TIMER-BASED PROTOCOLS IN AD HOC AND SENSOR NETWORKS

Except where reference is made to the work of others, the work described in this dissertation is my own or was done in collaboration with my advisory committee. This dissertation does not include proprietary or classified information.

_____

Bonam Kim

Certificate of Approval:

_____	_____
Chung-Wei Lee Assistant Professor Computer Science and Software Engineering	Min-Te Sun, Chair Assistant Professor Computer Science and Software Engineering
_____	_____
Yu Wang Assistant Professor Computer Science and Software Engineering	Stephen L. McFarland Acting Dean Graduate School

TIMER-BASED PROTOCOLS IN AD HOC AND SENSOR NETWORKS

Bonam Kim

A Dissertation

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Doctor of Philosophy

Auburn, Alabama
August 7, 2006

TIMER-BASED PROTOCOLS IN AD HOC AND SENSOR NETWORKS

Bonam Kim

_____

Signature of Author

_____

Date of Graduation

VITA

Bonam Kim was born in Seoul, Korea, on May 20, 1968. After completing high school at Seoul, Korea in 1987, she attended Dankook university in Seoul, Korea from 1987 to 1993. She graduated with a B.A. in 1991 and a M.S. in 1993. She then entered the Auburn University at Auburn in the Spring of 2000. She received a M.S. in Computer Science and Software Engineering from Auburn University in August 2003, and a Ph.D. in Computer Science and Software Engineering from Auburn University in August 2006.

DISSERTATION ABSTRACT

TIMER-BASED PROTOCOLS IN AD HOC AND SENSOR NETWORKS

Bonam Kim

Doctor of Philosophy, August 7, 2006
(M.A., Auburn University, 2003)
(B.S., Dankook University, 1991)

136 Typed Pages

Directed by Min-Te Sun

Wireless ad hoc and sensor networks present the next great challenge for distributed system research. In wireless ad hoc and sensor networks, each node participates the routing process that allows a packet to be forwarded from its source to the destination. Protocols that support communications in such networks have to take into account the mobility of the participants and the status of links between nodes.

In general, a distributed system is defined as a collection of autonomous components that are interconnected through a network and distributed middleware. A distributed system coordinates the events of components to share resources and give users the perception that the whole system is a single and integrated computing facility. According to this definition, wireless ad hoc and sensor networks can be considered as a special type of distributed systems where each node in the networks serves an autonomous component. In theory, the concepts involved in the design of distributed system should be easily adapted for wireless ad hoc and sensor networks.

In the design of a distributed system, the order of events plays an important role. To manipulate the order, one of the the well-known tools is the (defer) timer. Many distributed communication protocols have been designed using this tool.

In this dissertation, we investigated the use of the defer timer on the design of various protocols in wireless ad hoc and sensor networks. By properly setting up the defer timers, many difficult issues in ad hoc and sensor networks, such as the broadcast storm problem, the construction of a virtual backbone, and dynamic cluster formation can be easily addressed with only the help of simple localized information at each node. The simulations show the suggested timer-based protocols perform effectively in wireless ad hoc and sensor networks.

ACKNOWLEDGMENTS

This dissertation is dedicated to my parents who taught me to love learning and who always made my education one of their top priorities.

I would like to take the opportunity to thank people who guided and supported me during this process. Without their contributions, this research would not have been possible. First, I would like to thank my Ph.D. advisor, Dr. Min-Te Sun. I think in the past three years, Dr. Sun taught me how to do the research. In other words, he taught me how to make gold out of stones. Besides this, he was always patient and helpful whenever his guidance and assistant were needed. As an international student, I felt a certain difficulty in technical writing. Dr. Sun also contributed much of his valuable time to help me improve my writing skills. Therefore, I do want to show my deep appreciation to him. I am also grateful to Dr. Chung-Wei Lee and Dr. Yu Wang for serving on my Ph.D. committee. I really appreciate Dr. John Cochran for spending time reviewing this work and giving valuable suggestions and comments on my work.

There were helps from people in research lab in and outside of Auburn University. I can still remember the time that my former advisor, Dr. Kyungsan Cho Wells worked together with me at Dankook University in Korea. He was the first one that introduced the amazing world to me and encouraged me to explore the wonderful nature. I am also deeply grateful to my colleague, Junmo Yang who worked with me on this dissertation from the very beginning, and cared about it as though it were his own.

Additionally, I would like to show my great appreciation to my parents and mother-in-law for their continuous support during all these years. My brother and sisters always guide me during my explorations up to today, using their valuable experience and successful examples. Last, but not least, my husband, Myeongjoo Kang and my sun, Hyunkoo Kang, also contributed much of their time and efforts to support me during my study. Without any one of them, the work would not have been possible.

Style manual or journal used Journal of Approximation Theory (together with the style known as "aums"). Bibliograpy follows van Leunen's *A Handbook for Scholars*.

Computer software used The document preparation package TEX (specifically LATEX) together with the departmental style-file `aums.sty`.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

A wireless ad hoc network is a collection of mobile nodes such as handheld devices, mobile phones, and automotive telematics systems that communicate with each other by forming a multihop radio network. In such a network, each node functions as both a host and a router, and the control of the network is distributed among the nodes. The network topology (i.e., the connections between mobile nodes) in an ad hoc network is generally created and destroyed dynamically due to the departures and arrivals of mobile nodes.

The major advantage of ad hoc networks is that a fixed infrastructure is not needed at the time of network deployment. This makes ad hoc networks the best candidate for applications such as emergency services and disaster recovery, ad hoc meetings at business conventions, and communications in the battlefield. In the late 1990's, the so-called "smart dust" project [1, 2] at Berkeley introduced the concept of the sensor networks. As one of the major applications of wireless ad hoc networks, sensor networks share a similar definition, but impose more stringent resource constraints at each sensor node.

Many research issues have been raised concerning wireless ad hoc and sensor networks. These topics include virtual backbone construction [4, 5, 6, 7, 8], routing [9, 10, 11, 12, 13, 14], broadcast [15, 16, 17, 18], and network formation [19, 20, 21, 22, 23, 24, 25, 26]. Each requires a carefully designed protocol to address the issue, and many considerations and constraints are involved in the design of such a protocol. In addition to problems

such as wireless transmission errors, dynamic network topology, and power limitations, the following two key constraints have made the protocol design process particularly difficult:

1. The protocol should be distributed - In wireless ad hoc and sensor networks, each node acts both independently and collectively to accomplish the tasks. As opposed to wired networks, no central control is available in such networks.

2. The protocol should rely on only localized information - If a protocol requires global knowledge (e.g., the complete network topology), there will be tremendous overhead associated with learning such knowledge. Even if it were possible, it would require a lot of time and by the time the global knowledge had been learned, it would already be outdated.

Therefore, wireless ad hoc and sensor networks are generally considered as a special type of distributed systems. In theory, the concepts involved in the design of distributed systems should be easily adopted for wireless ad hoc and sensor networks.

In the design of a distributed system [27], the order of events plays an important role. For instance, assume that two events occur in a distributed system. If the first event needs the result of the second as its input, then a temporal correlation is formed. If the protocol has the second event scheduled to finish first, then these two events will never be completed. Additionally, if a protocol assigns high priority to the second and low priority to the first, the performance (i.e., the time needed to complete both events) is likely to be worse than if the protocol assigns high priority to the first and low priority to the second.

Note that although a physical clock is commonly available at each component in a distributed system, it cannot be used to manipulate the order of events for two reasons. First, it is difficult to align clocks between components precisely. Second, the delay caused by exchanges of clock information is difficult to estimate. Instead, the logical clock [28, 29, 30] and (defer) timer [31, 32] are commonly used to achieve these goals in a distributed system.

This dissertation is to investigate the use of a defer timer in the design of protocols for a set of problems in ad hoc and sensor networks. Although a defer timer has been used in some of existing network protocols, its power has yet to be fully utilized. Second, the type of research problems in wireless ad hoc and sensor networks that may benefit from the uses of a defer timer will be identified. Furthermore, several examples will be provided to demonstrate the power of a defer timer in the design of protocols for wireless ad hoc and sensor networks. By properly setting up the defer timers, many difficult issues in ad hoc and sensor networks, such as the broadcast storm problem [15, 16], the construction of the virtual backbone [4, 5, 6, 7, 8], and dynamic cluster formation [26, 33, 35] can be easily tackled with only the help of simple localized information at each node.

The remaining chapters are organized as follows. Chapter 2 presents background and general issues concerning current wireless networks. Chapter 3 introduces the order of events in a distributed system. The concepts of clock synchronization, physical and logical clocks, and defer timer are described and examples of existing defer timer based protocols are discussed. In Chapter 4, our proposed timer based dominating set construction

protocols are presented. In each section, different timer based DS/CDS protocols are discussed. A summary of the dissertation, the conclusions reached contributions to the field, and suggestions for future research are given in Chapter 5.

CHAPTER 2

BACKGROUND AND GENERAL ISSUE

This chapter provides an overview of wireless ad hoc networks. There are three types of wireless ad hoc networks: mesh networks, mobile ad hoc networks, and sensor networks. Each has important applications and supports a different degree of mobility for wireless devices. Section 2.1 discusses the relevant background and an overview of wireless ad hoc networks. Sections 2.2, 2.3, and 2.4 provide the definition, the standards, and research issues for each type of ad hoc network. In Section 2.5, the relationship between wireless ad hoc networks and distributed systems is presented.

2.1 Overview of wireless ad hoc networks

Like traditional wired networks, wireless networks are formed by routers and hosts. Typically, the routers are responsible for forwarding packets in the network and hosts may be sources or sinks of data flows. In a wireless network, the link between different network components can be wireless. This allows the components in wireless networks to enjoy a higher degree of travel freedom than those in wired networks. Wireless networks operate on radio frequencies (RF), and different RF spectrums are used for different wireless networks. The wireless links that operate in different parts of the spectrum demonstrate distinctly different physical characteristics. These characteristics, such as transmission range, power consumption, and propagation model (e.g., omnidirectional or directional), influence the design of an appropriate wireless protocol.

As a special types of wireless network, an ad hoc network consists of wireless devices that form temporary links at times when they are in close proximity. Each node in an ad hoc network acts not only as a host to generate or receive packets, but also as a router to help forward packets toward the destination. Due to the constraints imposed by power limitations or the implemented standards defined by various industry committees, each node in an ad hoc network has limited transmission range. As a result, a packet in an ad hoc network is likely to travel through several hops before it reaches its final destination.

An ad hoc network is both self-organizing and adaptive. This means that the network can adjust itself to changes in the environment. An ad hoc network is also standalone, which means that the operation of an ad hoc network does not depend on a pre-installed network infrastructure. A device in an ad hoc network must have the ability to detect the presence of other devices in order to communicate and share information and services with them. Depending on the application, the devices in an ad hoc network can vary significantly, from personal digital assistants (PDAs),and laptops to Internet mobile phones. Different type of devices offer very distinct attributes, including their computation, storage, and communications capabilities.

2.2 Wireless Mesh Network

Also referred to as community networks, wireless mesh networks are a special type of wireless ad hoc network that consists of wireless access points within a community that are designed to provide an alternative wireless communications infrastructure for the community's residents. Thanks to the widespread acceptance of the home networking concept, it

is now common for people to install a wireless access point for several Internet appliances (e.g., one or more personal computers, PDAs, printers, and gaming consoles) to share a single commercial broadband connection. Wireless mesh networks interconnect the access points across households to form a large multi-hop wireless network. This offers two advantages. First, instead of having each household subscribe to the Internet service, only one or few broadband connections are needed for the entire community if a wireless mesh network is in place. Second, wireless mesh networks extend the Internet coverage to areas where the original broadband service is not available.

A wireless mesh network is generally built on top of home networks, which are typically wireless local area networks (WLANs). A WLAN provides high data rate connections in a local area to the Internet. Most WLANs operate in unlicensed bands that are free of charge and rigorously regulated. The Institute of Electrical and Electronics Engineers (IEEE), European Telecommunications Standards Institute (ETSI), and HomeRF Working Group (HomeRF WG) have all been involved in developing standards for WLANs, but the ones that dominate the market are from IEEE. Currently there are four IEEE specifications for WLAN: 802.11b, 802.11a, 802.11g, and 802.11n. The WLANs based on these specifications are the building blocks for wireless mesh networks.

- IEEE 802.11b (also referred to as Wi-Fi)[36, 37, 38, 39] – IEEE 802.11b supports transmission rates of up to 11 Mbps within a range of 30 to 75 meters. This is comparable to a traditional Ethernet. IEEE 802.11b uses the unlicensed 2.4 GHz Industrial, Scientific, and Medical (ISM) spectrum. There are two immediate consequences of using an unlicensed band. First, the transmissions in a IEEE 802.11b

network are prone to interference from other devices utilizing the same spectrum, such as microwave ovens and cordless phones. Second, the transmission power of a IEEE 802.11b device has to meet certain regulation so that it will not be harmful to other devices using the same unlicensed band. To overcome these issues, a spread spectrum is primarily used in IEEE 802.11b. The choice of the unlicensed band for 802.11b is favorable to vendors because it greatly reduces their production costs.

- IEEE 802.11a [36, 37, 38, 40] – IEEE 802.11a supports high transmission rates of up to 54 Mbps within a range of 25 to 50 meters. Unlike IEEE 802.11b, IEEE 802.11a does not use a spread spectrum scheme, but rather uses an orthogonal frequency division multiplexing (OFDM) that operates in the 5 Ghz band. Due to the choice of the 5 GHz spectrum, IEEE 802.11a devices suffer from much less radio frequency (RF) interference than others (e.g. IEEE 802.11b) that utilize ISM spectrum. With high data rates and relatively less interference, IEEE 802.11a is especially suited to supporting multimedia applications in densely populated user environments.

- IEEE 802.11g [36, 37, 38] – Developed later than IEEE either 802.11b or 802.11a, IEEE 802.11g is an attempt benefit from the positive aspects of both the earlier standards. IEEE 802.11g supports a bandwidth of up to 54 Mbps within a range of 30 to 300 meters, and uses the 2.4 Ghz ISM spectrum. In place of the spread spectrum scheme, IEEE 802.11g is based on the use of OFDM modulation. IEEE 802.11g is compatible with 802.11b, meaning that 802.11g access points (AP) will also work with 802.11b wireless network adapters and vice versa.

- IEEE 802.11n [36, 37, 38]- IEEE 802.11n builds upon previous 802.11 standards by incorporating multiple-input multiple-output (MIMO) technology. IEEE 802.11n offers especially high transmission rates of 100Mbps to 200Mbps.

To improve the performance of a wireless mesh network, the access point is usually assumed to be equip with multiple wireless interfaces built on either the same or different WLAN technologies. The primary research issue in mesh networks is how to take advantage of the availability of multiple wireless interfaces at each access point to maximize the communication throughput [41].

2.3 Mobile Ad Hoc Networks

A mobile ad hoc network (MANET) is a wireless ad hoc network consisting of mobile hosts. In such a network, the network topology may change frequently. MANETs offer the most convenient tool for providing communications capability for mobile users at locations where a fixed network infrastructure is not available, such as in disaster recovery after a catastrophe and communications in the battlefield.

Supporting communications in a MANET is difficult due to the nature of its highly dynamic topology, which results in several challenges for the protocol design. First, wireless links will be created and destroyed from time to time as nodes join and leave the network. The connectivity between neighbors thus needs to be updated promptly for any protocol to function. Second, the network may be disconnected into groups at times so that a destination that was previously connected can become unreachable. These challenges render

traditional wired network protocols useless for MANETs. Three important research issues are critical in supporting communications in MANETs:

- Broadcasts - Broadcasting is a common operation in many applications. It is also widely used to resolve many network layer problems. For instance, in dynamic source routing protocols [42], a source broadcasts a query to find a path to the destination. In a MANET, because the network topology changes from time to time, it is expected that broadcasts must be performed more frequently. Broadcasts may also be used in LAN emulation [43] or serve as a last resort for providing multicast services in networks with rapidly changing topologies.

- Virtual backbone construction - As a special type of ad hoc network, the operation of MANETs assumes that no physical infrastructure is available, which increases the cost of communications. To reduce this cost, the use of virtual backbone schemes has been proposed [44]. In these schemes, the virtual backbone works in the similar manner as a physical backbone so that the resources (e.g., bandwidth and storage overhead) can be better utilized. Virtual backbones can be used to collect topology information for routing, to provide a backup route, and to multicast or broadcast messages. In general, the overhead associated with virtual backbone construction and maintenance is proportional to the size of the virtual backbone. In order to reduce the overhead, the size of the virtual backbone should be as small as possible.

Additionally, due to the functions of the virtual backbone, it is expected that the virtual backbone is connected. Hence, a connected dominating set (CDS) is considered a good candidate for use of a virtual backbone in MANETs.

- Network formation - Network formation problem is sometimes referred to as the cluster formation problem. There are two reasons why network formation is important in MANETs. First, some wireless standards that use frequency hopping (e.g., bluetooth) allow nodes to select a subset of nearby nodes as its neighbors. Hence, the network topology is determined by not only the locations of nodes, but also the network formation protocol. Second, a task may require the collaboration of several nodes to complete. In such a case, a protocol is needed to assign nodes appropriately for the incoming task.

Mobile ad hoc networks are expected to become an integral part of the future 4G architecture [45], which aims to provide pervasive computing environments that will enable support users to accomplish their tasks, access information, and communicate anytime, anywhere from any device. Currently, the specifications that support mobile ad hoc networks include the IEEE 802.11 family, IEEE 802.16e[46], and IEEE 802.20[47].

- IEEE 802.11 family - all the IEEE 802.11 standards support ad hoc modes for nodes in close proximity to each other. This scenario can be considered as single-hop MANETs.

- IEEE 802.16e [46]- IEEE 802.16e supports a transmission rate of 30 Mbps within the range of 50 km, achieving this by using Scalable OFDM (SOFDM) over the 2 to

6 GHz licensed bands. IEEE 802.16e supports broadband wireless access (BWA) for mobile users and is an amendment to IEEE 802.16, which is a standard defined for wireless metropolitan area networks (MANs). Coupled with 802.16e, IEEE 802.16 aims at supporting both fixed and mobile BWA in a metropolitan area with a single base station.

- IEEE 802.20 [47]- The IEEE 802.20 standard specifies new mobile air interfaces for wireless broadband. This specification fills the performance gap between the high data-rate low mobility services currently regulated by the IEEE 802.11 family and high mobility cellular networks. IEEE 802.20 focuses primarily on long-range and high-speed mobile connectivity and is designed to work in narrow bands that can be scavenged from amidst the existing spectrum allocations. The network is based on proprietary Flash-OFDM technology to deliver data to users. It operates using licensed bands below 3.5GHz and supports a transmission rate of 1 Mbps within a range of up to 250 km. Essentially, IEEE 802.16e supports nodes with lower mobility (e.g., a mobile user walking around carrying a PDA or laptop) over a wide area, while IEEE 802.20 focuses on high-speed mobility issues (e.g.,users in vehicles).

2.4 Wireless Sensor Networks

Wireless sensor networks are a special class of ad hoc networks that are used to provide a wireless communication infrastructure among sensors deployed in a specific application domain. A sensor network is composed of a large number of sensor nodes that are

densely populated either inside the phenomenon or very close to it. The positions of the sensor nodes need not be engineered or predetermined, allowing random deployment in inaccessible terrain or disaster relief operations.

Each node in a sensor network consists of three subsystems: the sensor subsystem which senses the environment, the processing subsystem which performs local computations on the sensed data, and the communication subsystem which is responsible for message exchanges with neighboring sensor nodes. While individual sensors have only a limited sensing region, processing power, and energy, networking a large number of sensors can result in a robust, reliable, and accurate sensor network covering a wide region. The types of sensors range from small passive microsensors (e.g, "smart dust") to larger scale, controllable weather-sensing platforms. At present, there are several standard and proprietary devices that support sensor networks. IEEE 802.15.1 (Bluetooth) and IEEE 802.15.4 (Zigbee) are the most promising standards for wireless sensor networks because Bluetooth and Zigbee devices are generally inexpensive and consume relatively little power, but motes [1], designed primarily by UC-Berkeley, have also been adopted by many sensor network applications.

- IEEE 802.15.1 (Bluetooth)[48] - Initially developed by the Bluetooth special interest group, Bluetooth is a wireless specification for wireless personal area networks (WPANs), which has characteristics such as short-range, low power, and low cost. Operating on the 2.4 GHz unlicensed ISM band, Bluetooth supports data rates up to 2.178 Mbps within distances of up to 100 m.

- IEEE802.15.4 (Zigbee)[48] - Initially developed by the Zigbee alliance, ZigBee is designed to support low data rate, low power consumption, and low cost wireless communications. The primary applications of Zigbee include automation and remote control. It supports a data rate of 250 kbps using 2.4 GHz unlicensed bands within a range of 10 to 75 m.

- Motes [1, 2]- A mote is a small, low cost, and low power device that incorporates a CPU, radio transmitter, and associated hardware to support its sensing capability. For example, the MICA mote is a commercially available product that has been widely used by researchers and developers. The MICA mote uses an Atmel ATmega 128L processor [1, 2] operating at 4 MHz. The 128L is an 8 bit microcontroller that has 128 KB of onboard flash memory to store the mote's program and maintain/buffer the collection of sensor measurements.

2.5 Wireless Mobile Ad Hoc and Sensor Networks and Distributed Systems

Due to the nature of wireless mobile ad hoc and sensor networks, it is impractical to assume the availability of a central coordinator or global view when designing a network protocol. In such networks, each station makes decisions independently, based strictly on localized information, in order to collectively achieve a goal for the whole network or resolve a global issue.

In general, a distributed system is defined as a collection of autonomous components that are interconnected through a network and distributed middleware. A distributed system

coordinates the events of components to share resources and give users the perception that the whole system is a single and integrated computing facility. Based on this definition, wireless mobile ad hoc and sensor networks are clearly a special type of distributed system.

In a distributed system [27], the order of events greatly influences the performance of the system. To manipulate the order, the best-known tools are the logical clock (for soft synchronization) [28, 29, 30] and the (defer) timer [31, 32]and many distributed communication protocols have been designed using these tools. However, when it comes to the field of wireless ad hoc and sensor networks, the uses of the defer timer have not yet been fully exploited. Part of the reason seems to be because the term *defer* intuitively infers a longer delay for communications, which is in general not a desirable property in the design of a network protocol. This is, however, not necessarily the case in many applications as will be shown in later chapters. In fact, the similarity between wireless mobile ad hoc and sensor networks and distributed systems suggests that the tools used in the design of distributed systems, such as the defer timer, could be borrowed to improve the protocol design of wireless ad hoc and sensor networks.

CHAPTER 3

THE ORDER OF EVENTS IN A DISTRIBUTED SYSTEM

At first glance, it seems easy to assign an order to a list of events in a distributed system by simply utilizing clock information. However, the situation is actually more complicated than it appears because typically the processes that trigger the events in the system are running on different components. While it is common for each component (e.g., a mobile station in a wireless ad hoc network) to incorporate a physical clock, these clocks are not necessarily synchronized; there is no global clock among the components in a system. In fact, even a simple status update across different components can be difficult lacking a global clock because it is hard to determine which update request is the most recent. Even if the components in the system can somehow be synchronized at one point in time, the global clock cannot be maintained for long because each physical clock in the different component is running at a slightly different speed. These clocks will gradually drift away from each other as time goes by. Even if all the components share a global clock, unpredictable delays (e.g., switching and propagation delays) incurred during data transmissions can still be a problem for the design of network protocols. For instance, assume two components are requesting a common resource from the system. The request sent earlier may arrive at the destination later due to unpredictable delays. In such cases, regardless of whether the components share a global clock, the order of events cannot be assigned by a simple exchanges of messages (i.e., by sending requests and expecting the resource to be granted based on the times the requests are sent).

3.1 Logical Clock

Without loss of generality, assume that a distributed system is composed of n components denoted by P_1, P_2, ..., P_n. For a component P_i, C_i denotes the logical clock associated with P_i. The logical clock of a component, which is typically implemented by a counter, *ticks* (i.e., increments its value) every time an event happens at the component. The primary difference between a logical clock and its physical counterpart is that the value held by a logical clock does not directly reflect the real physical time. In a distributed system, discrete event is defined as either the beginning or end of a process at a component. For instance, the execution of a procedure, the message transmission, and receiving are all considered events. If an event e happens at P_i, the timestamp of the event, denoted by $C(e)$, is defined as the value of the logical clock C_i at the time the event happens. Before a component sends a message, it puts a timestamp on the message using its own logical clock.

