertices from which it can directly receive messages, i.e., its in-neighbors $I(v)$. By checking if $v$ itself is included in the in-neighborhood $I(u)$ of an in-neighbor $u$, $v$ determines if $u$ is also an out-neighbor. Then $v$ incorporates the link type (`in-only` or `duplex`) information to the shared neighbor list.

We assume that there is a unique root vertex $r$ to coordinate the algorithm. A vertex has a distinct identity, a membership flag, and a domination status field. We denote the set of vertices currently being constructed by $S$. The membership flag of a vertex can be either `in_S` or `not_in_S`, indicating if it is in $S$ or not. The domination status of a vertex is recorded as `free` if it is neither dominated nor absorbed; it is `dominated` if it is dominated but not absorbed; it is `absorbed` if it is absorbed but not dominated; it is `twin` if it is both dominated and absorbed. A vertex's membership flag of value `in_S` implies that its status is `twin`. When a vertex $v$ changes its membership flag or status value, it sends a message to $O(v)$ so that all of its out-neighbors are informed of this change.

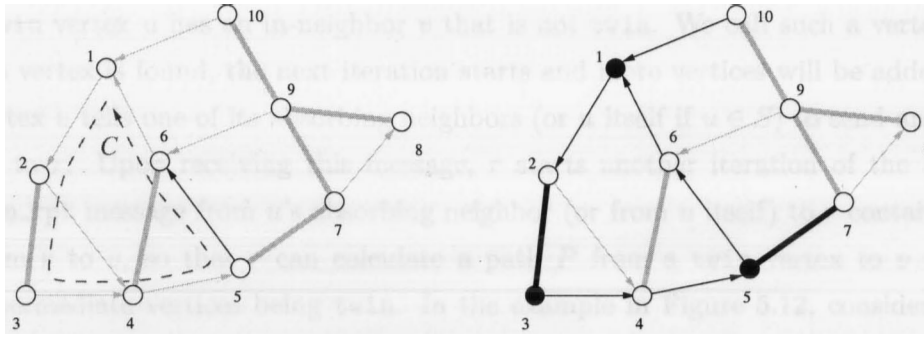



Figure 5.11: Cycle $C$ in iteration one.

In the beginning of the $\text{EAR}_\text{D}$ algorithm, root $r$ broadcast a message within $G$ to find a simple cycle. A vertex forwards this message to all of its neighbors only when it receives it for the first time. Each copy of the message records its travel history, i.e., a list of vertex ID's that it has traversed. The first copy of the message to return to $r$ reveals a simple cycle, denoted $C$. Root $r$ adds every other vertex to $S$, beginning with $r$ itself. Then $r$ informs the vertices newly added to $S$ (who change their membership flag to `in_S`) by sending an `alt_color` message along $C$. Routing information for the `alt_color` message is contained in the message itself so that every vertex receiving it knows whom to forward it to next. All the vertices on the cycle $C$ change status to `twin` after receiving this message. Note that vertices not on $C$ also change their status as a result of update to their neighbors. For example, in Figure 5.11, let the cycle $C$ be $v_1v_2v_3v_4v_5v_6v_1$ and $v_1$, $v_3$, and $v_5$ are added to $S$. Vertex $v_7$ changes its status from `free` to `twin` after learning that $v_5$ is added to $S$. Every vertex in $S$ also remembers the path to $r$ when it is initially added to $S$.

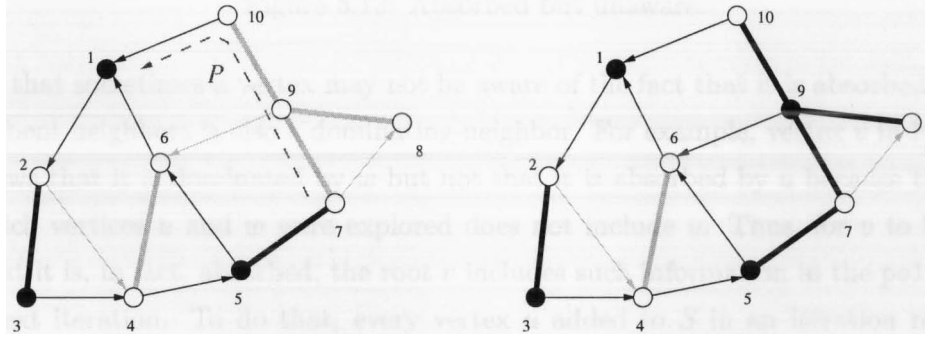



Figure 5.12: Path $P$ in iteration two.

