UNIVERSITY OF CALIFORNIA

Los Angeles

Energy Efficient Routing For Wireless Sensor Networks

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy in Electrical Engineering

by

Jay Lin Gao

2000

The dissertation of Jay Lin Gao is approved.

Mario Gerla

Mani Srivastava

Kung Yao

Gregory J. Pottie, Committee Chair

University of California, Los Angeles

2000

DEDICATION

To all from whom I have learned.

Contents

List of Figures

List of Tables

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor, Professor Greg Pottie, for his tutelage and trust. I would also like to thank Professor Mario Gerla, Professor Mani Srivastava, and Professor Kung Yao for their valueable input before and during the writing of this dissertation.

My appreciation also goes to John Siwko, Bruce Kwan, Chris Hansen, Keith Scott, Kathy Sohrabi, Vishal Ailawadhi, Dennis Connors, Tommy Yu, Tai-Lai Tung and many others too numerous to mention here, for sharing their research experiences and insights with me. It is a privilege to have them as colleagues and a blessing to know them as friends.

I am forever grateful to Maitra and Nan-Ping. Their constant encouragement has given me the clarity of purpose and the inspiration I need to complete this work.

Above all, I would like to express my deepest gratitude to my parents for bringing me to this world and for giving me the chance to learn and grow.

VITA

October 7, 1970	Born, Taipei, Taiwan, Republic of China
1993	B.S. Electrical Engineering magna cum laude University of California, Los Angeles Los Angeles, CA
1994	M.S. Electrical Engineering University of California, Los Angeles Los Angeles, CA

PUBLICATIONS

J. Gao, K. Sohrabi, V. Ailawadhi and G. Pottie, "A Self-Organizing Wireless Sensor Network," *Proc. 39th Annual Allerton Conference on Communication, Control, and Computing*, Urbana, Illinois, October 1999.

J. Gao, K. Sohrabi, V. Ailawadhi and G. Pottie, "An Energy Conscious Self-Organizing Wireless Sensor Network," *IEEE Personal Communications Magazine, October 2000.*

ABSTRACT OF THE DISSERTATION

Energy Efficient Routing For Wireless Sensor Networks

by

Jay Lin Gao

Doctor of Philosophy in Electrical Engineering
University of California, Los Angeles, 2000
Professor Gregory J. Pottie, Chair

As hardware technology pushes sensor performance near its fundamental limit, further improvement in sensor technology relies on the application of advanced signal processing techniques. The use of sensor arrays is one such example. As a further extension of the concept of sensor array, a sensor network seems to have greater potential for performance improvement due to its versatility to be deployed in a wide range of applications and its ability to monitor a large physical area. To tap into the full potential of sensor networks, we have to deal with the communication problem. The best long term solution will be based on an ad-hoc network architecture because of its ability to operate without infrastructural support. However, the stringent constraint on energy resources has become the most significant challenge in sensor network design.

In this dissertation, we anticipate a set of application level tasks and design energy efficient algorithms to fulfill their communication needs. One of the application level tasks is the multi-hop communication between a large number of sensors and an information gathering entity called the USER. We envision that the bulk of the long-range multihop traffic consists of short packets carrying target detection reports or network status queries and replies. Based on this assumption, we present a table-driven, multi-path network structure and a routing algorithm, based on the concept of a sequential stochastic assignment, that has demonstrated, through simulation experiments, significant performance improvement over the classic minimum metric routing algorithm.

Another major application level task in sensor network operation is the formation of an adaptive local network to perform various cooperative signal processing functions on detected targets, which is essential for target identification and tracking purposes. This task involved the selection of a central processing node and the construction of routes from each sensor to the central node. The size, location, and topology of the local network will depend on the signal strength, mobility and location of the target. Such networks typically have a short operational life time, just long enough for the necessary data to be gathered and uploaded to the central node. In this dissertation, we present an energy efficient and highly scalable election algorithm, called Single-Winner Election(SWE), to select the central node and build energy efficient paths from each sensor to the central node.

Chapter 1

Introduction

Sensor networks are basically monitoring networks with a wide range of applications. They can be as simple as the transducer circuits that are wired and installed in an automobile to gauge fuel level, engine temperature, and vehicle speed and display the results to the driver in real time, or they can be wireless networks that cover a vast geographical area, such as the ADRAMS and AARI-GPS buoy networks deployed by the International Arctic Buoy Program (IABP). These buoys collect data on barometric pressure, temperature, and ice motion in the Arctic Basin and relay the data through satellite. They can be deployed by small rockets or air dropped on location. All vital electronics, including batteries, data recording device, diagnostic circuitry, and antenna are housed inside a polycarbonate sphere to protect it from the harsh Arctic environment. These are examples of pure monitoring networks that are designed to measure well defined environmental parameters from clear and present targets.

In general, the targets of interest are not necessarily present and available, and the parameter of interest, such as target location, movement, and categories

are only indirectly linked to measurable environmental parameters. For example, in military applications, a sensor network can be used to detect the movement of vehicles or foot soldiers. Such events can only be *recognized* indirectly through the presence of a complex sequence of measurable acoustic or seismic vibrations that are generated by the event. In civilian applications such as search and rescue and law enforcement, the targets of interest can vary from simple human movement to the presence of weapon and explosives. In these cases, the linkage between measurable physical phenomena and their origin can only be discovered through the appropriate use of signal processing and hypothesis testing. This tells us that in order to extend the usefulness of sensor networks, it is no longer sufficient to just gather data, which is what the pure monitoring networks do, but it is vital to extract information meaningful to the application and present them in understandable form. As the nature of possible targets become more diversified, it is increasingly necessary to develop a new class of intelligent sensors that can synthesize sensing and processing capabilities in a single package. Recent progress in very-large-scale integration (VLSI) technology has spawned an entire generation of such sensors. One example is the low power integrated microsensor (LWIM) developed at UCLA. It features light-weight and easily deployable nodes with programmable microprocessor, low power radio communication, and highly sensitive sensor circuitry. In order to realize the full potential of this class of sensors, new software technology and communication protocol sensitive to the unique constraints and requirement of microsensor must also be developed.

1.1 Overview of Sensor Networks

In most applications, sensor networks have three functional components: (1) detection and data collection, (2) signal processing, and (3) notification. There are two basic approaches by which these three functions are implemented: (1) **single-sensor** approach and (2) **multi-sensor** approach.

The **single-sensor** approach centralizes all three functions on a single node. These nodes are equipped with ultra-sensitive sensors(e.g. arrays), powerful signal processing units, long range radio communication capability, and high capacity power supply. This approach is not robust because failure of a single component renders the entire node useless. Also, the large physical size of these nodes often makes them difficult to deploy and vulnerable to sabotage. Further, the propagation losses due to distance and physical obstructions can often render distant targets undetectable regardless of the sensor sensitivity.

The **multi-sensor** approach distributes sensing functions to a large number of small yet capable nodes, which significantly lowers the hardware requirement and manufacturing cost for each node. Failures caused by hardware or other environmental factors will most likely disable only a subset of the nodes, allowing most network operations to continue. Although each node has lower sensing and processing power than a single super node, networking functions allow them to operate as a single unit while covering a large geographical area. Environmental events can be detected at close proximity due to better sensor coverage, resulting in higher average signal-to-noise ratio. Applying signal processing techniques, each sensor can extract a small set of essential features from the raw data to determine the nature of the target, and when necessary, report its finding to a

user node through multi-hop routing. When a target is observed by multiple sensors, cooperative data fusion or beamforming techniques can be used to improve probability of detection, false alarm, and target identification accuracy [22]. To facilitate the operations for such sensor networks, routing protocols must be developed to handle the communication needs.

Sensors networks are most valued for their small size, signal processing capabilities, and versatility to form wireless ad hoc networks. Their ability to operate without pre-existing infrastructure makes them useful in virtually any environment. However, this independence from the infrastructure also imposes a new set of constraints. One such constraint is shorter radio range, which can lower network connectivity. Due to low antenna height, ground level features have a strong effect on propagation loss. Small rocks, plants, or even mild undulation in the terrain can create significant variations in the radio channel characteristics. Furthermore, radio signal strength tends to drop off with higher exponent at a smaller distance than those with higher antennae. Channel measurements [21] show that for an outdoor environment, near ground propagation loss exponent can vary from 3.5 to 5.0.

Another factor that affects network connectivity is node density. When density is low, the network can be severely fragmented due to a low average nodal degree - the average number of neighbors within radio transmission range. A simulation study shows that the average size of the maximal connected component - the largest subset of connected nodes in a network - cannot reach 90% until the average nodal degree goes above 7.(See Table 1.1) To compensate for low node density, radio power needs to be raised to extend transmission range.

The most significant constraint on network operation is energy limitation.

Network Size = 100							
Average Nodal Degree	0.79	1.54	2.54	3.80	5.31	7.07	9.08
Average Size of Maximal Connected Component	2.17	4.09	8.37	25.38	66.6	91.84	98.2
Standard Deviation of Connected Component	1.55	3.81	7.34	20.27	31.2	18.19	9.19

Table 1.1: Network Connectedness

Lacking infrastructural support, each node depends on small and low capacity batteries as the energy source, and cannot expect replacement when operating in hostile or remote regions. As nodes deplete their batteries, there will be a gradual reduction of network connectivity and sensor coverage, which degrades sensing and signal processing performance and introduces some dynamics in network topology. Most protocols, whether on the signal processing, networking, MAC, or physical levels, do not consider their impact on external situation such as network connectivity because they are designed for infrastructured networks in which any component failures are restored quickly, either by outside agents or sophisticated automatic recovery schemes. For such networks, loss of connectivity is a rare event that has little statistical significance to overall performance, therefore is not a factor in protocol design and testing. For mobile networks, many protocols attribute the loss of connectivity to the mobility of nodes, not to the energy depletion caused by the execution of the algorithm. Therefore the task of handling mobility assumes far more importance than energy conservation.

Because of their very limited energy resources and lack of maintenance after deployment, ad hoc sensor networks are required to operate under degraded capacity for a significant time duration such that the overall system performance becomes highly dependent on the energy efficiency of the algorithm.

1.2 Overview of Dissertation Topics

The role of any sensor network protocol is to support the detection notifications and cooperative signal processing functions under a stringent energy constraint. In a broader sense, a sensor network is formed from the integration of sensing, signal processing, and communication functions.(See Figure 1.2) It is the perfect platform on top of which hierarchical information processing [30] can take place. It allows information to be processed on different levels of abstraction, ranging from the detailed, microscopic examination of specific targets, to the macroscopic view on the aggregate behavior of targets. Any events in the environment can be processed on three levels: node level, local cluster level, and global level.

Figure 1.1 depicts a sensor network operating on these three levels in response to environmental stimuli. On the node level, data collection and processing occurs on each individual node, and no communications are required except for the notification of the final result to the USER. On the cluster and global level, routing functions are required for the gathering of raw or pre-processed data from individual or clusters of nodes to a central location for advanced processing such as data fusion or beamforming. Having a larger set of data to work with, detailed information on individual target as well as general, aggregate patterns in target dynamics are obtainable.

In this dissertation, we propose two routing algorithms that fulfill these communication needs: (1) an energy efficient multi-hop routing algorithm for long distance notification, and (2) an adaptive local routing algorithm for cooperative signal processing on the cluster level which can be extended to the global level as well.

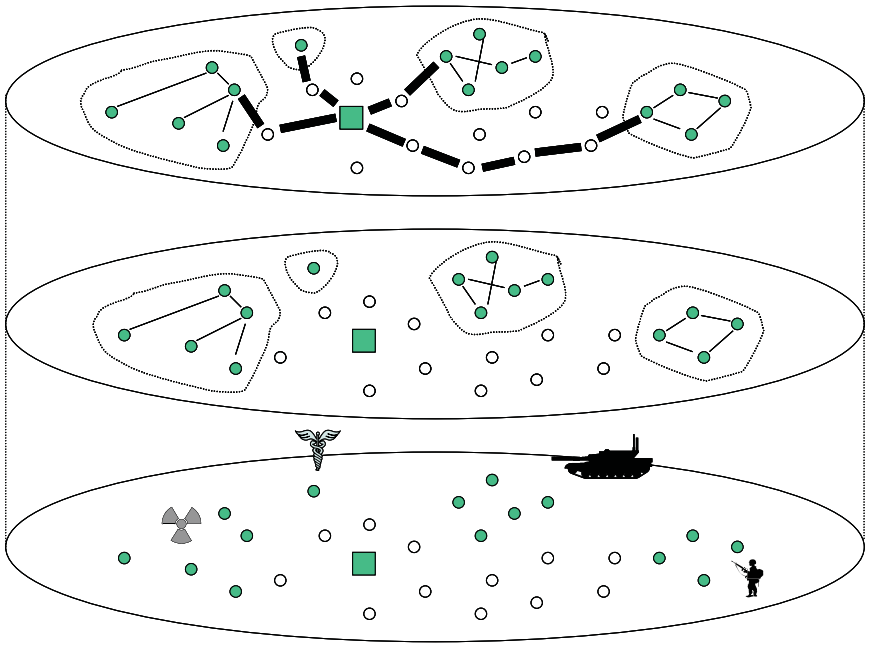

Figure 1.1: Hierarchical Information Processing

Chapter 2 presents the concept of ad hoc network and a simple survey on some of the major categories of routing algorithms, with comments on their individual strengths and weaknesses. We also briefly describe the concept of network scalability.

Chapter 3 presents an energy efficient multi-hop routing algorithm for sensor network applications. We discuss the issues and challenges involved in protocol design, such as assymetrical traffic loading, energy management, robustness, and the need to distinguish packets that require different quality of service (QoS) due to the content of its payload. We consider what network structure can best satisfy this set of requirements and constraints and propose a constrained overlapping tree structure for its simplicity, robustness, and low overhead. This tree structure can be constructed by a modified spanning tree algorithm for which we give a

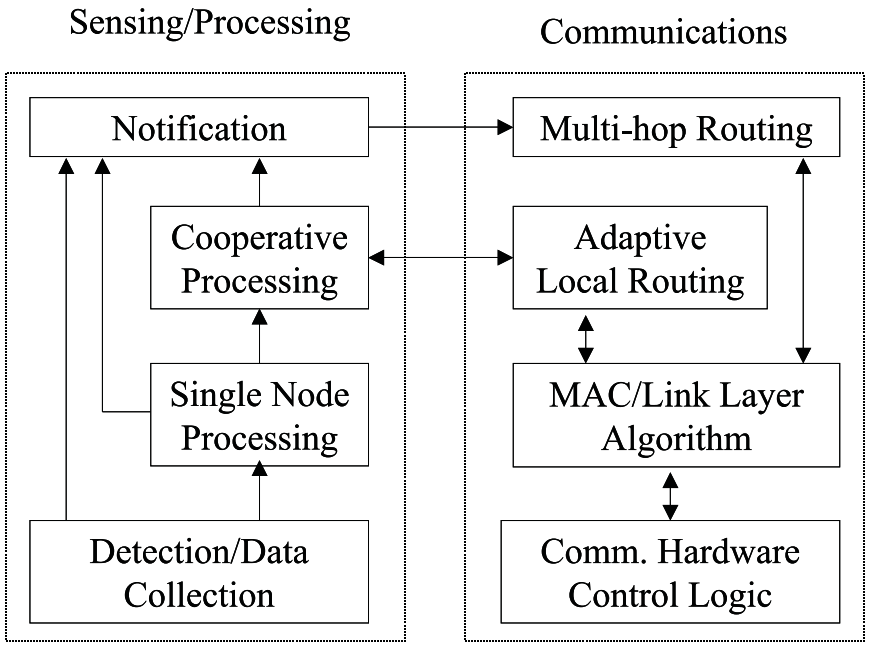

Figure 1.2: Integration of Sensing and Communication Functions

detailed description of its operation and proof of convergence and path restoration capabilities.

The proposed overlapping tree structure will provides a multi-path environment that enables priority routing and load sharing. To select the best path, each node has to consider the priority level of the packet and the QoS and capacity on each path. For this task we propose a sequential assignment routing (SAR) algorithm, and present its mathematical background and modification required for implementation. Simulation results provide performance and sensitivity comparison between SAR and the classic minimum metric algorithm.

Chapter 4 begins by the categorization of targets into two broad categories, (1) near-field (NF) and (2) far-field (FF), and is followed by comments on the

communication requirements for coherent and non-coherent cooperative information processing. We then present the second major network function: an adaptive local network formation/routing algorithms that facilitates non-coherent cooperative signal processing. A Single Winner Election (SWE) algorithm is used to select a central node (CN) distributively that maximizes a given election criterion among a small group of nodes that cooperate in information processing. This election criterion can be either SNR, node location, or any other characteristics associated with the node, as required by the particular cooperative function.

This election process is similar to a multi-source diffusing computation process [15]. In this distributed process, multiple local elections can start simultaneously, and by exchanging local election results, the global solution is found with lower overhead. By piggybacking routing information onto the signaling traffic of the election process, a minimum-hop spanning tree rooted at the CN is created simultaneously. We demonstrate the energy efficiency of SWE by a simple analysis under an information theoretic context. We show that further energy saving can be obtained by imposing voluntary delay on local election, thus demonstrating the energy-delay trade-off inherent to this type of distributed process. At the end of the election, a termination procedure will inform the winning CN about the completion of all election and routing computations, so the actual cooperative function can be initiated by the CN. This procedure can be implemented as a simple state machine on each node, and we provide a proof for its correctness. Finally, simulation results are presented to demonstrate the delay-overhead characteristic and scalability.

For coherent cooperative functions, traffic loading will increase substantially due to the necessity to transmit a large amount of raw data across the radio

channel. Therefore the focus of protocol design shifts from algorithmic efficiency to path optimization. In chapter 5, we describe a modified network building procedure. It follows the basic structure of the non-coherent case except that a membership trimming procedure is required to limit traffic loading. This trimming procedure can be implemented by a Multi-Winner Election (MWE) algorithm, which selects a limited number of source nodes (SN) that will eventually provide the raw data for cooperative processing. Then a modified SWE will select the optimal node among the original members as CN so that the total energy consumption required to gather raw data from each SN to this CN is minimized. Issues such as time delay and complexity will be discussed. Simulation results demonstrate the delay-overhead characteristics and lower scalability than the non-coherent case.

In chapter 6 we include some final remarks on our research findings and possible directions for future research.

Chapter 2

Background

Before we describe our sensor network protocols in detail, a few remarks on the possible underlying link level technologies that support network operations are warranted. In section 2.1, we briefly describe the concepts of wireless Ad-Hoc Networks and link level channel access control algorithms. Section 2.3 presents a simple survey on several major categories of routing techniques, and section 2.4 discusses the concept of *scalability*, which is an increasingly important factor in ad-hoc network design and performance evaluation.