The following rules, known as Lamport's scheme [49], are commonly adopted for the implementation of logical clocks:

- Let a and b be two successive events in P_i, then $C(a) < C(b)$.
- Let event a be the sending of message m by component P_i and event b be the receiving of m by component P_j, then $C(a) < C(b)$.

Logical clocks are based on artificial time rather than real-time. Instead of utilizing a physical clock or assuming a global clock at each component, logical clocks can be used to

determine a relative order of events. The following subsections will briefly introduce two logical clocks that are commonly used.

3.1.1 Lamport Clock

The local time at which each event occurs at different network components cannot be used to determine the order of events due to the absence of perfectly synchronized clocks and global time in a distributed system. However, Lamport's logical clock allows us to overcome these difficulties to learn the order of events in certain scenarios.

Lamport's logical clock is based on the idea of capturing the happened-before relationship. If event a must happens before event b, then $C(a) > C(b)$. The benefit of the happened-before relationship is its lack of dependence on physical clocks. A clock is considered accurate if all happened-before relationships are obeyed.

The happened-before relation captures the casual dependencies between events, i.e., whether two events are causally related or not. The relation, denoted by $\rightarrow$, is defined as follows:

- $a \rightarrow b$, if a and b are events in the same component and a occurred before b.
- $a \rightarrow b$, if a is the event of sending a message m by a component and b is the event of reception of the same message m by another component.
- if $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$, i.e., the " $\rightarrow$ " relation is transitive.

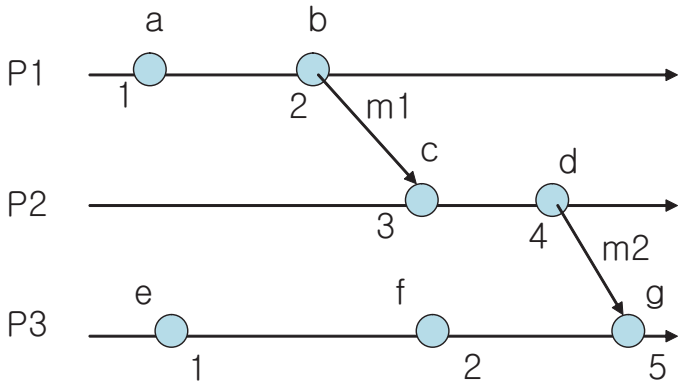

Figure 3.1: Lamport's logical clock

Using Lamport's scheme, for any pair of events e and e', if $e \rightarrow b$, then $C(e) < C(e')$. In addition, two events a and b are said to be concurrent if $a \nrightarrow b$ and $b \nrightarrow a$. In other words, concurrent events are not causally related to each other.

Figure 3.1 gives an example of Lamport's logical clock. In Figure 3.1 , a and b represent events in component P_1, c and d represent events in component P_2, and e, f, and g represent events in component P_3. Specifically, b is the event of sending the message m_1 at P_1 and c is the event of receiving m_1 at P_2. d is the event of sending the message m_2 at P_2 and g is the event of receiving m_2 at P_3. Assume all clock values C_1, C_2, C_3 are initialized as 0, and the value of the counter is incremented by 1 every time an event happens.

a and e are internal events in components P_1 and P_3 which cause both C_1 and C_3 to be incremented to 1. b is a send event in P_1. The message is assigned a timestamp of 2. The event c corresponding to the receive event of the message $m1$ increments the clock C_1 to 3. Similarly, d, f, and g are events in P_2 and P_3, resulting in clock values of 4, 2, and 5, respectively.

According to the definition of causal dependencies between events, $a \rightarrow b$, $c \rightarrow d$, $e \rightarrow f$, and $f \rightarrow g$ as causal dependencies reflect the order of events that happen at the same component. Additionally, $b \rightarrow c$ and $d \rightarrow g$ because the send event and receive event of the same message are causally related. According to the transitivity property of causal relationships, $b \rightarrow d$ because $b \rightarrow c$ and $c \rightarrow d$. Note that a and f are not causally related. In other words, a and f are concurrent.

There are a few problems with Lamport's logical clock. One of the problems is that distinct events (e.g. a and e) can have the same Lamport timestamp, as shown in Figure 3.1. Another problem with Lamport's clock is that for two events e and e', if $C(e) < C(e')$ it does not necessarily follow that $e \rightarrow e'$. For example, in Figure 3.1 $C(a) < C(f)$, but this pair of events are not causally related. The reason for the problems is that the clocks in each of the different component are incremented every time an event occurs in that component. Lamport's logical clock fails to distinguish between increments of the clock due to events involving a single component and those involving multiple components (e.g., sending or receiving messages).

3.1.2 Vector Clock

Another type of logical clock, known as the vector clock, was proposed by Mattern [50] and Fridge [51]. A vector clock in a system of N components is a vector of N integers. Each component maintains its own vector clock (V_i for component P_i) to timestamp the events.

As opposed to maintaining a single counter at each component, as in Lamport's logical clock, a component uses N counters as its vector clock in a system of N components. The i-th element of the vector (i.e., a counter) holds the estimated logical clock value for the i-th component in the system. For a component P_i, the only element in its vector clock that is guaranteed to be up-to-date is the i-th element.

At the initialization of the system, each component sets the elements of its vector clock as 0. When an event happens at P_i, P_i increments the i-th element of its vector clock. If a message is sent from a component, the component stamps its vector clock's current value to the message. When a component receives a message with a vector timestamp, it uses the timestamp on the message to update some counters in its vector clock. For each counter in the receiving component's vector clock, the component compares the counter value and the corresponding element value in the message's timestamp, and writes the larger value back to the counter. This ensures that the receiver has information that is at least as up-to-date as the sender of the message.

Given two events e and e', $V(e) = V(e')$ if and only if at least a pair of the corresponding elements of the timestamps are different; $V(e) \leq V(e')$ if and only if each element in V(e) is less than or equal to the corresponding element in V(e'); and $V(e) < V(e')$ if and only if $V(e) \leq V(e')$ and $V(e) \neq V(e')$. Using a vector clock, the causal relation (i.e., $\rightarrow$) between events is now redefined as $e \rightarrow e'$ if and only if $V(e) < V(e')$. Two events e and e' are said to be concurrent if neither $V(e) \leq V(e')$ nor $V(e') \leq V(e)$.

The vector clock is better able to reflect the causal relations between events. For instance, Figure 3.2, shows how vector clocks are used for exactly the same scenario (i.e.,

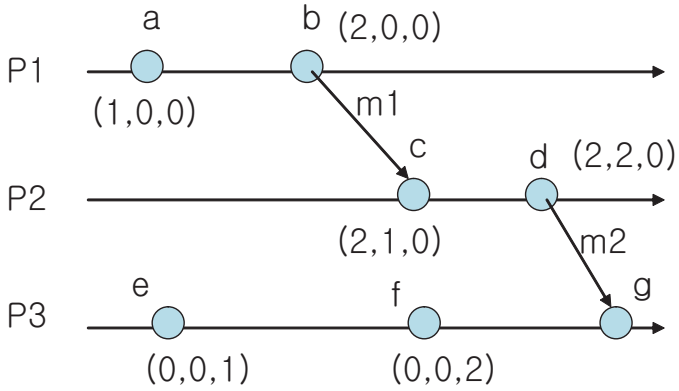

Figure 3.2: Example of vector clock

events and components) as Figure 3.1. As the figure 3.2 demonstrates, not only can the vector clock be used to find all the causal relations Lamport's logical clock is capable of identifying, but it can also tell that events a and f are concurrent because neither $V(a) \leq V(f)$ nor $V(f) \leq V(a)$.

The disadvantage of vector clocks is that more storage and message overhead are required because an entire vector rather than a single integer must be maintained.

3.1.3 Examples of protocols based on the use of logical clocks

Logical clocks have been used in the design of many distributed algorithms. For example, frequently a distributed system has one or several resources that are shared by multiple components. If the goal of a protocol is to grant the component that requested a resource the earliest the access right to the resource, the logical clock can be used. In [52], the proposed protocol first elects a coordinator among the components. If a component needs to access a resource, it sends a request to the coordinator. Upon the reception of multiple requests, the coordinator grants the access right to the component who sent the request

the earliest if the resource is available. After the component has finished accessing the resource, it sends an acknowledgement to the coordinator allowing it to release the resource to the next requestor. In this protocol, the logical clock can be used to determine the order of the requests.

The logical clock is also a powerful tool for analyzing and reasoning about distributed computations in general [53]. For instance, the global state of a distributed system [54] is defined as the collection of all state information. The consistency of a global state means that for every received message a corresponding send event is recorded in the global state. This consistency is important for a global state because it is required in the evaluation of global predicates (i.e., properties of the global state). To maintain the consistency of a global state, vector clocks can be used to check the causal relation of send and receive events. In [55, 56, 57], a consistent snapshot reflects a consistent global state and can therefore be determined using vector clocks.

In some cases, it is necessary to enforce causal order among a set of events. For instance, using a logical clock a receiver in a distributed system is able to sort the received messages according to the order they were sent. Another application of logical clocks is the debugging of distributed systems [58]. Using logical clocks, it is possible to show that some events cannot be the cause of an exception event. Logical clocks can also be used to reduce the size of the system trace as recording events with timestamps instead of raw data is sufficient for system rollback. Since causally independent events may be executed concurrently, logical clocks can also be used to determine the degree of parallelism of a computation [59].

3.2 Defer Timer and its application to network protocols

The defer time refers to the waiting time intentionally inserted between events in a distributed system. The defer timer is a countdown timer at each component. In general, before a defer timer expires, a component is forbidden to perform a certain set of events. By controlling the defer timer for each component, it is thus possible to determine the order of events.

The concept of the defer timer has been used in many network protocols. For instance, in the end-to-end communication protocols, the (defer) timer is used to guarantee the safe delivery of the packet and is normally set to be a constant [31]. The following subsections provide an overview of a few network protocols that are based on the use of defer timers. The first protocol is based on a constant defer timers, the second is based on a random defer timer, the third is based on an adaptive defer timer, and the fourth is based on a hybrid defer timer (i.e., both random and adaptive).

3.2.1 Inter Frame Spacing

In an IEEE 802.11b wireless LAN [39, 38, 37], a station listens to the wireless medium and ensures the medium is available before it transmits anything. This medium access control mechanism is referred to as carrier sensing multiple access with collision avoidance (CSMA/CA). The (waiting) time a station listens to the channel before it transmits is defined as its inter frame spacing (IFS).

There are four different IFS defined in the specification: the short interval inter frame spacing (SIFS), DCF inter frame spacing (DIFS), point coordination inter frame spacing (PIFS), and extended inter frame spacing (EIFS). SIFS is used before a station sends either an ACK, CTS, or DATA frame. PIFS is used when the access point coordinates all stations by polling. DIFS is used before a station starts a session (i.e., sending RTS or sending data without the preceding RTS/CTS). In the case of transmission failure, EIFS, the longest IFS, is used in place of DIFS to avoid further channel congestion due to immediate retransmissions. According to the IEEE 802.11b specification, in the case of a frequency hopping spread spectrum (FHSS), SIFS is 10 microseconds, PIFS is 30 microseconds, DIFS is 50 microseconds, and EIFS is 60 microseconds.

The reason for having four different IFSs in IEEE 802.11 is to assign different priorities to different frame transmissions. The longer the IFS is, the more likely a station is to hear another station's transmission and postpone its own transmission. In particular, the shortest SIFS gives the highest priority to signals (e.g., ACK) and the longest EIFS gives the lowest priority to retransmissions after a collision. An example of the IFS is shown in Figure 3.3.

3.2.2 Back-off Time

In IEEE 802.3 Ethernet [60] and IEEE 802.11 wireless LAN [38], defer time is used to avoid traffic congestion. In these systems, the medium (i.e., ethernet link and air) is shared by a set of stations. When a station encounters a packet loss (possibly due to collision), it is likely that many stations are in need of transmission at the same time. If all stations either

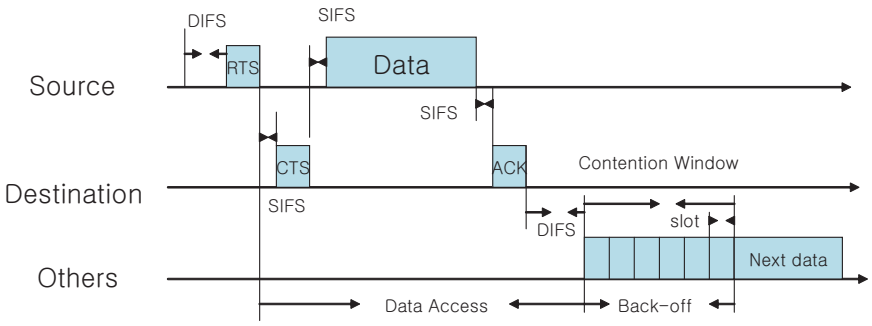

Figure 3.3: Inter Frame Spacing for medium priority transmissions

start transmitting immediately or simply wait for a constant amount of time (IFS) before transmission, a collision will occur.

To avoid this problem, a defer time (also called a back-off time) is inserted before the retransmission whenever a station finds a packet loss. After a station sends a packet, if it does not get back an ACK within a predefined period, the station concludes that the packet is lost. In such a case, the back-off time is set to be the product of the current contention window (CW) and a random variable uniformly distributed in $[0, 1)$. In general, the size of the contention window is doubled every time a packet is reported lost. The back-off time is decremented each time the channel is sensed to be idle. If the channel is sensed to be busy before the back-off time reaches zero, the decrementing process is frozen, resuming only when the channel is once again sensed to be idle for a predefined period. Before the back-off time expires, the station stops injecting traffic to the network.

Basically, a packet loss triggers the exponential growth of the contention window size, which in turn is likely to lead to a longer back-off time. The longer the back-off time a station has, the less frequently the packet is (re)transmitted. This helps alleviate network congestion.

3.2.3 Broadcasting

In a broadcast, a packet generated by the source station needs to reach all the other stations in the network. The easiest implementation of message broadcast is flooding i.e., each station retransmits a broadcast packet the first time it receives it. The main problem with flooding is that when it is adopted in MANETs, it typically causes an excessive number of retransmissions. This can result in high energy consumption, packet collisions and channel contention. This problem is referred to in the literature as the broadcast storm problem [15, 16]. Hence, the design of an efficient broadcast protocol to provide more reliable message broadcasts with fewer retransmissions is an important issue in MANETs.

In location-based ad hoc networks where each node equips a Global Positioning System [35, 62]. Assume that each station in the network knows the locations of itself and its neighbors. In Figure 3.4, assume that node S broadcasts a message to node A, B and C with TTL greater than zero. If nodes A and C retransmit earlier than node B, B does not need to retransmit since it can conclude (from the locations of neighbors S, A, and C encoded in the header of the (re)transmissions) that its coverage area (the shaded region) is completely covered by those of nodes S, A, and C. The message will reach the same area no matter whether node B retransmits or not. Therefore, if the order of the retransmissions is correct, node B can derive that its retransmission is redundant.

To make sure the order of retransmissions is favorable in finding redundant retransmissions, intuitively the retransmissions that cover more new area should happen earlier. Although computing the exact new coverage area for each retransmission is complicated and requires more than just localized information, the distance between the node itself and

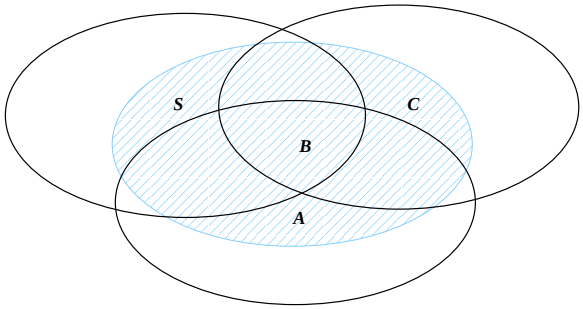

Figure 3.4: The retransmission of B is redundant if S, A, and C transmit before B

the sender where it received the broadcast from is a good indication for approximating the amount of new area. (i.e., the farther the node is away from the previous sender, the more new area its retransmission is likely to contribute.) Based on this observation, let R be the transmission radius, ΔD be the distance between the node itself and the neighbor who it received the broadcast from, and ΔT_{max} be the maximum value of the defer timer, the following simple formula can be used to set a defer timer for broadcast retransmission when a node receives a broadcast:

$$\Delta T = T_{max} \cdot \frac{(R - \Delta D)}{R} \tag{3.1}$$

Computer simulations reveal that the order of retransmissions following this defer timer can effectively help identify redundant retransmissions, thus reducing the number of retransmissions significantly without the need for collecting neighbor location information.

Figures 3.5 demonstrates how much performance gain the defer timer based broadcast protocol has achieved compared with simple flooding and the broadcasting protocols in [63] and [64] (denoted as Ni) and [4], [5], and [6](denoted as Wu). Notice that Wu's

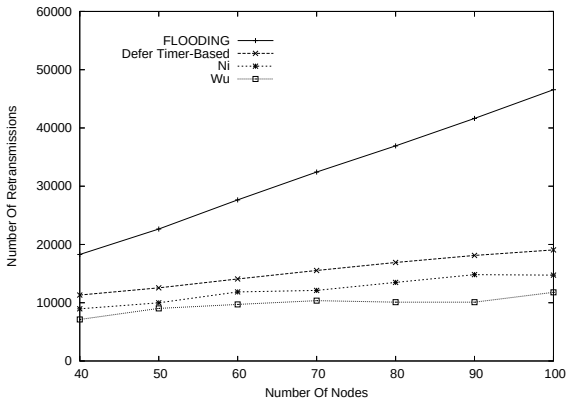

Figure 3.5: Number of retransmissions

protocol shows a low number of retransmissions because it requires 2-hop local topology information at each node, which will introduce an extra message overhead not counted in Figure 3.5.

3.2.4 Dynamic Cluster Formation in Sensor Networks

In a sensor network where each node is able to detect only the distance (not the direction) of the target, the sensor nodes need to collaborate in order to derive the location of the intruder. Hence, it is natural for the nodes participating in a computation to form a cluster (for collaboration). However, at the same time for the purpose of energy conservation, it is desirable that only the nodes near the target be activated for the computation. To form a cluster, the first step is to elect a cluster head. For the purpose of intruder detection, it seems reasonable to elect the node closest to the intruder as the cluster head for this computation.

Consider a set of n distinct sensor nodes deployed on a two-dimensional plane. The Voronoi diagram of the sensor network is the subdivision of the plane into n cells, one

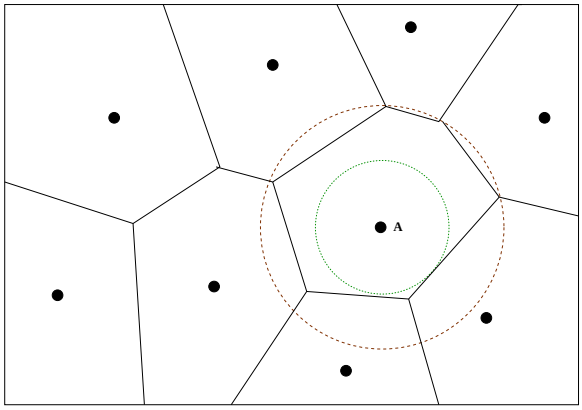

Figure 3.6: Voronoi diagram for a sensor network

for each node. Each cell in the Voronoi diagram represents the area closest to the sensor node located within. Assuming a certain localization service (e.g., [33], [34], and [35]) is available for sensor nodes at the time the network is deployed, it is possible to pre-compute the Voronoi diagram when the sensor network is first deployed as long as the sensor nodes are immobile.

In Figure 3.6, if node A detects the object to be outside the outer circle, it can safely conclude that it must not be the node nearest to that object. On the other hand, if node A detects that the object is inside the inner circle, it can also safely conclude that it must be the one closest to the object. The problem arises when the distance to the detected object falls in between the radii of these two circles.

To determine which node should be the cluster head, a defer timer can be used. Let the radius of the large and small circles be *R* and *r*, respectively, and $W_{max}(W_{min})$ be the maximum (minimum) defer time. For each node not able to make a definite choice, a defer timer D is set up based on the following linear formula:

$$D = W_{min} + (W_{max} - W_{min}) \cdot ((1 - Pr(i|d)) + U(W_{ran}), \tag{3.2}$$

Notice this formula is a simplified version of the one in [65] and contains two parts. The first two terms in equation 3.2 are the deterministic part that relates the estimated distance to the back-off delay value, and the third term accounts for the random part that prevents potential collision when the distances from the target to two or more cluster heads are approximately the same. According to this formula, the defer time is set to near W_{max} when the distance is closet to *R* and W_{min} when the distance is closer to *r*. Note that this formula contains two parts. If no other node volunteers to become the cluster head before a node's defer time expires, the node declares itself as the cluster head. In cases where more than one node declare themselves to be the cluster head, they exchange information on their distance to the intruder and the one with the smallest distance is declared to be the cluster head.

This simple defer timer allows the cluster head to be elected efficiently without the need to introduce extra signals. In [65], the computer simulation results indicate that this defer timer based protocol incurs the minimum location error, the smallest latency, and the least amount of message exchanges.

CHAPTER 4

TIMER-BASED VIRTUAL BACKBONE CONSTRUCTION PROTOCOLS

The Dominating set (DS) (also known as vertex cover) and connected dominating set (CDS) have found major applications in the area of mobile ad hoc network (MANETs) in the past few years. The ability to reach all nodes within one hop has made the DS/CDS strong candidates for the virtual backbone [44] in MANETs. The dominating set of a graph is a subset of nodes in the graph such that each node not in the subset has at least one direct neighbor that belongs to the subset. If the nodes in the dominating set induce a connected graph, the set is called a connected dominating set. This concept has been found to be extremely useful in routing [5, 6, 7, 66], message broadcasting [67, 68, 69], and collision avoidance [70].

Given a complete network topology, the problem of finding the minimum DS/CDS is known to be NP-hard [71, 72]. In addition, due to the nature of MANETs, it is impractical to assume that a node in a MANET has a complete network topology. As a result, a practical DS/CDS construction protocol for MANETs needs to be fully distributed and the DS/CDS correctly constructed based on localized information. In addition, it should possess the following properties.

- The resulting DS/CDS should be as small as possible – The impact of the size of the DS/CDS is two fold. First, when the network topology changes due to nodal movements, a smaller DS/CDS is easier to maintain. Second, the size of the DS/CDS is

closely related to the size of the routing table at each node. The smaller the DS/CDS is, the smaller the routing table that each node has and thus the more efficiently the message communications can be achieved.

- The protocol should avoid introducing extra messages – Bandwidth is a precious resource in wireless networks, and introducing any extra messages may degrade the performance of the system significantly. If possible, the protocol should be carefully designed so that the necessary local information can be collected solely via beacons.

- The protocol should adapt to station mobility – The protocol should maintain and incrementally adjust the DS/CDS under changes of network topology caused by nodes either leaving or joining the network after the construction of DS/CDS.

- The DS/CDS protocol should take into account the energy level at each node – In addition to nodal mobility, the energy level of nodes in a DS/CDS is an important factor in determining the lifespan of the DS/CDS. Since the process of constructing a DS/CDS is in general costly and time-consuming, in the case of a static network it is desirable to prolong the lifespan of the DS/CDS in order to avoid the need to reconstruct the DS/CDS frequently. Moreover, in the above routing and collision avoidance protocols, nodes in the DS/CDS are commonly used to forward more packets and participate in traffic management for the network, so they are likely to consume more energy than nodes not in the DS/CDS. In cases where nodal mobility frequently renders the existing DS/CDS obsolete, the DS/CDS protocol should act as a key that

evenly distributes the energy consumption to all the nodes in the network, thus allowing the network to remain operational for longer periods of time.

This dissertation presents three timer-based protocols, namely timer-based DS (TDS), timer-based CDS (TCDS), and timer-based energy aware CDS (TECDS). In these proposed timer based protocols, each node adaptively sets up a defer timer based on the number of uncovered neighbors and determines whether or not to join the DS/CDS when the timer expires.

In this dissertation, a mobile ad hoc network (MANET) is represented as an undirected graph $G = (V, E)$, where V is the set of all stations in the MANET and E is the edge set with $(u, v) \in E$ *if and only if* u and v are within each other's transmission range.