After the `alt_color` message returns to the root $r$, $r$ broadcasts a `poll` message to see if any `twin` vertex $u$ has an in-neighbor $v$ that is not `twin`. We call such a vertex $u$ *open*. If such a vertex is found, the next iteration starts and more vertices will be added. In this case, vertex $u$ tells one of its absorbing neighbors (or $u$ itself if $u \in S$) to send an `open_rpt` message to $r$. Upon receiving this message, $r$ starts another iteration of the algorithm. The `open_rpt` message from $u$'s absorbing neighbor (or from $u$ itself) to $r$ contains the full path from $r$ to $v$, so that $r$ can calculate a path $P$ from a `twin` vertex to $v$ with none of its intermediate vertices being `twin`. In the example in Figure 5.12, consider the path $P = v_7v_9v_{10}v_1$. The root $r$ then sends an `alt_color` message to add vertices of $P$ to $S$. It is possible that $r$ may receive multiple `open_rpt` messages from different sources, but it only

processes the first received message from the current `poll` message and discards the rest. Similarly, other vertices forward only the first `open_rpt` message that they receive related to the current `poll` and discard the others. The `alt_color` message is sent along $P$ and returns to $r$. As in the initial iteration, other vertices may also change status according to the membership flag changes of their neighbors. After $r$ receives this return `alt_color` message, it broadcasts another `poll` in $G$ to see if any non-`twin` vertices remain. This process continues until all vertices are `twin`.

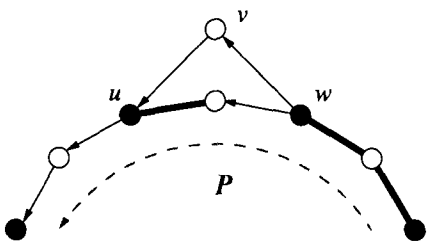



Figure 5.13: Absorbed but unaware.

Note that sometimes a vertex may not be aware of the fact that it is absorbed if none of its absorbent neighbors is also a dominating neighbor. For example, vertex $v$ in Figure 5.13 only knows that it is dominated by $w$ but not that it is absorbed by $u$ because the path $P$ with which vertices $u$ and $w$ were explored does not include $v$. Thus, for $v$ to be able to know that it is, in fact, absorbed, the root $r$ includes such information in the `poll` message of the next iteration. To do that, every vertex $u$ added to $S$ in an iteration reports the subset of $I(u)$ that becomes absorbed for the first time. This report is piggy-backed on the `alt_color` message that is sent along the path $P$ back to $r$. Then, $r$ compiles a list of vertices that are newly absorbed to send with the next `poll` message. For example, in the portion of the network depicted in Figure 5.13, vertex $u$ knows that vertex $v$ was not absorbed until $u$ itself is added to $S$. When $u$ receives the `alt_color` message, it attaches the information that $v$ is newly absorbed to this message. After this message returns to $r$, $r$ incorporates the list of all newly absorbed vertices to the `poll` message of the next iteration. Vertex $v$ will then learn that it is absorbed and change its status from `dominated` to `twin`. With this change, every vertex always has the current domination status of its in-neighbors when it receives the `poll` message of a new iteration.

When the root $r$ determines that all vertices are `twin`, the algorithm terminates and the vertices of $S$ comprise a gemini set of $G$.

The $\text{EAR}_\text{D}$ algorithm terminates since at least one vertex becomes twin-dominated in each iteration and there are a finite number of vertices in the input graph $G$. When the algorithm terminates, the vertices in $S$ form a gemini set. The union of the initial cycle and the subsequent exploration paths form a strong subgraph of $G$. Such a subgraph spans all the vertices in $S$. Other vertices that are not on the initial cycle or any exploration path are strongly connected to and from some vertex in $S$.

### 5.4.3 $\text{EAR}_\text{P}$

In this subsection, we present another distributed algorithm for constructing small gemini sets, denoted $\text{EAR}_\text{P}$. Here, we begin with a set of initiating root vertices $\{r_1, r_2, \ldots, r_a\}$, each of which will begin to construct the gemini set $S$ in parallel. The initiating roots begin essentially to execute the previous algorithm $\text{EAR}_\text{D}$ as if there was only one root. Once an exploration cycle or path is discovered by root $r_i$ and vertices on that cycle/path are added to $S$, all of the vertices on that cycle/path are known to be in the same strongly connected component of $\langle S \rangle_w$ as $r_i$. Furthermore, if any of the vertices on this cycle/path know that they are in the same strongly connected component as another root $r_j$, then all of the vertices on the cycle/path are in the same strongly connected component and, in fact, the two roots are in the same strongly connected component.

Ideally, all vertices in the same strongly connected component would immediately be informed when the component increases in size, so that subsequent exploration paths (from any of the roots in the component) are easily recognized.

If we were to use a single "component ID number" to label each strongly connected component, we would need to wait for this ID value to be updated and to propagate throughout the strongly connected component before beginning another iteration to form a new exploration path. To avoid this, we maintain at each vertex $v$ a list of root ID's $R_v$ of those roots known to be in the same strongly connected component as $v$. If $R_v = \emptyset$, then $v$ has not yet been included in any strongly connected component. If $R_u \cap R_v \neq \emptyset$, then $u$ and $v$ are in the same strongly connected component even if $R_u \neq R_v$. (This can happen due to propagation delays.) Note that it is possible that $R_u \neq \emptyset$, $R_v \neq \emptyset$, and $R_u \cap R_v = \emptyset$ but $u$ and $v$ are in the same strongly connected component and simply not aware of it.