2.1 Infrastructured & Ad-Hoc Networks

In an infrastructured network, there is a clear distinction between the *users*, which are entities that are source and/or the destination of information exchange, and the *infrastructure*, which consists of a set of nodes and links that store and relay information for the *users*. We note that in its purest form, an infrastructured network does not allow a **direct** connection between two users, so information

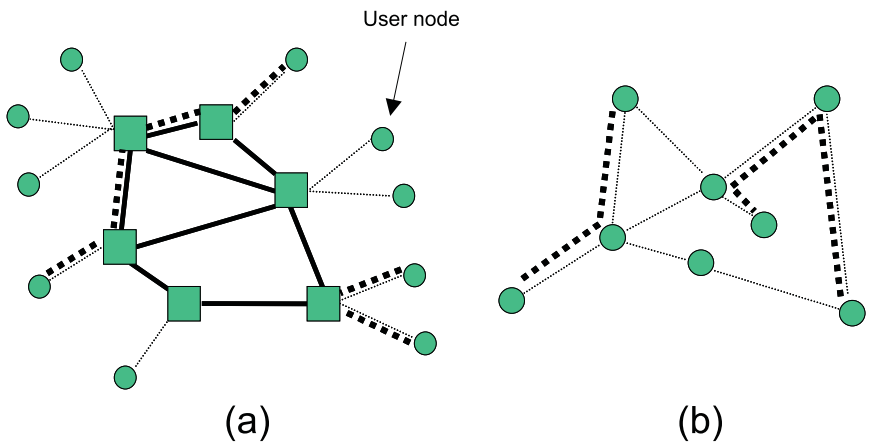

Figure 2.1: Infrastructured and Ad-Hoc Networks

exchange must always go through the network infrastructure. (See Figure 2.1(a)) The advantage of this approach is that the operational complexity involved in the actual information transfer is transparent to the users. A prime example of such a network is the telephone network, where the users can communicate by a simple analog device (telephone) that is plugged into a pre-installed outlet wired to the telephone company facility. The user and his simple device have no involvement in the sampling and digitization of the voice waveform, the calculation of routes, and the setup of the circuit that actually enables the conversation to take place. Although the cellular system required more sophisticated devices, they are only responsible for establishing a link level communication with the nearest base station beyond which the operations of the network remain transparent.

Although having an infrastructured network makes communication easy for the users, sometimes, due to physical limitations and user mobility, it is better to shift some of the networking tasks to the users themselves. An example of this is the ALOHA network developed in the early 1970's to provide communications

for the University of Hawaii's various computing facilities located on different islands. For ease of operation, a satellite repeater is used to create a single broadcast channel shared by all users. Although the satellite can be considered in principle as part of the "infrastructure", it does not regulate or control the way the users communicate with each other. The users will share the channel by following a channel access procedure later known as the ALOHA algorithm. This algorithm gives the users the ability to compete for access to the channel while keeping their behaviors in check. The DARPA Packet Radio Network [13] is an example of a network where each user can be called upon to "serve" the communication needs of other users by relaying messages and reporting changes in network conditions. In both cases, the users have more direct involvement in the operation of the networks.

Moving further toward a user-initiated and user-maintained operation, at the other end of the spectrum, we have the *ad-hoc networks*, where the users are also their own service providers. Each user, upon joining the network, earns the right to request service from other users and the responsibility to shoulder the cost of network maintenance and operation. Free from the reliance on pre-existing communication infrastructure, such a mode of operation is ideal for a mobile ad-hoc network or MANET, where network topology and membership change constantly.

2.2 Network Topology & Channel Access Technologies

The foundation of a communication network is the individual link that connects the nodes. Each link represents the fact that communication between the nodes joined are supported through some transmission medium by a link level Medium Access Control (MAC) protocol. In wireless ad-hoc networks, the transmission medium is the radio channel, and the task of MAC is to control the way individual nodes can access the radio channel. Two general approaches can be taken:

1. *Scheduling*

2. *Random Access*

In *scheduling*, communication resources can be divided along three dimensions: **time, frequency, and code**. In **Time Division Multiple Access (TDMA)**, a channel can be divided into time slots that are assigned to different links. In **Frequency Division Multiple Access (FDMA)**, each link uses a different frequency band. In **Code Division Multiple Access (CDMA)**, radio signals are spread across a large bandwidth by a linear transformation using a code, and each signal can be recovered by applying the corresponding reverse mapping.

The highest level of access control will create a **contention-free** system where scheduling is held unchanged. This is also called *fixed assignment*. The advantage of this approach is that upon the creation of a schedule, network operation can become very energy efficient. On the other end of the spectrum are the **contention-based** or *random access* schemes. Contention-based MAC is designed based on "self-regulated aggression," where users compete for resources

but try, whenever possible, to avoid a deadlock with other users. In both slotted and unslotted ALOHA protocols, users will compete for resources by using it whenever the need arises. If no competition occurs, then transmission is successful; if competition results in collision of the signal, then each competing user will voluntarily impose a delay for a random length of time, hoping to separate the time of their respective future retransmissions. To further reduce the chance of collision, some preventive measures can be used. In **Carrier Sense Multiple Access (CSMA)** systems, each user is required to listen to the channel for traffic before transmitting. This reduces the possibility of collision.

These schemes are often chosen for their simplicity and the benefit of "statistical sharing" of resources. Since in a fixed assignment system, users cannot voluntarily acquire additional resources outside those already scheduled, in anticipation of periods of demand surge caused by normal statistical fluctuation, each user is usually allocated bandwidth much higher than its statistical average. The result is inefficient bandwidth utilization. On the other hand, the contention-based system breaks away from the rigid structure of the fixed-assignment schedule so that all users share a common pool of resources through a contention process. The result is that while individual service demand still fluctuates, the aggregate demand shows much less fluctuation above or below the aggregate norm, as predicted by the Strong Law of Large Numbers. Therefore more users can be accommodated than what the fixed-assignment schedule will allow. However this contention process itself generates secondary traffic that may consume considerable amount of bandwidth. Furthermore, since a successful transmission can occur in any slot, each user has to keep its transceiver turned on to listen to the channel, which may sum up to considerable energy consumption over time.

To achieve adaptability to demand fluctuation without losing operational efficiency, a *demand assigned scheduling* can be used. One such scheme is called *reservation*, in which a TDMA, FDMA, or CDMA system can be used as the foundation. However, access to time slots, frequency bands, or codes is granted only after a user successfully makes a reservation through a contention process. By separating the contention process from the data transmission, energy consumption is lowered because smaller packets reduce the collision profile significantly. However, since the contention mechanism is still based on "collision," performance improvement can be quite limited.

To bring about a fundamental change in the contention mechanism, a "collision-free" process is introduced. A *token passing* system is one where a token (represented by a special packet) is circulating among the users. When a user is in possession of the token, it has exclusive use of the transmission medium. When it passes the token to others, it will lose access. Instead of directly competing with each other, users compete for resources by optimizing how long they will hold on to the token once they come into possession of it, and to whom they will pass the token. The optimal token passing protocol should allow each user fair and equal access to the channel relative to individual and aggregate demand.

Another scheme, which works like a centralized version of the token passing scheme, is called *polling*. In this scheme, a central entity polls each user, which in effect gives the user the opportunity to use the channel. When the user is done, the central entity will be informed so that others can be polled. The polling procedure can vary according to the overall objective of the protocol. In general, we note that the contention process is much more structured, and bandwidth consumption is reduced significantly.

2.3 Routing Algorithms

In this section, we present several dissection of routing algorithms based on their adaptability and features in their network structures. We explore the motivation, methodology, and pros and cons for each approach.

Adaptability

"Adaptability" refers to the responsiveness of the algorithm to changes in network topology, congestion, and other conditions that can impact routing performance. Despite the fact that there is always the possibility of change, non-adaptive routing algorithms are nevertheless used in many cases. Although they can perform rather poorly, their simplicity is highly desirable. One example of such an algorithm is *flooding*. Communication from the source node to the destination node is accomplished by saturating the network with data, so that as long as the source-destination pair is connected, delivery is guaranteed. This protocol is very inefficient, but extremely simple and robust. When system resources are abundant, it can be a viable option if the scope of flooding is controlled [12] and used infrequently as a backup.

In *fixed directory* systems, a network with fairly static conditions can operate according to a "directory" or "table" that lists for each source-destination pair a pre-determined out-going link. This requires some pre-programming before system activation, but no further overhead will be incurred. It has much lower robustness but higher efficiency than *flooding*. If network topology can be maintained for a long period of time and channel conditions and user demand are

well-behaved, a fixed directory system can produce nearly optimal performance.

Adaptive algorithms are those capable of self-adjustment in response to change of network conditions. They have higher complexity but better performance under changing conditions. Example of such algorithms are found under the broad category of *Link State* protocols. Such protocols keep track of the condition of all links (and nodes) in the network by flooding the network with update packets when change occurs. Although capable of making optimal adjustment, the large overhead involved in keeping each node informed of all changes is its most significant drawback.

To alleviate this situation, a *Distance Vector* approach can be used. In many networks, changes in network connectivity or channel noise level affect only those nodes near it. The exact scope of adjustment required is related to the degree of coupling in the network, which is reflected by the interdependence of a "distance vector," a quantity that represents the QoS expected when routing packets from a node to a list of destinations. When the network condition changes, update packets will start propagating outward, away from the location where the change occurs. Nodes with strong coupling to the change will make proper adjustment to their own vector values and inform others about this change. Nodes with weak coupling will see very little change in their vector values, and therefore stop further propagation of update packets. As result, the scope of the update process is limited and overhead is reduced.

Single Path & Multipath

In general, a collection of nodes and links through which a single packet

reaches its destination can be called a path. A path, as a networking construct, is not strictly required for routing (consider *flooding*.) However, it is very beneficial to work with this construct because it provides the necessary directional orientation for efficient routing. When the construct of path is applied, it can be done implicitly or explicitly. With an implicit path construct, each node sees only a portion of each path and makes routing decisions based on local knowledge. If an explicit path construct is used, each node will have some information regarding the *overall* condition along each path, such as the pathwise delay and QoS estimate. Most advanced routing techniques utilize the path construct as an aid to routing, and two approaches can be distinguished.

The *single path* approach establishes *one* path for each source-destination pair, and the *multipath* approach will allow several paths to be used simultaneously. The classic *shortest path* algorithm is an example of a single path algorithm. It finds the path that minimizes an additive metric between a pair of nodes. Because of its simplicity, quick convergence and good performance under mild loading and slow changing conditions [27], it is often one of the top candidates considered by designers as a starting point for developing new protocols.

However, there are many reasons one would consider taking a multi-path approach. One of these is load balancing, which can help keep queuing delay low. In *bifurcation* routing [16], at each node streams of data packets can be split among several outgoing links to balance queue lengths or as a way of diverting "overflow" traffic when the local queue is full. Another benefit is robustness against packet loss. Having multiple paths, the source node can send multiple copies of a message through several paths. The highest degree of protection against path failure is provided by a *node-disjoint multipath* algorithm. Since disjoint paths share no

common nodes, except the source and destination, any single link or node failure can disable at most one path at a time. If k node-disjoint paths exist, then delivery will be successful even if $k-1$ failures occur during the transmission session. However, the overhead and time complexity of disjoint path calculation is high. Multipath algorithms often need to be used in conjunction with a packet resequencing procedure because individual packets may arrive out of order. For applications such as real time voice or video playback, a single path algorithm will be more suitable.

Table-driven & Demand-driven

In any routing protocols, the designers have two options on when paths should be generated. In a *table-driven* system, paths connecting all possible source-destination pairs are pre-computed and recorded in a *routing table*. However, it is not particularly suitable under rapidly changing conditions due to the fact frequent calculation and revision of the routing tables can raise overhead significantly. There is also the challenge of maintaining routing table consistency so that the appearance of invalid routes such as loops or dead ends are rare and temporary.

When a *single-path* algorithm comes under heavy traffic, oscillatory behavior can also occur as traffic is shuttled back and forth between groups of nodes each time the routing table is revised [27]. Such oscillation creates a bursty loading pattern and long delay.

A *demand-driven* protocol seeks to circumvent frequent routing table update by doing away with it completely. The term demand-driven means that *no path*

calculation takes place until the moment it is actually required. In other words, instead of tracking the network conditions continuously, the *demand-driven* approach simply takes the most current snapshot of the network at the instant when routing service is required and builds a set of paths to fulfill that service requirement. In [35] a prototype demand-driven routing algorithm is described for the wireless mobile environment. When a source wishes to send packets to another node, a **path discovery** procedure takes place. During this procedure, the intention to find a connecting path to the destination is announced by flooding a query, or **QRY**, packet, that will invariably reach the destination node if it is still connected to the network. Then the destination node will send a reply, or **RPY** packet, to the source node by back-tracing the paths taken by the **QRY** packets. When the source node receives the **RPY** packets, it will have information on several paths that lead to the destination node at its current position. Then the source node can use these paths to send information to the destination node.

Here we see that the **path discovery** operation can have fairly high overhead since the *flooding* procedure is used, and overall delay will increase because it precedes every transmission session. Attempts to alleviate overhead during the **path discovery** have been made by keeping the route table of previously computed paths so that re-use is possible. This is only suitable if network topology did not change sufficiently to make the path invalid. Another approach, called Location Aided Routing [36], makes use of the last known location of the destination node as a starting point for a more focused search. A controlled flooding of the query packets, limited inside a **request zone** that contains the last known location of the destination node, make the **path discovery** procedure more energy efficient. Despite the possibility of high overhead, a demand-driven protocol is often the

only choice we have under fast changing network conditions.

Hierarchical Routing

In *hierarchical* routing, different kinds of protocols will be applied to the network under a logical hierarchy. The network is often divided by task, hardware and software technology, and/or physical location into several smaller autonomous groups. Within each group there is the capacity for routing, self-organization if it is ad-hoc, and independent MAC structure. These groups are then joined together by another network that consists of specially selected nodes from each individual group. Thus there are two network layers: one handles the group-to-group communications; the other the communications within a group.

Examples of such layered architectures can be found in the class of *clustering* networks. In [10], several distributed algorithms are presented so that a random collection of mobile nodes can organize themselves into a network of overlapping clusters. Each cluster consists of a **cluster-head**, and all nodes within its 1-hop neighborhood are considered members of the cluster. Since each cluster has a diameter no larger than 2, the **cluster-head** is given the special task of monitoring, controlling and when necessary relaying all intra-cluster traffic. The communication between different clusters is handled by nodes called **gateways**. These are selected from nodes that are in communication range with at least one other cluster. One nice property about this approach is that it allows a natural division of the channel resources for adjacent clusters.

The specialization of nodes into **cluster-heads** and **gateways** naturally lead

to a division between *inter-cluster* routing and *intra-cluster* routing and the formation of a two-level hierarchy. When two nodes belong to the same cluster, intra-cluster routing is all that is required; otherwise, they have to connect through intra-cluster routing to a gateway or a cluster-head in their individual clusters, and these gateways and/or cluster-heads will then be connected through a separate inter-cluster protocol. The hierarchical structure lends itself easily to hybrid schemes, where different MAC and routing protocols are used on the intra and inter-cluster levels.

One advantage of the cluster topology is that it buffers the routing protocols from overreacting to mobility. Small scale movement within the same cluster, however frequent, will remain transparent to inter-clustering routing. A larger cluster size ($k > 1$ hops) can be used to further decrease sensitivity to mobility if the pattern of node movement is localized and predictable. The trade-off for this increased manageability of mobile nodes is the overhead involved in creating and maintaining the clusters.

2.4 Scalability

The idea behind *scalability* can be simply understood as the ability of a network to "scale" to larger size. A perfectly scalable network is one where the average operational complexity, resource consumption, and performance level remains constant *for each individual node* despite the network size. Therefore, to increase network size from ten to one hundred, simply add ninety more identical nodes to it. One nice property of a scalable system is that it can be designed and tested on small scale, because its large scale performance can be easily projected.

Furthermore, if a network protocol is scalable, then network size can increase without raising the burden on individual nodes.

The keys to building a scalable protocol are two-fold: (1) the distribution of tasks and (2) the reduction of coupling. It is clear that when operation is not distributed, then some central entity will have to execute certain tasks for each node in the network. Such networks cannot possibly scale easily because the size of the network will be limited by the computational capacity of the central entity. Scalability essentially means that increase of network size is transparent to the individual members of the network. This is only possible when the network is loosely coupled. A network protocol will invariably create some degree of coupling, which is generally the parametric dependence among different nodes. Thus when a change occurs or when a new node is added, a chain reaction of adjustments has to be made. The stronger the coupling, the more significant and far-reaching is this reaction. A scalable network can have coupling, but the "zone of influence," within which coupling is strong, should be curbed from growing.

2.5 Interlayer Cooperation

The *OSI Reference Model* is a network architecture developed so that different systems can work together in an "open" environment. According this model, each node can be divided vertically into seven functional layers (from bottom to top): **Physical, Data Link, Network, Transport, Session, Presentation, Application**. Peer-to-peer communication can occur on each layer. However, the top four layers provide **end-to-end connections** for source-destination flow control or ARQ, and the lower three layers provide **hop-by-hop** communications

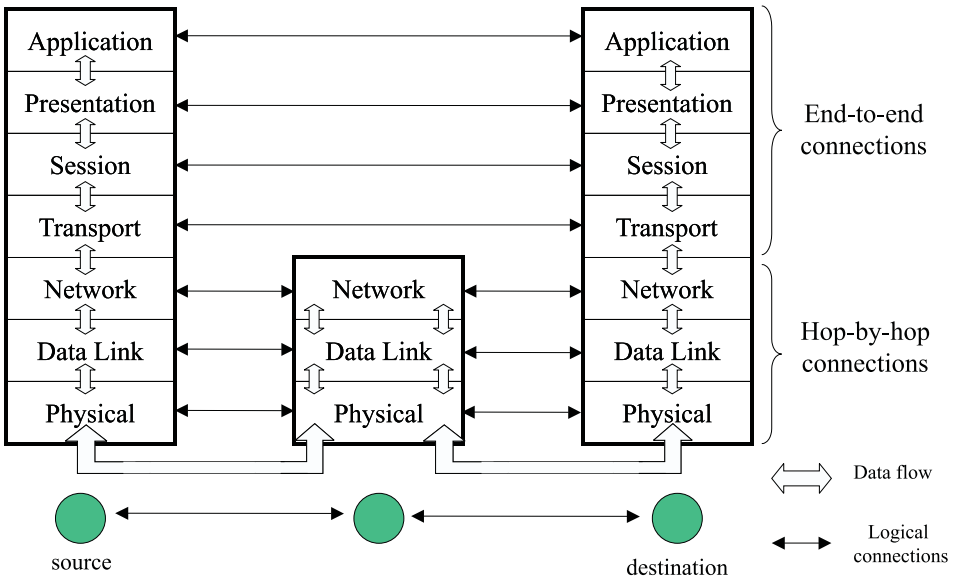

Figure 2.2: OSI Reference Model

needed between the source and destination node. (Figure 2.2) Ideally, with regard to the goal of interfacing different systems together, the operational details of the lower layers should be transparent to those on top. However, this should not be taken as a recommendation to develop each layer in isolation. Rather, we should note that there are often close relationships between the layers, and their division in the model is merely done for convenience for adaptation in an open system.

If we look at the relationship between network and data link layers, we see that they can be implemented as separate systems as long as a standardized interface is used. However they share many common design considerations such as QoS, energy consumption and delay on a hop-by-hop basis, so it is important that we consider how they might cooperate and enhance each other.

As an example, let us consider a system where the *flooding* algorithm is the best choice. This may occur in military applications where robustness and simplicity are the primary concerns. Although flooding works with any link level protocol, it is wise to take advantage of the omnidirectional propagation of radio waves by creating a broadcast channel. A broadcast channel allows each node to send data to all neighbors in one single transmission, thus reducing traffic load and time delay. We can see that if a point-to-point TDMA is used for MAC, then the broadcast procedure will be time consuming because multiple transmissions are required. On the other hand, if conserving energy is the primary concern for a network, an explicit or implicit path construct will be used to "direct" each packet from the source to the destination. In this case, a broadcast channel is not only unnecessary but also wasteful of energy because the radio has to stay on all the time to listen to broadcast messages. A structured MAC such as TDMA or FDMA will be better.