If G is connected, a set *DS* $\subset V$ is called a *dominating set* if for every vertex $v \in V - DS$, there exists a vertex $w \in DS$ such that $(v, w) \in E$. A dominating set is said to be *connected* if its induced graph in G is connected.

A node $u \in V$ is said to be in the state of *inDS*, covered (by *DS*), or uncovered (by *DS*) according to the following:

- *inDS*: if $u \in DS$;
- covered: if $u \notin DS$ and there is an edge $(u, v) \in E$ for some $v \in DS$;
- uncovered: if $u \notin DS$ and there is no edge joining u to any node in *DS*;

The state transition diagram is shown in Figure 4.1. There are four possible state for a node, namely init/uncovered, start/initiator, covered/marked/dominatee, and DS/dominator.

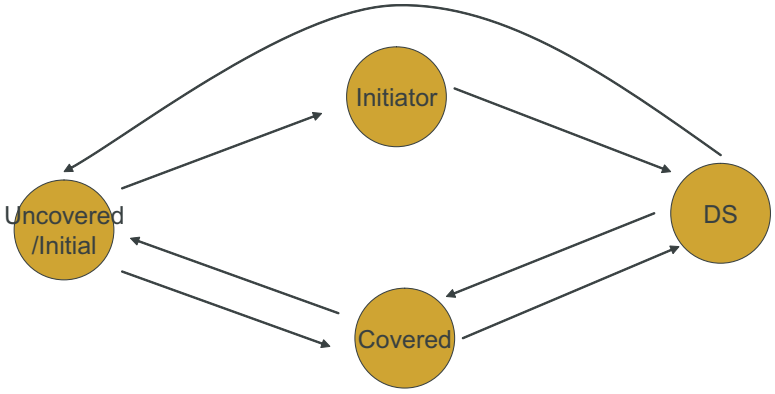

Figure 4.1: Possible states for a node and the transitions between states

4.1 Survey of Existing DS and CDS Protocols

Computing the minimum dominating set and the minimum connected dominating set are considered fundamental problems in computer science and these problem are known to be NP-hard [71, 72]. In [72, 73, 74, 75], several nondeterministic protocols were proposed to compute the minimum DS or CDS at a lower time complexity in a centralized manner. These protocols require a station to have the topology of the whole network prior to the computation. Hence they are not suitable for MANETs.

In the CEDAR routing protocol [76], the core extraction phase is basically a distributed generic dominating set construction protocol. In this protocol, each node exchanges its dominator, the number of neighbors, and the number of neighbors that use it as its dominator with all its neighbors via beacons and switches to become a dominator if some of its neighbors chose it as their dominator. The protocol in [76] offers two major advantages. First, they obtain the dominating set without introducing any extra messages. All the necessary information can be acquired through the beacons. Second, the size of the resulting

dominating set is very competitive. However, since in CEDAR each node joins the dominating set independently, the dominating set may contain many disconnected components. All nodes in the DS constructed by the CEDAR protocol are not guaranteed to fully connect the topology in a network. In other words, the dominating set constructed by this protocol is not routing friendly when it is used as virtual backbone in ad hoc networks.

The protocols in [77] [78] construct the CDS by expending the maximal independent set. The construction has two phases. In the first phase, if a node finds its unique MAC address is minimum among its neighbors, it adds itself into a set and removes all its neighbors from the consideration of the set members. This process is repeated until the resulting set becomes the maximal independent set, which is also a (non-connected) dominating set. In the second phase, the nodes in the set use local topology information for up to 3 hops to add gateway nodes to the set until the set becomes a CDS. Because in these protocols the node needs to collect local topology information within three hops, extra messages need to be introduced. However, the key advantage of these protocols is that the size of the resulting CDS is bounded by a constant times the size of the minimum CDS.

The CDS construction protocol proposed in [4, 5, 6] provides the CDS by eliminating nodes from the network. These protocols also have two phases. In the first phase, a node exchanges the neighbor list with its neighbors. If a node finds that all its neighbors are pairwise connected, it removes itself from the consideration of the CDS. In the second phase, some heuristic rules are applied to further reduce the size of the set. The protocols are simple, distributed, and most of time compute a CDS with a very competitive size. However, they also introduce extra messages between immediate neighbors (to exchange neighbor

list). Additionally, when the topology changes, the protocols do not have a mechanism to maintain the CDS.

Since energy (i.e., battery power) is a limited resource for nodes in MANETs and sensor networks, the energy-aware protocols for MANETs and sensor networks [79, 80, 81] have always received special attention. In general, nodes in the CDS forward more packets and participate network management, so they tend to consume more energy than those outside of the CDS. However, none of the above CDS protocols takes nodal energy into consideration when constructing the CDS. In [82, 83], Wu et al proposed an extended marking process that constructs an energy-aware CDS for MANETs. This extended marking process is aimed at both reducing the size of CDS and evenly distributing the energy consumption to all nodes in the network. In [82, 83], the simulation results show that Wu's energy aware CDS protocol allows the network to go through more of CDS reconstruction steps, an indication that the protocol prolongs the operation of the network. However, similar to [4, 5, 6], this protocol also requires exchanges of extra messages between immediate neighbors.

4.2 Timer-based Dominating Set Protocol (TDS)

This section presents the new Timer-based Dominating Set (TDS) protocol.

4.2.1 TDS Protocol Description

This protocol is based on a very simple greedy strategy. That is, in order to obtain a smaller dominating set, a node with more neighbors should have a higher chance to be

included in the dominating set than a node with fewer neighbors. While this strategy is simple and valid, the question is how to design a distributed protocol to automate such a process.

To achieve this goal, the idea of the *distance defer transmission* broadcast protocol proposed in [68] could be borrowed. This broadcast protocol intentionally inserts a *defer timer* for nodes to defer message retransmission. Nodes receiving the broadcast message farther away from the sender defer the retransmission less. By doing this, the nodes covering more new area are more likely to retransmit the broadcast message. This strategy greatly reduces the number of retransmissions for a broadcast without affecting its reachability. If the same idea of deferring can be used to differentiate the nodes with more neighbors and the nodes with fewer neighbors, a simple and robust distributed protocol can be constructed to efficiently compute the dominating set.

Similar to every existing wireless system, including IEEE 802.11 [61] and Bluetooth [84], assume each node has an unique value or identifier in the network, such as its MAC address. The following refers to a node's unique identifier as its id. A node transmits a beacon at every fixed time interval. Before a node transmits its beacon, it encodes its own id, dominator, and current state in the header of the beacon. By doing this, each node learns its neighbors, the dominator and the state of the neighbors without introducing extra messages. Since normally the beacon already includes a node's id, the only extra fields are its current state (3 bits to represent 6 possible states) and its dominator's id (32 bits if it is MAC address). Using IEEE 802.11b as an example, Figure 4.2 shows the extension of the IEEE 802.11b beacon format.

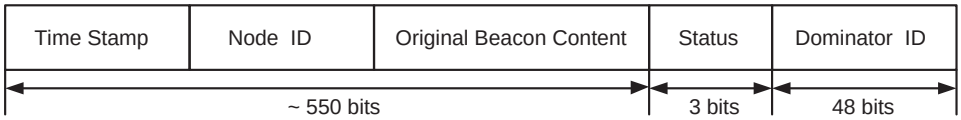

Figure 4.2: Extended IEEE 802.11b Beacon Format

Every node starts from the init state. After waiting for a fixed time interval (to collect information from its neighbors' beacons), if an init node finds its id is the minimum among its neighbors, it switches to start state. If not, the node switches to the uncovered state and waits for further signals from its neighbors to wake it up. Note that there may be several start nodes in the networks. A start node calculates the ΔT according to the following formula and then switches to the covered state. In contrast, an uncovered node waits until one of its neighbors turns *DS* before it calculates the ΔT using the same formula and then switches to the covered state.

$$\Delta T = T_{max} \cdot \frac{1}{(number\ of\ uncovered\ neighbors)^{\alpha}} \tag{4.1}$$

Clearly nodes with more neighbors result in shorter defer times compared with nodes with fewer neighbors as long as $\alpha > 0$ in Equation 4.3.

A *covered* node waits for its ΔT to expire and then checks if all of its neighbors are covered. If this is the case, the covered node switches to the covered state and uses the dominator neighbor that triggered it to switch from the uncovered to covered state earlier as its own dominator. If not, the covered node joins the dominating set by switching to the dominator state and, consequently, uses itself as its own dominator.

If nodal movements cause a covered node to lost its own dominator, it simply switches to init state and restarts the protocol again. Similarly, if nodal movements cause a dominator to lose all its covered nodes, it switches to the covered state to reduce the size of the dominating set providing one of its neighbors is in the dominator state.

The pseudo code of the above Timer-based Dominating Set Construction Protocol is given in the following.

```
/* node x in wireless ad hoc network executes the following
procedure until x has a dominator */
while ( x has no dominator )
    init state :
        on receiving a signal from neighbors
            if ( x.id is smallest among its neighbors )
                then state ¡= start
                else state = uncovered

    start state :
        start ΔT
        state = covered

    uncovered state :
        on receiving a beacon from a dominator neighbor
            start ΔT
            state = covered
```

covered state :

if (x has an uncovered neighbor)

then state = dominator

else state = covered

covered state :

if (x's dominator is no longer a neighbor of x)

then state = init

dominator state :

if (none of x's neighbors use x as its dominator) *and*

(a neighbor of x, y, is a dominator)

then x's dominator = y and state = covered

Timer-based Dominating Set Construction Protocol

4.2.2 Correctness of the TDS protocol

This subsection tests whether the new TDS protocol successfully computes the dominating set for the whole network.

Theorem 4.1 *If the network topology remains stable for a period of time, the collection of all nodes in the dominator state that result from the TDS protocol forms a dominating set for the entire network.*

Proof.

First, if the network topology is unchanged eventually all nodes the will end up remaining in either the dominator or covered state stably.

Notice that although in Figure 4.1 there are two outgoing links from the dominator and covered states, the only case where a node would leave the dominator or covered state is when the network topology is changed. In other words, to prove that all nodes in the network end up being in either the dominator or covered state, it is only necessary to show that every init node will eventually visit either the dominator or covered state.

According to the TDS protocol, a init node has to switch to either start or uncovered after a fixed period of time. The start node goes through covered and then visits either the dominator or covered state after its ΔT expires. The uncovered node, however, needs to wait until one of its neighbors becomes a dominator before it can move on to the covered state and, later, the dominator or covered state. It is necessary to show that, after a certain amount of time, one of the uncovered node's neighbors must become a dominator.

Notice that the criteria for becoming a start node (i.e., a local minimum id) guarantees that there must be at least one start node in the network. Now it is necessary to show that an uncovered node will eventually turn covered by performing an induction over the number of hops an uncovered node is away from the closest start node.

When an uncovered node is one hop away from a start node, as described earlier, the start node will eventually switch to the dominator state after its ΔT expires. Once the original start node becomes a dominator, its beacon will force the uncovered nodes to switch to covered.

Assuming any uncovered node less than n hops away from the closest start node will eventually become covered, it seems reasonable to suggest that an uncovered node p that is n hops away from the closest start node will also become covered. Let the set S be the subset of p's neighbors that are $n-1$ hops away from the start node. According to the induction hypothesis, nodes in S will eventually become either dominators or covered. However, according to the TDS protocol, if p stays uncovered, this implies that all nodes in S are covered. This cannot be the case, since if when the first node in S ends its defer timer ΔT it has at least one uncovered neighbor, p, that will force it to switch to dominator state. In other words, p will also switch to covered as well. This shows that any uncovered node, no matter how far away it is from the start node, will become covered. Notice that this induction is valid even when there are multiple start nodes.

Based on the above arguments, all the nodes in the network will eventually stay in either the dominator or covered state stably if the network topology remains unchanged. The covered node has at least one dominator neighbor as its dominator. Hence, the collection of all dominator nodes forms the dominator set for the entire network. Q.E.D.

■

4.2.3 Example of the TDS Protocol Execution

To better understand the TDS protocol, an example is provided in this subsection to demonstrate how it functions. Figure 4.3 shows the network topology. The top of Figure 4.3 illustrates the patterns used to represent various node states.

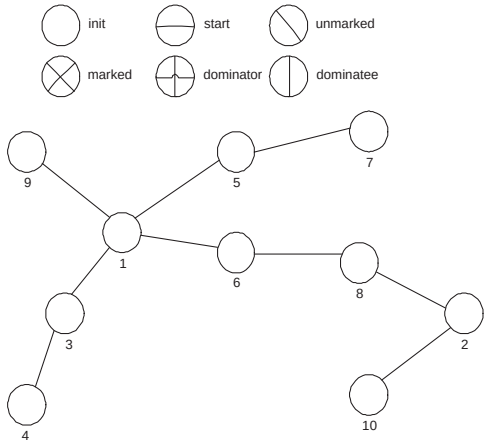

Figure 4.3: Initial network topology

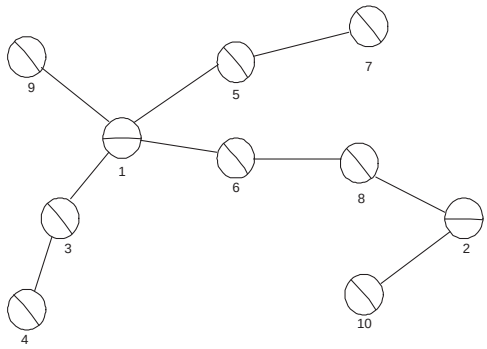

Figure 4.4: Node 1 and node 2 switch to the start state

In Figure 4.3, the number for each node represents the id for the node. At the beginning of the process, all the nodes are in the initial state. According to the TDS protocol, after a period of time node 1 and node 2 soon discover that their id is the local minimum among their neighbors and thus switch to the start state. All the other nodes switch to the unmarked state. This is shown in Figure 4.4.

After nodes 1 and 2 compute their ΔT, they again switch to the marked state and wait for their timer ΔT to expire. Obviously, node 2 will defer longer than node 1 since node 1

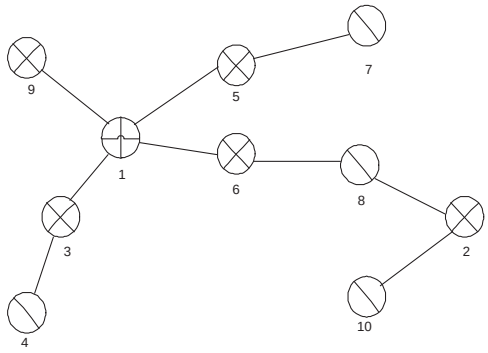

Figure 4.5: Node 1 joins the dominating set and forces its neighbors to switch to the covered state

has more unmarked neighbors than node 2. When node 1's ΔT expires, it switches to the dominator state and then causes its neighbors, nodes 3, 5, 6, and 9, to compute their ΔT and switch from the unmarked to the marked state. This occurs when node 2 is still in the marked state. The result of the network is shown in Figure 4.5. Notice that the ΔT of nodes 3, 5, and 6 will be shorter than that of node 9 since they have more unmarked neighbors.

Now let the ΔT of node 2 expire, after which it switches to the dominator state because nodes 8 and 10 are not yet covered. This again forces nodes 8 and 10 to switch from the unmarked to the marked state. Notice that even though nodes 8 and 10 have different numbers of neighbors, their ΔT will be the same since both have no unmarked neighbors. Additionally, let the ΔT of nodes 3, 5 and 6 expire, after they then switch to the dominator state and force nodes 4 and 7 to switch to the marked state. The state of the network is now shown in Figure 4.6. Notice that node 8 is already a marked node and thus will ignore the beacons received from node 6 after node 6 becomes a dominator.

After the ΔT of the rest of the marked nodes expire, they will find that all their neighbors are in either the dominator or dominatee state and thus will switch to the dominatee

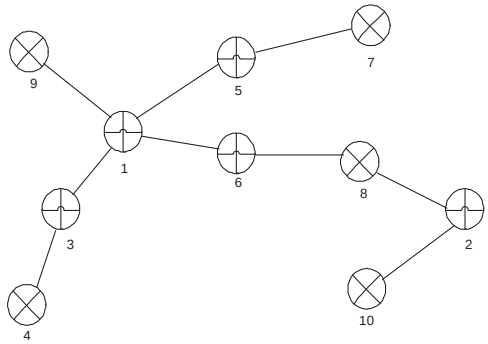

Figure 4.6: Network status after nodes 2, 3, 5, and 6 join the dominating set

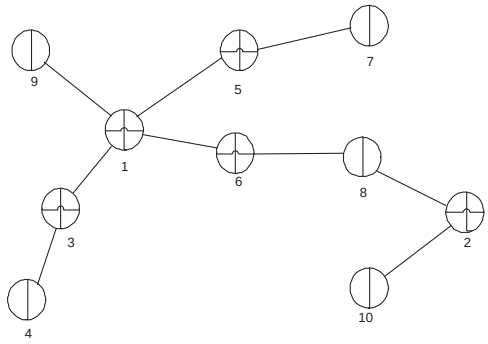

Figure 4.7: Final network status

state. The resulting network status is shown in Figure 4.7. If the network topology is not changed, the dominating set obtained by the TDS protocol is the collection of nodes in the dominator state, which is the set $\{1, 2, 3, 5, 6\}$. Nodes 4, 7, 8, 9, and 10 are dominatees.

4.2.4 Comparison of Simulation Results for the TDS Construction Protocols

4.2.4.1 Simulation Setting and Parameter Consideration

Table 1 shows a list of the parameters used in the simulations, if not specified otherwise.

Nodes are generated randomly on a 4 by 4 square plane. The plane is wrapped vertically and horizontally to eliminate the effects of the edge. Each node has the same range of transmission. If the generated network is partitioned into pieces, it is discarded and a new network topology is generated to ensure the connectivity of the whole network. The value T_{max} is chosen to be 100 time units. The last parameter to be determined in the TDS protocol is the value of α in Equation 4.3. In order to select a good value for α, computer simulations were carried out. The results of these simulations are shown in Figure 4.8.

Table 1: Simulation Parameters

Parameter	Value
Space	4 by 4 (in unit)
Transmission Radius	1
Number of Nodes	50
T_{max}	100

In Figure 4.8, the x-axis is the value of α and the y-axis is the size of the dominating set for 50 nodes. Since the standard deviation of the size of the resulting dominating set is only 0.17 when α is in the range between 1 and 5, it seems likely that the value of α does not have a significantly impact on the performance of the protocol. Hence, a value of 1 for was used for α for the remaining simulations of the TDS protocol.

4.2.4.2 Simulation Results and Performance Evaluation

In addition to the TDS protocol, three other dominating set protocols were implemented, namely Wu's protocol [4], CEDAR's protocol [76], and the first phase of Wan's

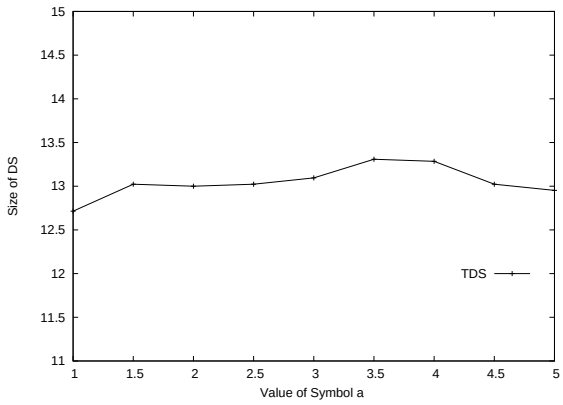

Figure 4.8: The value of α vs the size of the dominating set

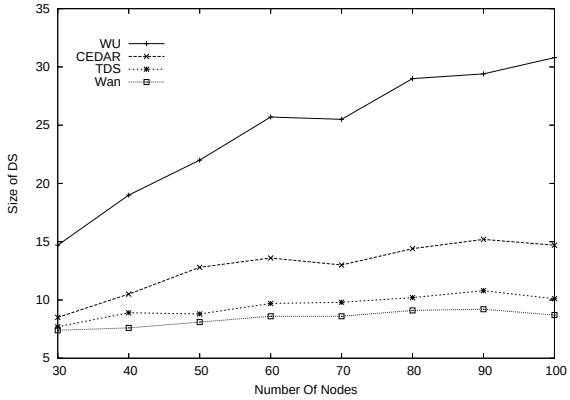

Figure 4.9: The size of the dominating set for different protocols

protocol [77] [78] (the second phase of Wan's protocol is not needed since it is for connected dominating sets). The optimal dominating set was computed using brute force to show how far the results of each protocol departed from the ideal. All simulations were executed under the same parameters and network topology. The sizes of the resulting dominating sets for networks with a size between 30 and 100 nodes were used to test the performance of the new TDS protocol. The simulation results are shown in Figure 4.9.

In Figure 4.9, the x-axis is the size of the network and the y-axis is the size of the resulting dominating set for the different protocols. Clearly, the size of the connected dominating set obtained by the Wu's is at least twice as large as the size of any dominating set obtained by other generic dominating set construction protocols, regardless of the size of the network. Notice that Wu's protocol assumes that each node knows the local network topology 2 hops away from itself. Even with such a strong assumption, the size of the connected dominating set is still too big. This shows that it is not always true that the connected dominating set is always better than the generic dominating set, since the bigger connected dominating set clearly will require much more overhead to maintain it.

In addition, Figure 4.9 shows that the dominating sets obtained by Wan's and the TDS protocols are very close to the minimum dominating set. The size of the dominating set obtained by the CEDAR's protocol is considerably larger than that of Wu's and ours by approximately 10% to 45%. The difference becomes more and more obvious as the size of the network increases. This suggests that the Wan's and TDS protocols are more scalable than CEDAR's. Although the result of Wan's protocol is slightly ahead of TDS in terms of the size of the dominating set, the difference is marginal (i.e., less than 1 node in most of the cases).

As described earlier, if the dominating set is more connected, it would be more friendly to the routing protocols. Now consider the connectivity nature of the dominating set obtained by Wan's and the TDS protocol. In Figure 4.10, the x-axis represents the size of the network and the y-axis is the number of disconnected components for the dominating set. It is obvious that the dominating set obtained by the TDS protocol is more connected than

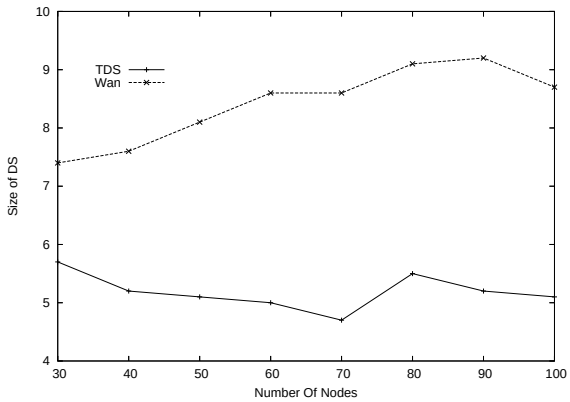

Figure 4.10: The number of disconnected components for Wan's and TDS's dominating set

the dominating set obtained by Wan's. The connectivity difference increases from 30% to 70% as the size of the network increases from 30 to 100. The reason for this is simple. The dominating set obtained by Wan's protocol is the maximal independent set, where all nodes in the set are isolated. In contrast, the dominating set obtained by the TDS protocol contains just few disconnected trees rooted at the start nodes. The simulations reveal that the number of start nodes is insensitive to the size of the network. This suggests that the connectivity of the dominating set obtained by the TDS protocol is scalable. In other words, the dominating set obtained by the TDS protocol is more routing friendly than the dominating set obtained by Wan's.

The only disadvantage of the TDS protocol is that it introduces a delay for dominating set construction. If the ad hoc network has a diameter of n, the worst case for TDS to obtain the dominating set is bounded by $n \cdot T_{max}$. However, since T_{max} is at most the time for a few rounds of beacons (i.e., a fraction of a second in IEEE 802.11b wireless networks

[61]), such delays should not be a major concern even when the nodes are vehicles moving at highway speed.

In summary, the new TDS protocol is a simple and efficient protocol capable of handling nodal mobility. When the size of the network is large, the TDS protocol generates a considerably smaller dominating set compared with CEDAR's. The dominating set obtained by TDS is more connected compared with the dominating set obtained by Wan's and is therefore routing-friendlier.

4.3 Timer-based Connected Dominating Set Construction Protocol (TCDS)

This section presents the new *Timer-based Connected Dominating Set Construction Protocol*. Like many other CDS protocols, the distributed Timer-based Connected Dominating Set Construction protocol (TCDS) also has two phases. The first phase is to elect an initiator in the MANET; the second phase is to construct a CDS from the initiator.