When an exploration cycle/path is discovered, the root $r_i$ may learn of some new roots that are known to be in its strongly connected component, that is, $R_{r_i}$ may increase in size. This new $R_{r_i}$ is propagated through the strongly connected component as follows: First,

$R_{r_i}$ is sent to all of the vertices on the exploration cycle/path as new elements of $S$ are added. Note that as $R_{r_i}$ propagates along the cycle/path, a receiving vertex may, in fact, have learned of some additional roots. In this case, these roots are added to the set $R_{r_i}$ to propagate further. As this message continues around the cycle/path, each vertex also forwards the root ID list to all of its out-neighbors. If such an out-neighbor $w$ has root ID list $R_w$ that is not identical to but shares elements with the incoming root ID list, then $w$ forwards the union of these lists to its out-neighbors. This process continues throughout the strongly connected component.

As we assume that $G$ is strongly connected, the process will continue until no root can locate a new exploration path and all vertices in $G$ have the same root ID list which will contain all of the initiating roots.

## 5.5 Comparisons

Here, we compare the sizes of the twin-dominating sets generated by the approximation algorithms to those of the gemini sets generated by various algorithms and heuristics. We also consider the degradation of graph connectivity resulting from using only those arcs dominated or absorbed by the vertices of the gemini sets. In particular, we investigate the average vertex distance and the average number of arc-disjoint paths between vertex pairs in the original graph and in $\langle S_g \rangle_w$.

As in previous chapters, we generate random digraphs as follows. For any given graph size $n$, we place $n$ vertices on an 800 by 600 plane uniformly at random. We assign a random transmission to a vertex with an expected value pre-calculated to produce a desired graph density. There is a simplex link from vertex $u$ to vertex $v$ if $u$'s transmission range is at least as large as the distance between $u$ and $v$. Two oppositely directed simplex links comprise a duplex link.

We generated two levels of network density for each graph size. The graph sizes that we generated in the simulation vary from 20 to 200 for the sparser graphs and from 40 to 200 for the denser graphs. For each graph size/density combination, we repeated the experiment for 100 times. The transmission ranges in the simulation are the same as those in Chapter 2 to allow comparisons of these results to those for the undirected case. Note that, due to the introduction of unidirectional (simplex) links, the digraphs are always denser than undirected graphs. In particular, we observe that about 15% more unidirectional links must

be added to the undirected graphs of the same size and transmission range.

The twin-dominating set algorithm used in this simulation is $GDT_{SEQ}$ (denoted `TWIN` in the data series). The GDG algorithm (denoted `GDG`) is based on the $GDT_{SEQ}$ algorithm; it adds more vertices if the twin-dominating set returned by $GDT_{SEQ}$ is not yet a gemini set. The $EAR_C$ algorithm (denoted `EARC`) is the centralized heuristic loosely based on ear-decomposition. The $EAR_P$ algorithm (denoted `EARP`) is the multiple initiator distributed implementation of $EAR_C$. In our implementation of $EAR_P$, we randomly choose 5% of the vertices as the initiating roots.

As illustrated in Figures 5.14 and 5.15, twin-dominating sets are always the smallest. As the algorithms become less centralized and more parallel, the gemini sets returned become larger. Note that for the dense digraphs (Figure 5.15), the gemini sets returned by the GDG algorithm are only slightly larger than the twin-dominating sets returned by the $GDT_{SEQ}$ algorithm. Indeed, according to our experiments, the twin-dominating sets generated by $GDT_{SEQ}$ are, in fact, gemini sets in most cases.

We compare the average vertex distances and the average number of arc-disjoint paths in the original digraphs and the sub-digraphs weakly induced by the gemini sets returned by GDG, $EAR_C$, and $EAR_P$ in Figures 5.16, 5.17, 5.18, 5.19. Not surprisingly, we again observe the trade-off between the size of the gemini sets and the connectivity of the weakly-induced subgraphs.

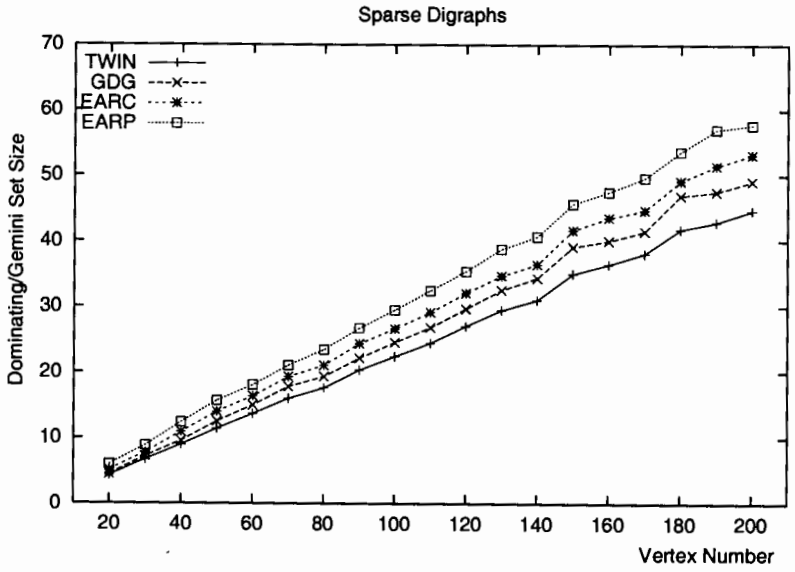



Figure 5.14: Dominating/gemini set size — sparse digraphs.