2.6 Summary

In this section we briefly described the concept of ad-hoc networks and some basic issues in MAC and routing protocols. We also discussed the concept of scalability and the necessity of interlayer cooperation between MAC and networking. In the coming chapters, we will use many of these concepts and ideas to design a new set of routing and network formation protocols for autonomous sensing applications.

Chapter 3

Multi-hop Routing

This chapter presents a multi-hop routing algorithm designed for wireless ad hoc sensor networks. We start with an overview of the key assumptions, requirements, and constraints in protocol design, and propose a network structure and routing algorithm to meet our objectives.

In order to tie multiple sensors into a cohesive sensing/processing unit, two modes of operation must be supported by a multi-hop routing algorithm: (1) *collection* mode and (2) *distribution* mode, with respect to a **USER** node that serves as an access point through which outside agents can interact with the sensors. This USER node can be a stationary long range radio deployed permanently as part of the sensor network or a mobile unit that can extract data by moving into communication range with the network. When a USER node comes into communication range of the network, it will initiate a distributed computational process to construct routes from each sensor node to itself. If it then moves to a different location and wishes to extract sensor data again, the same computation process can be repeated at this new location. In the *distribution* mode, command

or query packets are multi-casted to selected groups of sensors to trigger sensing, processing, or communication functions or request sensor information and signal processing results. In the *collection* mode, a network USER acts as a "sink" node that gathers from individual sensors information regarding physical environment, target activities and network status.

These two distinct modes of operations create two major traffic flow components: (1) **sensor-to-user** and (2) **user-to-sensor**. The user-to-sensor traffic usually consists of single-packet command messages issued sporadically by USERs who, in most cases, assumes a passive role as a gatherer of sensor data. Therefore the bulk of network traffic belongs to the sensor-to-user component. Furthermore, it can be observed that the user-to-sensor flow pattern is inherently more energy efficient than sensor-to-user flow, as illustrated by Figure 3.1, which shows the difference in loading pattern when sending one packet from each sensor to USER and from the USER to each sensor. The sensor-to-user pattern is clearly assymetric, putting disproportionally heavier strain on the energy resources of the immediate neighborhood of the USER. To prevent early energy depletion and maintain connection, particular attention must be focused on loading balancing inside this neighborhood.

Our basic approach to algorithm design is to optimize performance on the sensor-to-user traffic under a stringent constraint on energy. Sensor-to-user traffic usually has low intensity because packets are generated in response to environmental events. With the aid of hierarchical information processing, sensor data is highly compressed so that only short summary reports containing the most relevent information is sent to the distant USER. One possible performance

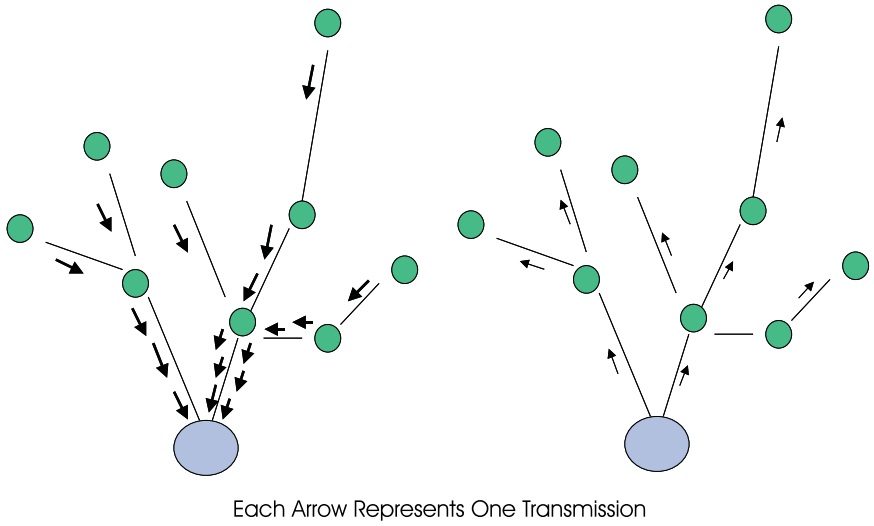

Figure 3.1: Collection and Distribution Mode

criterion is packet delay. For a lightly loaded system, delay is due less to queueing than factors such as link-level ARQ and/or low duty-cycle MAC scheduling. Therefore, the overall delay on each path can be reasonably estimated by a link level additive quality-of-service (QoS) metric. Having the QoS on each path, priority service can be provided to each packet according to the nature of its payload. For example, a detection report for a large battalion of enemy troops will require quicker delivery than the detection of, say, the temperature conditions.

Robustness is another key challenge in network design. Besides environmental factors and probabilistic failure of hardware components, sensor networks can fail rather quickly due to wasteful energy expenditure. To raise robustness, one has to take up preventive measure such as building multiple paths and corrective measures that restore failed paths. It will be a challenge to find energy efficient ways to implement them.

In general energy efficiency can be improved in two areas: (1) route computation and (2) route maintenance. However efficiency in one area often come at the expense of the other. Complex computations may find minimum energy paths, but they can be expensive to maintain as network topology changes. So energy efficiency should be emphasized in each area to the degree that appropriately matches its importance in meeting the overall objective. If we assume most sensors are not mobile, then network topology should change much slower. In this case, more resources can be spent on route computations to provide priority service and long term failure protection, while simple, distributed and localized procedures should be sufficient for route maintenance.

3.1 Network Structure: Table Driven and Multipath

To determine what network structure is appropriate for sensor networks, we turn to the research literature for suggestions and insights. In recent years, mobile ad-hoc networks (MANET) has emerged as the most versatile networking platform for the future. It is designed to provide peer-to-peer multimedia communication between mobile users. Two multi-hop routing algorithms have been proposed for MANET : Ad-hoc On-Demand Distance Vector (ADOV) routing and Temporally Ordered Routing Algorithm (TORA). Both are examples of demand-driven systems that eliminate the overhead associated with table update in high mobility scenarios. However, they have high energy cost during the path discovery phase that must preceed each transmission session. Since our system does not deal with high mobility and multimedia traffic, it is in the interest of energy efficiency to

take the table driven approach.

While the MANET has to deal with mobility by a demand driven approach despite its high cost, the Power-Aware Routing [28] is designed to minimize power consumption under more static network topology. It applies the minimum metric algorithm (shortest path) on two different power metrics:

1. Minimum energy per packet, and

2. Minimum cost per packet.

The first metric is intuitive and produces substantial energy saving **while the network retains full connectivity**; however, possible performance degradation due to node/link failure has not been accounted for. The minimum cost metric is obtained by weighting the first metric by the energy reserve on each node. It has the nice property of load balancing by steering traffic away from low energy nodes. However since the single path algorithm makes a "discrete routing decision" by sending packets on one single path, the cost metric has to be continuously updated to allow a smooth distribution of traffic load. A more energy efficient approach is to create multiple paths so that less route updating is required.

There are many multipath network structures mentioned in the research literature [25, 23, 24]. They mostly deal with the generation of multiple node-disjoint or edge-disjoint paths from each node to one single destination. The degree of failure protection is directly related to the degree of disjointness k. A k-disjoint structure can protect against failures on k links or nodes. Path computations generally start with a graph labeling procedure, at the end of which a path from each node to the destination node is found, and then followed by another round

of graph labeling to generate a new set of paths that are disjoint from the previous set. As a rule of thumb, k iterations will be required to generate a k-disjoint structure. Each iteration will have roughly the same complexity as a shortest path algorithm, and the disjoint property creates strong coupling between routing tables that makes a localized recovery scheme nearly impossible. After the $k+1$ failures occured, a global scale recomputation with complexity k times that of a shortest path algorithm will be required. The key to reduce overhead is to loosen up this coupling effect by relaxing the disjointness requirement. Although the degree of failure protection is lower, one can compensate by a localized path restoration procedure at much lower energy cost.

3.1.1 Tier Structure & Overlapping Trees

Before the network starts to build multiple routes from each sensor to the USER, a minimum-hop spanning tree is used to create a logical tier structure centered around the USER. Tier n will comprise all nodes whose minimum hop-distance from the USER is n. Having the hop-distance allows each node to estimate its topological distance and direction relative to the USER. On top of this tier structure, multiple trees, each rooted from a 1-hop neighbor of the USER, are built. The number of trees generated is equal to the number of 1-hop neighbors the USER has at its present location. Each tree will be forced to grow outward from the USER by successively branching, in the uptree direction, to nodes with higher hop-distances. This serves two purposes: (1) to restrict the coverage of each tree and therefore the degree to which the trees overlap each others, and (2) to control the number of paths each node has to the USER. When tree coverage is too high, the overhead associated with building each tree approaches that of

a global tree; when the tree coverage is too low, there will be an insufficient number of paths available for failure protection. To fine tune the size of each tree, a *TREE_DEPTH* parameter can be used to set the hop-distance above which each tree can branch to nodes with higher or equal hop-distance under certain conditions. A larger *TREE_DEPTH* parameter will reduce the degree of tree overlap. At the end of the procedure, most nodes will belong to multiple trees and thus having multiple paths to the USER. Paths provided by each tree from any node to the USER are node-disjoint inside the 1-hop neighborhood of the USER. The advantage of this structure is that it allows each sensor indirect control of which 1-hop neighbor of the USER will relay each message by selecting the corresponding tree. This is very important for traffic management near the USER, and the more trees a sensor belongs to, the more choices it has.

3.1.2 Tree Building Procedure

For each node, the tier structure and the overlapping trees are computed based on a modified version of a distributed spanning tree (ST) algorithm. There are several advanced features of the modified spanning tree (MST) algorithm. The first feature is that after any topological change, as long as the resulting network is still connected to the USER, path restoration automatically occurs and converges to a feasible solution without the need for a separate recovery algorithm. The second feature is that branching decisions are made by considering multiple metrics, rather than minimizing just one metric. Detailed descriptions of the procedure will be provided in Section 3.1.3, and proof of convergence and path restoration will be given in Section 3.1.4.

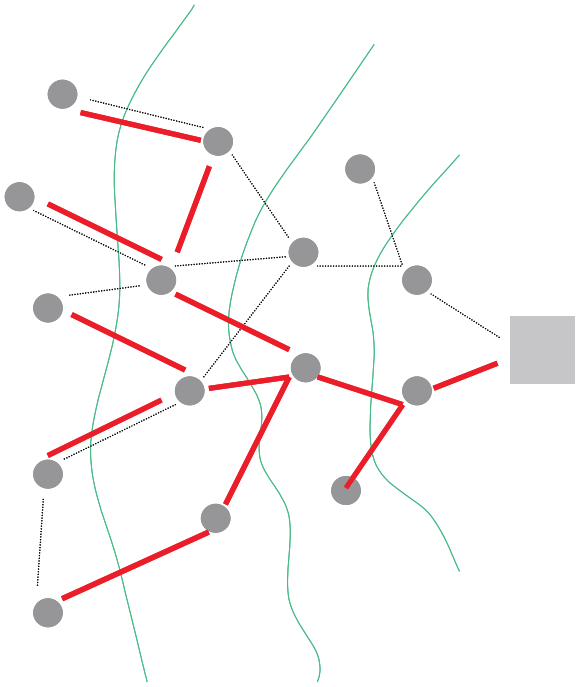

Figure 3.2: Two overlapping trees on a network with four tiers

Spanning Tree Metrics & Branching Metrics

As we know from graph theory, in a spanning tree each node will have a single path to the root node. That implies there will be only **one** single outgoing link registered in each routing table for the root. If, for example, in a spanning tree, node i has node j as its outgoing neighbor to the root, then node j is called the *successor* of i, and node i is the *predecessor* of j. In any tree, each node, other than the root, **must** have one *successor* and any number of *predeccessors*. Any path from a node to the root is defined by the *predecessor-successor* pairings that connect it to the root. Therefore the spanning tree algorithm is basically a *successor* selection algorithm. To facilitate the *successor* selection, two metrics are used: (1) **spanning tree (ST) metric** and (2) **branching metrics**. The

ST metric is usually an additive metric that is summed along a path, defined by the its *predecessor-successor* pairs, to reflect certain characteristics associated with that path. The branching metric is used for *successor* selection; it is a quantitative measure of the "desirability" of a neighbor as *successor*, which can depend on the ST metric and some other parameters.

For the tier structure computation, the ST metric and the branching metric are the same, i.e., the hop-distance from the USER. However, in the overlapping tree computation, the ST metric is the QoS on each path, and the branching metric will be both a function of the QoS and path capacity. The QoS metric and path capacity estimate can be computed during the tree building process. Since the QoS metric is additive, higher QoS metric represents lower performance. Path capacity is the estimated number of packets that can be routed from each node to the USER on a single tree before it causes energy depletion to occur. It can be approximated given the average transmitter power setting and current energy reserve level on each node along the path. Although the tier structure and the overlapping trees are created based on different ST and branching metrics, the algorithm is essentially the same. Therefore, it makes sense to first present the different metrics used in both cases, and then present description of the algorithm in general. In Table 3.1, we list some of the notation and definitions that will be used throughout this chapter.

We also define a function I such that:

$$I(A) = \begin{cases} 1 & \text{A is true} \\ \infty & \text{A is false} \end{cases}$$

E	the set of directed links
V	the set of nodes in the network
(i,j)	edge connecting i and j;
n_i	ST metric associated with node i;
$l_{i,j}$	ST metric associated with the directed link from i to j;
$H_k^i = H_{k,a}^i$	minimum hop-distance from k to a as registered in node i;
$H_i = H_i^i$	minimum hop-distance from i a;
$Q_k^i = Q_{k,a}^i$	QoS metric from k to USER on tree a as registered in node i;
$Q_i = Q_i^i$	QoS metric from i to USER on tree a;
$L_k^i = L_{k,a}^i$	path capacity from k to USER on tree a as registered in node i;
$L_i = L_i^i = L_{i,a}$	path capacity from i to USER on tree a;
$s^i = s_a^i$	node i's successor on tree a;
$P_i = P_{i,a}$	directed path from node i to USER, as implied by current chosen successor;
$M_k^i = M_{k,a}^i$	ST metric of k on tree a as registered in node i;
$M_i = M_{i,a} = M_{i,a}^i$	ST metric from i to USER on tree a;
$B_k^i = B_{k,a}^i$	Branching metric associated with k on tree a as computed by node i;
N_l^i	all nodes l-hop away from node i;
$N^i = N_1^i$	1-hop neighbor of i;
N^{*i}	set of "feasible" neighbors of i. A neighbor, say j, is feasible if $B_j^i < \infty$;

Table 3.1: Notation & Definitions

Tier Structure Computation

To apply the MST algorithm to create a tier structure, we initialize the ST metric associated with node i to zero: $n_i = 0, \forall i \in V$ and the ST metric associated with the directed link (i,j) to 1: $l_{i,j} = 1, \forall (i,j) \in E$. Then the ST metric, branching metric, and successor selection will be triggerred from the root and computed distributively. We note that at each node i, the ST metric represent its minimum

hop distance from the root: $M_i = H_i$. The branching metric associated with j computed by i is:

$$B_j^i = M_j^i + l_{i,j} \tag{3.1}$$

A node, say node i, will choose its successor:

$$s^i = j \text{ such that } B_j^i = \min_{k \in N^i}\{B_k^i\} \tag{3.2}$$

Its ST metric will then be:

$$M_i = M_{s^i}^i + l_{i,s^i} = M_{s^i}^i + 1 \tag{3.3}$$

Overlapping Tree Computation

To show how branch decisions are made for each overlapping tree, let us consider the network shown in Figure 3.3, which shows two overlapping trees a and b. Node i has n neighbors, $N^i = \{1, 2, \cdots, n\}$, where node $\{1, 2, \cdots, m\}$ belong to tree a and node $\{k, \cdots, n\}$ belong to tree b. In this case, n_i and $l_{i,j}$ are the QoS metrics associated with node i and link (i, j), respectively. Then for node i, the ST metrics associated with each neighbor on tree a and b are:

$$M_{j,a}^i = M_{j,a} + l_{i,j}, j \in \{1, 2, \cdots, m\} \tag{3.4}$$

$$M_{j,b}^i = M_{j,b} + l_{i,j}, j \in \{k, \cdots, n\} \tag{3.5}$$

Let $p(i, j)$ by the average energy consumption per packet on link (i, j), and b(i) be the battery reserve in node i at the time of the computation, then the path capacity can be estimated by:

$$L_{j,a}^i = \min\left\{\frac{b(i)}{p(i,j)}, L_{j,a}\right\}, j \in \{1, 2, \cdots, m\} \tag{3.6}$$

$$L_{j,b}^i = \min\left\{\frac{b(i)}{p(i,j)}, L_{j,b}\right\}, j \in \{k, \cdots, n\} \tag{3.7}$$

where $L_{j,a}$ and $L_{j,b}$ are the capacity estimates from node j to the root on tree a and b, respectively.

On any given path, lower metric and higher path capacity are both desirable. Therefore, we want both factors included in the branch metric. For this purpose, a function Γ used to combine both considerations into a single quantity for branching decision. In general, Γ should be non-decreasing with respect to the ST metric and non-increasing with respect to the path capacity. In our algorithm, we choose a simple form:

$$\Gamma(x, y) = \frac{x}{y+1} \tag{3.8}$$

and evaluate it for each neighbor of node i on each tree:

$$B^i_{j,a} = \Gamma(M^i_{j,a}, L^i_{j,a}) I(H_i > H_j), j \in \{1, 2, \cdots, m\} \tag{3.9}$$

$$B^i_{j,b} = \Gamma(M^i_{j,b}, L^i_{j,b}) I(H_i > H_j), j \in \{k, \cdots, n\} \tag{3.10}$$

The term $I(H_i > H_j)$ is used to limit the tree size and can be turned into a $\geq$ relation, as described in Section 3.1.1. The successor and ST metric for node i on tree a and b will be:

$$s^i_a = j \text{ such that } B^i_{j,a} = \min_{k \in N^i} \{B^i_{k,a}\} \tag{3.11}$$

$$s^i_b = j \text{ such that } B^i_{j,b} = \min_{k \in N^i} \{B^i_{k,b}\} \tag{3.12}$$

$$M_{i,a} = M^i_{s^i_a,a} + n_i \tag{3.13}$$

$$M_{i,b} = M^i_{s^i_b,b} + n_i \tag{3.14}$$

The path capacity for tree a and b between node i and the USER will be $L^i_{s^i_a,a}$ and $L^i_{s^i_b,b}$, respectively.