Like many other CDS protocols, the distributed Timer-based Connected Dominating Set Construction protocol (TCDS) proposed here also has two phases. The first phase is to elect an initiator in the MANET; the second phase is to construct a CDS from the initiator.

4.3.1 The TCDS protocol

Similar to the TDS protocol, the TCDS protocol is also based on a simple greedy strategy. That is, in order to to obtain a smaller connected dominating set, a node with more *uncovered* neighbors should have better chance to be included in the dominating set than a node with fewer *uncovered* neighbors. The protocol basically works as follows.

Starting from an initiator as the first node in the CDS, the direct neighbors are covered as *covered* nodes. For each *covered* node, a timer is set based on the number of *uncovered* neighbors. Nodes with more *uncovered* neighbors are given a smaller timer value, and hence will expire earlier. When the timer expires, a node enters the CDS if it still has *uncovered* neighbors.

According to IEEE 802.11 wireless LAN specification [61], a node periodically broadcasts its beacon. The timers used in this protocol have an initial value of -1. A positive integer is assigned when a timer is started. The timer value will go down each beacon period. When the value reaches 0, the timer expires and the value stops at 0.

4.3.1.1 Initiator Election

The initiator election selects one unique initiator. Here the node ID (for example, the MAC address) is used as the metric to select the node with smallest ID as the initiator. Other metrics can be used as long as they are unique. The initiator will send out an $ANNOUNCE$ message every $initMax$ beacon periods, and it will refresh the $InitTimer$ of other stations which expire after $2 * initMax$ beacon periods. It is a soft state protocol, and the expiration of InitTimer for stations other than the initiator implies that the initiator leaves the MANET. The nodes will wait $2 * initMax$ to make sure all the $InitTimer$s expire, the initiator election process starts again.

The following is the pseudo code for the initiator election phase of the TCDS protocol.

/* node i in MANET executes the following: */

Initialization :

$initiator(i) \leftarrow MAXINIT$

status$(i) \leftarrow$ *uncovered*

color$(i) \leftarrow 0$

DSTimer$(i) \leftarrow -1$

ODSTimer$(i) \leftarrow -1$

$InitTimer(i) \leftarrow initMax$, start InitTimer

InitTimer expires

if initiator(i) = MAXINIT then /* initial announcement */

$initiator(i) \leftarrow i$

$announce(i, i, color(i))$

$InitTimer(i) \leftarrow 2 * initMax$, start InitTimer

else if $initiator(i) = i$ then /* initiator is selected */

$color(i) \leftarrow color(i) + 1$

$announce(initiator(i), i, color(i))$

$InitTimer(i) \leftarrow initMax$, start InitTimer

/* refresh message*/

else if $initiator(i) \neq i$ then /* re-elect initiator */

status$(i) \leftarrow$ *uncovered*

$initiator(i) \leftarrow MAXINIT$

$color(i) \leftarrow 0$

wait for $2 * initMax$

$initiator(i) \leftarrow i$

$announce(initiator(i), i, color(i))$

$InitTimer(i) \leftarrow 2 * initMax$, start InitTimer

Node i receiving $announce(j, k, c)$ /* $i \neq j$ */

if $initiator(i) > j$ or

$initiator(i) = j$ and $color(i) \neq c$ then

$initiator(i) \leftarrow j$

$color(i) \leftarrow c$

$announce(j, i, c)$

$InitTimer(i) \leftarrow 2 * initMax$, start InitTimer

Initiator Election Protocol

4.3.1.2 Connected Dominating Set Construction

After the initiator is detected, the initiator first enters *DS*, broadcasting to its neighbor about its *inDS* status. A neighboring $uncovered$ node becomes *covered* after receiving the message. Then the *covered* node calculates the ΔT according to the following formula if it still has *uncovered* neighbors and start its *DSTimer*.

$$\Delta T = T_{max} \cdot \frac{1}{(number\ of\ uncovered\ neighbors)^{\alpha}} \tag{4.2}$$

It is clear that nodes with more *uncovered* neighbors result in shorter defer times compared with nodes with fewer *uncovered* neighbors as long as $\alpha > 0$ in Equation 4.3.

When the *DSTimer* expires, the node enters *DS* and broadcasts to its neighbors about its *inDS* status.

The following is the pseudo code for the second phase of the TCDS protocol after the unique initiator has been identified.

/* node i executes the following */

Node i detects itself as initiator :

$status(i) \leftarrow$ *inDS*

$color(i) \leftarrow color(i) + 1$

broadDS(i, color(i))

Receiving broadDS(j,c) :

if $color(i) < c$ then

$color(i) \leftarrow c$

if $status(i) =$ *uncovered* then

$status(i) \leftarrow$ *covered*

start *CoveredTimer* with δt

Node i in *covered* state :

if (i hears from two neighbors with large color difference)

$status(i) \leftarrow$ *inDS*

broadDS(i)

if (i does not have any *uncovered* neighbor)

DSTimer(i) = -1

else if $ODSTimer(i) = -1$ then

$DSTimer(i) \leftarrow \Delta T$, start *DSTimer*

$ODSTimer(i) \leftarrow \Delta T$

else if $ODSTimer < \Delta T$ then

$DSTimer(i) \leftarrow \Delta T$, start *DSTimer*

$ODSTimer(i) \leftarrow \Delta T$

Node i in *inDS* state :

if (i does not have any *covered* neighbor) and

(i has at least one *inDS* neighbor) then

$status(i) \leftarrow$ *covered*

DSTimer expires :

$status(i) \leftarrow$ *inDS*

broadDS(i)

CoveredTimer expires :

$status(i) \leftarrow$ *uncovered*

Timer-based Connected Dominating Set Construction Protocol

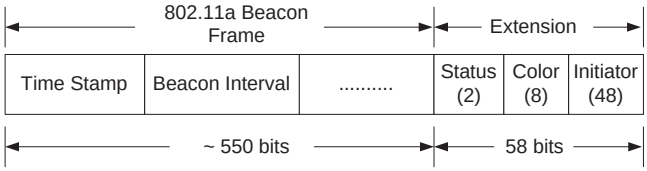

Figure 4.11: Beacon Frame Format

4.3.2 Beacon Frame Extension

Note that there is no need for any new control messages with the TCDS protocol. All the messages mentioned in the pseudo code are in fact beacon frames with slight extensions. The extended beacon frame format is shown in Figure 4.11.

In the extended beacon format, a total of 58 bits are needed for the node to collect necessary information from its neighbors. The *status* bits are always present, and allow the neighbor nodes to know the status of each other. The value "01" means *uncovered*, "10" means *covered* and "11" means *inDS*.

A $broadDS(i, c)$ message is simply a beacon with status value "11" and color value c where i is the source address (SA). Because it is already part of the beacon, it does not need to be included in the extension.

An $announce(initiator, sender, color)$ is a beacon with all the extensions. The sender is the SA, so only *initiator* and *color* must be added, along with the *status*. Since the $announce$ message is sent out every $initMax$ BPs, the overhead of the beacon extension is not significant.

4.3.3 Correctness of the TCDS Protocols

This subsection tests that the TCDS protocol generates a CDS as long as the network topology remains connected and stable for a period of time.

Lemma 1 *A unique initiator will be selected by the initiator election protocol within bounded beacon periods if the network topology remains connected and stable.*

Proof.

It is necessary to prove that the node with the smallest ID (say, node i) will be selected as the initiator. Initially each node sets its initiator to $MAXINIT$, which means no initiator has yet been elected. When the $InitTimer$ expires, it will identify itself as the initiator and announce this fact. If a neighbor has a larger initiator, it will accept the new initiator value and forward the message further. Looking at the node i, each of i's neighbors will set its initiator to i. The initiator value will stay as i because i is the smallest. This process repeats until it reaches all the nodes in the network. Since the network is connected and stable, each node will receive the announcement and set its initiator as i. If $initMax$ is set large enough to ensure that the announcement of the initiator from i reaches every other node in the network, the whole network will agree that i is the initiator. When the $initTimer$ of node i expires and finds that its initiator is itself, node i knows the initiator is elected. The next phase of the protocol (dominating set construction) can then start.

■

Lemma 2 *A new initiator will be selected by the initiator election protocol within bounded beacon periods if the original initiator crashes or leaves the network, assuming that the network remains connected and stable otherwise.*

Proof.

The initiator election protocol specifies that the original initiator will send out an $ANNOUNCE$ message every $initMax$ BPs. Other nodes only forwards an $ANNOUNCE$ message if its initiator is bigger than the initiator in the message or the color value is different. If the original initiator is active, its color value will change every $initMax$ BPs, and the refreshed $ANNOUNCE$ will keep the $InitTimer$ of other nodes from expiring.

When the original initiator crashes or leaves the network, all the other nodes agree on the initiator and the color value of these nodes will also be the same. Their $initTimer$ will expire since no more refreshed $ANNOUNCE$ messages are coming. After the expiration, all the state information will be reset to their initial values and the initiator election will kick in again. According to Lemma 1, a unique initiator will be selected.

■

It is also necessary to prove that the TCDS protocol successfully determines the connected dominating set for the whole network.

Theorem 4.2 *If the network topology remains stable for a period of time, the collection of all inDS nodes that results from the TCDS protocol forms a connected dominating set for the entire network.*

Proof.

If the network topology is unchanged, eventually all the *inDS* nodes will form a dominating set and the induced graph is connected.

Proving this by induction for a number of stations n.

Induction base: when $n = 1$, the only station is the initiator. The initiator enters *inDS* after election which is a connected dominating set. The above claim is therefore true.

Induction hypothesis: assume the CDS covers up to $k - 1$ nodes and the graph with $k - 1$ nodes can be denoted as $G(k - 1)$.

Induction step: assume a new station (node k) joins the network. If one of its neighbors is an *inDS* node, then the CDS covering $G(k - 1)$ will cover $G(k)$ and is still a CDS.

If none of its neighbor is an *inDS* node, all the neighbors of k are *covered* nodes due to the existence of CDS in $G(k - 1)$. According to the protocol, one or more of k's neighbors will have its *DSTimer* expire and enter *DS*. After their access, there will be no more *uncovered* nodes and a dominating set results. Since the newest *inDS* node enters *DS* from the *covered* status, it must have an *inDS* neighbor. (A node enters *covered* status only after it receives a $broadDS$ message from an *inDS* node) The CDS covering $G(k - 1)$ plus the new *inDS* node will therefore be a CDS covering $G(k)$ and the *DS* is connected.

■

4.3.4 Example of the TCDS Protocol Execution

To better understand the TCDS protocol, an example is provided in this subsection to demonstrate how it functions. Figure 4.12 shows the initial network topology. The left side of Figure illustrates the patterns used to represent various node states. In the figure, the

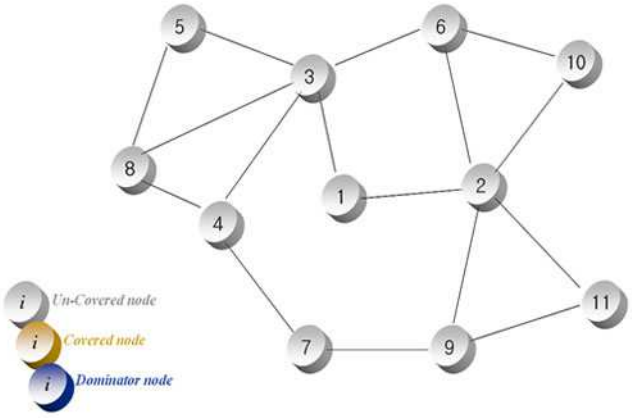

Figure 4.12: Initial state

number for each node represents the id for the node. At the beginning of the process, all the nodes are in the initial state. Based on the initiator election phrase, after a period of time node 1 discovers that its id is the smallest among all the nodes in the network. It is thus selected to serve as the initiator and switches to the DS state in Figure 4.13.

As shown in Figure 4.14, nodes 2 and 3 then switch to the covered state. At this stage, all the remaining nodes are still in the uncovered state.

After nodes 2 and 3 have computed their ΔT, they wait for their timer ΔT to expire. Nodes 2 and 3 have the same ΔT since nodes 2 and 3 have the same number of uncovered neighbors. When ΔT for nodes 2 and 3's expires, node 2 switches to the DS state first because of its node ID. At the same time, nodes 6, 9, 10, and 11 switch from the uncovered to the covered state and then compute their ΔT, as shown in Figure 4.15.

Figure 4.16 depicts how node 3 switches to the DS state, causing its neighbors, nodes 4, 5, 6, and 8, to compute their ΔT and switch from the uncovered to the covered state.

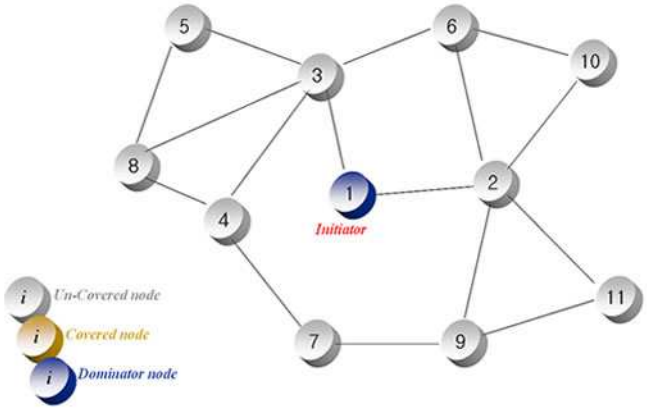

Figure 4.13: Node 1 is selected as the initiator

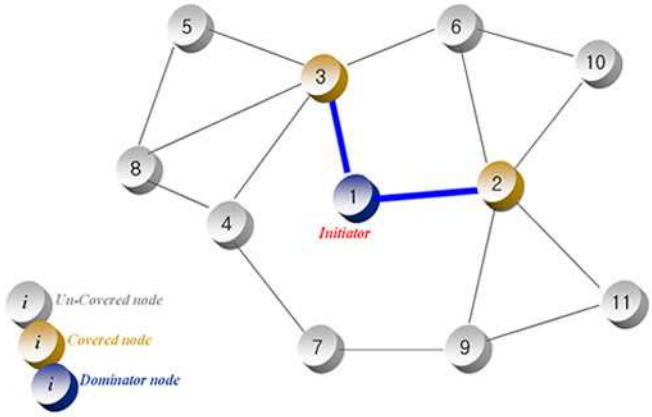

Figure 4.14: Nodes 2 and 3 switch to the covered state

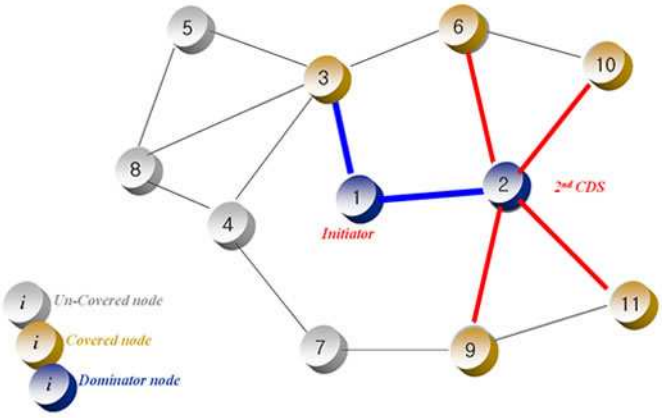

Figure 4.15: Node 2 switches to the DS state and nodes 6, 9, 10, and 11 switch to the covered state

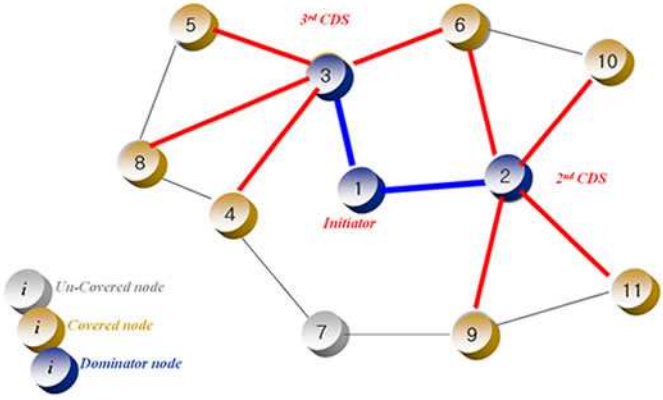

Figure 4.16: Node 3 switches to the DS state and nodes 4, 5, 6, and 8 switch to the covered state

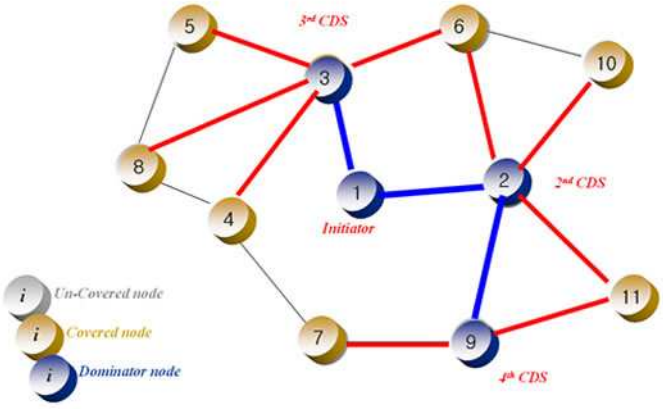

Figure 4.17: The resulting network topology

Once the ΔT of node 9 expires, it switches to the DS state and forces node 7 to switch to the covered state. Notice that even though node 4 and 9 have the same number of uncovered neighbors, node 7 is covered by node 9 because the ΔT of node 9 expires earlier than that of node 4. Additionally, even though nodes 6, 10 and 11 have different numbers of neighbors, their ΔT will be the same since none have no uncovered neighbors. After the ΔT of the remaining covered nodes expires, These three nodes will find that all their neighbors are in either the DS state or covered state and thus will switch to the covered state. The resulting network status is shown in Figure 4.17. If the network topology remains unchanged, the connected dominating set obtained by the TCDS protocol is the collection of nodes in the DS state, namely the set $\{1, 2, 3, 9\}$. Nodes 4, 5, 6, 7, 8, and 10 are covered nodes.

4.3.5 Station Mobility

Next it is necessary to show that the TCDS protocol can adapt to accommodate station mobility. This is done by showing that the protocol successfully maintains a CDS under changes of network topology and include the following four cases:

1. The initiator leaves the network.
2. A new node joins the network after the construction of the CDS.
3. A redundant *inDS* node can change its status while still retaining the dominating set connection.
4. An *inDS* node in the CDS leaves the network

Case 1 is proved in Lemma 2, and Case 2 is the exact situation in Theorem 4.2. From the protocol, if a node has only *inDS* neighbors, it will change its $status$ to *covered*. The total number of *inDS* nodes is thus reduced. This can happen if several *inDS* nodes movement.

Case 4 is handled by the color scheme. From the protocol, we can see that a node changes its color only after receiving a larger color value from an *inDS* node. Since the initiator keeps increasing its color value, if the CDS is intact, the color difference of neighboring nodes should be very small. If the CDS is broken into several segments, the segment that contains the initiator will keep increasing its color value while other segments will have their color value unchanged. When a node sees two neighbors with a large color difference,

it can consider itself a border node between the disconnected segments and enters *inDS*. This process will continue until a CDS has been re-constructed.

4.3.6 Implementation Considerations

There are two implementation issues to consider in our protocol: The first is identifying a suitable value for initMax in the initiator election phase; the second is the trade-off between the size of the beacon and the size of the dominating set.

The value of initMax plays an important role in the initiator election and it has to be big enough to allow minimum id information to propagate throughout the network. Intuitively, if the node with minimum id is at the boundary of the network, the value of initMax has to be at least the diameter of the network for the protocol to come up with a single initiator. If the size of initMax is too small, the first phase will end up with multiple initiators and thus cannot guarantee that the dominating set obtained at the end of the second phase is still connected. On the other hand, if initMax is set too big, it is likely that the protocol (especially the first phase of it) will take too much time. Additionally, in case of topology changes due to nodal mobility, a protocol with a large initMax value may take some time to converge. In theory, the value of initMax should be proportional to the diameter of the network. However, since it is too costly for each node to learn the size of the network, it is not practical to determine the value of initMax at run time.

Although this issue indicates that our protocol may not be scalable, in reality, this is not a serious problem even for a relatively large ad hoc network, as the because IEEE 802.11 beacon period is only 0.1 seconds. When the node with minimum id transmits

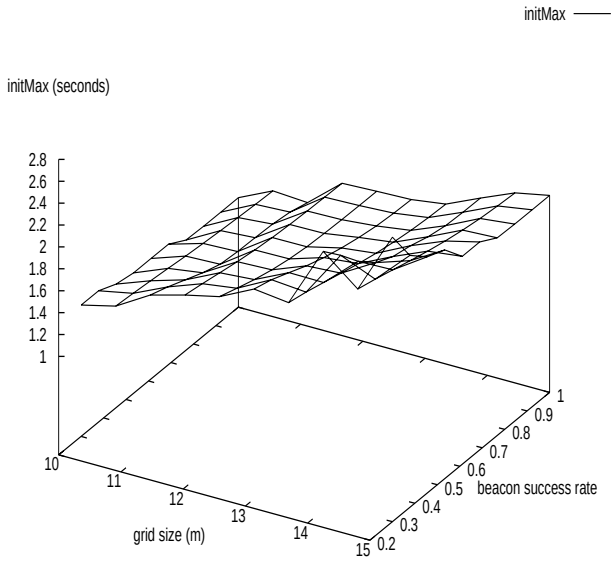

Figure 4.18: initMax for $m \times m$ grid under various beacon success rate

its beacon, its id information basically floods network. In other words, even though the beacon transmissions are not reliable, the minimum id information will reach other nodes through all possible routes and thus will not be affected significantly by unreliable beacon transmissions. For instance, for an ad hoc network with 1000 stations uniformly distributed in a 15×15 grid, setting initMax to 3 seconds (i.e., 30 beacon periods) almost guarantees that the minimum information will reach all other nodes in time even when the beacon's success rate is as low as 0.3. This is shown in Figure 4.18.

Another issue is the trade-off between the size of the beacon and the size of the dominating set when the topology changes. When the network topology changes due to nodal mobility, as described in Section 4.4.4.2, our protocol is still able to come up with a connected dominating set. In a few cases a non-dominator will switch to the dominating state

(e.g., a new station enters the network or some *covered* stations become *uncovered* due to movement and their *covered* neighbors may switch to the dominating state). However, the only case where a node in the dominating set will switch to the covered state is when all its neighbors are in the dominating set. It seems that as time goes by, when network topology changes occur, the size of the CDS may grow bigger and bigger.

The reason some nodes in the CDS cannot be removed from the set is actually due to a lack of information. For instance, assuming each node randomly selects one dominator neighbor and includes that neighbor's id in the beacon, a dominator can safely switch to the covered state when the following statements hold.

- The beacons it received from all its neighbors have a different dominators.
- It has at least one dominator neighbor.

Although this approach does not introduce any new messages, it certainly increases the size of each beacon by 48 bits and thus may not be justifiable. Of course, another easy resolution to this issue is to simply set a timer at each node when running this protocol. When this timer expires, the protocol executes from scratch again to ensure the resulting CDS remains a competitive size.

4.3.7 Comparison of Simulation Results for the TCDS Construction Protocols

4.3.7.1 Simulation Setting and Parameter Consideration

The MANET used in the simulation consisted of n stations randomly distributed over a $m \times m$ area. The transmission range of a single station was normalized to 1 unit of

distance. The simulations were run on both 4 and 8×8 grids. Our protocol was compared with Wu's [5] and Wan's [78] protocols. Let $T_{max} = 100$, $initMax = \delta t = 20$, $\alpha = 1$.

4.3.7.2 Simulation Results and Performance Evaluation

For a 4 grid, TCDS's DS size ranged between 11 to 13, Wan's protocol ranged between 13 to 22 and Wu's protocol ranged between 16 to 55. TCDS thus easily outperformed the other two protocols.

Two characteristics of the TCDS are worth elaboration. First, TCDS is not overly sensitive to the number of stations. The curve is very flat, which implies very good scalability. Secondly, the DS size is even smaller when $n = 256$ than for $n = 128$, which is counter intuitive. A careful examination of the simulation reveal that the favorable location of the initiator in the 256-node network was the reason for the slight difference. Overall TCDS is very scalable, and while the location of initiator has some impact, this impact is quite minor if the nodes are randomly distributed and the number of stations is large.