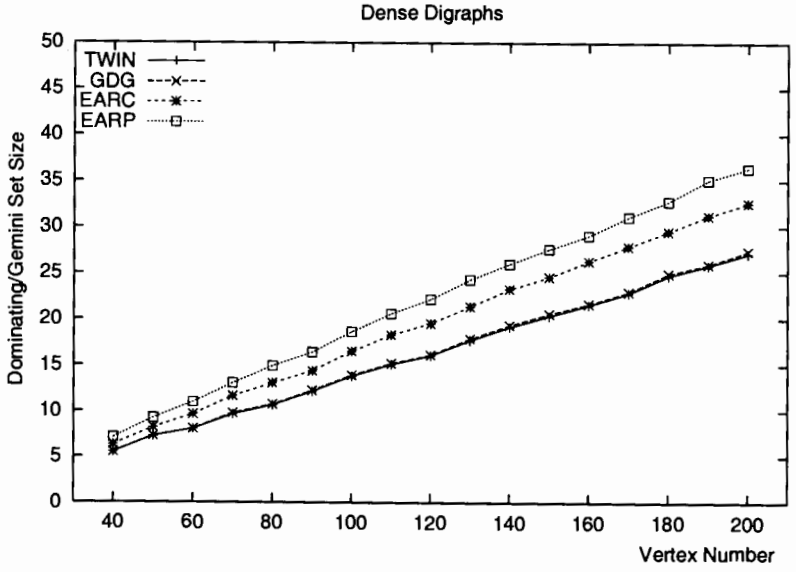



Figure 5.15: Dominating/gemini set size — dense digraphs.

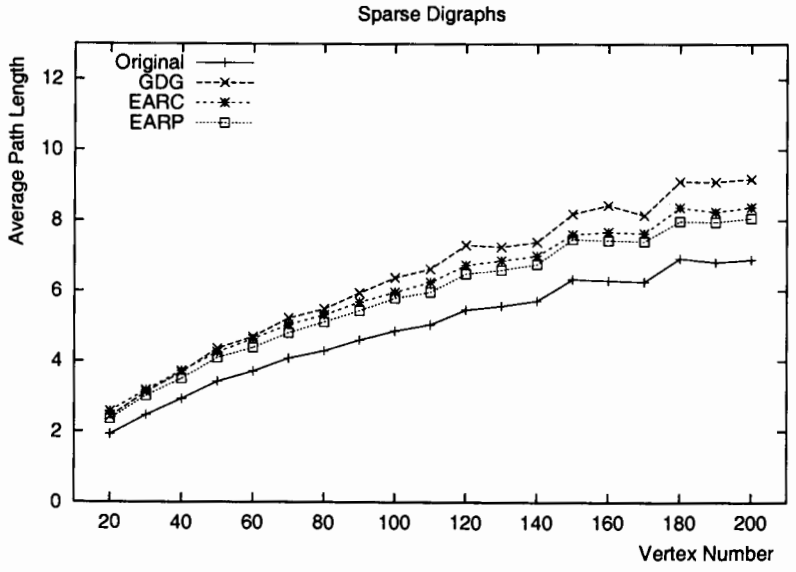



Figure 5.16: Average vertex distance — sparse digraphs.

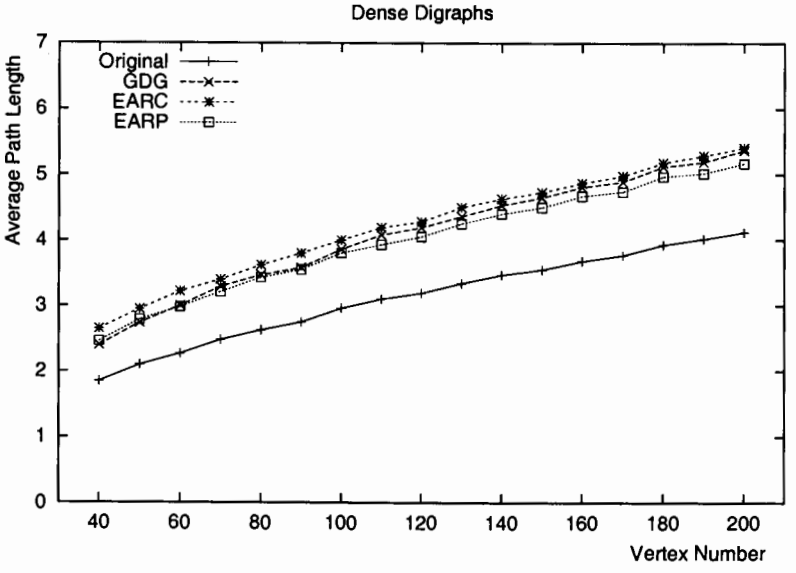



Figure 5.17: Average vertex distance — dense digraphs.