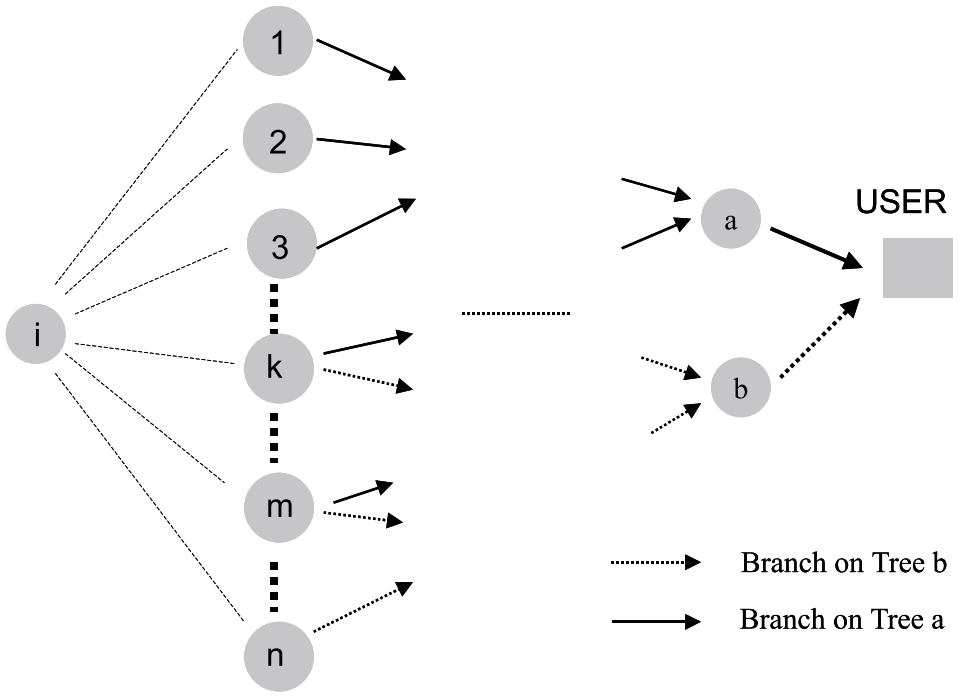

Figure 3.3: Overlapping Trees: Selecting a Successor Node

3.1.3 Modified Spanning Tree Algorithm (MST)

The modification to the ST algorithm involves a hand-shaking procedure that "enforces" ST metric consistency between any *predecessor-successor* pairs. For tier computation, the ST metric is just the hop-distance; for the overlapping trees, the ST metric is the QoS metric. During the operational lifetime of the network, the MST process can be triggered by the root node many times to either create new trees or reset ST metrics on an existing tree. We define a computational *epoch* to be the time duration between two successive MST procedures triggered from the root. Each epoch represents a completely fresh new cycle, whose operation will be independent of the result of the last epoch. Therefore, it is sufficient to

describe the behavior of the MST during a single epoch.

The MST algorithm requires two buffers: (1) First-In-First-Out (FIFO) buffer to hold MST and data messages, and (2) Random Access Memory (RAM) buffer to hold *ACK* and *NACK* messages only. The network function can retrieve messages from either buffers, depending on its current state. The algorithm can operate in one of two possible states: (1) **ACTIVE** state, during which both MST messages and data messages are processed sequentially as they are retrieved from the FIFO buffer, and (2) **WAIT** state, during which a node attempts to establish a new *successor* by sending a request message and waiting for either an *ACK* or *NACK* message for a limited time duration. The *ACK* signifies acknowledgement and agreement to the proposed change by the new successor node, and *NACK* or timer expiration will cancel the proposed successor change. In MST, there are three basic message types:

1. *ST_Metric_Update* - Reports ST metric changes to neighboring nodes.

2. *Link_List_Update* - Reports topologicial changes to neighboring nodes.

3. *Successor_REQ, ACK, NACK* - Signalling messages used during the *predecessor-successor* hand-shaking procedure.

and the basic entries in the routing table for each tree are:

- *counter* - An integer value set by the USER node after initiating a global computation, used to distinguish a new computational epoch from the old.

- *current_time_stamp* - Time stamp value of the last MST message processed, except *Successor_ACK, Successor_NACK*. It is used to filter out old MST

messages that carries outdated information regarding network state during the current epoch.

- *root* - ID no. of the root node (the 1-hop neighbors of the USER). (default = *null*.)
- *successor* - current successor node ID. (default = *null*.)
- *state* - **ACTIVE** or **WAIT**.
- *current_ST_metric* - current ST message as computed based on current *successor*. (default = ∞.)
- *reported_ST_metric* - list of the latest reported ST metric values for each neighbor. (default = ∞.)
- *current_neighbor_list* - list of neighbors and the latest reported link level metric values associated with each neighbor.

A flow chart is given in figure 3.4, and the following is a brief description of the MST procedure:

- **Step 0:** Each sensor node will initialize all entries in the routing table to their default values; each root node, as instructed by the USER, broadcasts *ST_Metric_Update* to all neighbors to initiate a new computational *epoch*. All nodes in **ACTIVE** mode.
- **Step 1:** Each node retrieves a MST message from the FIFO buffer; if the buffer is empty, then wait. For each message retrieved, if it belongs to the previous *epoch* or has an older time stamp then the last processed MST

message, discard it and retrieve another; if it has a new *counter* value, then reset routing table to the default values. This represents the beginning of a new *epoch* at this node.

- **Step 2:** If a node received *Successor_REQ*, then go to Step 7; if *ST_Metric_Update* or *Link_List_Update* is received, it will update the routing table accordingly and recompute the branching metric. If *successor* change is not required, go to step 6.

- **Step 3:** A *Successor_REQ* message is sent to the best "feasible" successor candidate, then enters **WAIT** state for a *Successor_ACK* or *Successor_NACK* message from the RAM buffer. If no "feasible" candidate is available go to step 5.

- **Step 4:** If *Successor_ACK* is received before time out, set state:=**ACTIVE**, update routing table, register new *successor* information, go to step 6; if Time-Out or *NACK* is received, go to step 3.

- **Step 5:** Update *successor*:=*null*, state:=**ACTIVE** and set *current_ST_metric*:= ∞.

- **Step 6:** Compute *Current_ST_metric* and if necessary broadcast new value by sending *ST_Metric_Update* messages to all neighbors. Return to step 1.

- **Step 7:** If the neighbor sending *successor_REQ* is the current successor or the current ST metric is ∞, then the request is denied, by a *NACK* to prevent route table conflict; otherwise request is granted by *ACK*. Return to step 1.

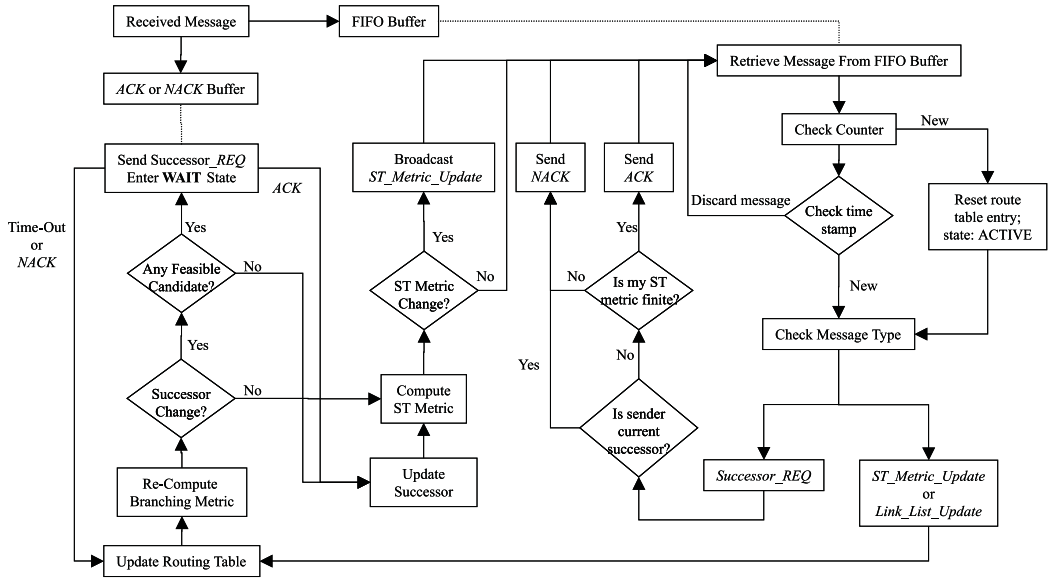

Figure 3.4: Procedure for Modified Spanning Tree Algorithm

3.1.4 Proof of Convergence & Path Restoration for MST

In this section, we provide a proof of convergence for the MST using the branching metric Γ described in Section 3.1.2. We will only consider the behavior of MST within a single computational *epoch*. At any time t, a node will be in one of the following three categories in relation to the root:

I. Topologically disconnected.

II. Topologically connected but has no valid route at time t according to its routing table.

III. Topologically connected with valid route at time t.

Condition **(I)** usually occurs after long period of network operation, when energy resource in its surrounding region is completely exhausted. Once a node falls

into this category, it will be difficult to restore connection by itself. One possibility is to raise transmitter power in the hope of connecting with some distant nodes. However, such effort is usually not feasible because those distant nodes will also have to raise transmitter power in order to establish duplex connection. The high energy consumption required and possible increase in co-channel interference make such desperate efforts less appealing. A more likely scenario is when another batch of new nodes are deployed in the same region, thus restoring network connectivity. However, as long as a node stays in category **(I)**, it will not participate in the multi-hop operation. Therefore our proof refers exclusively to nodes in category **(II)** and **(III)** only.

General approach for the proof

We will prove the convergence property of MST by showing that if at time $t = 0$, a node belongs to category (II) and that no further topological changes occur after time $t = 0$, then it will *eventually enter and stay in category (III) after finite time.* We start by proving a series of propositions that shows the handshaking procedure used to establish each *predecessor-successor* pair will enforce ST metric consistency. Therefore, the ST metric for any node in (II), as a function of time, will be lower bounded by a monotone increasing sequence that converges to ∞. Using this fact in several lemmas, we can show that any node with neighbors in category (III) will eventually select its successor exclusively from them, thus become a permanent member in category (III) itself. We then apply the last argument iteratively to show that any node in category (II) will be in category (III) in finite time. A node can be in category (II) for many reasons. It could that at time t the spanning tree has not yet been built, or that a topological change

occurs before t which caused its path to the root to become invalid temporarily. In any case, our proof will show that as long as a "feasible" path exists, the MST algorithm will converge in finite time. This proof shows both the convergence of the MST when started at the beginning of an *epoch* and its ability to automatically restore paths in finite time after some failures occur.

Notations & Definitions:

- $G = (V, E)$ is a connected network that contains the root node r at time t.
- V is the set of "feasible" nodes with non-zero but finite energy resource.
- E is the set of "feasible" edges with non-zero but finite link metric.
- $\forall i, j \in V$, $(i, j) \in E$ implies there is an edge connecting i and j.
- Let r denote the root node of a spanning tree.
- If $s^i = j$, then the node j is the successor of node i.
- Let $C_i(t), i = 1, 2, \cdots$ represent mutually disjoint maximal connected subgraphs in G such that: $\forall$ node $j \in C_i(t)$, the path defined by its successors: $\{s^j(t), s^{s^j}(t), s^{s^{s^j}}(t), \cdots\}$ does **not** lead to r.
- Let $C(t)$ be the union of all $C_i(t)$ in G.
- Let $M^*(t)$ be the minimum ST metric in $C(t)$, and if $C(t) = \emptyset$, then $M^*(t) = \infty$.
- $i^*(t)$ is the node in $C(t)$ having the minimum ST metric $M^*(t)$; if two nodes have the lowest ST metric in $C(t)$, then an arbitrary but fixed rule

independent of the MST process and parameters are used to select $i^*(t)$. Therefore, $i^*(t)$ is unique at all times.

- $N(t_1, t_2)$ is the number of times i^* changes during $(t_1, t_2]$.
- $N_i(t_1, t_2)$ is the number of times i^* **changes into** i during $(t_1, t_2]$.
- $N_{\max}$ is the maximum number of neighbor for each node.
- $l_{\min}$ is the minimum link metric in E.
- t_{out} is the time out for **WAIT** state.
- C^c is the complementary set of C in G.
- A node i becomes "permanently" in C^c after time t_o if and only if $\forall t > t_o, i \in C^c(t)$.

Assumptions:

- The branching metrics and ST metrics are those described in Section 3.1.2.
- $\forall i, j \in V$ and $(i, j) \in E, n_i$, $l_{ij}(> 0)$, $b(i)$, $p(i, j)$ are assigned at the beginning of a computational *epoch* and held constant by the MST algorithm during that *epoch*.
- Time progression and delay account for only the processing of MST messages or idle periods; time consumed by *data* packet processing and transmissions incurred by the link level protocol are excluded.
- Examine only the behavior of MST during one computational *epoch*.
- Assumes each *epoch* has infinity time duration.

- Assumes topological changes occurs before $t = 0$ created a non-empty $C(t)$ at $t = 0$, and not topological changes occurs after $t = 0$.

Proposition 3.1 *If $s^i(t) = j$, then $s^j(t) \neq i$.*

Proof: Step 7 of MST algorithm prevents any two nodes from setting each other as successors. See Section 3.1.3.

Proposition 3.2 *If $i \in C(t), j \in C^c(t)$, and $(i, j) \in E$, then $s^i(t) \neq j$ and $s^j(t) \neq i$.*

Proof: We can prove by contradiction. Since $j \in C^c(t)$, if $s^i(t) = j$, then i has a path to r which suggest i cannot be in $C(t)$. Similarly, since $i \in C(t)$, if $s^j(t) = i$, then j has no path to r which implies $j \in C(t)$.

Proposition 3.3 *If $C(t) \neq \emptyset$ and $M^*(t) < \infty$, then* **(i)** *$i^*(t)$ has successor $j \in C(t)$, and* **(ii)** *there exists a routing table inconsistency: $M_j(t) + l_{i^*j} > M_{i^*}(t) = M^*(t)$.*

Proof: Again, we prove by contradiction. For (i), if $i^*(t)$ has no successor, step 5 of MST algorithm will set ST metric to ∞. For (ii) if $M_j(t) + l_{i^*j} \leq M_{i^*}(t) = M^*(t)$, then $M_j(t) \leq M_{i^*}(t) - l_{i^*j} \leq M^*(t)$, which contradicts the fact that $M^*(t)$ is the minimum ST metric in $C(t)$.

Proposition 3.4 *Let $\Delta t = t_{out} N_{\max}$. If $C(t) \neq \emptyset$ and $M^*(t) < \infty$, then $\exists t_o \in (t, t + \Delta t]$ such that:* **(i)** *$M^*(t_o) \geq M^*(t)$,* **(ii)** *let t_1 be the first instance after t_o that $i^*(t_1) = i^*(t)$, and if $M^*(t_1) < \infty$ and then $M^*(t_1) \geq M^*(t) + l_{\min}$, and* **(iii)** *if $M^*(t_o) < \infty$, then $i^*(t) \neq i^*(t_o)$.*

Proof: By Proposition 3.3, at time t, i^* has a route table inconsistency. Therefore, according to the MST algorithm, it must be either re-calculating the metric or in **WAIT** state for successor acknowledgment. The ST metric for i^* will not change untill either (1) a new successor k is found, or (2) no successor candidate is found among the neighbors. Both (1) and (2) **must** occur before $t + \Delta t$.

Let t_o denote the earliest instant in $(t, t + \Delta t]$ when either (1) or (2) occurs. For easy of argument, we prove (iii) first. Consider case (1), since node $i^*(t)$ found a new successor at time t_o, then that new successor must have finite ST metric. This implies the minimum ST metric for $C(t_o)$ must also be finite. For case (2) we show that the contrapositive of (iii) is true. Suppose $i^*(t) = i^*(t_o)$ and no successor is found, then at time t_o, the ST metric for node $i^*(t)$ is set to ∞. However, that would also imply the ST metric for $i^*(t_o)$ is ∞ because $i^*(t) = i^*(t_o)$. This proves (iii).

By our selection of t_o, node $i^*(t_o)$ must have ST metric greater than or equal to that of $i^*(t)$ during time interval (t, t_o). If $i^*(t_o)$ changed ST metric during $(t, t_o]$, then it cannot become lower or equal than $M^*(t)$ because that would suggest it has a neighbor whose ST metric is lower than $M^*(t)$ prior to time t_o and contradicting the premise. In other words, $M_{i^*(t_o)}(\tau) \geq M^*(t), \tau \in (t, t_o]$, and therefore $M^*(t_o) \geq M^*(t)$. This proves (i).

To prove (ii), we consider both case (1) and (2):

- If (1) occurs, then $s^{i^*(t)}(t_o) = k$, and from (i) we know that:

$$\begin{aligned} M_{i^*(t)}(t_o) &= M_k(t_o) + l_{i^*(t)k} \\ &\geq M^*(t_o) + l_{\min} \\ &\geq M^*(t) + l_{\min} \end{aligned} \tag{3.15}$$

Since the ST metric for node $i^*(t)$ is finite at time t_1, then by (iii) we know that during (t_0, t_1), node $i^*(t)$ does not have the minimum ST metric in C. Therefore the inequality in Eq 3.15 remains true for $\tau \in (t_0, t_1)$:

$$\begin{aligned} M_{i^*(t)}(\tau) &\geq M^*(\tau) + l_{\min} \\ &\geq M^*(t_o) + l_{\min} \\ &\geq M^*(t) + l_{\min} \end{aligned} \tag{3.16}$$

At t_1, we know node $i^*(t)(= i^*(t_1))$ again has the minimum ST metric. According to proposition 3.3, a routing table inconsistency occurs at node $i^*(t)$, and it will enter **WAIT** state until successor is either found or determined to be unavailable. Therefore the ST metric will stay the same. Thus we have:

$$\begin{aligned} M^*(t_1) &= M_{i^*(t_1)}(t_1) \\ &= M_{i^*(t)}(t_1) \\ &= M_{i^*(t)}(t_1^-) \\ &\geq M^*(t_o) + l_{\min} \\ &\geq M^*(t) + l_{\min} \end{aligned}$$

- If (2) occurs, then $M_{i^*(t)}(t_o) = \infty$, so that Eq 3.15 obviously holds. Regardless of whether a new successor is found during (t_0, t_1), Eq 3.16 also holds, therefore, we will reach the same conclusion about M^* at time t_1.

This proves (ii).

Proposition 3.5 *Given $C(t_2) \neq \emptyset$ for $t_2 > t_1 \geq 0$, then $\exists i \in C(t_2)$ such that $N_i(t_1, t_2) \geq \frac{N(t_1,t_2)}{|V|}$.*

Proof: (By contradiction) Suppose $\forall i \in C(t_2), N_i(t_1,t_2) < \frac{N(t_1,t_2)}{|V|}$, then

$$N(t_1,t_2) = \sum_{i \in V} N_i(t_1,t_2) < \sum_{i \in V} \frac{N(t_1,t_2)}{|V|} = N(t_1,t_2)$$

.

Remark: Up to this point, we have shown some of the basic properties of the set $C(t)$. Especially important is Proposition 3.4, which provides three key insights.

- The minimum ST metric in C, $M^*(t)$, if finite, will increase by at least $l_{\min}$ each time $i^*(t)$ visits the same node.
- If $M^*(t)$ remain finite is still finite, then $i^*(t)$ will change its identity at least once during any time period of length Δt.
- $M^*(t)$ is non-decreasing as a function of t.

The next series of Lemmas will show that M^* converges to ∞, which will force all nodes in C to find a feasible path to r.

Lemma 3.1 $\lim_{t \to \infty} M^*(t) = \infty$.

Proof: It is sufficient to show that $\forall \eta > 0, \exists t_o > 0$ such that $\forall t > t_o, M^*(t) > \eta$. Let us choose $t_o = |V| \Delta t \frac{\eta}{l_{\min}}$, then there are three possible cases:

Case 1: $(C(t_o) = \emptyset)$ Since $M^*(t_o) = \infty$ by definition, then $M^*(t) = \infty > \eta$, $\forall t > t_o$ since M^* is non-decreasing.