For the 8×8 grid, TCDS's DS size ranged between 44 to 49, Wan's protocol ranged between 61 to 75 and Wu's protocol ranged between 72 to 132, following the same performance pattern.

On average, MTDS's DS size is about 65% of that of Wan's protocol and about 40% of Wu's protocol.

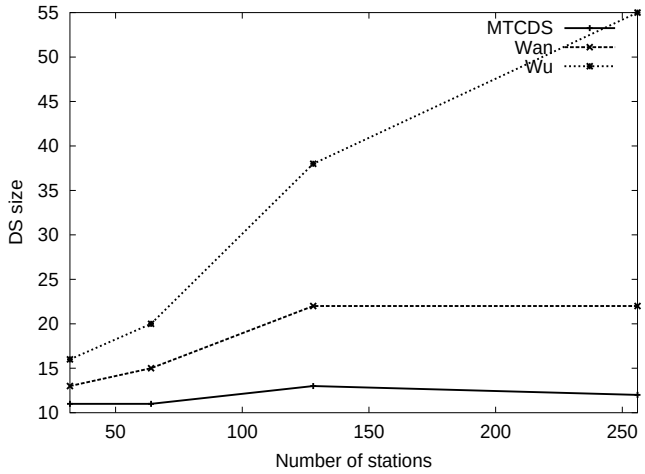

Figure 4.19: DS size in 4 x 4 square

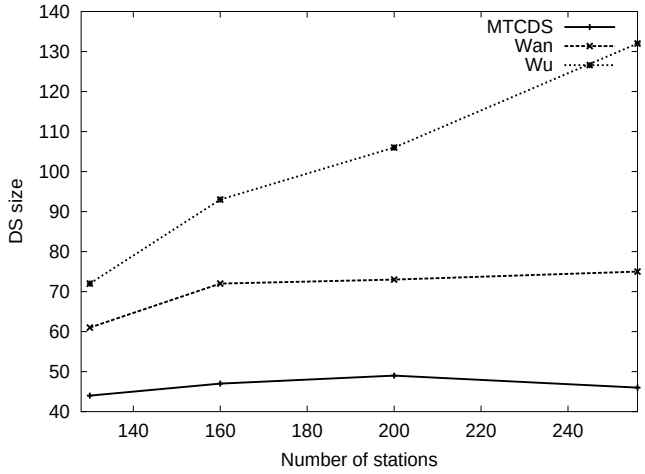

Figure 4.20: DS size in 8 x 8 square

4.4 Time-based Energy Aware Connected Dominating Set Construction Protocol (TECDS)

This section presents two *Timer-based Energy-aware Connected Dominating Set Protocols*which extend the Timer-based Connected Dominating Set protocol (TCDS) so that the energy level at each node is taken into account when constructing the CDS. Simulation results have shown that our protocols effectively construct an energy-aware CDS with a very competitive size and prolong the network operation under different levels of nodal mobility.

4.4.1 TECDS protocol

Like TCDS, the Timer-based Energy aware Connected Dominating Set protocols (TECDS) use the same notations and definition is as those given in Section **??**. The TECDS protocols also have two phases: initiator election and CDS construction. In the first phase, an unique initiator is elected; in the second phase, the CDS is constructed rooted from the initiator. The major difference between TCDS and our TECDS protocols is that the energy level at each node is taken into consideration in both phases.

Assume that each node in the network has the same transmission range. As with every wireless network system, each node periodically broadcasts a beacon signal. Two types of beacon signals are used in the protocols: the regular beacon and the $announce$ beacon. In the regular beacon, a node's MAC address, status (i.e., uncovered, covered, or *inDS*), and color value (used to detect if the initiator is still active) are included in the header of the

beacon. In the "announce" beacon, a node encodes those included in the regular beacon as well as the energy level and number of neighbors for its initiator (for the possible election of a new initiator) in the header of the beacon. Notice that a $broadDS$ message is actually a regular beacon encoded with *inDS* status. The reason for introducing two different beacon formats is to reduce the overhead required by the protocols. Since the announce beacon carries more information, it is larger than the regular beacon. The protocols are carefully designed so that the announce beacon is sent every initMax regular beacon period.

4.4.2 Consideration of Initiator Election

The TECDS protocols are based on a similar greedy strategy to that used in TDS and TCDS protocols. In TCDS, the node with the minimum MAC address is picked as the initiator. In TECDS, however, the objective is to create a CDS with a smaller size that contains nodes with a higher energy level, so two new criteria are used when the protocol picks the initiator: the number of neighbors and the energy level. Depending on the order of consideration for these two criteria, two different versions of TECDS, namely TECDS1 and TECDS2, are introduced. In TECDS1, the node with the most energy is picked as the initiator. In cases where multiple nodes have the same energy level, the one with the most neighbors is picked as the initiator. In TECDS2, the node with the most neighbors is picked as the initiator. When multiple nodes are found to have the same most neighbors, the one with the highest energy level is then elected as the initiator. In both protocols, when multiple nodes have the same number of neighbors and the same energy level, the node with the minimum MAC address is picked as the initiator to break the tie.

The following is the simplified version of the initiator election phase for the TECDS1 protocol. The protocol is similar to the initiator election phase for the TCDS given in Section 4.3.1.1. Differences between the TECDS protocols and TCDS's are highlighted.

/* node i in MANET executes the following: */

On initialization:

All variables in TCDS are initialized the same way as in TCDS. In addition, node i initializes energy(i) as its own energy level and nbrNum(i) as the number of its neighbors.

On InitTimer expiration:

Other than initiator(i), i's id, and color(i), the announcement message now also includes energy(i) and nbrNum(i).

On receiving announce(j, k, c, e, r):

Node i compares the energy(j), numNBR(j), and MAC id received from all its neighbors to elect a initiator.

Initiator Election Phase

4.4.3 The TECDS Construction

After the initiator is selected, the initiator enters *DS* first. It broadcasts to its neighbors information about its *inDS* status. A neighboring $uncovered$ node becomes *covered* after receiving the message. The *covered* node then calculates ΔT according to the following formula if it still has *uncovered* neighbors and starts its *DSTimer*.

$$\Delta T = T_{max} \cdot \frac{1}{N_{uncovered}} \cdot \frac{1}{E} \tag{4.3}$$

In Equation 4.3, the term $N_{uncovered}$ represents the number of uncovered neighbors and E is the energy level. This equation uses both the number of uncovered neighbors and *the energy level* at each node to compute ΔT. It is obvious that nodes with more *uncovered* neighbors or higher energy levels result in shorter defer times compared with nodes with fewer *uncovered* neighbors and lower energy levels. This is the major difference between TCDS and our TECDS protocols in the CDS construction phase.

Other than Equation 4.3, the pseudo code of the CDS construction phase for the TECDS protocols is exactly the same as that for TCDS.

As long as the network topology remains stable for a period of time, it can be shown that both TECDS1 and TECDS2 always generate the CDS for any connected MANET. In addition, it can be proved that both TECDS1 and TECDS2 are capable of incrementally maintaining the CDS under changes of network topology by utilizing the color value, state of neighbors, and timers such as InitTimer. All four of the following different topology changes are well supported by our protocols:

1. The initiator leaves the network.

2. A new node joins the network after the construction of the CDS.

3. A redundant inDS node can change its status while still preserving the dominating set connection.

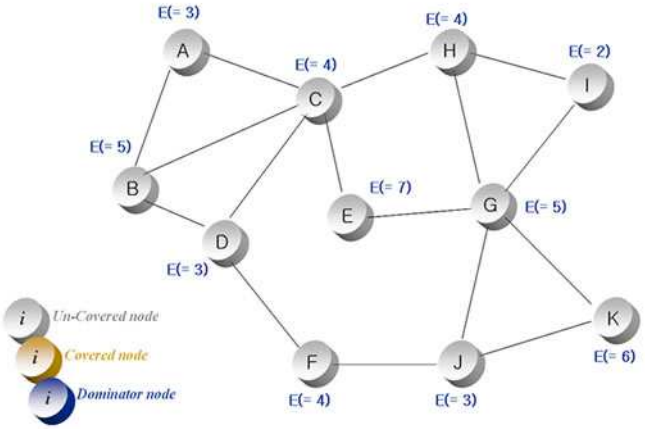

Figure 4.21: Initial state

4. The inDS node in the CDS leaves the network

Since both TECDS1 and TECDS2 are extension from TCDS, the proof of mobility support for both TECDS protocols is essentially identical to that given for TCDS. Several examples of the station mobility handing will be presented in 4.4.4.2.

4.4.4 Example of the TECDS Protocol Execution and Station Mobility Handling

4.4.4.1 Example of the TECDS Protocol Execution

In TECDS, two criteria are used when the protocol picks the initiator as explained in 4.4, namely the number of neighbors and the energy level. An example of TECDS1 is provided to demonstrate how the protocol functions. Figure 4.21 shows the network topology. The left side of Figure 4.21 illustrates the patterns used to represent various node states.

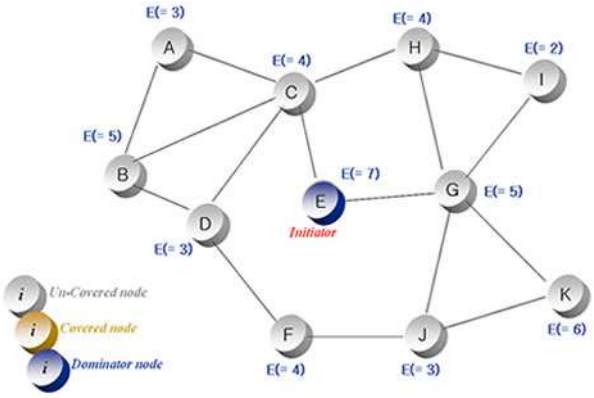

Figure 4.22: Nodes E is selected as an initiator

In Figure 4.22, the number and alphabet for each node represents the $energylevel$ and the id of the node, respectively. At the beginning of the process, all nodes are in the initial state. According to the TECDS protocol, after a specified period of time node E discovers that it has the highest energy level and thus becomes the initiator, switching to the DS state while all the other nodes switch to the uncovered state. This is shown in Figure 4.22.

In Figure 4.23, nodes C and G have switched to the covered state. All the other nodes are still in the uncovered state.

After nodes C and G compute their ΔT, they wait for their timer ΔT to expire. Obviously, node C will defer longer than node G since node C has a higher energy level than node G. When nodes G's ΔT expires, node G switches to the DS state and causes its neighboring nodes H, I, J, and K to compute their ΔT and switch from the uncovered to the covered state. This occurs when node C is still in the covered state. The resulting network is shown in Figure 4.24.

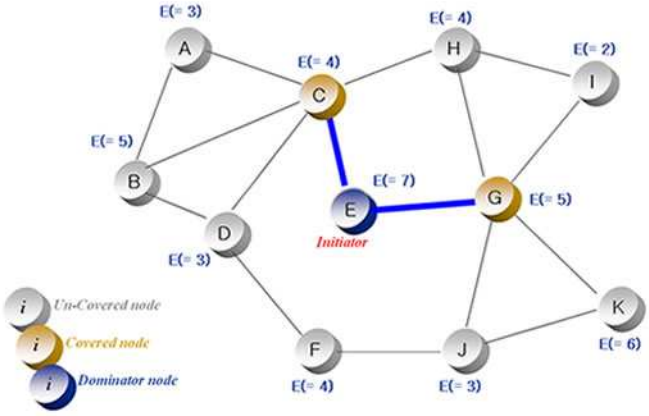

Figure 4.23: Nodes C and G switch to the covered state

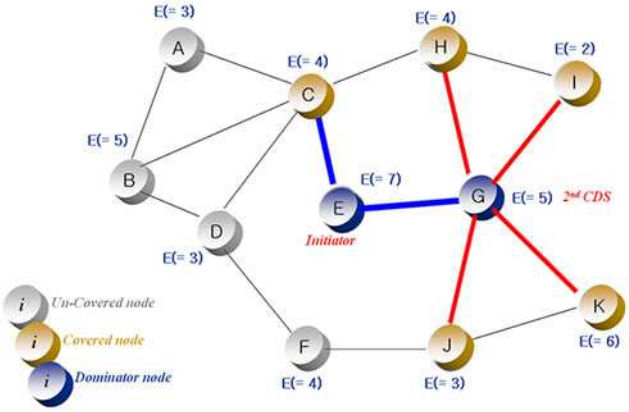

Figure 4.24: Node G switches to the DS state and nodes H, I, J, and K switch to the covered state

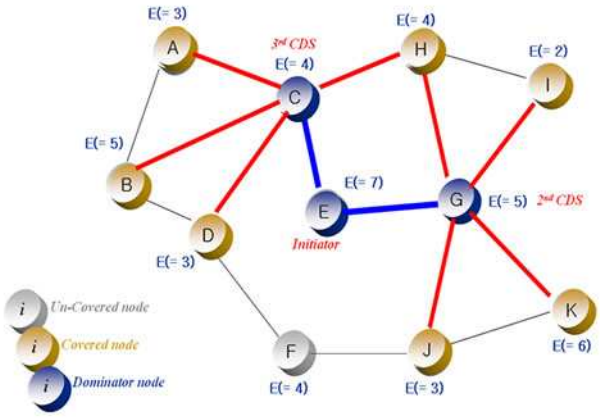

Figure 4.25: Node C switches to the DS state and nodes A, B, D, and H switch to the covered state

In Figure 4.25, node C switches to the DS state, causing its neighbors, nodes A, B, D, and H, to compute their ΔT and switch from the uncovered to the covered state.

Once the ΔT of node J expires, it switches to the DS state and forces node F to switch to the covered state. Notice that even though nodes J and D have the same number of uncovered neighbor nodes, node F is covered by node J because the ΔT of node J expires earlier than that of node D. Additionally, even though nodes H, I and K have different numbers of neighbors, their ΔT will be the same since none have uncovered neighbors. After the ΔT of the remaining covered nodes expires, they will find that all their neighbors are in either the DS state or the covered state and thus will switch to the covered state. The resulting network status is shown in Figure 4.26. If the network topology is not changed, the connected dominating set obtained by the TECDS protocol is the collection of nodes in the DS state, namely the set $\{E, C, G, J\}$. Nodes A, B, D, F, H, and I are covered nodes.

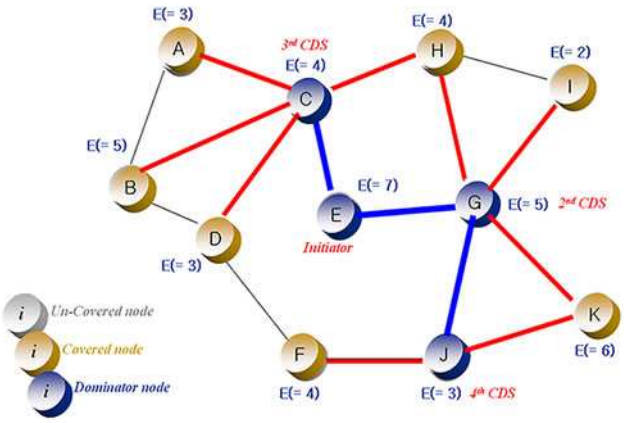

Figure 4.26: The resulting network topology

4.4.4.2 Example of Station Mobility

Figure 4.27 is the resulting network status as from the example outlined in 4.4.4.1 after the construction of CDS is completed.

Figure 4.28 shows the Case 2 in 4.4.3, when a new node W joins the network after the construction of CDS.

After node W joins the network, the node K calculates the ΔT and then switches to the DS state because it has an uncovered neighbor node W, as shown in Figure 4.29.

Once the ΔT of node K expires, it switches to the DS state and forces node W to switch to the covered state as shown in Figure 4.30.

Figure 4.31 shows Case 3 in 4.4.3, when an DS node in the CDS leaves the network. Here, the node F switches from the covered to the uncovered state because its dominator node J has left the network as shown in Figure 4.32. The node D calculates the ΔT and

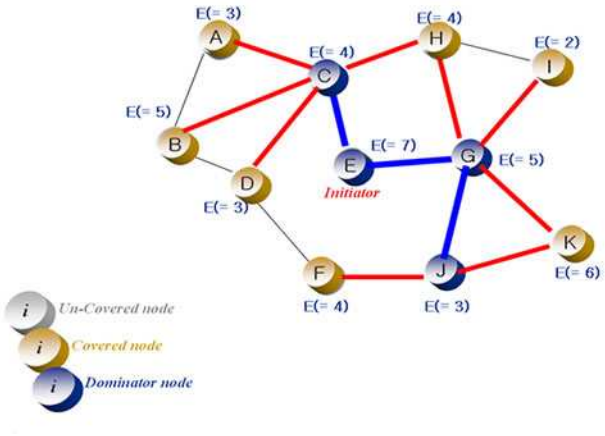

Figure 4.27: The network after the construction of CDS

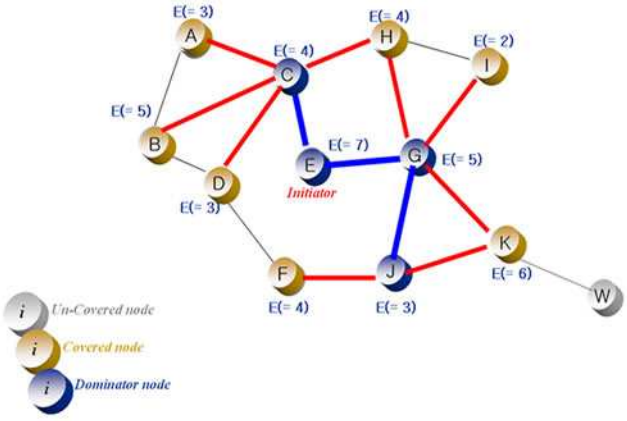

Figure 4.28: A new node W joins the network

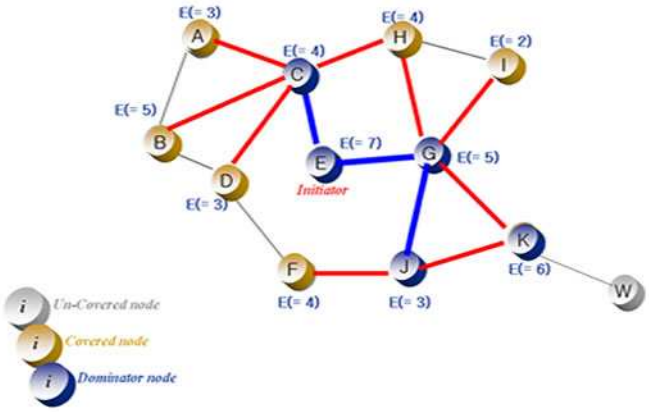

Figure 4.29: Node K switches to the covered state

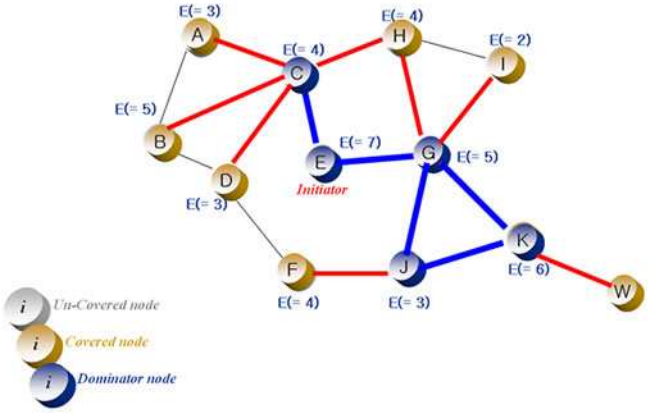

Figure 4.30: Node W switches to the covered state

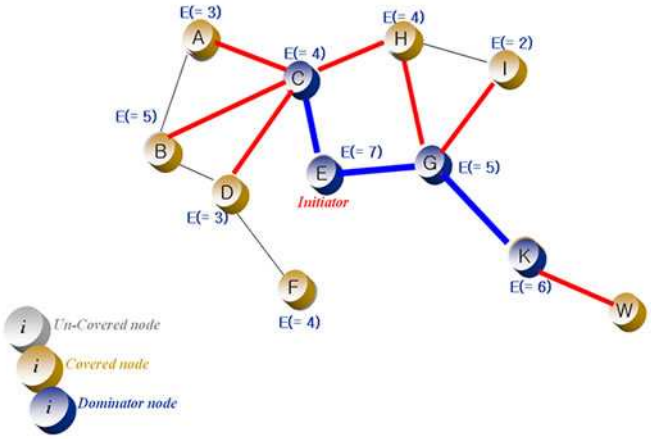

Figure 4.31: For the case where an DS node J leaves the network

then switches to the DS state, forcing node F to switch to the covered state, as shown in Figure 4.33.

After the node W leaves the network, Figure 4.34 and Figure 4.35 show Case 4 in 4.4.3 when a redundant DS node can change its status while keeping the dominating set connected. In the case, node K switches from the DS status to the covered state, as shown in Figure 4.35.

4.4.5 Comparison of Simulation Results for the Energy Aware CDS Construction Protocols

In addition to the two new TECDS protocols(i.e., TECDS1 and TECDS2), three other CDS protocols, namely TCDS [7], Wu's original connected dominating set protocol [5] (referred as Wu1 hence after), and Wu's extended protocol [82] (referred as Wu2 hence after). Wu et al actually proposed two sets of extended rules based on the node's energy

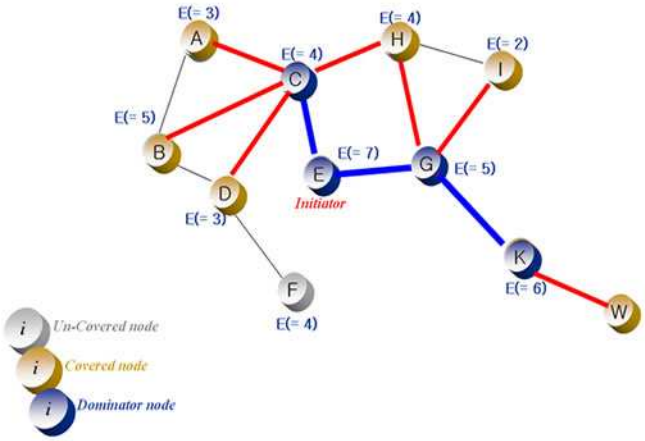

Figure 4.32: Nodes F switches to uncovered state

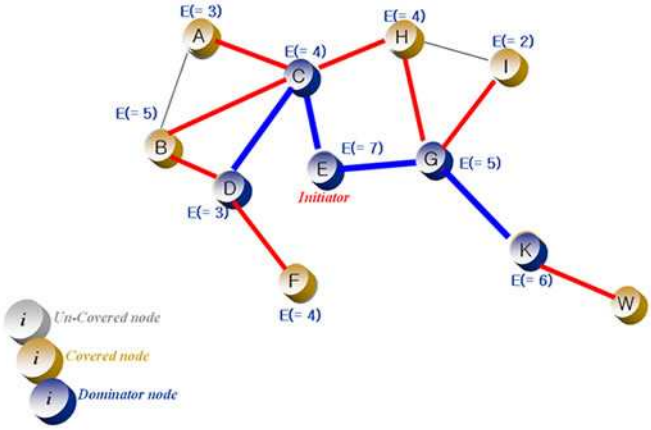

Figure 4.33: Node D switches to the DS state and node F switches to the covered state

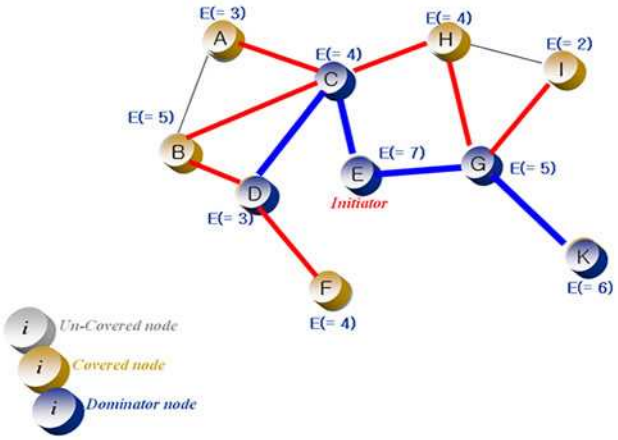

Figure 4.34: Node W leaves the network

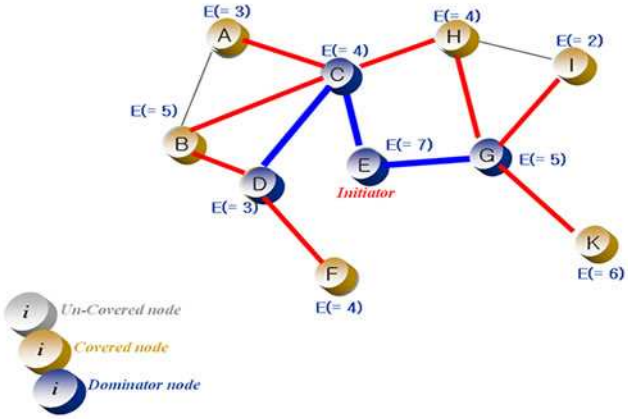

Figure 4.35: Node K switches to the covered state

levels in [82]. Both sets of extended rules were implemented and found to show almost the same results in terms of the metrics used for performance comparisons (e.g., the size of CDS, the average energy level of nodes in the CDS, etc). Hence, only the results of applying Wu's first set of extended rules are shown here.