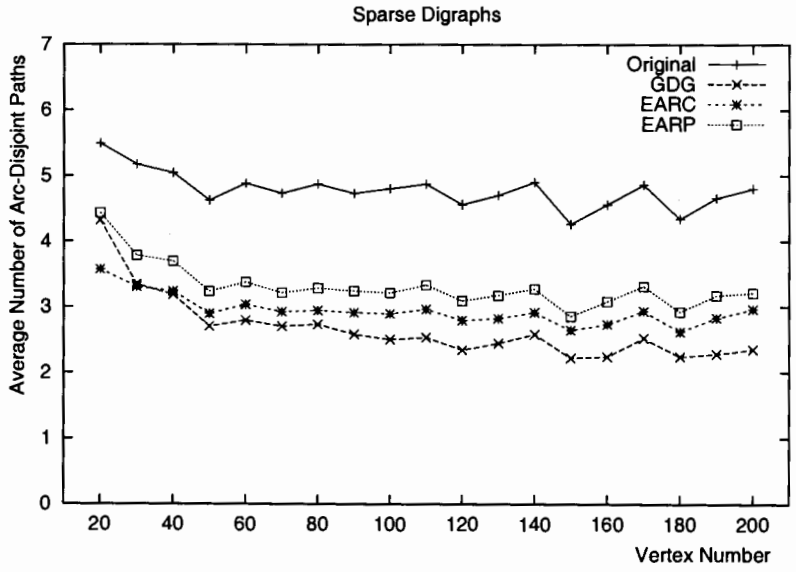



Figure 5.18: Average number of arc-disjoint paths — sparse digraphs.

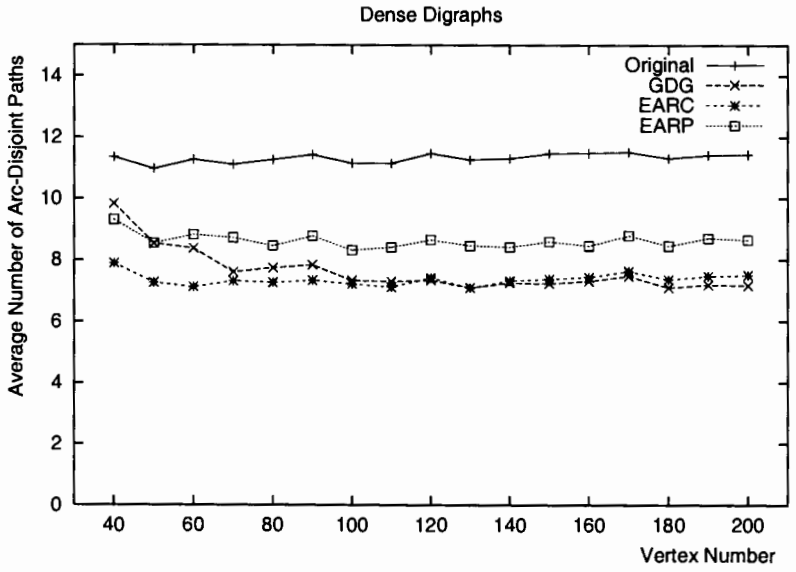



Figure 5.19: Average number of arc-disjoint paths — dense digraphs.

# Chapter 6

# Future Work

The thesis has proposed solutions to some interesting problems on clustering mobile ad hoc networks with duplex and simplex links. There are still many intriguing questions in this area.

## 6.1 Parallel solutions

We were not able to show a non-trivial bound on the approximation ratio of the synchronous parallel algorithm, $DW_{SYNC}$, presented in Chapter 2. The difficulty seems to lie in the combination of greediness and parallelism. When multiple vertices are added to the weakly-connected dominating set being constructed in a single iteration, the techniques used in proving the performance ratios of $GW_M$ and $GW_S$ fail.

The implementations of the $GW_M$, $GW_S$, and $DW_{SYNC}$ algorithms generate identical weakly-connected dominating sets frequently in our experiments. However, we can construct an example in which the size of the weakly-connected dominating set generated by $GW_S$ and $DW_{SYNC}$ differ significantly. For the graph in Figure 6.1, the first iteration of $DW_{SYNC}$

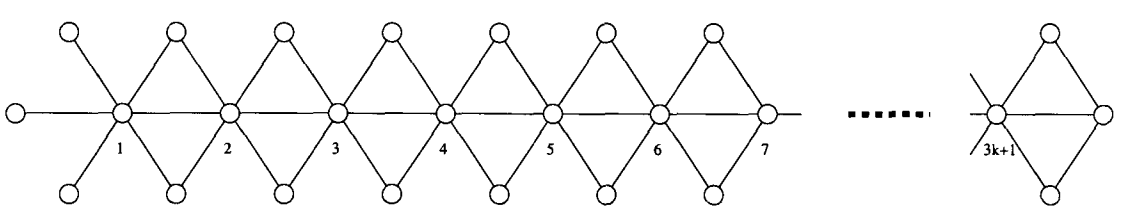



Figure 6.1: Linear difference.

can choose vertices 1, 4, 7, 11, ..., $3k+1$ by setting the vertex ID's to favor these vertices. In later iterations, at least $\frac{k}{2}$ more vertices need to be added. The $\mathrm{GW_M}$ algorithm may choose vertices 1, 3, 5, ..., $2l+1$ in $l+1$ iterations. The ratio between the sizes of these two resultant weakly-connected dominating sets is $\frac{3}{2}$.