Case 2: $(C(t_o) \neq \emptyset$ **and** $M^*(t_o) = \infty)$ Essentially the same situation as Case 1.

Case 3: $(C(t_o) \neq \emptyset$ **and** $M^*(t_o) < \infty)$ If at time t_o, M^* is still finite, based on its non-decreasing property, we conclude that it must be finite prior to

t_o as well. In other words, $M^*(t) < \infty, \forall t \leq t_o$. Using this fact and proposition 3.4(iii), we see that i^* must change identity at least once within any duration of length Δt. Therefore we have

$$N(0, t_o) \geq \frac{t_o}{\Delta t} = \frac{\eta |V|}{l_{\min}}$$

By proposition 3.5, we choose $i \in C(t_o)$ such that

$$N_i(0, t_o) \geq \frac{N(0, t_o)}{|V|} \geq \frac{\eta}{l_{\min}}$$

Let us now define a sequence of real numbers $\{T_1, T_2, T_3, \cdots, T_{N_i(0,t_o)}\}$ to designate the times that i^* makes a transition to i. Then by proposition 3.4(ii), the following inequalities are true:

$$\begin{aligned}
M^*(T_{N_i(0,t_o)}) &\geq M^*(T_{N_i(0,t_o)-1}) + l_{\min} \\
&\geq M^*(T_{N_i(0,t_o)-2}) + 2l_{\min} \\
&\geq \underbrace{M^*(T_1)}_{>l_{\min}} + (N_i(0, t_o) - 1) l_{\min} \\
&> N_i(0, t_o) l_{\min} \\
&\geq \frac{\eta}{l_{\min}} l_{\min} \\
&= \eta
\end{aligned}$$

Again by the non-decreasing property of M^*, we have

$$M^*(\tau) \geq M^*(t_o) \geq M^*(T_{N_i(0,t_o)}) > \eta, \forall \tau > t_o$$

This proves Lemma 3.1.

Lemma 3.2 *Once a node, say, i is "permanently" in C^c after t', then $M_i(t)$ is bounded above by a finite number for $t > t'$.*

Proof: Since i is in C^c after t', a feasible path to the root will be maintained by MST for at all times after t'. The longest possible path that can be found and maintained by MST for i will have less than $|E|$ links and $|V|$ nodes. Assume that this extreme case occurs, such path still has finite length, the ST metric will be a finite sum and therefore bounded.

Remark: The reason that a node in C does not have bounded ST metric is because its *successor* selection cannot lead to the root. Therefore any path inside C is invalid in that it will either form a loop, or terminate at some node with infinity ST metric. In both cases, no upper bound on ST metric can be found.

Lemma 3.3 *For overlapping trees, the capacity estimate at any node will be bounded above and below by a finite number during each computational epoch.*

Proof: Since the MST algorithm will not change $b(i)$ and $p(i,j)$ for any i and j after an epoch begins, and they are both non-zero finite numbers, then the following inequality holds for any $i \in V$:

$$L_{upper} = \frac{\max_{i \in V} b(i)}{\min_{(i,j) \in E} p(i,j)} \geq L_i \geq \frac{\min_{i \in V} b(i)}{\max_{(i,j) \in E} p(i,j)} = L_{lower} > 0$$

Lemma 3.4 *If $i \in V$ has a neighbor j that is "permanently" in C^c after time t', then eventually, i will also be "permanently" in C^c.*

Proof: Since j is permanently in C^c. Let $\hat{M}_j$ be the upperbound on its ST metric in C^c.

- (Tier structure case:) The ST metric is the same as the branching metric in this case, and the link metric is always 1. By Lemma 3.1, $\exists t_o$ such that $M^*(t_o) > \hat{M}_j + 1$. Since M^* is non-decreasing in time, $M^*(t)$ will always

be greater than $\hat{M}_j + l_{\max}$ for $t > t_o$. Therefore the branching decision in node i will never select any node in $C(t)$ as successor over j after time t_o because node j is always a better choice than those neighbors in $C(t)$. So we have $s^i(t) \in C^c(t), \forall t > t_o$, and i will be permanently in C^c.

- (Overlapping tree case:) By Lemma 3.1, $\exists t_o$ such that:

$$\begin{aligned} M^*(t_o) &> \underbrace{(L_{\max}+1)\frac{\hat{M}_j + l_{\max}}{L_{\min}+1} - l_{\min}}_{\text{constant}} \\ \frac{M^*(t_o)+l_{\min}}{L_{\max}+1} &> \frac{\hat{M}_j + l_{\max}}{L_{\min}+1} \\ \Gamma(M^*(t_o)+l_{\min}, L_{\max}) &> \Gamma(\hat{M}_j + l_{\max}, L_{\min}) \end{aligned}$$

where Γ is defined in Equation 3.8. Now recalled the definition of branching metric B, we have:

$$\begin{aligned} \min_{k\in C(t)} B_k^i(t) &\geq \Gamma(M^*(t_o)+l_{\min}, L_{\max}) \\ &> \Gamma(\hat{M}_j + l_{\max}, L_{\min}) \\ &\geq \max_{l\in C^c(t)} B_l^i(t) \ \forall t \geq t_o \end{aligned}$$

Since for $t \geq t_o$, branching metric to any neighbor in C will be higher than those in C^c, successor $s^i(t)$ cannot belong to C after t_o. Therefore i is permanently in C^c after t_o.

This proves Lemma 3.4.

We now have all the necessary facts to prove MST convergence.

Theorem 3.1 *If at any time t, $C(t) \neq \emptyset$, then C will be empty in finite time given no further topological changes occur after t.*

Proof: At any time t, the root node will be permanently in C^c. By Lemma 3.4, the 1-hop neighbor of root will be permanently in C^c in finite time. Then applying the same argument repeatedly, we can show that, the 2-hop, 3-hop, up to d-hop neighborhood will be permanently in C^c in finite time, where d is the diameter of the network. Since the network has finite size and therefore finite diameter, it takes finite time to cover all nodes in V. This proves the convergence of MST.

Further Remarks on the Proof

In any network, the set $C(t)$ represents those nodes that, although still belonging to the same connected graph as the root node, do not have a valid (i.e., finite metric) path to the root node at time t according its routing table. $C(t)$ can be non-empty under two conditions: (1) when the spanning tree is not completely built, or (2) after link or node failures occur.

In either case, we have proved that the MST algorithm can complete the tree building procedure and restore existing trees in finite time. However, we have also made two key assumptions: (1) each *epoch* has infinte duration and (2) topological changes do not occur after time $t = 0$. We would like to give a few remarks to show that these assumptions do not hinder the effectiveness of MST in realistic applications.

Each epoch has infinite time duration - This assumption effectively means that each computational *epoch* will last sufficiently long for convergence to take place, and simulation study has confirmed that this is almost always the case. Although it is possible that path restoration procedure tiggerred by topological changes at the very end of the current *epoch* may not have

enough time for convergence, the start of the next *epoch* will ensure that a set of valid routes are generated in finite time. Therefore in reality, the finite duration of each *epoch* will not affect MST convergence.

No topological changes occur after $t = 0$ - This assumption simply shifts the time axis so that we only examine the behavior of MST when network topology, perhaps after a rapid succession of changes, becomes stable for a sufficiently long period of time that convergence can take place. This is actually a reasonable assumption because during simulation trials, the observed rate of convergence for MST is always much faster than the rate of topological changes which is primarily the result of energy depletion. If the rate of topological changes is very high, it is usually due to mobility. In those cases a demand-driven protocol would have been used instead of the MST.

One will also notice that although the MST is convergent for both tier structure and overlapping tree calculations, the latter is dependent on the former because the branching metric for each overlapping tree uses the hop distance provided by the tier structure. There are two methods we can use to make sure that the overlapping trees are built **after** a valid tier structure is in place:

Delay - At the beginning of an *epoch*, or immediately after topological changes, each node will carry out each step of MST with a small time delay imposed on the overlapping tree calculation so that it will occur after all necessary tier computations are completed. In simulation experiments, we see that the delay required to separate out these two calculations is rather small, roughly that which is required to relay packets for three or four hops.

Buffer Management - As a safeguard, a buffer management scheme can be used so that when MST messages are retrieved from the FIFO buffer, priority can be given to tier structure computation packets over those for the overlapping tree.

In our simulations, we implemented sufficient delay and checked for the existence of tier structure before computing overlapping trees. No compromise to MST convergence was observed.

3.1.5 Network Failure and Loop Prevention

When a sufficient number of node or link failures occur, the network can become fragmented into mutually disjoint clusters that may or may not be connected with the USER. Although these isolated nodes are temporarily separated from the USER, they can still perform important sensing functions either as individual nodes or in a cooperative fashion and report their findings after re-establishing connection with the rest of the network. Therefore it is important to take a look at the energy consumption incurred by the MST in these isolated clusters.

When a cluster is disconnected from the USER topologically, the MST algorithm will try to restore a path to the USER. However, the convergence of MST is based on the premise that a node is still topologically connected to its root (a 1-hop neighbor of the USER), and this is a form of **global** information that cannot be determined locally. In other words, short of gathering the topological data for the entire network, as a *link state* algorithm would do, most sensors will not be able to determine with certainty whether any feasible path to the root exists. However, within an isolated cluster, each node will typically observe significant

ST metric increase in short period of time. These increases are caused by metric updates occuring inside the cluster as each node tries to find a new route to the root node, and since no valid path can be found, the ST metric has no choice but to increase until reaching infinity. This is clearly predicted by Lemma 3.1, where we showed that $M^*(t) \to \infty$ as $t \to \infty$.

The rate at which the ST metric reaches infinity will determine how soon the path-restoration effort will stop and therefore how much energy will be consumed. One way the ST metric can reach infinity quickly is when it has no neighbor or all neighbors have reported infinite ST metric. The likelihood of this happening will depend on the internal topology of the cluster and many other factors. There is also the possibility that the propagation of *ST_Metric_Update* messages will be trapped in a loop. In this case the ST metric will increase by a finite amount each time the *ST_Metric_Update* message goes around the loop. Such a loop can stay intact for a prolonged period of time, draining significant energy resources on radio transmissions.

One can see that the key to increase the rate of converge to infinity is to either prevent or break up loop formations that would have otherwise stayed intact for a long time. In [19], loops are prevented by suppressing all path restoration attempts on the local level. Whenever a link failure occurs, the root node will be notified and a new computation epoch will begin. This means every failure will require a global recomputation, a high energy cost procedure itself. Instead, we allow local recovery to take place but only try to detect loop formations by setting a threshold value on the ST metric. Any time the ST metric goes above this threshold, the MST algorithm will stop, declare infinity metric to its neighbors, and only resume operation when a neighboring node presents a path

with ST metric lower than the threshold.

By adding a threshold value on ST metric, each node can reach infinite metric much more easily. This will raise the rate of convergence for $M^*(t)$ toward infinity. Beside the obvious benefit of stopping MST quickly when no valid path exists, it is also helpful to the rate of convergence when a valid path does exist because the faster $M^*(t)$ reaches infinity, the faster a node will be "permanent" in set $C^*(t)$.(See Lemma 3.4)

3.2 Sequential Assignment Routing (SAR) Algorithm

For most networks, a minimum QoS metric routing approach is usually the first choice to consider. It is a simple greedy algorithm based on the principle: *Use the optimal path whenever it is available.* Under most circumstances, it generalizes quite well into global strategies that have fairly good performance. Furthermore, if the optimal path is guaranteed to be available at all times, then the greedy principle becomes a globally optimal strategy. Once such an optimal path is found, it can be used forever.

However, in sensor network applications, any path can only operate for a limited time depending on the availability of energy resources, and the more a path is used, the less capacity is available to serve future demand. Under such conditions, the greedy principle will not generalize well unless each packet is treated the same and the best service is provided on a first-come-first-serve basis. For sensor networks, however, packets with different payloads have different requirements, and preferences should be given to those with higher priorities. To

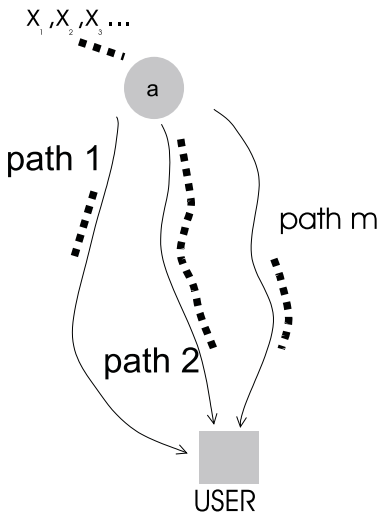

Figure 3.5: Single Source Ideal Scenario

assign the limited energy resources on paths with different QoS to a random set of service requests, a first-come-first-serve approach is no longer efficient. On close inspection, the sensor routing problem is very closely related to a sequential stochastic assignment problem.

3.2.1 Ideal Scenario: Single source, disjoint paths

In the ideal environment, we have a fully disjoint multi-path network and know the exact sequence of packets that we need to route ahead of time. Suppose in response to events detected, a sequence of packets is generated by node a, which has unlimited energy, with priority $\{X_n(\omega) = x_n, n = 1, 2, 3 \ldots\}$ for some $\omega \in \Omega$. Assume node a belongs to m trees providing m disjoint paths such that the maximum number of packets it can route is $N(m) = \sum_{j=1}^{m} L_{a,j}$. Let p_i represent the tree selected by the SAR for the ith packet. Let $W(x)$ be the weight

coefficient associated with priority x, which scales the QoS metric. If there is no other traffic on the network, then node a will exhaust the energy resource on each path by routing $N(m)$ packets to the USER. For performance evaluation, we define the **average weighted metric** to be:

$$\bar{M} = \sum_{i=1}^{N(m)} W(x_i) M_{a,p_i} \tag{3.17}$$

We can interpret the quantity,$\bar{M}$, as the average QoS that is provided to each packet *relative* to its priority level. To maintain the same average weighted QoS, or $\bar{M}$, high priority packets should be given better QoS then lower priority packets. The weight coefficient $W(x_i)$ scales the contribution each packet made to $\bar{M}$, giving high priority packets significant influence over the lower priority packets. Since higher metric represents lower QoS, the SAR algorithm will make routing decisions with the goal of minimizing $\bar{M}$. Let us now define a **resource matrix**:

$$\mathbf{r} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} \tag{3.18}$$

$$= \begin{bmatrix} M_{a,1} \ldots M_{a,1} & M_{a,2} \ldots M_{a,2} & \ldots & M_{a,m} \ldots M_{a,m} \\ \underbrace{1 \ldots\ldots 1}_{L_{A,1}} & \underbrace{2 \ldots\ldots 2}_{L_{A,2}} & \ldots & \underbrace{m \ldots\ldots m}_{L_{A,m}} \end{bmatrix} \tag{3.19}$$

Each column of $\mathbf{r}$ represents the capacity to route one packet from node a to USER. Note that $\forall i, r_{1,i}$ is the metric value associated with $r_{2,i}$, the tree ID number. To map the tree selection p_i to its corresponding column index in the resource metric $\mathbf{r}$, we define: q_i be any permutation of $\{1, 2, 3, \ldots, N(m)\}$ such that $N(p_{i-1}) \leq q_i \leq N(p_i)$. Then Eq (3.17) is equivalent to:

$$\bar{M} = \sum_{i=1}^{N(m)} W(x_i) r_{1,q_i} \tag{3.20}$$

The optimal algorithm will minimize the quantity in Eq (3.20). Without loss of generality, assume the sequence $\{W(x_1), W(x_2), \ldots, W(x_{N_m})\}$ is in ascending order. Lemma 7.1 tells us that to minimize the product sum of two vectors, one should be sorted in ascending order, the other in descending order. This is proved in Appendix A by mapping the product sum into a Gilmore-Gomory matching problem [2]. Therefore, we know that $\bar{M}$ is minimized by finding a sequence q_i^* such that r_{1,q_i^*} is in descending order. The corresponding **optimal tree assignment** will be $p_i^* = r_{2,q_i^*}$.

In the simplest terms, to minimize the average weighted metric, the higher the priority, the better the path that is assigned, and the lower the priority, the lower the QoS that will be provided, even when high QoS paths are available. In the ideal situation, when the capacity and QoS for each path and the priorities of the entire sequence of packets are known, optimal assignment can be achieved.

3.2.2 Real Time Implementation

However, in practical situations, paths are not necessarily disjoint because the overlapping tree structure has relaxed the disjointness requirement in the interest of energy efficiency. Also, traffic can be generated from multiple regions inside the network, not just from a single location. Consequently, parameters for different paths usually interact in a way that cannot be tracked locally. For example, if two trees share a common node, it is very likely that this node will have higher loading because it has to relay traffic from both trees. This is not observable to most upstream nodes in both trees, so their capacity estimates tends to be more optimistic. Although each node can use a linear projections on the local traffic to predict the path capacities available in the future, correction must be made to

keep these local estimates accurate.

The most accurate method of keeping accurate path capacity is to report down stream traffic activities to the upstream nodes. There are several possible ways to accomplish this: (1) piggybacking energy information on the USER-to-sensor traffic or (2) periodic recomputation of capacity estimates.

The first method looks promising because the potential energy saving can be significant, but its feasibility will depend on the frequency of user-to-sensor traffic and whether the packet size is fixed or flexible. Since user-to-sensor traffic usually consists of small packets, if the packet size has to increase to accommodate the additional information, then the overall packet size will have to increase. The receiver will have to turn on its radio for longer time to listen for these additional information. Also the user-to-sensor traffic may be sporadic and covers only a small subset of nodes, and the results maybe unpredictable results.

Therefore it is safer to make no assumptions on the frequency of USER-to-sensor traffic and track the network state primarily by explicit periodic updates. The hope is that since the underlying structure is multipath, enough traffic balance occurs that errors in capacity estimates will not significantly impact the routing decisions. If this is true, then these periodic updates can be less frequent.

In real time implementation, routing decisions must be made on a packet-by-packet basis, without the opportunity to observe the full sequence. The decision process of matching arriving packet X_n to a finite set of resources are made based on predictions of the statistics of future traffic according to the observation of $W(x_i)$ in the past. Such predictive decisions are similar to the **sequential**

stochastic assignment policy described in [31], in which a set of positive numbers $p_1, p_2, \ldots, p_n$ is assigned to another sequence of positive iid random variables $X_1, X_2, \ldots, X_n$ whose CDF is $G_X(z)$, in a sequential manner under an optimal strategy that maximizes the expected value of the product sum, $E\left[\sum_{j=1}^{n} p_{i_j} X_j\right]$. The optimal policy is as follows:

For each $n \geq 1$, there exist numbers

$$-\infty = a_{0,n} \leq a_{1,n} \leq a_{2,n} \leq \ldots \leq a_{n,n} = +\infty$$

such that when ever there are n stages to go and $p_1 \leq p_2 \leq \ldots \leq p_n$ then the optimal choice in the initial stage is to use p_i if the random variable X_1 is contained in the interval $(a_{i-1,n}, a_{i,n}]$. The $a_{i,n}$ depend on G_X but are independent of the p's. Define $a_{0,n} = -\infty, a_{n,n} = +\infty$. Then:

$$a_{i,n+1} = \int_{a_{i-1,n}}^{a_{i,n}} z dG_x(z) + a_{i-1,n} G(a_{i-1,n}) + a_{i,n}[1 - G(a_{i,n})] \quad (3.21)$$

for $i = 1, 2, \ldots, n$, where $-\infty \cdot 0$ and $\infty \cdot 0$ are defined to be 0.