4.4.5.1 Simulation Setting and Parameter Consideration

It is possible to assume a link between two nodes only if their geometric distance is less than the wireless transmission range. In this simulation, the transmission range of a single station was normalized to 1 unit of distance. Random network topologies were generated by randomly placing nodes in 4×4 and 8×8 square grids of a two-dimensional simulation area. The values of the x and y coordinates were uniformly distributed. The value of T_{max}, $initMax$, and δt were chosen to be 100, 20, and 4 time units respectively. Two scenarios were considered and simulated differently as follows:

1. Static Networks

   static network or network with low mobility - 20 different network topologies were randomly generated. The energy level at each node was generated in a normal distribution, with an average 7.0 and a variance of 2.0. The performance of the protocols was assessed by the average size of CDS, the average energy level of the nodes in the CDS, the minimum energy level of the node in the CDS, and the variance of the minimum energy level of the nodes in the CDS for both 4×4 and 8×8 grids with various nodal densities. Notice that when the network

topology was static, the minimum and average energy level of the nodes in the CDS could be considered an indication of the lifespan of the CDS.

2. Mobility Networks

when nodes are mobile - the simulation started by initializing each node with the same energy level of 100 units. The nodes in the CDS were subtracted by 2.0 and the nodes not in the CDS were subtracted by 0.1 each time the CDS was reconstructed due to nodal movement. The reconstruction continued until a node reached energy level 0 and the number of rounds is counted. This number can be used to indicate how long the network remained operational after a series of CDS reconstructions. The performance of the protocols was assessed by comparing the average sizes of CDS and the number of rounds under random network topologies.

4.4.5.2 Simulation Results and Performance Evaluation

1. Static Networks

In Figures 4.36 and 4.37, the $x-$axis represents the size of the network and the $y-$axis shows the size of the resulting CDS from the five different protocols. It is clear that TCDS consistently generates the smallest CDS and Wu1 consistently generates the largest CDS among all the protocols for both the 4×4 and 8×8 grids. The sizes of the CDS generated by TECDS1 and TECDS2 are very close (mostly within 10%) to those generated by TCDS. When the scale of the network

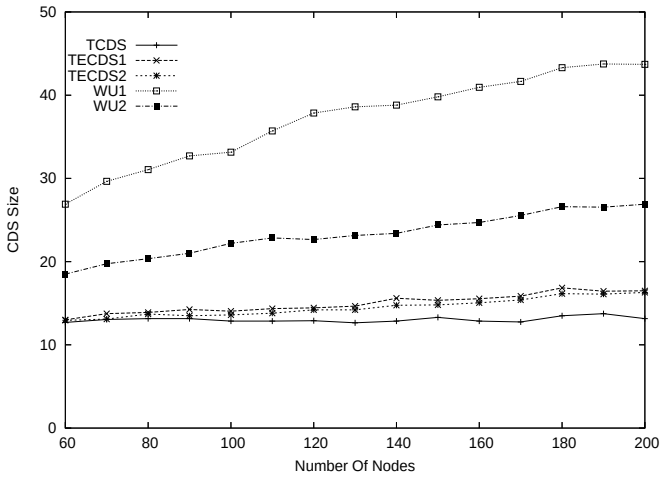

Figure 4.36: CDS size in 4 x 4 square : static network

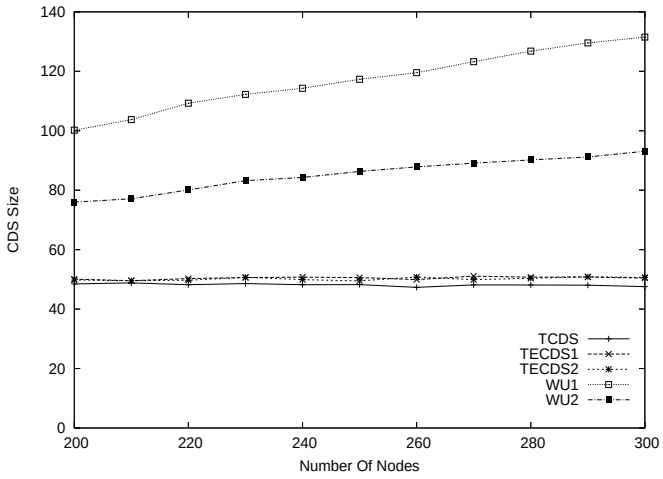

Figure 4.37: CDS size in 8 x 8 square : static network

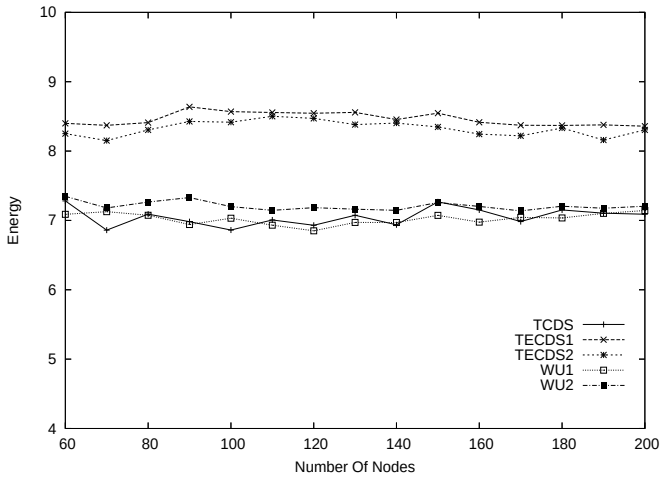

Figure 4.38: CDS Average Energy in a 4 x 4 square : static network

increases from the 4×4 to the 8×8 grid, the performance difference between TCDS and the energy-aware TECDS1 and TECDS2 becomes less covered. Although Wu2 significantly reduces the size of the CDS compared to Wu1, its CDS size is still 40 to 50% larger than the CDS generated by TECDS1 and TECDS2.

In Figures 4.38 and 4.39, the $x-$axis represents the size of network and the $y-$axis shows the average energy level of the nodes in the resulting CDS from the five different protocols. Hence, TECDS1 and TECDS2 are able to achieve an approximately 20% higher average energy level of nodes in CDS than the others for both the 4×4 and 8×8 grids by slightly increasing the CDS size from TCDS. In contrast, Wu1 and TCDS have the lowest average energy level among all the protocols for both the 4×4 and 8×8 grids. It is surprising to find that even though Wu2 considers the energy level at each node, it does not significantly improve the average energy level, as shown in Figures 4.38 and 4.39, where it is a mere 5% better than Wu1.

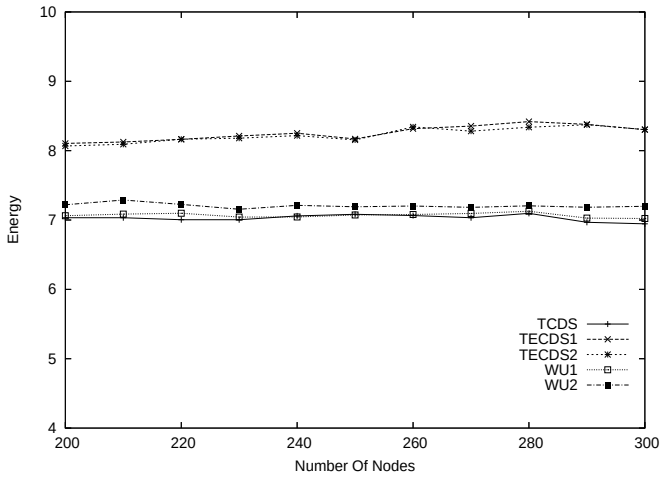

Figure 4.39: CDS Average Energy in an 8 x 8 square : static network

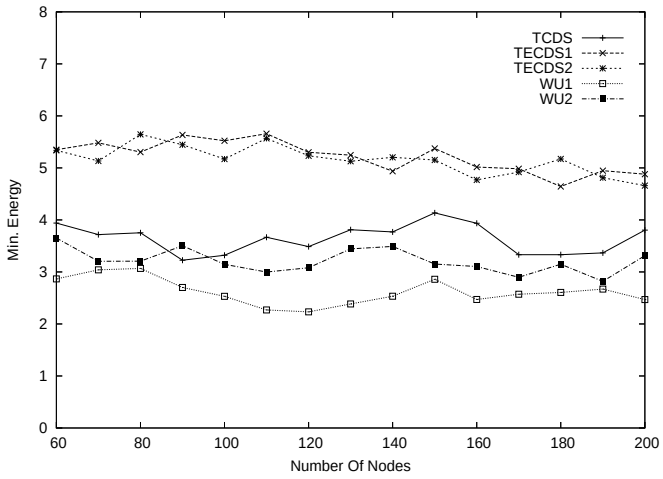

Figure 4.40: CDS Min Energy in a 4 x 4 square : static network

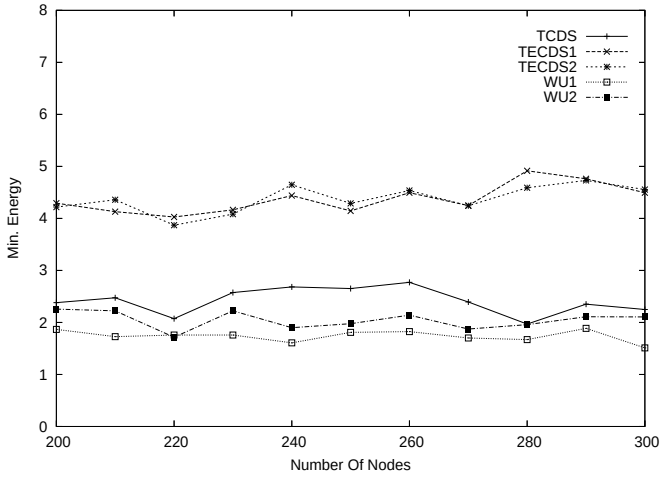

Figure 4.41: CDS Min Energy in an 8 x 8 square : static network

In Figures 4.40 and 4.41, the $x-$axis shows the size of the network and the $y-$axis is the minimum energy level of the nodes in the resulting CDS from five different protocols. From these figures, the energy-aware TECDS1 and TECDS2 protocols select the nodes with higher minimum energy levels than any of the others, while Wu1 selects the nodes having the lowest minimum energy level. These figures indicate that the CDS created by the energy-aware TECDS1 and TECDS2 protocols live longer than any other protocols under a static network. In the 4×4 grid, the minimum energy level of the nodes in the CDS generated by TECDS2 and Wu1 are about 5.1 and 2.6, respectively. This means the minimum energy level of the nodes in the CDS generated by TECDS2 is around 50% higher than that generated by the Wu1 protocol. For the 8×8 grid, the performance of the minimum energy level for different protocols shows a similar trend.

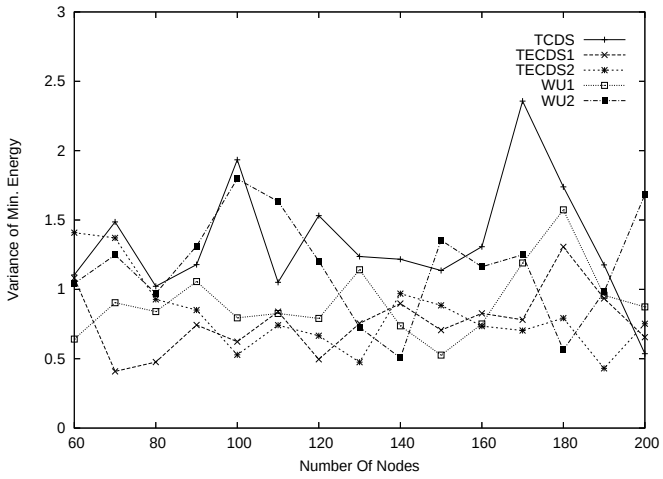

Figure 4.42: Variance of Min Energy in a 4 x 4 square : static network

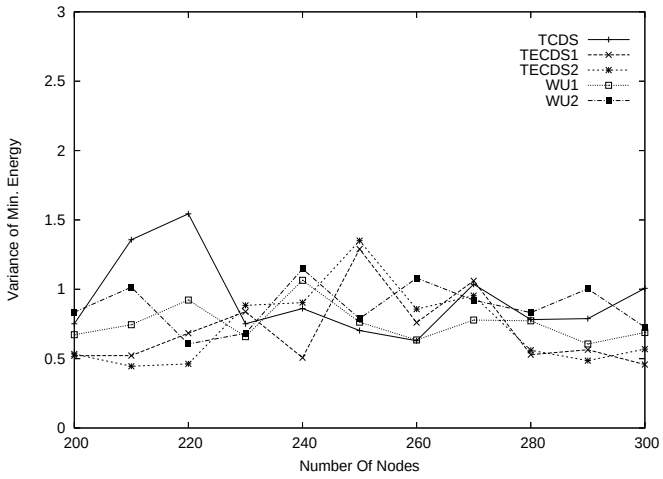

Figure 4.43: Variance of Min Energy in an 8 x 8 square : static network

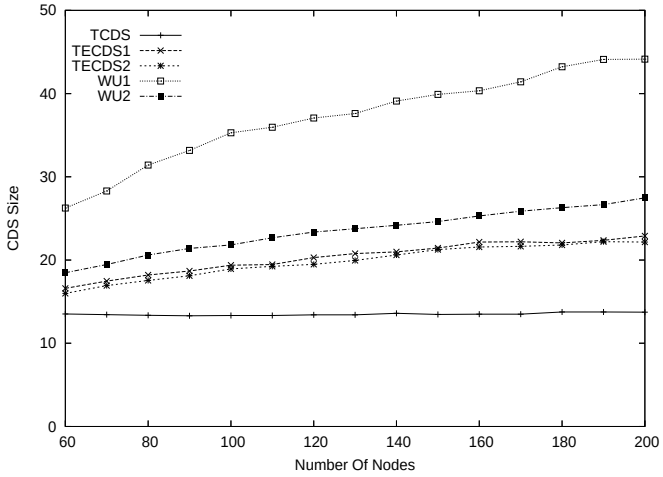

Figure 4.44: CDS size in a 4 x 4 square : mobile network

In Figures 4.42 and 4.43, the $x-$axis is once again the size of the network and the $y-$axis shows the variance of the minimum energy level of the nodes in the resulting CDS from the five different protocols. In these figures, TECDS1 shows a smaller variance than the other protocols when the network size is small. This implies that the performance of TECDS1 is more stable than that of other protocols when the network size is small. However, when the network size is bigger than 100 nodes, both TECDS1 and TECDS2 both demonstrate relatively smaller variances than the other protocols. It is interesting to see that the variances of Wu2 fluctuate and, in general, are slightly bigger than those of TECDS1 and TECDS2 regardless of the size of the network.

2. Networks with High Mobility

In Figures 4.44 and 4.45, the $x-$axis shows the size of the network and the $y-$axis is the size of the resulting CDS from five different protocols. For a 4×4 grid, the

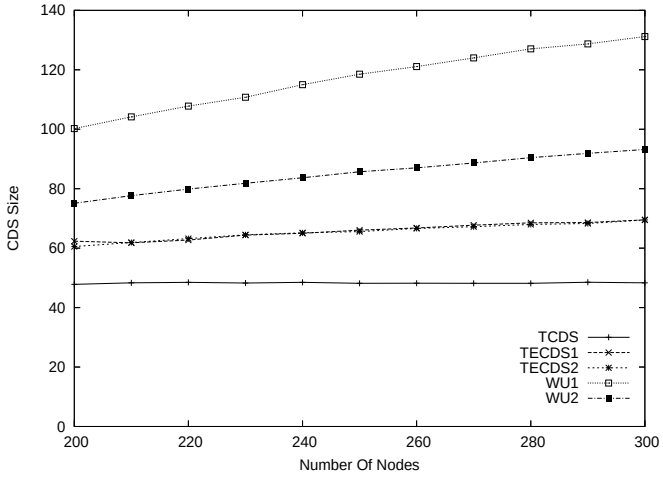

Figure 4.45: CDS size in an 8 x 8 square : mobile network

size of the CDS obtained by Wu1 is at least twice as large as the size of the CDS generated by TCDS, regardless of the size of the network. While the size of the CDS generated by TECDS1 and TECDS2 is approximately 20 to 40% larger than that generated by TCDS, it is always at least 20% smaller than that generated by Wu2. The same performance pattern for these protocols is shown in the case of the 8×8 grid.

In Figures 4.46 and 4.47, the $x-$axis represents the size of the network and the $y-$axis shows the number of rounds from the five CDS protocols. Here, the number of rounds can be thought of as the lifespan of the network when the network topology changes frequently. In both figures, it is obvious that the number of rounds obtained by the energy-aware TECDS1 and TECDS2 protocols are both 40 to 50% higher than that obtained by Wu2 for both the 4×4 grid and 8×8 grid scenarios. TCDS and Wu1 do not consider energy levels when they construct the CDS, so both can

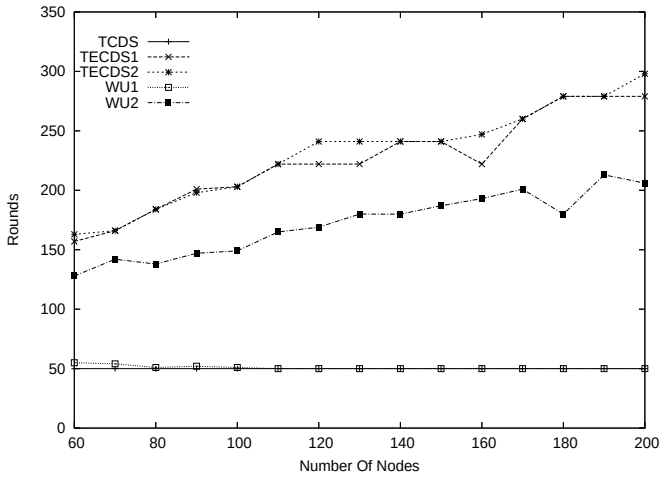

Figure 4.46: The number of rounds in a 4 x 4 square : mobile network

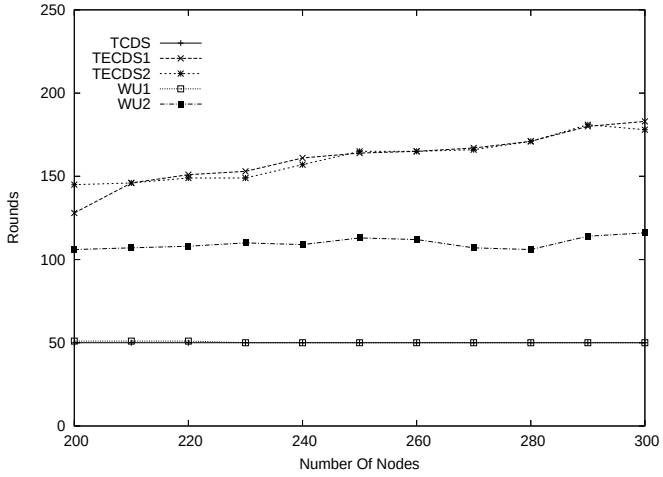

Figure 4.47: The number of rounds in an 8 x 8 square : mobile network

only sustain approximately 50 rounds regardless of the size of the network. There are two reasons why the TECDS1 and TECDS2 protocols produce a higher number of rounds. First, since the size of the CDS generated by TECDS1 and TECDS2 is always smaller than that generated by Wu1 and Wu2 (as shown in Figures 4.44 and 4.45), the energy consumption of the network per unit time is considerably less if our protocols are used. Second, our protocols successfully distribute the CDS load to every node in the network, so that the lifespan of the whole network is improved.

The results show that the proposed energy-aware TECDS1 and TECDS2 protocols produce a better CDS in terms of both the average/minimum energy levels of nodes in the CDS and the number of rounds that represent the lifespan of the CDS and the network. In addition, the size of the CDS generated by both TECDS1 and TECDS2 is consistently smaller than that generated by either Wu1 or Wu2.

CHAPTER 5

CONCLUSION

This chapter summarizes the research and highlights its contributions to the field of mobile ad hoc and sensor networks, and discusses future research direction.

5.1 Summary and Contribution

As a special type of distributed systems, wireless ad hoc and sensor networks accomplish tasks using various distributed protocols. In these protocols, the order of events has a significant impact on the performance. To obtain and manipulate the order of events in a distributed system, both the logical clock and defer timer are commonly used.

In this dissertation, the uses of the defer timer were reviewed along with its effect on the design of various protocols in ad hoc and sensor networks. Based on these observations, the concept of the defer timer was extended to the construction of the virtual backbone for wireless ad hoc and sensor networks. Specifically, three timer based protocols were proposed: TDS for dominating set construction, TCDS for connected dominating set construction, and TECDS for energy aware CDS construction. In these protocols, each node adaptively sets up a defer timer based on the number of uncovered neighbors and determines whether or not to join the DS/CDS when the timer expires. The proposed protocols possess all the desired properties for an ideal DS/CDS construction protocol. Using the defer timer, it was possible to design a set of DS/CDS protocols that are simple, distributed, inexpensive (i.e., introduce no extra messages on need for computation), and adaptive to

nodal mobility. In addition, the TECDS protocols take into account the energy level at each station. The simulation results showed that the dominating set and connected dominating set constructed by the timer-based protocols are very competitive in terms of size and other properties.

In general, the protocols based on the defer timer can easily accommodate a mixed set of considerations as long as each consideration can be measured quantitatively. For instance, in this dissertation, with minimum modifications from TCDS, TECDS was able to take each node's energy level into consideration. This is one of the most important advantages of timer-based protocols.

5.2 Future Research Direction

As pointed out in Section 2.2, wireless mesh networks have received increasing research attention in recent years due to its ability to provide broadband networking infrastructure for large business enterprizes and Internet access to residence in rural areas. The virtual backbone construction is one of the important research issues in wireless mesh networks [87]. Our CDS construction protocols for wireless networks is fully distributed, based on localized information, and is known to generate very competitive size of CDS. If the gateway in a wireless mesh network is selected as a initiator for constructing a CDS, our timer-based DS and CDS construction protocols can be the best candidates for the virtual backbone construction for wireless mesh networks. However, as indicated in [88], each mesh router in a mesh network may be equipped with multiple radio interfaces and each radio interface may have different transmission radius. This suggests that a mesh router in

a mesh network may have different sets of neighbors depending on what types of the radio interfaces it has. How to properly define the DS and CDS under such an environment and how to modify our timer-based protocols for such a multi-radio mesh network can be an interesting future direction of this research.

Research on Media Access Control (MAC) in wireless networks is a another challenging field due to the difficulties caused by transmission errors, collisions, and hidden nodes. These difficulties become even more severe when support is provided for multicast communication in wireless networks. Such support is necessary for delivering acceptable quality of service in many applications of wireless communications, such as emergency reporting and video conferencing. For these applications, the MAC protocol should have the ability to avoid interference and collisions, and at the same time provide a reliable multicast service. The proposed multicast MAC protocols in [89, 90] are unreliable in that when a multicast is done, the sender does not know whether every intended receiver has received the data successfully. To achieve our goals, the concept of defer timer can be used to provide a simple coordination mechanism between multicast receipents at the MAC layer.

BIBLIOGRAPHY

[1] J.M.Kahn, et.al., "Next Century Challenges: Mobile Networking for Smart Dust," ACM Mobicom, 1999.

[2] J. M. Kahn, R. H. Katz and K. S. J. Pister, "Emerging challenges: Mobile Networking for Smart Dust", Journal of Communications And Networks, vol. 2(3), pp. 188-196, September 2000.