Do these parallel solutions have similar logarithmic approximation ratios as the centralized algorithms?

## 6.2 Digraph sparsity

The central idea of the weakly-connected dominating set algorithm of Dubhashi, et al. [25] is to connect the vertices in a small dominating set $S$ by adding at most $2|S|$ more vertices. This is based on a crucial observation that an undirected graph $G$ on $n$ vertices with girth $g$ has at most $n^{1+\frac{2}{g-1}}+n$ edges (Lemma 15.3.2 [54]). Unfortunately, directed graphs do not have such a nice property to allow the techniques of Dubhashi, et al. to be used to construct small gemini sets for ad hoc networks with unidirectional links. Simple generalizations of the notion of "girth" in undirected graphs do not guarantee sparse digraphs.

One natural generalization of girth is the length of the shortest directed cycle. However, there are digraphs on $n$ vertices and of $\Omega(n^2)$ arcs with shortest cycles of length $\Omega(n)$.

For a second generalization of girth, we consider, given an arc $(u,v)$, the length of the shortest path from $u$ to $v$ not containing arc $(u,v)$. If no arc in a digraph has such a path of length under some (relatively large) threshold, one would expect the digraph to be sparse. Again, unfortunately, there are digraphs on $n$ vertices and of at least $\Omega(n^{3/2})$ arcs with the lengths of such paths for all arcs being at least $\Omega(\sqrt{n})$

Are there any good local or sub-global properties to guarantee the sparsity and connectivity of digraphs at the same time?

## 6.3 Modeling mobile ad hoc networks

A fundamental problem in ad hoc networking research is to design a single model which can be used for various problems in this area. Such a general framework would simplify and standardize research approaches. Ideally, a model should be mathematically succinct and manipulable and also precise enough to reflect the nature of ad hoc networks. However, given the inherent complexity of ad hoc networking systems, this does not seem immediately

achievable.

## 6.4 Efficient utilization of unidirectional links

Unidirectional links may occur in actual ad hoc networks due to the technology. Most researchers simply do not try to use these links. We have seen that using these links is possible, although it may incur some overhead. Are there more efficient ways to incorporate these links?

# Bibliography


[1] Hosame Abu-Amara and Arkady Kanevsky. On the complexities of leader election algorithms. In *Proceedings of the Fifth International Conference on Computing and Information (ICCI'93)*, pages 202–206, 1993.

[2] Micah Adler and Christian Scheideler. Efficient communication strategies for ad-hoc wireless networks. In *Symposium on Parallelism in Algorithms and Architectures*, pages 259–568, 1998.

[3] Khaled M. Alzoubi, Peng-Jun Wan, and Ophir Frieder. Distributed heuristics for connected dominating set in wireless ad hoc networks. *KICS Journal of Communications and Networks*, 4(1), March 2002.

[4] Khaled M. Alzoubi, Peng-Jun Wan, and Ophir Frieder. Message-optimal connected dominating sets in mobile ad hoc networks. In *The Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'02)*, pages 157–164, June 2002.

[5] Khaled M. Alzoubi, Peng-Jun Wan, and Ophir Frieder. Weakly-connected dominating sets and sparse spanners in wireless ad hoc networks. In *The 23rd International Conference on Distributed Computing Systems (IEEE, ICDCS)*, May 2003.

[6] A.D. Amis, R. Prakash, T.H.P. Vuong, and D.T. Huynh. Max-min d-cluster formation in wireless ad hoc networks. In *Proceedings of IEEE INFOCOM*, Tel Aviv, March 2000.

[7] Beongku An and Symeon Papavassiliou. A mobility-based clustering approach to support mobility management and multicast routing in mobile ad-hoc wireless networks. *International Journal of Network Management*, 11(6):387–395, 2001.

[8] Onur Arpacioglu, Tara Small, and Zygmunt Haas. Notes on scalability of wireless ad hoc networks. *IETF Internet draft, draft-irtf-ans-scalability-notes-01.txt, December 2003, (Work in Progress).*

[9] Baruch Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems (detailed summary). In *the 19th ACM Symposium on Theory of Computing (STOC)*, pages 230–240, New York City, NY, May 1987.

[10] Dennis J. Baker and Anthony Ephremides. The architectural organization of a mobile radio network via a distributed algorithm. *IEEE Transactions on Communications*, COM-29(11):1694–1701, 1981.

[11] S. Banerjee and S. Khuller. A clustering scheme for hierarchical routing in wireless networks. Technical Report CS-TR-4103, University of Maryland, College Park, February 2000.