Although the optimal strategy is for a maximization problem, it can be shown that it is also the optimal strategy for the minimization problem through a linear mapping. Again, let $p_1 \leq p_2 \leq \ldots \leq p_n$ be a set of positive number that needs to be assigned in sequential order to a sequence of positive iid random variables: $\grave{X}_1, \grave{X}_2, \ldots, \grave{X}_n$. Since n is finite, let $\grave{X}_{\max} = \max\{\grave{X}_1, \grave{X}_2, \ldots, \grave{X}_n\}$, then we know that there exists positive sequence $X_1, X_2, \ldots, X_n$ such that $\forall j \leq n$, $\grave{X}_j = \grave{X}_{\max} - X_j$. Let i_j be the index of the p's assigned to the jth $\grave{X}$, then

$$E\left[\sum_{j=1}^{n} p_{i_j} \grave{X}_j\right] = \underbrace{\grave{X}_{\max} \sum_{j=1}^{n} p_{i_j}}_{constant} - E\left[\sum_{j=1}^{n} p_{i_j} X_j\right] \quad (3.22)$$

To minimize the left side of Eq (3.22), we can apply the optimal policy for the maximization problem to X_j's, whose distribution can be easily translated from that of the $\grave{X}_j$'s. Therefore the basic approach is the same except that we have to note the proper sign and index reversal.

However, the direct application of the optimal strategy is still difficult for several reasons. First of all, the calculation of $\{a_{i,n}, i = 1, 2, \ldots, n\}$ must start from $n = 1$ and progressively increase up to the maximum, say N. This requires repeated computation of an integral and the memory capacity to hold approximately $N^2/2$ real numbers. As complex as it already is, by removing the independent assumption on the X_n's, as shown in [32], the calculation of $\{a_{i,n}, i = 1, 2, \ldots, n, n \leq N\}$ will become that for the optimal stopping problem of finding the k largest values from $X_1, X_2, \ldots, X_N$ for $k = 1, 2, \ldots, N$. Either way, the processing and memory requirements are too high.

The alternative is to examine what are the functions of the set $\{a_{i,n}, i = 1, 2, \ldots, n\}$ and try to develop a simpler approach to calculate them. The $a(i, n)$'s are a set of numbers that partition the real number into n intervals, and when an observed X_j falls into the i interval, it is then assigned to the ith highest value in the sequence $p_1, p_2, \ldots, p_n$. Comparing this to the optimal policy in the ideal scenario described in Section 3.2.1, where the path with the ith *highest* metric value is assigned to a packet with the ith *lowest* priority, we can see that $\{a_{i,n}, i = 1, 2, \ldots, n\}$ is simply used as a *ranking predictor* on the observed X_j.

In order to avoid the overhead required by the optimal policy, we need to construct a simpler predictor on the ranking of the priority of the arrival packets. To construct such a predictor, assume that $W = \{W_i = W(X_i), i = 1, 2, \cdots, n\}$ is a sequence of iid random variables with CDF $F_W(w)$. The basic concept in

relative frequency tells us that the number of observations out of n realizations of W_j's that falls inside the interval $\left(-\infty, F_W^{-1}(\frac{i-1}{n})\right]$ can be approximated, when n is large, by $n \cdot F_W(F_W^{-1}(\frac{i-1}{n}) = n \cdot \frac{i-1}{n} = i-1$, and that for $\left(F_W^{-1}(\frac{i}{n}), +\infty\right)$ it is $n-i$. Thus, if W_j is observed in the interval $\left(F_W^{-1}(\frac{i-1}{n}), F_W^{-1}(\frac{i}{n})\right]$ then the magnitude of W_j should be close to the ith largest value in $\{W_1, W_2, \ldots, W_n\}$. This predictor is suboptimal in the sense that it is only good under large n, but very simple to compute. Using this predictor, we propose the following suboptimal policy:

> *For any node, say a, assume, without loss of generality, that $M_{a,1} \geq M_{a,2} \geq \ldots \geq M_{a,m}$, so that $r_{2,j}$'s are also in descending order. If by the best estimate, the network is determined to have resource capacity to route total of n packets and $w_i \in (F_W^{-1}(\frac{j-1}{n}), F_W^{-1}(\frac{j}{n})]$, where $1 \leq j \leq n$, then the path assignment will be $r_{1,j}$ with QoS metric value of $r_{2,j}$.*

3.3 Simulation Results

We simulate a randomly deployed sensor network with one USER and 37 sensor nodes. Its topology is shown in Figure 3.6. Each edge represents a bidirectional communication link. Node 15 is detached from the rest of the network. Node 1 is the USER. Assume network traffic is generated by the subgroup consisting of nodes {37,10,17,21,27}, which detects some events in the southwest corner. Each node has equal probability of generating packets, and each packet has three possible priority settings, {low,medium,high}. In our simulation test, 30% of the packets are low priority packets, representing house keeping messages, such as status report, maintenance command, or synchronization packets.

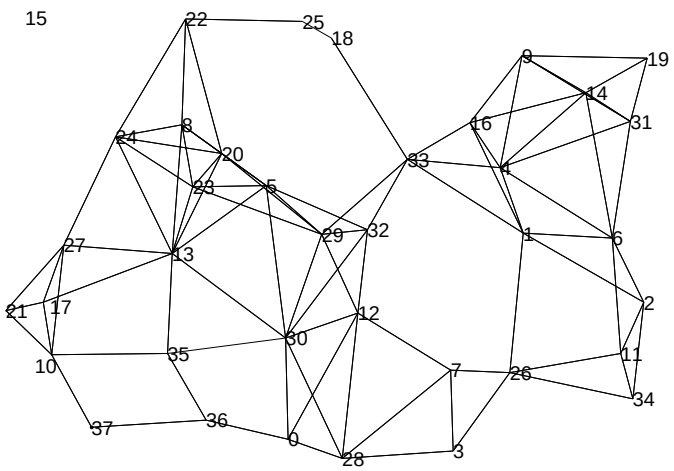

Figure 3.6: Simulated Network Topology

3.3.1 Average Weighted Metric & Capacity

Each link is assigned with one of two possible metric values to reflect different performance levels under general criteria. For the network simulated, a "good zone" is defined that includes nodes {36,30,12,0,28,7,3,26}. For each link that terminated in this zone, the metric is set to 0, for links that terminate outside this zone, the metric is 30. Radios are assumed to operate at fixed power level and the per packet energy consumption is 10 energy units. Each node has metric of 1.0 and an identical power source with 10^6 units of energy. Global metric recomputation is initiated by the USER after every 5000 sensor packets were received. This is a relatively long interval compared to the average radio transmission capacity on each node, which is no more than 10^5 packets.

In our simulation, we focus on how SAR performs under different high priority traffic settings. There are two major traffic parameters: p_h, the probability that a packet generated is high priority, and w_h, the weight coefficient assigned to it.

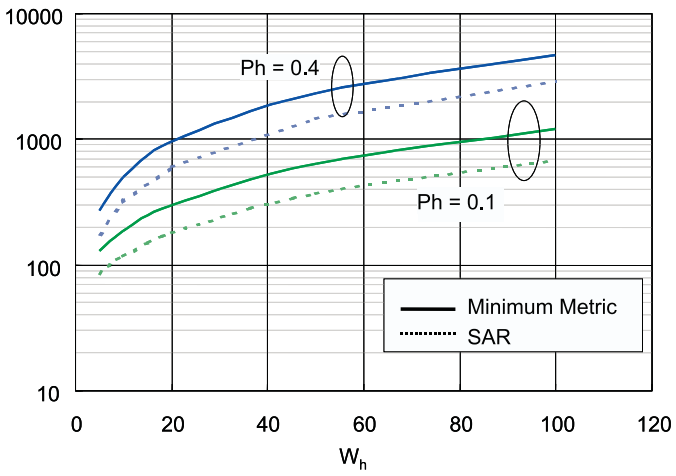

Figure 3.7: Average Weighted Metric & W_h

Higher p_h means more high priority packets will be sent through the system, and their contribution to the average weighted metric will be significant. If a high priority packet is generated at node i and travels on a path with ST metric M_i, then it contributes $M_i w_h$ to the total weighted metric sum. Higher w_h raises the impact each high priority packet has on the average weighted metric. In our simulation, the average weighted metric $\bar{M}$ is compared between the SAR and the minimum metric algorithm, under different p_h and w_h values.

In Figure 3.7, w_h is between 5 and 100. For the top two curves, p_h is 0.4; for the bottom two curves, $p_h = 0.1$. As expected, in both cases, average per packet metric increases with w_h. When $p_h = 0.4$, SAR provides 34 to 39% reduction in average metric; when $p_h = 0.1$, 33 to 44% reduction can be achieved.

In Figure 3.8, the same comparison is made for $p_h \in [0.05, 0.4]$. As expected, the average weighted metric increases when more high priority packets are generated. For $w_h = 50$, SAR is 38% to 40% better than minimum metric; for $w_h = 10$,

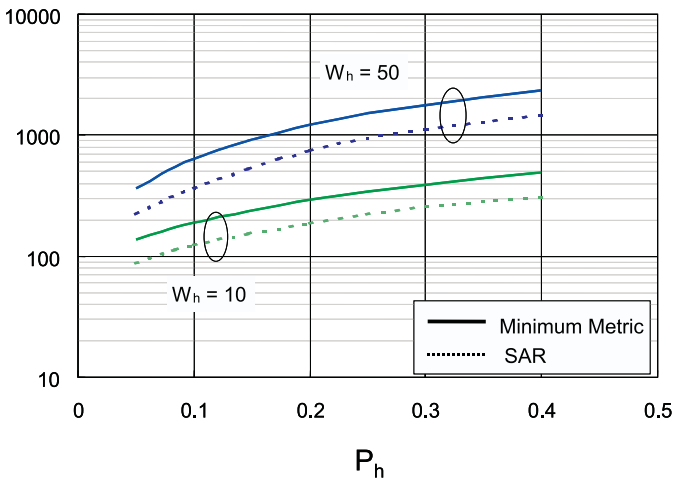

Figure 3.8: Average Weighted Metric & P_h

SAR is about 37.5% better. For both SAR and minimum metric, the metric values grow roughly linearly with p_h and w_h. The SAR algorithm is able to produce a lower metric value because it preserves better paths by limiting their usage, thus making them available for priority packets, which is more important to the overall system performance. On the other hand, the minimum metric algorithm makes no such distinction based on priorities. It assigns many low and medium priority packets to the best path, which degrades the performance in the long run. Frequent recomputation can significantly reduce network capacity. Figure 3.9 shows that for both algorithms, the average network capacity takes a 20% loss when the frequency of recomputation increases by one order of magnitude.

3.3.2 Sensitivity Study

The performance of any algorithm depends heavily on the validity of its input parameters. Input errors will generally lead to output errors to some degree.

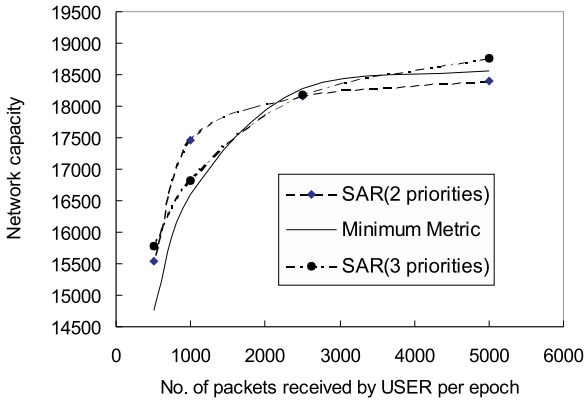

Figure 3.9: Network Capacity & Epoch Duration

The "sensitivity" of an algorithm can be understood in terms of how detrimental input error will be to the output. Over-estimation or under-estimation on either the QoS or capacity metric will cause traffic imbalance and affect the average weighted QoS. The primary function of a global re-computation is to reset algorithm parameters based on the current condition of the network, and keep the routing tables on track. The period between two successive recomputations is called an *epoch*, and it is during this period that the network goes through changes that the algorithm is not aware of. The deviation of these parameters from the actual state of the network is the reason for performance degradation. In our simulation study, we look at the effect of (1) capacity estimate error and (2) channel metric error separately.

In SAR, path capacity is computed during the tree building process. Although it can be updated later on during a path restoration procedure, it only reflects

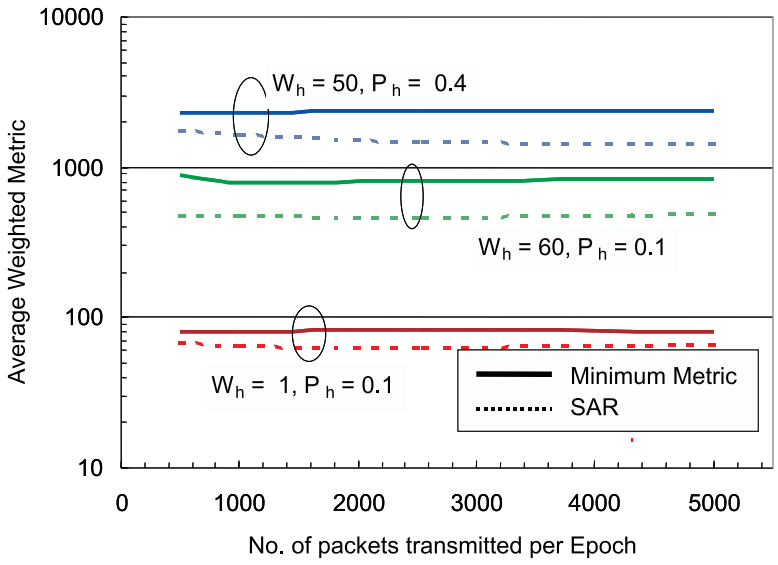

Figure 3.10: Average Weighted Metric & Epoch Duration

local changes. Therefore, it is of considerable interest to see how much degradation is possible when the number of recomputations is reduced. In general, metric recomputation is triggered by the USER. Therefore a natural criterion used to determine the length of each *epoch* is the number of packets a USER receives during that *epoch*. Furthermore, since other energy consuming activities are not simulated in our study, this criterion can provide an accurate measure of capacity change during each *epoch*.

To study the effect of energy capacity error, three different traffic settings under fixed link metrics are used. We change the duration of a computational *epoch* and compare average weighted metrics $\bar{M}$ for SAR and minimum metric. Figure 3.10 shows the simulation results. The two curves on top have $p_h = 0.4$ and $w_h = 50$. No significant change in $\bar{M}$ is observed for the minimum metric algorithm. However, SAR algorithm suffers a 20% **increase** when recomputations

are more **frequent**, which is not what we expected. A possible explanation is that when recomputation is more frequent, communication overhead will increase, resulting in a significant reduction in energy resource available to high priority traffic. The minimum metric algorithm makes no such distinction, therefore it has less sensitivity as well. The other two pairs of curves, for $p_h = 0.1$ and $w_h = 60$ and $w_h = 1$, showed no observable metric increase because at $p_h = 0.1$, high priority packets have less impact on the average metric. Overall, it seems that both SAR and minimum metric algorithm have fairly low sensitivity to error in capacity estimates.

To examine the sensitivity to link metric accuracy, we simulate the same network under fixed *epoch* duration and dynamic channel metrics. The radio channel is allowed to change, after the initial route computation, among three states: {**good**, **average**, **poor**} with metric value of 1, 30 and 60, respectively. Each link is allowed to make a transition from its current state to one of the other two states with probability: $\frac{p}{2}$ whenever a packet is routed through it. The higher the value of p, the less accurate the metric estimates in the routing table will be. We compared the performance of both algorithms under two different settings: $p_h = 0.1$, and $p_h = 0.3$ under $w_h = 10$ and allow p to vary from 1% to 50%. The result is summarized in Table 3.2.

For smaller p, the gap between the average weighted metric is large, with the minimum metric algorithm producing approximately 31% to 37% higher metric than SAR. But with increased p, the gap shrinks considerably to somewhere between 3.5% to 8.6%. The change is expected if one considers the fact that the more the channel changes, the more random the routing decisions will become.

Thus both algorithms approach a random decision scheme. The alternative solution is to dynamically track the channel state and update the routing table more frequently, but any gain we may obtain could easily be offset by the unavoidable high overhead required by such an operation.

p: channel metric transition probability						
$p_h = 0.1, w_h = 10$						
p	0.01	0.02	0.05	0.1	0.2	0.5
Min Metric	173	180.02	185.53	205.45	219.48	266.4
SAR	132.22	135	140.41	151.4	181.37	245.6
Reduction %	24%	25%	24%	26%	17%	8%
$p_h = 0.3, w_h = 10$						
Min Metric	354	360	367.2	401.5	436.89	536.2
SAR	258.6	257.92	282.8	331.65	359.9	518.4
Reduction %	27%	28%	23%	17%	18%	3%

Table 3.2: Average Weighted Metric & p

3.4 Summary

In this chapter, we presented a sensor network routing algorithm designed to provide multihop priority routing service between individual sensors and a USER node, under significant energy limitation. The underlying network structure is multi-path, generated by multiple overlapping spanning trees rooted at the

USER. The network construction uses a Modified Spanning Tree (MST) algorithm that is guaranteed to converge to a feasible solution and provide automatic path restoration. On top of this multi-path structure, we apply a Sequential Assignment Routing (SAR) algorithm that dynamically adapts to network condition to make route assignments based on a combined consideration of the packet priority and the QoS and capacity on each path.

Chapter 4

Adaptive Local Routing for Non-Coherent Cooperative Functions

To increase the signal processing capabilities, a group of sensors can form a local network to collect, exchange, and analyze sensor data in a cooperative fashion. Such cooperative signal processing techniques can reduce the probability of false alarm, estimate signal source location and enhance overall SNR [22]. The creation and operation of such local networks must be supported by a set of network functions that facilitate the necessary internodal signaling and data transfer. In contrast to the multihop routing functions required for global data collection, cooperative functions usually involve a small set of nodes near the target location and operate for a relatively short time span. They need to adapt temporally and spatially to the pattern of target appearances and the nature of the signal processing techniques used. Although the multihop functions are also adaptive,

one key distinction is that while it adapts to the traffic pattern over time, local network adapts *a priori*, by generating paths based on traffic loading declared by participating sensors before the formation process begins.

4.1 Overview and Assumptions

Cooperative functions can extract valuable target information that individual sensor cannot obtain. The decision to form a local network and engage in a particular cooperative function can be made by a remote USER after receiving a target detection report or by a built-in algorithm in each sensor when pre-defined conditions are met. In any case, we assume that a high level algorithm or outside agent will determine what cooperative function is needed and trigger the network formation process. We assume the sensor network is connected, and a link level channel access control algorithm provides contention-free point-to-point duplex communications between any "connected" pair of nodes.

In an energy constrained network, a point-to-point protocol has the attractive feature of not requiring the radio to stay on for long duration in "listening mode." This allows the network to run at very low duty-cycle and extends the operational life of the network. Also, we assume that the packet loss probability is sufficiently low on each link so that the energy consumption is linearly proportional to the number of successful transmissions. The term "local network" refers specifically to any connected subnetwork consisting of sensors that detect a common target, and we may sometimes simply refer to it as the "network" to reduce verbiage. Before describing the network formation algorithm in detail, a few remarks on the basic categories of environmental stimuli and cooperative functions are warranted.