[3] J. M. Kahn, R. H. Katz and K. S. J. Pister. "Next Century Challenges:Mobile Networking for Smart Dust," ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 99). pp. 217-278. August 1999.

[4] J. Wu, F. Dai, M. Gao, and I. Stojmenovic, "On Calculating Power-Aware Connected Dominating Sets for Efficient Routing in Ad Hoc Wireless Networks," Journal of Communications and Networks, Vol. 4, No. 1, March 2002.

[5] J. Wu and H. Li, "A Dominating-Set-Based Routing Scheme in Ad Hoc Wireless Networks," special issue on Wireless Networks in the Telecommunication Systems Journal, Vol. 3, 2001, 63-84. 7.

[6] J. Wu, "Extended Dominating-Set-Based Routing in Ad Hoc Wireless Networks with Unidirectional Links," IEEE Transactiosn on Parallel and Distributed Computing, 22, 1-4, 2002, 327-340.

[7] D. Zhou, M. Sun, and T. Lai, "A Timer-based Protocol for Connected Dominating Set Construction in IEEE 802.11 Multihop Mobile Ad Hoc Networks," IEEE SAINT 2005, pp.2-8, Trento Italy,

[8] M. Sun, S. Wang, C.K. Chang, T.H. Lai, S. Hiroyuki, and H. Okada, Interference-Aware MAC Scheduling and SAR Policies for Bluetooth Scatternets," Proc. IEEE GLOBECOM 2002, pp. 11-15, Taipei, Taiwan, Nov. 2002.

[9] D. Johnson and D. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," T. Lmielinski and H. Korth, Eds. Mobile Computing, Ch.5, Kluwer, 1996.

[10] C. Perkins and P. Bhagwat. "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," In ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications, pages 234 - 244, 1994.

[11] C.E. Perkins and E.M. Royer, "Ad-hoc on-demand distance vector routing," Proc. WMCSA '99, pp. 90 - 100, Feb 1999

[12] Nikaein Navid, Wu Shiyi, and Christian Bonnet. "HARP: Hybrid Ad hoc Routing Protocol," International Symposium on Telecommunications, IST 2001, 2001.

[13] Zygmunt J. Haas and Marc R. Pearlman, "The Performance of Query Control Schemes for the Zone Routing Protocol," IEEE/ACM Transactions on Networking, Aug 2001.

[14] L. Kleinrock and F. Kamoun, "Hierarchical Routing for large networks; performance evaluation and optimization," Computer Networks, Vo 1, pages 155-174, 1977.

[15] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu, "The Broadcasting Storm Problem in Mobile Ad Hoc Network," In Int'l Conf. on Mobile Computing Network (MOBICOM), pp.151-162, Aug. 1999.

[16] Y.-C. Tseng, S.-Y. Ni, and E.-Y. Shih, "Adaptive Approaches to Relieving Broadcasting Storms in a Wireless Multiple Ad Hoc Network," In 21st Int'l Conf. on Dist. Computing Systems, pp.481-488, Apr.2001.

[17] Wei Peng, Xi-Cheng Lu, "On the Reduction of Broadcast Redundancy in Mobile Ad Hoc Networks," IEEE Proc. MobiHoc 2000.

[18] A. Ephremies, T. Truong, "Scheduling broadcasts in multihop radio network," IEEE Trans. On Communications, Vol.2, No. 4, Apr. 1990.

[19] D.J Baker and A. Ephremides, "A Distributed Algorithm for Organizing Mobile Radio Telecommunication Networks," In Proceedings of the Second International Conference on Distributed Computing Systems, April 1981, 476-483.

[20] Chen, W.P., Hou, J., Sha, L., "Dynamic clustering for acoustic target tracking in wireless sensor networks," 11th IEEE International Conference on Network Protocols, pp. 284 - 294, 2003.

[21] http://www.bluetooth.com, "Specification of the Bluetooth System," Volume 1, Core, Version 1.1, February 22, 2001.

[22] B.N. Clark, C.J. Colburn, and D. S. Johnson, "Unit dist graphs," Discrete Mathmatics, 86, pp. 165-167, 1990.

[23] G. Miklos et al., "Performance Aspects of Bluetooth Scatternet Formation," in Proceedings of ACM MobiHoc 2000.

[24] T. Salonidis et al., "Distributed Topology Construction of Bluetooth Personal Area Networks," Proc. IEEE Infocom, pp. 1577-1586, April 2001.

[25] M. Bahramgiri, M.T. Hajiaghayi, and V.S. Mirrokni. "Fault-tolerant and 3-Dimensional Distributed Topology Control Algorithms in Wireless Multi-hop Networks." In IEEE Int. Conf. on Computer Communications and Networks (ICCCN02), pp. 392–397, 2002.

[26] M. Sun, C.K. Chang, and T.H. Lai, "A Self-Routing Topology for Bluetooth Scatternets," Proc. I-SPAN 2002, pp. 13-18, Manila, Philippines, May 2002.

[27] Gerard Tel, "Introduction to Distributed Algorithms," 2nd edition, Cambridge University Press, 2000.

[28] Lamport, L. "Time, Clock, and the Ordering of Events in a Distributed System," Communications of the ACM, vol.21, no.7, July 1978, pp.558-565.

[29] Mattern, F., "Virtual Time and Global States of Distributed Systems," Parallel and Distributed Algorithms, Elsevier Science, North-Holland, 1989, pp. 215-226.

[30] Mukesh S., Niranjan G., "Advanced Concepts In Operating Systems: Distributed, Database, and Multiprocessor Operating Systems," McGraw-Hill Series in Computer Science, 1994.

[31] Fletcher, J. G. and Watson, R. W. "Mechanisms for a reliable timer-based protocol," Computer Networks 2, pp.271-290, 1978.

[32] Tel, G., "Topics in Distributed Algorithms," Vol. 1 of Cambridge Int. Series on Parallel Computation, Cambridge University Press, pp. 240, 1991.

[33] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan, "The Cricket Location - Support system," Proc. 6th ACM MOBICOM, Boston, MA, August 2000.

[34] N. B. Priyantha, A. Miu, H. Balakrishnan, and S. Teller, "The Cricket Compass for Context-Aware Mobile Applications," Proc. 7th ACM MOBICOM, Rome, Italy, July 2001.

[35] http://www.navcen.uscg.gov/pubs/gps/sigspec/gpsspsa.pdf, "Global Positioning System Standard Positioning Service Specification", 2nd Edition, June 2, 1995.

[36] http://standards.ieee.org/getieee802/ "IEEE 802 working group"

[37] IEEE LAN MAN Standards Committee, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std. 802.11-1997, The Institute of Electrical and Electronics Engineers, New York, New York, 1997.

[38] http://standards.ieee.org, IEEE standards organization.

[39] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, IEEE Std 802.11b-1999/Cor 1-2001 (R2003).

[40] IEEE LAN MAN Standards Committee, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std. 802.11a-1999, The Institute of Electrical and Electronics Engineers, New York, New York, Dec. 1999.

[41] F. Andre, J.-M. Bonnin, B. Deniaud, K. Guillouard, N. Montavont, T. Noel, L. Suciu. "Optimized Support of Multiple Wireless Interfaces within an IPv6 End-Terminal", in : Smart Object Conference (SOC), Grenoble, France, May, 2003.

[42] D. Johnson and D. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," T. Lmielinski and H. Korth, Eds. Mobile Computing, Ch.5, Kluwer, 1996.

[43] R. Kreula and H. Haapasalo, "Transfer delay at ATM LAN emulation and classical IP over ATM," in Int. Conf. on Communication, 1997, pp. 478-482.

[44] S. Butenko, X. Cheng, D.Z. Du, and P. Pardalos, "On the Construction of Virtual Backbone for Ad Hoc Wireless Networks," In Proc. 2nd Conference on Cooperative Control and Optimization.

[45] Jeroen Hoebeke, Ingrid Moerman, Bart Dhoedt and Piet Demeester, "An overview of mobile ad hoc networks: applications and challenges", FITCE Annual Conference, Belgium, September 8-11, 2004.

[46] Kim, K., Kim, C., and T. Kim, "A Seamless Handover Mechanism for IEEE 802.16e Broadband Wireless Access", International Conference on Computational Science, vol. 2, pp. 527-534, 2005.

[47] http://standards.ieee.org/announcements/p80220app.html

[48] http://www.ieee802.org/15/, IEEE 802.15 Working Group for WPAN.

[49] L. Lamport, Time, Clocks, and the Ordering of Events in a Distributed System, Communications of the ACM 21 (7), 558-565, 1978.

[50] Mattern, F., "Virtual Time and Global States of Distributed Systems" (M.Cosnard et al.eds.), Parallel and Distributed Algorithms, Elsevier Science, North-Holland, 1989, pp.215-226.

[51] Fidge, J., "Partial Orders for Parallel Debugging," Proceedings of the ACM SIGPLAN-SIGOPS Workshop on Parallel and Distributed Debugging, 1988, pp.183-194.

[52] Singhal M. Advanced Concepts in Distributed Operating Systems. McGraw Hill, 1994.

[53] R. Schwarz and F. Mattern, Detecting Causal Relationships in Distributed Computations. in Search of the Holy Grail, Distributed Computing 7, 149-174, 1994.

[54] Carroll Morgan. Global and logical time in distributed algorithms. Information Processing Letters, 20:189–194, May 1985.

[55] K.M. Chandy and L. Lamport. Distributed Snapshots: Determining Global States of Distributed Systems. ACM Transactions on Computer Systems, Vol. 3, No. 1, pp. 63-75, Feb. 1985.

[56] J. Fowler and W. Zwaenepoel. Causal Distributed Breakpoints. Proc. 10th Int. Conference on Distributed Computing Systems, Paris, France, pp. 134-141, May 1990.

[57] F. Mattern. Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation. Journal of Parallel and Distributed Computing, to appear, 1993.

[58] Fidge, J., "Partial Orders for Parallel Debugging", Proceddings of the ACM SIGPLAN/SIGOPS Workshop on Parallel and Distributed Debugging," 1988, pp.183-194.

[59] R. Schiefer, P. van der Stok. "VIPER: a tool for the visualisation of parallel programs," pdp, p. 540, 3rd Euromicro Workshop on Parallel and Distributed Processing, 1995.

[60] Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications, IEEE 802.3 Std, March 2002.

[61] B. O'Hara and A. Petrick, The IEEE 802.11 Handbook: A Designer's Companion, IEEE Press, 1999.

[62] G.M. Djuknic and R.E. Richton, "Geolocation and Assisted GPS," IEEE Computers magazine, pp.123-125, Feb, 2001.

[63] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu, "The Broadcasting Storm Problem in Mobile Ad Hoc Network," In Int'l Conf. on Mobile Computing Network (MOBICOM), pp.151-162, Aug. 1999.

[64] Y.-C. Tseng, S.-Y. Ni, and E.-Y. Shih, "Adaptive Approaches to Relieving Broadcasting Storms in a Wireless Multiple Ad Hoc Network," In 21st Int'l Conf. on Dist. Computing Systems, pp.481-488, Apr.2001.

[65] Chen,, W.P., Hou, J., Sha, L., "Dynamic clustering for acoustic target tracking in wireless sensor networks," 11th IEEE International Conference on Network Protocols, pp. 284 - 294, 2003.

[66] S. Datta and I. Stojmenovic "Internal Node and Shortcut Based Routing with Guaranteed Delivery in Wireless Networks," Cluster Computing 5, 169-178, 2002.

[67] J. Cartigny, D. Simplot, and I. Stojemenovic, "Localized Minimum-energy Broadcasting in Ad-hoc Networks," IEEE INFOCOM 2003.

[68] M. Sun, W. Feng and T.H. Lai, "Location Aided Broadcast in Wireless Ad Hoc Networks," Proc. IEEE GLOBECOM 2001, pp. 2842-2846, San Antonio, TX, November 2001.

[69] Mario Cagalj , Jean-Pierre Hubaux, and Christian Enz, "Minimum-Energy Broadcast in All-Wireless Networks : NP-Completeness and Distribution Issuses," Mobicom 2002,September,Atlanat,GA,USA.

[70] Y. Wang J. J. Garcia-Luna-Aceves, "Collision avoidance in multi-hop ad hoc networks," IEEE MASCOTS 2002, pp. 145 -154.

[71] M. R. Garey and D.S.Johnson, "Computers and Intractability-A Guide to the Theory of NP-Completeness," San Francisco, CA:Freeman, 1979.

[72] D. S. Hochbaum, "Approximation Algorithms for NP-Hard Problems," Boston, MA, PWA Publishing Company, 1995.

[73] Sudipto Guha and Samir Khuller, "Approximation Algorithms for Connected Dominating Sets," European Symposium on Algorithms, pp. 179-193, 1996.

[74] Sudipto Guha and Samir Khuller "Approximation Algorithms for Connected Dominating Sets", Algorithmica 20(4): 374-387, 1998.

[75] B. S. Baker, "Approximation algorithm for NP-complete problems on planar graphs", J. the ACM, 41(1), pp.153-180, 1994.

[76] Raghupathy Sivakumar, Prasun Sinha, and Vaduvur Bharghavan, "CEDAR: A Core-Extraction Distributed Ad Hoc Routing Algorithm", IEEE Journal on Selected Area in Communication, Vol. 17, No. 8, August, 1999.

[77] Khaled M. Alzoubi, Peng-Jun Wan, and Ophir Frieder, "Message-Optimal Connected Dominating Sets in Mobil Ad Hoc Networks", MOBIHOC'02, June 9-11, 2002.

[78] Khaled M. Alzoubi, Peng-Jun Wan, and Ophir Frieder, "Distributed Heuristics for Connected Dominating Sets in Wireless Ad hoc Networks", Journal on Communication Networks, 2002.

[79] C.-K. Toh, "Maximum battery life routing to support ubiquitous mobile computing in wireless ad hoc networks," IEEE Commun. Maga., pp. 138. 147, June 2001.

[80] N. Bambos, "Toward power-sensitive network architectures in wireless communications: Concepts, issues, and design aspects," IEEE Personal Commun., pp. 50.59, June 2000.

[81] L. M. Feeney, "A QoS Aware Power Save Protocol for Wireless Ad Hoc Networks," in IFIP Med-hoc-Net 2002, September 2002.

[82] J. Wu, Fei Dai, Ming Gao, and Ivan Stojmenovic, "On Calculating Power-Aware Connected Dominating Sets for Efficient Routing in Ad Hoc Wireless Networks," Journal of Communications and Networks, Vol4,No1, March 2002.

[83] J. Wu, B. Wu, and I. Stojmenovic, "Power-Aware Broadcasting and Activity Scheduling in Ad Hoc Wireless Networks Using Connected Dominating Sets," Wireless Communications and Mobile Computing, a special issue on Research in Ad Hoc Networking, Smart Sensing, and Pervasive Computing, 3, 4, June 2003, 425-438.

[84] Bluetooth SIG, The Official Bluetooth Wireless Info Site, 2003. Avaliable at www.bluetooth.com.

[85] Padhye, J., Draves, R. and B. Zill, "Routing in multi-radio, multi-hop wireless mesh networks", Proceedings of ACM MobiCom Conference, September 2003.

[86] C.Perkins, E.Templine, S.Das, "Ad hoc on-demand distance vector (AODV) routing, IETF RFC3561, Jult 2003.

[87] Ian F. Akyildiz, and Xudong Wang, "A Survey on Wireless Mesh Network", IEEE Radio Communications, September 2005.

[88] R. Draves, J. Padhye, and B. Zill, Routing in multi-radio, multi-hop wireless mesh networks, Proceedings of ACM MobiCom, Philadelphia, PA, September 2004.

[89] K. Tang and M. Gerla, MAC Layer Broadcast Support in 802.11 Wireless Networks, Proc. IEEE MILCOM 2000, pp. 544-548, Oct. 2000.

[90] K. Tang and M. Gerla, Random Access MAC for Efficient Broadcast Support in Ad Hoc Networks, Proc. IEEE WCNC 2000, pp. 454-459, Sep. 2000.