[12] Jørgen Bang-Jensen and Gregory Gutin. *Digraphs: theory, algorithms and applications.* Springer, London; New York, 2001.

[13] R. Bar-Yehuda, O. Goldreich, and A. Atai. On the time complexity of broadcasting in radio networks: an exponential gap between determinism and randomization. *Journal of Computer and System Sciences (JCSS)*, 45:104–126, 1992.

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

[15] Elizabeth M. Belding-Royer. Multi-level hierarchies for scalable ad hoc routing. *Wireless Networks*, 9(5):461–478, 2003.

[16] Alan A. Bertossi. Dominating sets for split and bipartite graphs. *Information processing letters*, 19:37–40, 1984.

[17] Vaduvur Bharghavan, Alan J. Demers, Scott Shenker, and Lixia Zhang. MACAW: A media access protocol for wireless LAN's. In *Proceedings of SIGCOMM*, pages 212–225, 1994.

[18] Tracy Camp, Jeff Boleng, and Vanessa Davies. A survey of mobility models for ad hoc network research. *Wireless Communication and Mobile Computing: Special Issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, 2002.

[19] Gary Chartrand, Peter Dankelmann, Michelle Schultz, and Henda C. Swart. Twin domination in digraphs. *Ars Combinatoria*, 67, 2003.

[20] Geng Chen, Fabian Garcia Nocetti, Julio Solano Gonzalez, and Ivan Stojmenovic. Connectivity-based k-hop clustering in wireless networks. In *Proceedings of the 35th Hawaii International Conference on System Sciences (HICSS-35)*, January 2002.

[21] Yuanzhu Peter Chen and Arthur L. Liestman. Approximating minimum size weakly-connected dominating sets for clustering mobile ad hoc networks. In *The Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'02)*, pages 165–172, June 2002.

[22] Yuanzhu Peter Chen and Arthur L. Liestman. A zonal algorithm for clustering ad hoc networks. *International Journal of Foundations of Computer Science*, 14(2):305–322, 2003.

[23] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Mathematics*, 85:165–177, 1990.

[24] Bevan Das and Vaduvur Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *IEEE International Conference on Communications (ICC'97), Vol.1*, pages 376–380, June 1997.

[25] Devdatt Dubhashi, Alessandro Mei, Alessandro Panconesi, Jaikumar Radhakrishnan, and Aravind Srinivasan. Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 717–724, 2003.

[26] U. Feige. A threshold of ln n for approximating set cover. In *28th ACM Symposium on Theory of Computing*, pages 314–318, 1996.

[27] James A. Freebersyser and Barry Leiner. A DoD perspective on mobile ad hoc networks. In Charles Perkins, editor, *Ad Hoc Networking*. Addison-Wesley, 2001.

[28] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning tree. *ACM Transactions on Programming Languages and Systems*, 5(1):66–77, January 1983.

[29] M. L. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, 1979.

[30] M. Gerla, L. Kleinrock, and Y. Afek. A distributed routing algorithm for unidirectional networks. In *Proceedings of IEEE GLOBECOM*, pages 654–658, 1983.

[31] Mario Gerla, Taek Jin Kwon, and Guangyu Pei. On demand routing in large ad hoc wireless networks with passive clustering. In *Proceedings of IEEE WCNC*, September 2000.

[32] Mario Gerla and Jack Tzu-Chieh Tsai. Multicluster, mobile, multimedia radio network. *Wireless Networks*, 1(3):255–265, 1995.

[33] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs.* Academic Press, San Diego, CA, 1980.

[34] Sudipto Guha and Samir Khuller. Approximation algorithms for connected dominating sets. Technical Report 3660, University of Maryland, College Park, June 1996.

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

[36] Peter Gvozdjac. Modeling communications in low-earth-orbit satellite networks. Technical report, Ph.D. Thesis. School of Computing Science, Simon Fraser University, August 2000.

[37] Jaap Haartsen. Bluetooth – the universal radio interface for ad hoc, wireless connectivity. *Ericsson Reviews*, (3):110–117, 1998.

[38] Zygmunt J. Haas and Marc R. Pearlman. ZRP: A hybrid framework for routing in ad hoc networks. In Charles Perkins, editor, *Ad Hoc Networking.* Addison-Wesley, 2001.

[39] Teresa W. Haynes, Stephen T. Hedetniemi, and Peter J. Slater. *Domination in graphs, Advanced Topics.* Marcel Dekker, Inc., 1998.

[40] Teresa W. Haynes, Stephen T. Hedetniemi, and Peter J. Slater. *Fundamentals of Domination in Graphs.* Marcel Dekker, Inc., 1998.

[41] Ting-Chao Hou and Tsu-Jane Tsai. An access-based clustering protocol for multihop wireless ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 19(7):1201–1210, July 2001.

[42] Lujun Jia, Rajmohan Rajaraman, and Torsten Suel. An efficient distributed algorithm for constructing small dominating sets. In *Proceedings of the 20th ACM Symposium on Principles of Distributed Computing (PODC'01)*, 2001.