4.1.1 Target & Cooperative Function Categories

In general, environmental stimuli can be separated into two major categories: (1) near-field (NF) and (2) far-field (FF). Near-field stimuli have short range relative to the baseline width of sensor groups within detectable distance. Accurate localization and identification are possible if the target is located inside the convex hull of the network. Depending on its signal strength, the number of sensors that can detect its presence varies greatly. We can model the SNR at each sensor by an inverse relationship to some power of the distance because propagation loss is dominated by the line-of-sight component. However, more sophisticated models will have to take other factors into account. For example, for accoustic sensors, wind, microphone height, and physical orientation of the microphone will have a significant effect on the SNR of the data. It is also possible that a NF target will see reverberance due to the accoustic structure of the surrounding environment. Another major difficulty is the SNR degradation caused by low-pass filtering of the transmission medium. Nonetheless we can still expect the best data to be gathered by sensors that are closer to the target.

Far-field targets are located at much farther distance relative to the baseline width of the network. For these targets, source localization and range estimation can be difficult. Due to their physical distance from the network, seismic or accoustic signal propagation may have a strong multipath component. Therefore the SNR measured at each sensor can no longer be accurately modeled by a simple inverse relation the physical distance. Instead, a Raleigh random variable will be used to capture the multi-path effect.

There are two types of cooperative signal processing techniques: (1) non-coherent and (2) coherent. For non-coherent processing such as data fusion, raw

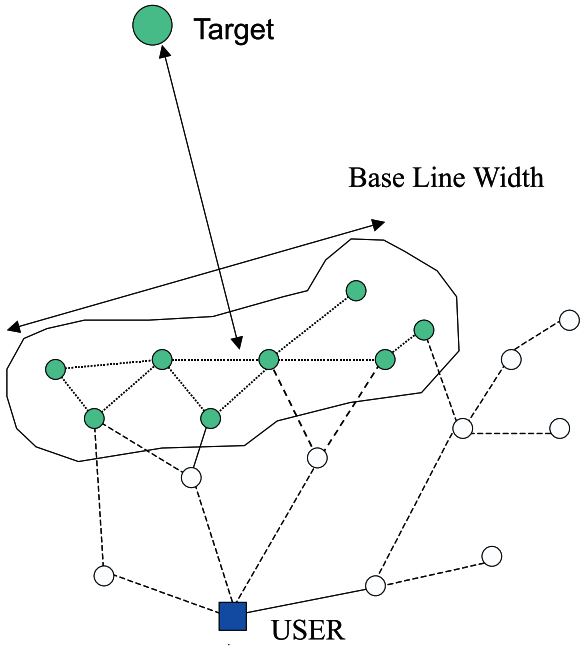

Figure 4.1: Target and Local Network

sensor data will be preprocessed at each node to extract a small set of parameters to be forwarded to a central node (CN) for further processing; for coherent processing like beam-forming, raw sensor data, after minimal pre-processing, will be tagged with a time stamp and uploaded through the local network to the CN for more intensive computations. The key difference is that coherent processing will generate a much higher traffic load in the local network compared to the non-coherent case. One particularly important case for protocol design is when a target with very strong signal strength is present. In this case, the entire sensor network may be awakened, and there is a potential that too much network resources will be spent on a single target when coherent processing is used. In the interest of energy efficiency, the number of participating sensors must be limited

as a safeguard.

Although energy efficiency is the ultimate goal, different approaches can be used for different cooperative functions. As we mentioned before, non-coherent functions generate light data traffic loading. Therefore energy efficiency cannot be won through generating minimum energy paths, rather, algorithmic efficiency should be emphsized. There are two approaches by which algorithmic efficiency can be raised:

1. Reduce signalling overhead (number of transmissions), and

2. Reduce average energy consumption for each signaling message.

To reduce energy consumption for each signaling message will require the construction of a low-energy signaling network, which may further complicate the algorithm and raise the overhead. The additional complexity required may wipe out any gain achieved. Besides raising overhead, a separate signaling network will not do much good if an energy conserving protocol is already used on the link level. In such a case, the network will be operating on top of a set of energy efficient links which will serve both the signaling and data communication needs. For these reasons, we will attempt to achieve energy efficiency mainly through overhead reduction and from time to time, use the terms "efficiency," "overhead efficiency," and "energy efficiency" almost interchangeably throughout this chapter.

On the other hand, the coherent cooperative function requires much higher data traffic, and energy efficiency must be achieved by path optimality and not algorithmic simplicity. Therefore, energy consumption on each link as well as the data traffic loading generated by each participating sensor must be explicitly

accounted for to produce a set of minimum energy paths for data transfer. For clarity of presentation, we shall separate the discussion of the coherent and non-coherent case into two chapters. For the rest of this chapter, we will focus our discussions on the non-coherent case and will defer detailed discussions on the coherent case until the next chapter.

4.1.2 Network Formation Process for Non-coherent Co-operative Functions

In general, there are three phases in the formation process of a local network:

I. Target Detection, Data Collection, Pre-Processing

II. Membership Declaration

III. Central Node Election

During phase I, a target is detected, its data collected and pre-processed. Although the USER node can override any decision made on the local level, the results of pre-processing can serve as good indicators whether a node should participate in a cooperative function. One such indicator is the Signal-to-Noise Ratio (SNR) of the sensor measurement, another would be the length of the data. Other factors such as energy reserve or the location of the sensor node can also be included in the overall considerations. In our study, we assume that SNR is used as the primary indicator because it is the only characteristic of the waveform that is modeled in our simulations and a good indicator on the usefulness of each piece of sensor data. It can serve as **a proxy for knowledge of the likelihood function for a target**. When a node decides to participate in a

cooperative function, it will enter phase II of the process and declare its intention to all neighbors. This should be done as soon as possible so that each sensor knows which neighbors are part of the local network and eliminate unnecessary signaling traffic with non-participating neighbors.

Phase III of the formation process is the CN election algorithm. A Central Node (CN) is needed to process data from different sensors. Since the CN is selected to perform more sophisticated information processing, it must have sufficient energy reserves and computational capability. It can also be selected based on its location, especially when it is centrally situated in the local network where average hop distance to each member node is minimized. If the data traffic is expected to be high, then explicit calculation of total transmission energy must precede the election algorithm so that a CN can be chosen to minimize this overall energy consumption. Fortunately, for the non-coherent cooperative function, data traffic is fairly low, therefore we make no special effort to find the optimal CN location and paths. Rather, we try to raise energy efficiency by simplifying the election process itself.

4.1.3 Overhead efficiency of distributed election algorithm

An election algorithm can be either centralized or distributed. Centralized election will take place at a single entity where all CN candidate information is gathered and compared. Then the result of the election will be broadcasted to each sensor in the local network. However, such a process cannot be energy efficient unless supported by a set of pre-existing routes connecting each sensor to the election entity. Otherwise information exchange has to rely on flooding techniques whose minimum complexity is $O(|V'|^2)$, where $|V'|$ is the number of

nodes in the local network. To develop an autonomous election algorithm that can function with as little network support as possible, distributed election is a better choice.

There are two approaches by which a distributed election can be conducted. The first one is a "virtual centralized election," where each node gathers full information about all CN candidates in the local network and then selects the CN. However, since the initial data exchange still relies on a flooding algorithm, its overhead is rather substantial. The second approach is similar to a "diffusing computation," where elections are held locally at lower overhead and their results are exchanged later on. The advantage of such an approach is that by sharing local election results, losing candidates are eliminated from contention from the very beginning of the election process, thus reducing overhead dramatically. We can illustrate this advantage with a simple analysis:

Consider a distributed election process occurring on a network of n nodes. Let $\{Q_i, i = 1, 2, \cdots, n\}$ denote the election criterion associated with each node i. We also assume that Q_i's are iid random variables, and $P(Q_i = Q_j) = 0, i \neq j$. Let us define $CN_{m_1,m_2}, m_1 \leq m_2$, so that

$$Q_{CN_{m_1,m_2}} = \max_{i=m_1,\cdots,m_2} Q_i \tag{4.1}$$

So the goal of any CN election algorithm is to find $CN_{1,n}$. Since Q_i's are iid, then we know that $\forall m_1 \leq j \leq m_2$:

$$P(CN_{m_1,m_2} = j) = \frac{1}{m_2 - m_1 + 1} \tag{4.2}$$

and the joint distribution for $CN_{1,n}$ and $CN_{1,m}$ is:

$$P(CN_{1,n=i}, CN_{1,m} = j) \quad = \quad P(CN_{1,n} = i | CN_{1,m} = j) \underbrace{P(CN_{1,m} = j)}_{\frac{1}{m}}$$

$$= \begin{cases} \frac{1}{nm} & i \neq j, i > m, j \leq m \\ 0 & i \neq j, i \leq m, j \leq m \\ \frac{1}{n} & i = j, i \leq m \\ 0 & i = j, i > m \end{cases} \tag{4.3}$$

We can compute the *entropy* of CN_{m_1,m_2}:

$$\begin{aligned} H(CN_{m,n}) &= -\sum_{i=m_1}^{m_2} P(CN_{m_1,m_2} = i) \log_2 P(CN_{m_1,m_2} = i) \\ &= -\sum_{i=m_1}^{m_2} \frac{1}{m_2 - m_1 + 1} \log_2 \frac{1}{m_2 - m_1 + 1} \\ &= \log_2(m_2 - m_1 + 1) \end{aligned} \tag{4.4}$$

and the *mutual information* between $CN_{1,n}$ and $CN_{1,m}$ is:

$$\begin{aligned} I(CN_{1,n}; CN_{m,n}) &= \sum_{i,j} P(CN_{1,n} = i, CN_{1,m} = j) \log_2 \frac{P(CN_{1,n} = i, CN_{1,m} = j)}{P(CN_{1,n} = i) P(CN_{1,m} = j)} \\ &= \sum_{i \neq j, i > m, j \leq m} \frac{1}{nm} \log_2 \frac{1/nm}{\frac{1}{n}\frac{1}{m}} + \sum_{i = j, i \leq m} \frac{1}{n} \log_2 \frac{1/n}{\frac{1}{n}\frac{1}{m}} \\ &= \frac{m}{n} \log_2 m \end{aligned} \tag{4.5}$$

Now consider a local network $G' = (V', E')$, where $V' = \{1, 2, \cdots, n\}$. Let E_{cut} be any cutset that separate G' into two connected subgraphs: $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, and without loss of generality, we assume $V_1 = \{1, 2, \cdots, m\}$ and $V_2 = \{m + 1, \cdots, n\}$. Then we have $V' = V_1 \cup V_2$ and $E' = E_1 \cup E_2 \cup E_{cut}$.(See Figure 4.2)

Equation 4.5 tells us that to lower the *uncertainty* of the election result by $\frac{m}{n} \log_2 m$ bits, the entire local network must be informed of the set $\{Q_1, Q_2, \cdots, Q_m\}$ if the "virtual centralized" approach is taken, or $\{CN_{1,m}, Q_{CN_{1,m}}\}$ if the "diffusing computation" approach is chosen. Assuming the flooding algorithm is used, and

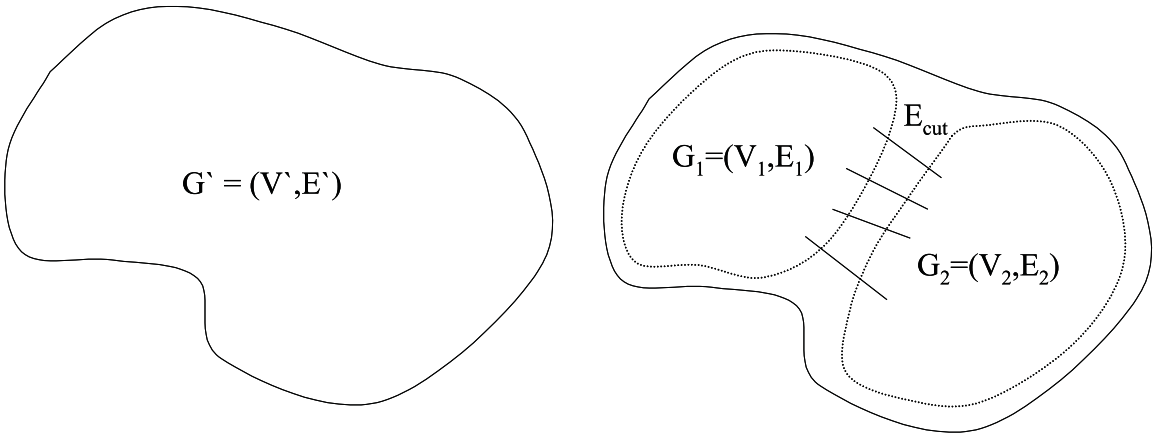

Figure 4.2: Distributed Central Node Election

the optimal lower bound is achieved, then the communication overhead required is:

$$\underbrace{|V_1|(|V'|-1)}_{\text{updating } G_1 \text{ to } G'} = m(n-1) \text{ packets} \tag{4.6}$$

for the "virtual centralized" approach, and

$$\underbrace{|V_1|(|V_1|-1)}_{\text{updating within } G_1} + \underbrace{|V_2|}_{\text{update election result to } G_2} = m(m-1)+n-m \text{ packets} \tag{4.7}$$

for the "diffusing computation" approach. We can now compute the overhead efficiency of both approaches:

$$\eta_{vc}(m,n) = \frac{\frac{m}{n}\log_2 m}{m(n-1)} \text{ bits/packet} \tag{4.8}$$

$$\eta_{dc}(m,n) = \frac{\frac{m}{n}\log_2 m}{m(m-1)+n-m} \text{ bits/packet} \tag{4.9}$$

Since overhead efficiency depends on both m, the size of the G_1, and n, the size of G'. Then we can define the optimal size $m^*(n)$ of G_1 for the diffusing computation election as:

$$\frac{\eta_{dc}(m^*(n),n)}{\eta_{vc}(m^*(n),n)} = \max_{m=1,\cdots,n} \frac{\eta_{dc}(m,n)}{\eta_{vc}(m,n)} \tag{4.10}$$

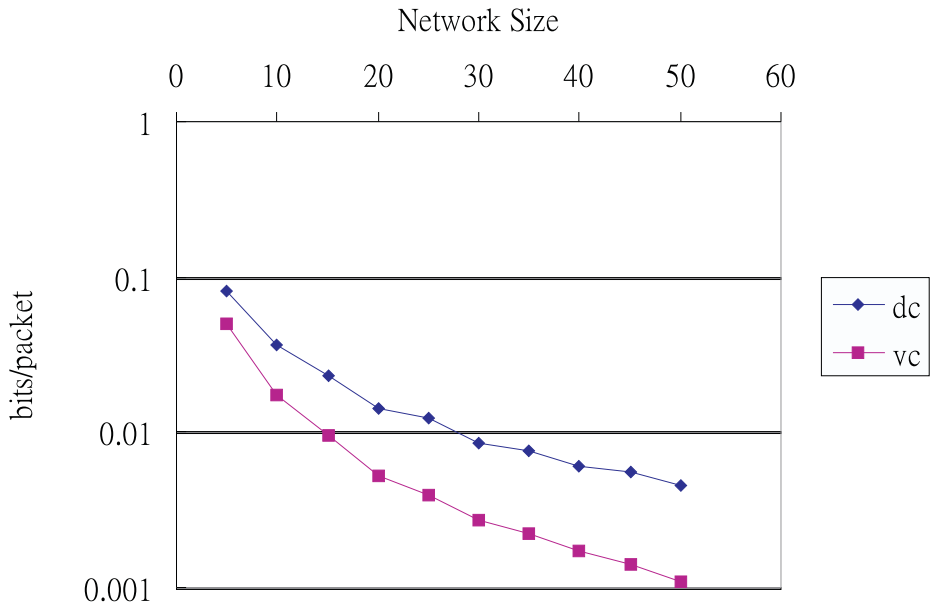

Figure 4.3: Overhead Efficiency for Distributed Election Algorithm

so that maximum gain in overhead efficiency is achieved. In Figure 4.3, we plotted $\{m^*(n), n = 5, 10, 15, \cdots, 50\}$ to compare the optimal energy efficiency achievable for these two approaches. There are several observations we can make here:

Observation 1 It is more energy efficient to have local elections and then exchange their results than having one single large scale election.

Observation 2 The smaller the size of the election, the higher the energy efficiency(bits/packet).

We can infer from observation 1 that since the division of G' into G_1 and G_2 results in higher overhead efficiency on G', then further divisions of G_1 and G_2 into

smaller sub-elections should raise the overhead efficiency on G_1 and G_2 respectively. Besides gaining efficiency by exchanging local election results, observation 2 shows us that the smaller the size of a local election, the higher the overhead efficiency for that local election. In our algorithm, we attempt to achieve the highest election efficiency by allowing local elections to take place between any two nodes, and let these election results "diffuse" throughout the network.

4.2 Central Node Election Algorithm

Since we assume that for cooperative functions, the data from several sensors are gathered at a central location for processing, the election of central node and the construction of a set of paths leading from each sensor to the CN will be the core objectives. This central node is different from the USER node in the multihop case in the sense that every sensor node has the potential to be selected as a central node. The CN election algorithm we propose has three components:

1. Single Winner Election (SWE) algorithm,
2. Spanning Tree (ST) algorithm, and
3. Termination procedure.

The first component handles the necessary signaling that facilitates the exchange of local election results throughout the network to find the CN; the second component computes a minimum hop spanning tree rooted at the CN; the termination procedure allows the elected CN to know that the network formation process is complete so it can initiate the cooperative function. Although it is rather intuitive to implement these three algorithms sequentially, by bundling election and

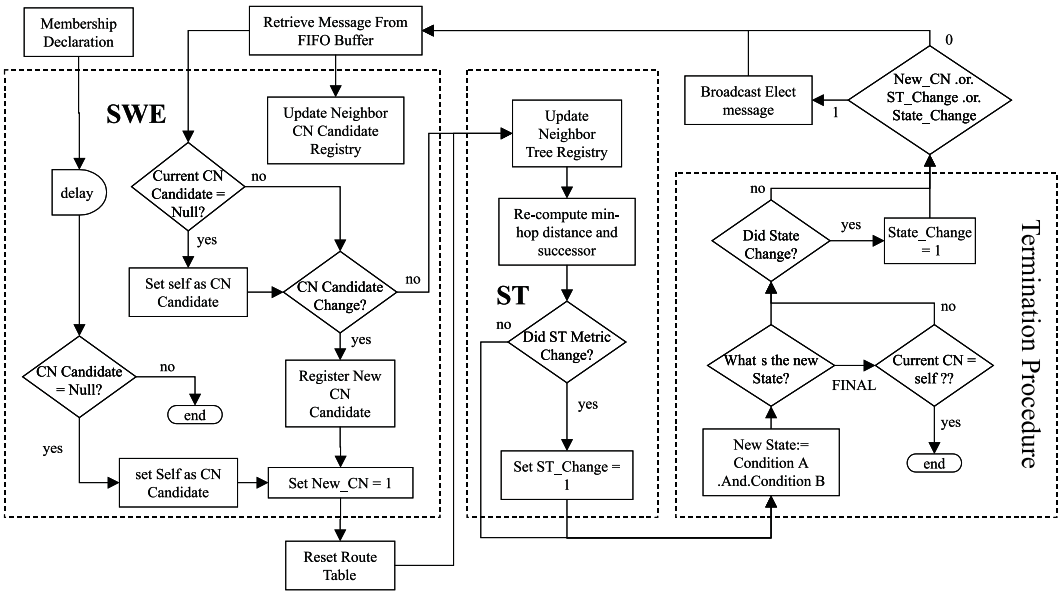

Figure 4.4: Flow Chart of the Network Formation Process

spanning tree information in a general **Elect** message, it is actually possible to combine all three component into one single algorithm. Figure 4.4 shows the flow chart of the combined algorithm. However, for presentation clarity, we will describe each component individually and omit the details of the ST algorithm since it is already well-known.