APPENDICES

APPENDIX A

SOURCE CODE OF THE TECDS IN THE STATIC NETWORK

```
// TECDS1.cpp : Timer-based Energy aware CDS protocol
// Network environment: Static network
// Initiator election : the node with most energy is picked as the initiator

#include <fstream>  // For input and output file
#include <math.h>
#include <stdlib.h>
#include <algorithm>
#include <iostream>
#include <set>
#include <queue>

#define MAXNODE 2000    // maximum number of nodes
#define DEBUG 1
#define MAXCOUNT 30
#define MAXINIT 3000
#define tmax 100
using namespace std;
enum StationStatus {uncovered, covered, bridge, gateway, inDS};
typedef struct node {
        int index;      // index of the node
        double x, y;    // x,y coordinate
        int degree;
        int status;
        int inds;       // 1: node in DS; 0: node not in DS
        int covered;    // 1: covered by DS; 0: not covered by DS
        int visited;    // used for checking connected graph
        int on;         // the mobile station on/off flag
        int valid;      // node used/unused for connectivity check
        set<int> nbr;   // neighbor set
        int initiator;  // initiator
        int initCount;
        int DSTimer;    // will enter DS when DSTimer is 0
        int ODSTimer;   // DSTimer value when first set
        double energy;  // Energy
} Node;

struct ANSWER {
 double size;
 double energy;
 double min;
 double var;
};

ANSWER answer[20];
Node station[MAXNODE]; // maximum number of stations
int m;  // total number of stations
int h;  // number of units X/Y
int n;  // total number of stations switched off

double distance(double x1, double y1, double x2, double y2) {
     return sqrt((x1 - x2)*(x1 - x2) + (y1 - y2)*(y1 - y2));
}

// connectivity check
int connect(){
        int curr; // first node of the queue

        // mark all nodes not visited
        for (int i = 0; i < m; i++) {
                station[i].visited = 0;
        }

        // find the first valid node
        for (i = 0; i < m; i++) {
                if (station[i].valid == 1) {
                        curr = i;
                        break;
                }
        }

        if ((curr < 0) || (curr >= m)) {
                cout << "No valid node!" << endl;
                exit(-1);
        }

        queue<int> q;
        q.push(curr);
        station[curr].visited = 1;

        while ( !q.empty() ) {
                curr = q.front();
                q.pop();
                for (set<int>::const_iterator siter=(station[curr].nbr).begin();
                siter!=(station[curr].nbr).end(); ++siter) {
                        if ((station[*siter].visited==0)&&(station[*siter].valid==1)){
                                station[*siter].visited = 1;
                                q.push(*siter);
                        }
                }
        }
        for (i = 0; i < m; i++) {
                if ((station[i].visited == 0) && (station[i].valid == 1)) {
                        return 0;
                }
        }
        return 1;
}
```

```
//Check DS connectivity
void checkDS(int n){
        int dscount = 0;
        set<int> union_set;
        insert_iterator<set<int> > union_ins(union_set, union_set.begin());
        union_set.clear();

        for (int i = 0; i < n; i++) {
                if (station[i].status == inDS)  {
                        dscount++;
                        set_union(union_set.begin(), union_set.end(),
                                (station[i].nbr).begin(), (station[i].nbr).end(),
                                union_ins);
                }
        }

        //cout << "DS count = " << dscount << endl;
        //cout << "cover count == " <<  union_set.size() << endl;
        if ( union_set.size() == n ){
                //cout << "It is a DS " << endl;
        } else {
                //cout << "It is Not a DS" << endl;
        }
}


extern int announce(int i, int j);


//anounce station i is the initiator, j is the announer
int announce(int i, int j){
       for (set<int>::const_iterator siter=(station[j].nbr).begin();
   siter!=(station[j].nbr).end(); ++siter) {
             if ((station[*siter].on > 0) && (station[*siter].initiator == MAXINIT)
                  && (station[*siter].initiator>= i) ) {
    station[*siter].initiator = i;
    station[*siter].initCount = MAXCOUNT;
       }
   if ((station[*siter].on > 0) && (station[*siter].initiator != MAXINIT)
                        && (station[*siter].initiator >= i)){
    station[*siter].initiator = i;
    station[*siter].initCount = MAXCOUNT;
    // announce(i,*siter);
   }
 }
 return 1;
}



// i is the DS node
void broadDS(int i){
 for (set<int>::const_iterator siter=(station[i].nbr).begin();
                siter!=(station[i].nbr).end(); ++siter) {
                        if ((station[*siter].on > 0) && (station[*siter].status == uncovered) ){
   station[*siter].status = covered;
   }
 }
}


// get number of uncovered nodes of i
int getUncovered(int i){
       int num = 0;

 for (set<int>::const_iterator siter=(station[i].nbr).begin();
                siter!=(station[i].nbr).end(); ++siter) {
                        if ((station[*siter].on > 0) && (station[*siter].status == uncovered)){
   num++;
      }
 }
 return num;
}



// return 1 if the DS is conencted
int checkConnectedDS(int num){
 int result = 1;
 int found = 0;

   for(int i=0; i < num; i++){
  if (station[i].status == inDS){
   found = 0;
   for (set<int>::const_iterator siter=(station[i].nbr).begin();
                 siter!=(station[i].nbr).end(); ++siter) {
                 if ((station[*siter].on > 0) && (station[*siter].status == inDS) ) {
    found = 1;
    break;
    }
   } // end for each neighbor
   if (found == 0) return 0;
  }
 }
}
```

```
// normal distribution to generate energy to each node
// added this line Mar.30
double normal(double avg, double std)
{
   static int parity = 0;
   static double nextresult;
   double sam1, sam2, rad;


   if (std == 0) return avg;
   if (parity == 0) {
     sam1 = 2*((double)(rand())/RAND_MAX) - 1;
     sam2 = 2*((double)(rand())/RAND_MAX) - 1;
     while ((rad = sam1*sam1 + sam2*sam2) >= 1) {
       sam1 = 2*((double)(rand())/RAND_MAX) - 1;
       sam2 = 2*((double)(rand())/RAND_MAX) - 1;
     }
     rad = sqrt((-2*log(rad))/rad);
     nextresult = sam2 * rad;
     parity = 1;
     return (sam1 * rad * std + avg);
   }
   else {
     parity = 0;
     return (nextresult * std + avg);
   }
}

// select the initiator by energy
// added this line Mar.30
void SelectInitiatorByEnergy() {

 int i;
 int initiator;
 double largestEnergy;
 int largestnbrNum;
 int firstTime = 0;


  largestEnergy = station[0].energy;

  for(i = 1; i < m; i++) {
  if ( station[i].energy > largestEnergy) {
   largestEnergy = station[i].energy;
   firstTime = i;
  }
     }

//  cout << "Largest nbr " << largestnbrNum << endl;
  //cout << "  energy = " << largestEnergy << endl;

  // add to calaulate largest energy
  largestnbrNum = (station[firstTime].nbr).size();
  initiator = firstTime;

     for(i = firstTime; i < m; i++) {
   if ( station[i].energy == largestEnergy && (station[i].nbr).size() > largestnbrNum) {
   largestnbrNum = (station[i].nbr).size();
   initiator = i;
  }
     }
  //cout << "  size = " << largestnbrNum << endl;

  for(i = 0; i < m; i++) {
  station[i].initiator = initiator;
    }

}

int main(int argc, char *argv[])
{

 int i;
 int numOfRound = 1;
 int goExit = 0;
 int goWhile = 0;
 double totalDSCount = 0;
 double totalEnergy = 0.0;
 int round =1;

 if (argc != 4) {
  //cout << "usage: peter stations hops, where" << endl;
  //cout << "\tstations: total number of stations" << endl;
  //cout << "\thops: number of units X/Y, total hops * hops units" << endl;
           return(EXIT_FAILURE);
 }


      m = atoi(argv[1]); // total number of stations
      h = atoi(argv[2]); // number of units X/Y


  // For output File
  char resultFile[50];
  strcpy(resultFile,argv[3]);
```

```
while(1) //add wile loop Apr. 15
  {
   int temp=0;
   ofstream outFile;
   outFile.open(resultFile, ios::out | ios::app);
   srand(numOfRound);
   //srand(4);


   for(i= 0; i < m; i++) {
    (station[i].nbr).clear();
   }


   //assign enenrgy level to each node
  for(i = 0; i < m; i++) {
   station[i].energy = normal(7.0, 2.0);
   //station[i].energy=100.00;
   //cout << "station[" << i << "] Energy Level=" << station[i].energy << endl;
  }


 // randomly generates the positions of all nodes
 // added this line Mar.30
      for(i = 0; i < m; i++) {
 station[i].index = m;
        station[i].x = (double)(h*rand())/RAND_MAX;
          station[i].y = (double)(h*rand())/RAND_MAX;
 }



 // sets the neighbors of all nodes
 for(i = 0; i < m; i++) {
  for(int j = i + 1; j < m; j++)
   if (distance(station[i].x, station[i].y,
         station[j].x, station[j].y) <= 1) {
    (station[i].nbr).insert(j);
    (station[j].nbr).insert(i);
   }
  //cout << "i = " << i << "  Neighbours:";
                          //copy((station[i].nbr).begin(),(station[i].nbr).end(),
                            //   ostream_iterator<int>(cout," "));
                          //cout << endl;

  //cout << "i = " << i << "  size = " << (station[i].nbr).size();

                          //cout << endl;
 }
 // set the valid flag, degree etc
 for (i = 0; i < m; i++) {
  station[i].valid = 1;
  station[i].on = 1;
  station[i].degree = (station[i].nbr).size();
  station[i].inds = 0; station[i].covered = 0;
  station[i].initiator = MAXINIT;
  station[i].initCount = MAXCOUNT;
  station[i].status = uncovered;
  station[i].DSTimer = -1;
  station[i].ODSTimer = -1;

 }

 if ( 0 == connect() ) {
  cout << "The graph is not connected" << endl;

  //return(EXIT_FAILURE);
  goWhile = 1 ;
 }
 else{

 //initiator election and announce to the whole topology (by calling anounce(i,i) function)
 for (int j=0; j < 20; j++){ // j: round number
  if (j== 100){
   station[0].on = 0;
   station[0].initiator = MAXINIT;
  }
  for (int i = 0; i < m; i++){
   if (station[i].initiator == MAXINIT  && station[i].on ){
    station[i].initiator = i;
    // cout << "announce itself i=" << i << "init=" << station[i].initiator << endl;
    announce(i,i);
   }
   if (station[i].initiator < i  && station[i].on ){
    announce(station[i].initiator,i);
    // cout << "forward announce i=" << i << "init=" << station[i].initiator << endl;
   }
   station[i].initCount--;
   //cout<< "i="<<i <<"round=" << j << "initCount=" <<  station[i].initCount << endl;
   if (station[i].initCount == 0 && station[i].on > 0){
    station[i].initiator = i;
    announce(i,i);
   }
  }
 }
```

```
// call function for selecting initiator
 // added this line Mar.30
 SelectInitiatorByEnergy();
 //for (i = 0; i < m; i++){
   cout << "station[" << i << "].initiator =" << station[i].initiator;
                cout << "initCount=" << station[i].initCount << endl;
   //}


 int unum = 0; // number of uncovered nbrs for each node
 int unum1 = 0;
 for (j = 0; j < 2000; j++){
  for (int i = 0; i < m; i++){
   if ( station[i].on > 0 && i == station[i].initiator){
    station[i].status = inDS; // the node move to the dominating set
    broadDS(i);
   }
   if ( station[i].on > 0 && station[i].status == covered ){
    unum = getUncovered(i);
    if (unum > 0){
     unum1 =  tmax /(getUncovered(i)*station[i].energy);

     if (station[i].ODSTimer == -1){
      station[i].DSTimer = unum1;
      station[i].ODSTimer = unum1;
      // cout << "station["<<i<<"].DSTimer=" << station[i].DSTimer << endl;
     } else {
      if (station[i].ODSTimer < unum1){
      station[i].DSTimer = unum1;
                                                  station[i].ODSTimer = unum1;
      }
     }
    } else {
      station[i].DSTimer = -1;
    }
   }
   if (station[i].DSTimer > 0) station[i].DSTimer--;
   if ( station[i].on > 0 && station[i].DSTimer == 0 && station[i].status != inDS){
    station[i].status = inDS;
    broadDS(i);
     cout << "station " << i << "enter DS at round " << j << endl;
   }
  }
 }

 checkDS(m);
 for (i =0; i < m; i++){
  if (station[i].status == inDS) {
   station[i].valid = 1;
  } else {
   station[i].valid = 0;
  }
 }
 int con = connect();
 if (con > 0) {
  //cout << "This DS is connected" << endl;
 } else{
  //cout <<  "This DS is NOT connected" << endl;
 }

  if ( 0 == connect() ) {
  cout << "The graph is not connected" << endl;
  //return(EXIT_FAILURE);
 }
 else{

 /*
  if ( DEBUG ) {
                  for (int i = 0; i < m; i++) {
   if (station[i].status == inDS){
                          cout << "i = " << i << "  Neighbours:";
                          copy((station[i].nbr).begin(),(station[i].nbr).end(),
                                  ostream_iterator<int>(cout," "));
                          cout << endl;
   }
                  }
        }*/


 int dsCount = 0;

 for (i = 0; i < m; i++){
  if (station[i].status == inDS){
   dsCount++;
   //cout << "station " << i << " is DS node" << endl;
   //cout << "i = " << i << "  energy:" << station[i].energy << endl;

   //cout << i << "\t" << station[i].energy << endl;


  }
 }
```

```
//cout << "DS Size =" << dsCount << endl;


  //ofstream outFile;
//outFile.open(resultFile, ios::out | ios::app);
double max=0.0, min=10000.0, avg=0.0, sum=0.0, mint=10000.0, td, tn;

for (i = 0; i < m; i++){
 if (station[i].status == inDS){
  //dsCount++;
  //outFile << i << "\t" << station[i].energy << endl;
  sum=sum+station[i].energy;
  if(max < station[i].energy) {
   max = station[i].energy;
  }
  if(min > station[i].energy) {
   min =station[i].energy;
  }

  td=station[i].energy / 2;

  if ( mint > td)
  mint=td;
 }
 else {
  tn=station[i].energy / 1;
  if ( mint > tn)
  mint=tn;


 }


}

avg=sum/dsCount;

double varSum=0.0, var=0.0;
for (i = 0; i < m; i++){
 if (station[i].status == inDS){
  varSum=varSum+pow((station[i].energy-avg),2);


 }
}

var=varSum/dsCount;


answer[round-1].energy = avg;
answer[round-1].size = dsCount;
answer[round-1].min = min;

int junmo;
double sumEnergy = 0.0;
double sumSize = 0.0;
double sumMin = 0.0;
double sumVar = 0.0;

double avgEnergy = 0.0;
double avgSize = 0.0;
double avgMin = 0.0;
double avgVar = 0.0;

if ( round ==10) {
 for(junmo = 0; junmo < 20; junmo++)
 {
  sumEnergy = sumEnergy + answer[junmo].energy;
  sumSize = sumSize + answer[junmo].size;
  sumMin = sumMin + answer[junmo].min;
 }

 avgEnergy = sumEnergy/round;
 avgSize = sumSize/round;
 avgMin = sumMin/round;


 for(junmo = 0; junmo < 20; junmo++)
 {
  sumVar = sumVar + (avgMin-answer[junmo].min)*(avgMin-answer[junmo].min);
 }

 avgVar = sumVar/round;

 outFile << "Number of Nodes\t" << m << endl;
 outFile << "round " << round << endl;
 outFile << "DS Size \t" << avgSize << endl;
 outFile << "Average \t" <<  avgEnergy << endl;
 outFile << "Min \t" << avgMin << endl;
 outFile << "Var \t" << avgVar << endl << endl;

 return(EXIT_SUCCESS);
}
else{
 round++;
 outFile << "round " << round << endl;
}
```

```
/*
 if ( 0 == connect() ) {
  //cout << "The DS is not connected" << endl;
        //return(EXIT_FAILURE);
  goWhile =1;

 }

*/
 // Calculate life span
/*
 for (i= 0; i < m; i++) {
  if (station[i].status == inDS)  {
   station[i].energy = station[i].energy - 2;
   if (station[i].energy <= 0) {
    goExit = 1;
    break;
   }
  } else {
   station[i].energy = station[i].energy - 0.1;
   if (station[i].energy <= 0) {
    goExit = 1;
    break;
   }
  }
    }*/


 totalDSCount = totalDSCount + dsCount;
 totalEnergy = totalEnergy + avg;

 if(goExit)
 {
  //cout << "Number of Nodes\t" << m << endl;
  //outFile << "Number of Nodes\t" << m << endl;
  //outFile << "Total Number of Round \t" << numOfRound << endl;
  //outFile << "Total Average DS Size \t" << totalDSCount/numOfRound << endl;
  //outFile << "Total Average Energy of DS \t" << totalEnergy/numOfRound << endl<<endl;
  //outFile.close();
  return(EXIT_SUCCESS);
 }

} //for cds
} //for topology


 if(goWhile){
  outFile << " skip this round " << endl;
  goWhile = 0;
 }

 outFile.close();
 numOfRound++;

 } // End of While

 // whether it is DS
 return(EXIT_SUCCESS);
}
```

APPENDIX B

SOURCE CODE OF THE TECDS IN THE MOBILE NETWORK

```
// TDMS2.cpp : Defines the entry point for the console application.
// This program for the mobile network

#include <fstream>  // For input and output file
#include <math.h>
#include <stdlib.h>
#include <algorithm>
#include <iostream>
#include <set>
#include <queue>
#define MAXNODE 2000    // maximum number of nodes
#define DEBUG 1
#define MAXCOUNT 30
#define MAXINIT 3000
#define tmax 100
using namespace std; //add this line by Bonam & Junmo
enum StationStatus {uncovered, covered, bridge, gateway, inDS};
typedef struct node {
        int index;      // index of the node
        double x, y;    // x,y coordinate
  int degree;
  int status;
        int inds;       // 1: node in DS; 0: node not in DS
        int covered;    // 1: covered by DS; 0: not covered by DS
        int visited;    // used for checking connected graph
        int on;         // the mobile station on/off flag
        int valid;      // node used/unused for connectivity check
        set<int> nbr;   // neighbor set
    int initiator;  // initiator
        int initCount;
    int DSTimer; // will enter DS when DSTimer is 0
    int ODSTimer; // DSTimer value when first set
    double energy;  // Energy

} Node;

Node station[MAXNODE]; // maximum number of stations
int m;  // total number of stations
int h;  // number of units X/Y
int n;  // total number of stations switched off

// Calculate the node distance
double distance(double x1, double y1, double x2, double y2) {
     return sqrt((x1 - x2)*(x1 - x2) + (y1 - y2)*(y1 - y2));
}

// connectivity check
int connect()
{
        int curr; // first node of the queue
        // mark all nodes not visited
        for (int i = 0; i < m; i++) {
                station[i].visited = 0;
        }
        // find the first valid node
        for (i = 0; i < m; i++) {
                if (station[i].valid == 1) {
                        curr = i;
                        break;
                }
        }
        if ((curr < 0) || (curr >= m)) {
                cout << "No valid node!" << endl;
                exit(-1);
        }
        queue<int> q;
        q.push(curr);
        station[curr].visited = 1;
        while ( !q.empty() ) {
                curr = q.front();
                q.pop();
                for (set<int>::const_iterator siter=(station[curr].nbr).begin();
                siter!=(station[curr].nbr).end(); ++siter) {
                        if ((station[*siter].visited == 0) &&
                        (station[*siter].valid == 1)) {
                                station[*siter].visited = 1;
                                q.push(*siter);
                        }
                }
        }
        for (i = 0; i < m; i++) {
                if ((station[i].visited == 0) && (station[i].valid == 1)) {
                        return 0;
                }
        }
        return 1;
}
```

```
void checkDS(int n){
        int dscount = 0;
        set<int> union_set;
        insert_iterator<set<int> > union_ins(union_set, union_set.begin());
        union_set.clear();
        for (int i = 0; i < n; i++) {
                if (station[i].status == inDS)  {
                        dscount++;
                        set_union(union_set.begin(), union_set.end(),
                                (station[i].nbr).begin(), (station[i].nbr).end(),
                                union_ins);
                }
        }
        if ( union_set.size() == n ){
                //cout << "It is a DS " << endl;
        } else {
                //cout << "It is Not a DS" << endl;
        }
}
extern int announce(int i, int j);

int announce(int i, int j){ //anounce station i is the initiator, j is the annouNER
 for (set<int>::const_iterator siter=(station[j].nbr).begin();
   siter!=(station[j].nbr).end(); ++siter) {
            if ((station[*siter].on > 0) && (station[*siter].initiator == MAXINIT)
&& (station[*siter].initiator>= i) ) {
    station[*siter].initiator = i;
    station[*siter].initCount = MAXCOUNT;
                }
   if ((station[*siter].on > 0) && (station[*siter].initiator != MAXINIT)
&& (station[*siter].initiator >= i)){
    station[*siter].initiator = i;
    station[*siter].initCount = MAXCOUNT;
   }
 }
 return 1;
}
void broadDS(int i){ // i is the DS node
 for (set<int>::const_iterator siter=(station[i].nbr).begin();
                siter!=(station[i].nbr).end(); ++siter) {
                        if ((station[*siter].on > 0) && (station[*siter].status == uncovered)){
   station[*siter].status = covered;
   }
 }
}
int getUncovered(int i){ // get number of uncovered nodes of i
int num = 0;
 for (set<int>::const_iterator siter=(station[i].nbr).begin();
                siter!=(station[i].nbr).end(); ++siter) {
                        if ((station[*siter].on > 0) && (station[*siter].status == uncovered)){
   num++;
  }
 }
 return num;
}

int checkConnectedDS(int num){ // return 1 if the DS is conencted
 int result = 1;
 int found = 0;
   for(int i=0; i < num; i++){
  if (station[i].status == inDS){
   found = 0;
   for (set<int>::const_iterator siter=(station[i].nbr).begin();
                 siter!=(station[i].nbr).end(); ++siter) {
                 if ((station[*siter].on > 0) && (station[*siter].status == inDS) ) {
    found = 1;
    break;
    }
   } // end for each neighbor
   if (found == 0) return 0;
  }
 }
}

// normal distribution to generate energy to each node
// added this line Mar.30
double normal(double avg, double std)
{
   static int parity = 0;
   static double nextresult;
   double sam1, sam2, rad;
   if (std == 0) return avg;
   if (parity == 0) {
     sam1 = 2*((double)(rand())/RAND_MAX) - 1;
     sam2 = 2*((double)(rand())/RAND_MAX) - 1;
     while ((rad = sam1*sam1 + sam2*sam2) >= 1) {
       sam1 = 2*((double)(rand())/RAND_MAX) - 1;
       sam2 = 2*((double)(rand())/RAND_MAX) - 1;
     }
     rad = sqrt((-2*log(rad))/rad);
     nextresult = sam2 * rad;
     parity = 1;
     return (sam1 * rad * std + avg);
   } else {
     parity = 0;
     return (nextresult * std + avg);
   }
}
```

```
// select the initiator by energy
// added this line Mar.30
void SelectInitiatorByEnergy() {

 int i;
 int initiator;
 double largestEnergy;
 int largestnbrNum;
 int firstTime = 0;


  largestEnergy = station[0].energy;

  for(i = 1; i < m; i++) {
  if ( station[i].energy > largestEnergy) {
   largestEnergy = station[i].energy;
   firstTime = i;
  }
     }

//  cout << "Largest nbr " << largestnbrNum << endl;
  //cout << "  energy = " << largestEnergy << endl;

  // add to calaulate largest energy
  largestnbrNum = (station[firstTime].nbr).size();
  initiator = firstTime;

     for(i = firstTime; i < m; i++) {
   if ( station[i].energy == largestEnergy && (station[i].nbr).size() > largestnbrNum) {
   largestnbrNum = (station[i].nbr).size();
   initiator = i;
  }
     }
  //cout << "  size = " << largestnbrNum << endl;

  for(i = 0; i < m; i++) {
  station[i].initiator = initiator;
    }

}

int main(int argc, char *argv[])
{

 int i; //add this line by Bonam & Junmo
 int numOfRound = 1;
 int Round =1;
 int goExit = 0;
 int goWhile = 0;
 double totalDSCount = 0;
 double totalEnergy = 0.0;
 int t=0;

 if (argc != 4) {
  //cout << "usage: peter stations hops, where" << endl;
  //cout << "\tstations: total number of stations" << endl;
  //cout << "\thops: number of units X/Y, total hops * hops units" << endl;
          return(EXIT_FAILURE);
 }


      m = atoi(argv[1]); // total number of stations
      h = atoi(argv[2]); // number of units X/Y



  // For output File
  char resultFile[50];
  strcpy(resultFile,argv[3]);

  //assign enenrgy level to each node
  for(i = 0; i < m; i++) {
   //station[i].energy = normal(7.0, 2.0);
   station[i].energy=100.00;
   //cout << "station[" << i << "] Energy Level=" << station[i].energy << endl;
  }


  while(1) //add wile loop Apr. 15
  {
   int temp=0;
   ofstream outFile;
   outFile.open(resultFile, ios::out | ios::app);
   srand(numOfRound);
   //srand(4);


   for(i= 0; i < m; i++) {
    (station[i].nbr).clear();
   }
```

```
// randomly generates the positions of all nodes
 // added this line Mar.30
      for(i = 0; i < m; i++) {
  station[i].index = m;
         station[i].x = (double)(h*rand())/RAND_MAX;
           station[i].y = (double)(h*rand())/RAND_MAX;
  }



 // sets the neighbors of all nodes
 for(i = 0; i < m; i++) {
  for(int j = i + 1; j < m; j++)
   if (distance(station[i].x, station[i].y,
        station[j].x, station[j].y) <= 1) {
    (station[i].nbr).insert(j);
    (station[j].nbr).insert(i);
   }
  //cout << "i = " << i << "  Neighbours:";
                          //copy((station[i].nbr).begin(),(station[i].nbr).end(),
                                 //ostream_iterator<int>(cout," "));
                          //cout << endl;

  //cout << "i = " << i << "  size = " << (station[i].nbr).size();

                          //cout << endl;
 }


 // set the valid flag, degree etc
 for (i = 0; i < m; i++) {
  station[i].valid = 1;
  station[i].on = 1;
  station[i].degree = (station[i].nbr).size();
  station[i].inds = 0; station[i].covered = 0;
  station[i].initiator = MAXINIT;
  station[i].initCount = MAXCOUNT;
  station[i].status = uncovered;
  station[i].DSTimer = -1;
  station[i].ODSTimer = -1;

 }
///////
 if ( 0 == connect() ) {
  cout << "The graph is not connected" << endl;
  //return(EXIT_FAILURE);
  goWhile = 1 ;
 }
 else{


 //initiator election and announce to the whole topology (by calling anounce(i,i) function)
 for (int j=0; j < 20; j++){ // j: round number
  if (j== 100){
   station[0].on = 0;
   station[0].initiator = MAXINIT;
  }
  for (int i = 0; i < m; i++){
   if (station[i].initiator == MAXINIT  && station[i].on ){
    station[i].initiator = i;
    // cout << "announce itself i=" << i << "init=" << station[i].initiator << endl;
    announce(i,i);
   }
   if (station[i].initiator < i  && station[i].on ){
    announce(station[i].initiator,i);
    // cout << "forward announce i=" << i << "init=" << station[i].initiator << endl;
   }
   station[i].initCount--;
   //cout<< "i="<<i <<"round=" << j << "initCount=" <<  station[i].initCount << endl;
   if (station[i].initCount == 0 && station[i].on > 0){
    station[i].initiator = i;
    announce(i,i);
   }
  }
 }

 // call function for selecting initiator
 // added this line Mar.30
 SelectInitiatorByEnergy();
 for (i = 0; i < m; i++){
   //cout << "station[" << i << "].initiator =" << station[i].initiator
<< "initCount=" << station[i].initCount << endl;
   //cout << "station[" << i << "].initiator =" << station[i].initiator
<< "Energy Level=" << station[i].energy << endl;
 }


 int unum = 0; // number of uncovered nbrs for each node
 int unum1 = 0;
```

```
for (j = 0; j < 2000; j++){
  for (int i = 0; i < m; i++){
   if ( station[i].on > 0 && i == station[i].initiator){
    station[i].status = inDS; // the node move to the dominating set
    broadDS(i);
   }
   if ( station[i].on > 0 && station[i].status == covered ){
    unum = getUncovered(i);
    if (unum > 0){
     unum1 =  tmax /(getUncovered(i)*station[i].energy);

     if (station[i].ODSTimer == -1){
      station[i].DSTimer = unum1;
      station[i].ODSTimer = unum1;
      // cout << "station["<<i<<"].DSTimer=" << station[i].DSTimer << endl;
     } else {
      if (station[i].ODSTimer < unum1){
      station[i].DSTimer = unum1;
                                                  station[i].ODSTimer = unum1;
      }
     }
    } else {
      station[i].DSTimer = -1;
    }
   }
   if (station[i].DSTimer > 0) station[i].DSTimer--;
   if ( station[i].on > 0 && station[i].DSTimer == 0 && station[i].status != inDS){
    station[i].status = inDS;
    broadDS(i);
    // cout << "station " << i << "enter DS at round " << j << endl;
   }
  }
 }

 checkDS(m);
 for (i =0; i < m; i++){
  if (station[i].status == inDS) {
   station[i].valid = 1;
  } else {
   station[i].valid = 0;
  }
 }
 int con = connect();
 if (con > 0) {
  t++;
  cout << "This DS is connected " << t << endl;
 } else{
  cout <<  "This DS is NOT connected" << endl;
 }

 if ( 0 == connect() ) {
  cout << "The graph is not connected" << endl;
  //return(EXIT_FAILURE);
  goWhile=1;
 }
 else{

 int dsCount = 0;
 for (i = 0; i < m; i++){
  if (station[i].status == inDS){
   dsCount++;
  }
 }


 dsCount = 0;
   //ofstream outFile;
 //outFile.open(resultFile, ios::out | ios::app);
 double max=0.0, min=10000.0, avg=0.0, sum=0.0, mint=10000.0, td, tn;

 for (i = 0; i < m; i++){
  if (station[i].status == inDS){
   dsCount++;
   //outFile << i << "\t" << station[i].energy << endl;
   td=station[i].energy / 2;

   sum=sum+station[i].energy;
   if(max < station[i].energy) {
    max = station[i].energy;
   }
   if(min > station[i].energy) {
    min =station[i].energy;
   }

   //td=station[i].energy / 2;

   if ( mint > td)
   mint=td;
  }
  else {
   tn=station[i].energy / 1;
   if ( mint > tn)
   mint=tn;
  }
 }
```

```
avg=sum/dsCount;

double varSum=0.0, var=0.0;
for (i = 0; i < m; i++){
 if (station[i].status == inDS){
  varSum=varSum+pow((station[i].energy-avg),2);


 }
}

var=varSum/dsCount;


/*
outFile << "Number Of Round" << numOfRound << endl;
outFile << "DS Size \t" << dsCount << endl;
outFile << "Average \t" << avg << endl;
outFile << "Max \t" << max << endl;
outFile << "Min \t" << min << endl;
outFile << "Var \t" << var << endl << endl;*/


//outFile << "min t=" << mint << endl;
//outFile.close();




/*
 if ( 0 == connect() ) {
  //cout << "The DS is not connected" << endl;
        //return(EXIT_FAILURE);
  goWhile =1;

 }*/


 // Calculate life span for mobile network
 if( 0 != connect())
 {
 for (i= 0; i < m; i++) {
  if (station[i].status == inDS)  {
   station[i].energy = station[i].energy - 2;
   if (station[i].energy <= 0) {
    goExit = 1;
    break;
   }
  } else {
   station[i].energy = station[i].energy - 0.1;
   if (station[i].energy <= 0) {
    goExit = 1;
    break;
   }
  }
    }


 totalDSCount = totalDSCount + dsCount;
 totalEnergy = totalEnergy + avg;

 if(goExit)
 {
  //cout << "Number of Nodes\t" << m << endl;
  outFile << "Number of Nodes\t" << m << endl;

  outFile << "Total Number of Round \t" << Round << endl;
  outFile << "Total Average DS Size \t" << totalDSCount/Round << endl;
  outFile << "Total Average Energy of DS \t" << totalEnergy/Round << endl<<endl;
  outFile.close();
  return(EXIT_SUCCESS);
 }
 }
 else {
  goWhile =1;
 }

}//for cds
}//for topology

 if(goWhile){
  outFile << " skip this round " << endl;
  Round--;
  goWhile = 0;
 }
 outFile.close();
 Round++;
 numOfRound++;

 } // End of While
 return(EXIT_SUCCESS);
}
```