[43] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad-hoc wireless networks. In Mobile Computing, editor, *T. Imielinski and H. Korth*, pages 153–181. Kluwer Academic Publishers, 1996.

[44] Robert E. Kahn, Steven A. Gronemeyer, Jerry Burchfiel, and Ronald C. Kunzelman. Advances in packet radio technology. *Proceedings of the IEEE*, 66(11):1468–1496, November 1978.

[45] Phil Karn. MACA – a new channel access method for packet radio. In *Proceedings of the 9th ARRL Computer Networking Conference*, pages 76–84, London, Ontario, Canada, 1990.

[46] Dariusz R. Kowalski and Andrzej Pelc. Deterministic broadcasting time in radio networks of unknown topology. In *Proc. 43rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 63–72, 2002.

[47] P. Krishna, N. H. Vaidya, M. Chatterjee, and D. K. Pradhan. A cluster-based approach for routing in dynamic networks. *Computer Communication Review*, 49:49–64, 1997.

[48] Xiang-Yang Li. Topology control in wireless ad hoc networks. In Stefano Basagni, Marco Conti, Silvia Giordano, and Ivan Stojmenovic, editors, *Ad Hoc Networking.* IEEE Press, 2003.

[49] Ben Liang and Zygmunt J. Haas. Virtual backbone generation and maintenance in ad hoc network mobility management. In *INFOCOM*, pages 1293–1302, 2000.

[50] Chunhung Richard Lin and Mario Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal on Selected Areas in Communications*, 15(7):1265–1275, September 1997.

[51] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41:960–981, 1994.

[52] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, Inc., San Francisco, California, 1996.

[53] Mahesh K. Marina and Samir Das. Routing performance in the presence of unidirectional links in multihop wireless networks. In *The Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'02)*, pages 12–23, June 2002.

[54] Jiri Matousek. *Lectures on discrete geometry. Graduate Texts in Mathematics.* Springer, New York, 2002.

[55] Shree Murthy and J. J. Garcia-Luna-Aceves. An efficient routing protocol for wireless networks. *Mobile Networks and Applications*, 1(2):183–197, 1996.

[56] Sanket Nesargi and Ravi Prakash. A tunneling approach to routing with unidirectional links in mobile ad-hoc networks. In *Proceedings of the IEEE International Conference on Computer Communications and Networks (ICCCN)*, October 2000.

[57] S.Y. Ni, Y.C. Chen, and J.P. Sheu. The broadcasting storm problem in a mobile ad hoc network. In *Proceedings of MobiCom*, pages 151–162, Seattle, 8 1999.

[58] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of INFOCOM*, April 1997.

[59] David Peleg. *Distributed Computing, a Locality-Sensitive Approach.* SIAM, Philadelphia, PA, 2000.

[60] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Computer Communication Review*, pages 234–244, 1994.

[61] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. *IETF Internet draft, draft-perkins-manet-aodvbis-00.txt*, October 2003, (Work in Progress).

[62] C. E. Perkins, E. M. Royer, and S. R. Das. Ad-hoc on-demand distance vector routing. In *Proceedings of 2nd IEEE Workshop on Mobile Computing Systems and Applications*, February 1999.

[63] Charles Perkins, editor. *Ad Hoc Networking.* Addison-Wesley, 2001.

[64] Rajmohan Rajaraman. Topology control and routing in ad hoc networks: A survey. *SIGACT News*, 33:60–73, June 2002.

[65] Venugopalan Ramasubramanian, Ranveer Chandra, and Daniel Mosse. Providing a bidirectional abstraction for unidirectional ad hoc networks. In *Proceedings of INFOCOM*, 2002.

[66] Elizabeth M. Royer and Chai-Keong Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications*, 6(2):46–55, April 1999.

[67] Raghupathy Sivakumar, Prasun Sinha, and Vaduvur Bharghavan. CEDAR: a core-extraction distributed ad hoc routing algorithm. *IEEE Journal on Selected Areas in Communications*, 17(8):1454–1465, August 1999.

[68] Martha Steenstrup. Cluster-based networks. In Charles Perkins, editor, *Ad Hoc Networking.* Addison-Wesley, 2001.

[69] Ivan Stojmenovic, Mahtab Seddigh, and Jovisa Zunic. Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):14–25, January 2002.

[70] John Sucec and Ivan Marsic. Hierarchical routing overhead in mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 3(1):46–56, January-March 2004.

[71] Peng-Jun Wan, Khaled M. Alzoubi, and Ophir Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. In *IEEE INFOCOM*, June 2002.

[72] Douglas West. *Introduction to Graph Theory – Second edition.* Prentice Hall, Upper Saddle River, N.J., 2001.

[73] Jie Wu and Hailan Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *DIAL-M'99*, pages 7–14, Seattle, 1999.

[74] Jie Wu and Hailan Li. A dominating-set-based routing scheme in ad hoc wireless networks. *Special Issue on Wireless Networks, Telecommunication Systems Journal*, 3:63–84, 2001.

# Index