The formation process uses an **Elect** message for signaling purposes. Each **Elect** message contains the following basic information:

sender_ID - identify which neighbor sends this information.

current_CN - the CN candidate currently selected by the *sender* node.

current_CN_metric - the election metric associated with the *current_CN*.

election_state - the election algorithm can be in one of two states: FINAL or TENTATIVE, depending on how far the CN election has progressed.

min_hop - current minimum hop distance estimate produced by the ST algorithm with respect to the *current_CN*.

The routing table contains the same basic information as well as the *current_successor* in the spanning tree and a list of neighbors who are pariticipating in the cooperative function. Each **Elect** message identifies a potential CN candidate and a set of parameters that serve as the election criteria (or metric) by which candidates are compared.

In the initial stage of the SWE process, each node may impose a voluntary delay of varying length before annoucing itself as a CN candidate by broadcasting **Elect** messages. In response to the first batch of **Elect** messages, those nodes that received them will start comparing the proposed CN candidates with itself and respond with a second batch of **Elect** messages, which carries the result of this initial comparison. The second batch of message passing will spawn further exchange of messages. During this process, for each **Elect** message that presents a better candidate, the information in the message will be recorded in the registry and then be forwarded to all neighbors; otherwise the message is discarded. The continuing exchange and forwarding or discarding of **Elect** messages represents a diffusion of local election results, and they will continue until one node emerges as the winner, as its **Elect** message propagates throughout the network and eliminates all other **Elect** messages. At the same time, a minimum-hop spanning tree rooted at the winning CN will gradually increase its coverage. By the end of the SWE process, a minimum-hop spanning tree will completely cover the local

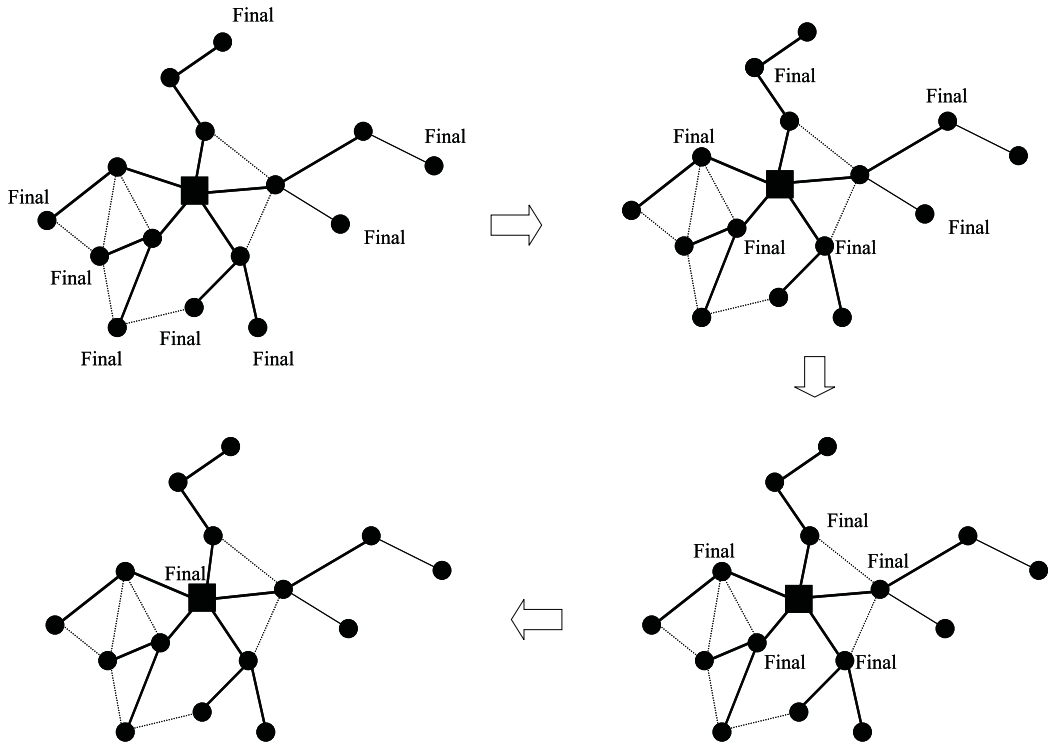

Figure 4.5: Termination Procedure

network as well.

The termination of the election process can be detected in a distributed fashion by running a simple state machine with two states: {**TENTATIVE**, **FINAL**}. The default state for each node is **TENTATIVE**, and it can change to **FINAL** when two conditions are met:

Condition A - All neighboring nodes have chosen the same CN candidate.

Condition B - Each neighbor, if it has higher hop distance from the CN candidate, is in state **FINAL** .

When a node reaches state **FINAL**, it will notify all neighbors with lower hop distances from the CN candidate by an **elect_final** message. A node in state **FINAL** can revert back to state **TENTATIVE** when either Condition A or B is violated. When an election process comes to an end, nodes at the periphery of the local network will reach state **FINAL** first, and start backpropagating **elect_final** messages toward the CN. When the CN itself receives **elect_final** from all neighbors, it will reach state **FINAL** last. Although some additional overhead and delay is required, such a termination detection procedure is far more energy efficient than polling and more reliable than a time-out scheme. It is also desirable to end the formation process at CN, because the next phase, which is the actual execution of the cooperative function, naturally must be initiated by the CN.

4.2.1 Overhead-delay trade-off

An energy-delay trade-off occurs during the CN election process. There are two factors that can affect overhead and delay: (1) the size of the local network that responds to the target and (2) the overhead efficiency of the election procedure. Larger local networks would have more latency and higher communication overhead because there are more nodes involved in the election process. However, the average per node overhead can be relatively unaffected if the election process is scalable.

Overhead efficiency has to do with the elimination of extra signaling traffic generated by nodes that will eventually lose the election, and this can be accomplished by making the process more *sequential*, thus giving better candidates a head start. As described in Section 4.2, for each node the initiation time of the

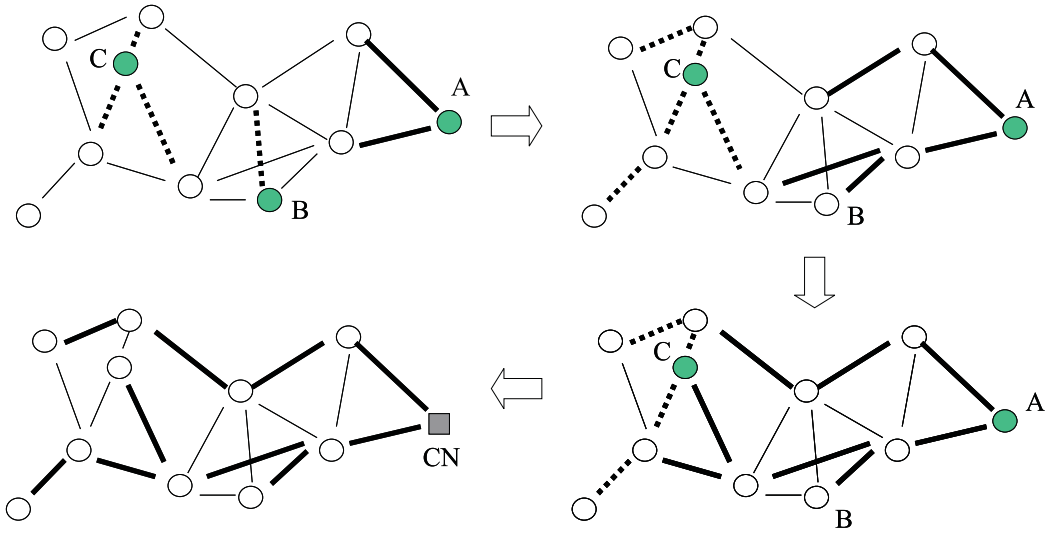

Figure 4.6: SWE without Voluntary Delay

election process is controlled by a voluntary delay timer and the reception time of the first **Elect** message. If the timer expires first, then election is triggered voluntarily; otherwise, the process is initiated by *winning the first election* triggered by the reception of the an **Elect** message. Thus, by setting longer voluntary delay on nodes less likely to win the election, it raises the liklihood that they will receive **Elect** messages generated by better candidates prior to expiration of their delay timer, lose the first election and therefore be eliminated from contention at the very beginning. This is the most efficient way to reduce signaling overhead. However, building such a delay mechanism into the protocol will increase overall network formation time as well. We can illustrate this trade-off by the example of a simple network.

Suppose a distributed election is conducted without imposing voluntary delay. Therefore, each node, after declaring membership, immediately broadcasts **Elect**

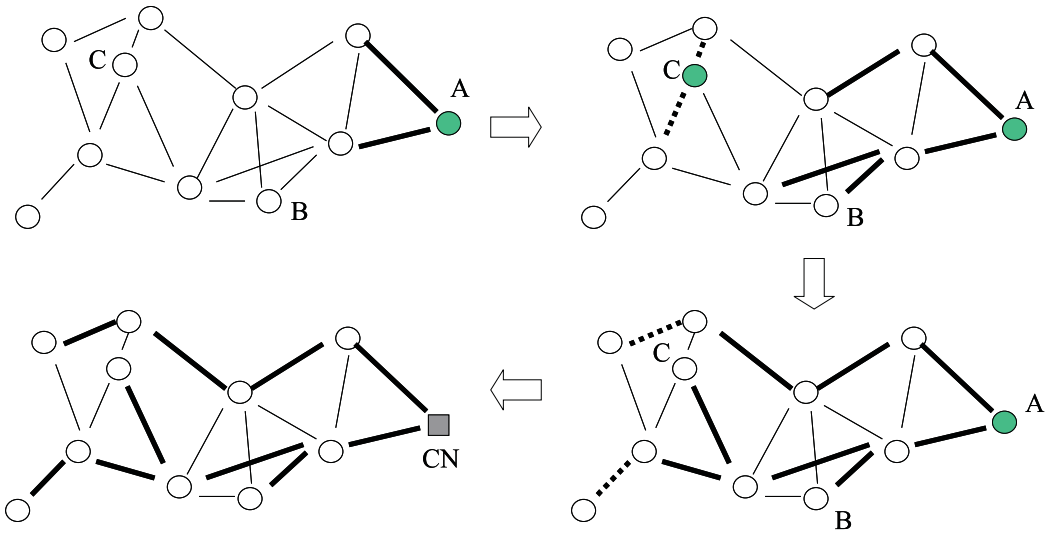

Figure 4.7: SWE with Voluntary Delay

messages to all neighbors. Suppose that node A has the highest SNR, and nodes B and C have the second and third highest SNR, respectively. After the initial SNR comparisons, all three nodes win local elections because their SNR values are locally maximum. At the same time, each node starts to built a minimum hop spanning tree as well. However, as **Elect** messages continue to propagate, further SNR comparison will tear down the spanning trees built by node B and C. Eventually, node A and its spanning tree will dominate the local network as illustrated by figure 4.6. The formation of spanning trees for node B and C can be consider a "waste" of energy because they are temporary. To reduce overhead, we can suppress the election process at node B and node C by taking voluntary time delay. As illustrated in figure 4.7, the election process in node A, after being given a head start, can quickly dominate the network before node B and C have the chance to build their own spanning trees. Ideally, if the **Elect** message of

node A can arrive at B and C before their delay timers expire, then all overhead will be eliminated from the election process.

However, each node must operate without global SNR information in a distributed system, so it cannot rule out the possibility of winning or losing the election. Therefore each node will impose a non-zero but finite voluntary delay that is inversely related to its own SNR level, and an *initial delay*, the period during which no election activites takes place in the local network, is unavoidable. Figure 4.8 shows a time diagram of the network formation process. The overhead efficiency of our algorithm comes from the *delay differential* that exists among the competing candidates as the previous example has illustrated, and the larger the *delay differential*, the lower the overhead. Although the *delay differential* does not contribute to the latency of the formation process, it will indirectly lengthen or shorten the *initial delay*. To see this, we can consider a delay function of the following form:

$$D(s) = \frac{D_o}{s}, s > s_{\min}$$

where $s_{\min}$ is the minimum SNR value required to participate in the cooperative function. If the two best CN candidates have SNR separation $\Delta s = s_1 - s_2$, then the *initial delay*:

$$D_{init} = D(s_1) = \frac{D_o}{s_1}$$

and the *delay differential* is:

$$\Delta D = D(s_2) - D(s_1) = \frac{D_o}{s_2} - \frac{D_o}{s_1} = \frac{D_o \Delta s}{s_1 s_2} = D_{init} \frac{\Delta s}{s_2}$$

Therefore we see that given no changes occur to SNR, the *initial delay* is directly proportional to the *delay differential*.

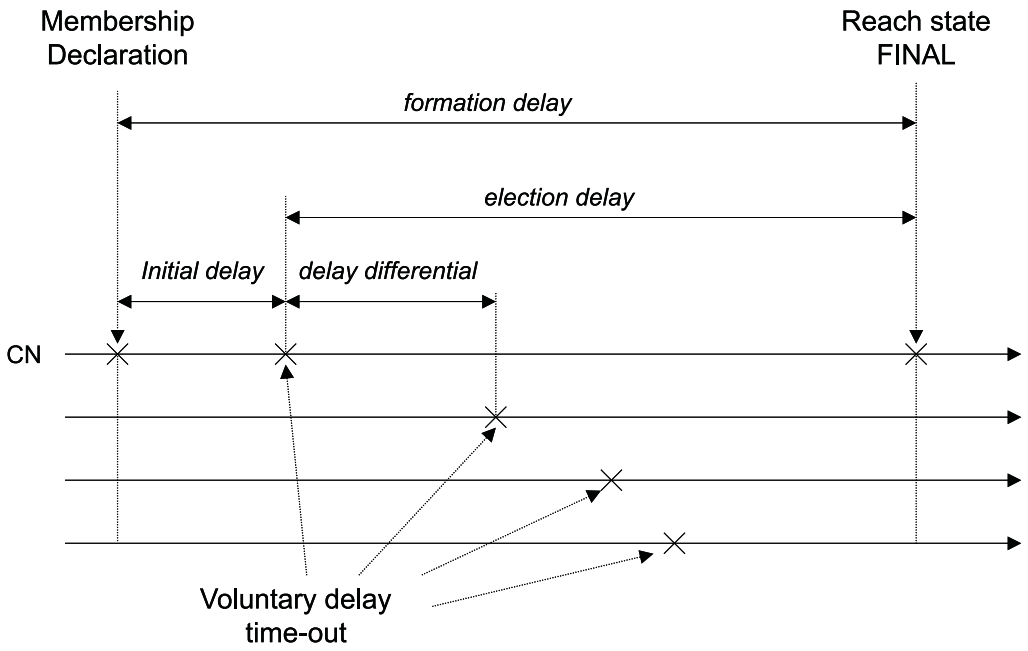

Figure 4.8: Time Diagram of Formation Process

The size of the local network can affect the overhead-delay trade-off in different ways when the target is either NF or FF. If a NF target has strong signal strength, the local network will naturally be larger. However, higher SNR nodes still tend to cluster around the target because the SNR is highly dependent on the line-of-sight component, which means the *delay differential* required to eliminate competing CN candidates remains small, and the *initial delay* should also remain small. We can expect some overhead increase simply because there are more nodes on the periphery. However, for a strong FF target, as the local network becomes larger, higher SNR nodes will tend to be more spread out. In such a case, longer *delay differential*, therefore longer *formation delay*, would be necessary to ensure overhead efficiency.

4.2.2 Proof of Correctness

In this section, we would like to present a proof of correctness for our algorithm by showing a CN and a spanning tree will be built in finite time, and that the termination procedure will properly conclude the formation process with the CN reaching state FINAL last.

Definitions and Assumptions

1. No topological change occurs during the formation process.
2. Transmission and processing delay per hop is fixed at Δt.
3. Let the *successor* of j with respect to i be denoted as s_i^j; let the k-hop *successor* of j be denoted as $s_i^{j,k}$.
4. Let $d_{j,i}$ be the hop distance from j to i.
5. A *non-directed* or ND-Path $P_{j_1,j_n} = \{j_1, j_2, j_3, \cdots, j_n\}$ is one where the hop distance is non-increasing: $\forall\ n \geq k > l \geq 1$ such that $d_{j_k,j_n} \leq d_{j_l,j_n}$ and acyclic.
6. A *directed* or D-Path $P_{j,i}$ is defined by *predecessor-successor* assignment in the routing table $\{j, s_i^j, s_i^{j,2}, s_i^{j,3}, \cdots, s_i^{j,d_{j,i}} = i\}$. Therefore, we see that $|P_{j,i}| = d_{j,i}$.
7. A *hybrid* path or H-Path consists of two parts:

   $$P_{j,i} = \underbrace{\{j, \cdots, j'\}}_{\text{ND-Path outside tree } i} \cup \underbrace{P_{j',i}}_{\text{D-Path inside tree } i}$$

8. A node enters state FINAL_i means it enters state FINAL with node i as its CN candidate.

Lemmas and Theorems

Lemma 4.1 *$\forall j, i, d_{j,i}(t)$ is non-increasing if i is the winning CN.*

Proof: Since $d_{j,i}(0) = \infty$ by default, it is sufficient to show that if $d_{j,i}(t) = m < \infty$, then $d_{j,i}$ is non-increasing after time t. We know that node j must have received an **Elect** message at some time prior to t from its current successor. This message must have traveled through $m-1$ hops to reach node j. Let time $t_1 < t$ be the latest instant prior to t that node j received an **Elect** message with *min_hop* $= m-1$ from its current successor $s^j(t)$. Since we assume the topology does not change throughout the network formation process, then $d_{j,i}(t)$ can increase only if node $s^j(t)$ reports a hop distance increase. This is clear because if node $s^j(t)$ remains at $m-1$ hops, then the ST algorithm will not update $d_{j,i}$ to a higher value above m.

Let $j_1 := s^j(t)$, then we note that $d_{j_1,i}(t_1) = m-1 < \infty$. Applying the same arguement, we see that there also exists neighbor $s^{j_1}(t_2)$,$t_2 < t_1$, with hop distance $m-2$ at time $t_2 < t_1$, where t_2 is selected in the same fashion as t_1. It is also clear that $d_{j_1,i}(t_1)$ can increase only if neighbor $s^{j_1}(t_1)$ reports an increase. Continuing this argument, eventually we have to conclude for $j_{m-1} = s^{j_{m-2}}(t_{m-2})$ that i has to report an hop distance increase to node j_{m-1} at some time $t_{m-1} < t_{m-2} < \ldots < t$. This is clearly a contradiction because node i is the root. This proves Lemma 4.1.

Lemma 4.2 *$\forall j$, if i is the winning CN and i selected j as CN candidate at time t_o, then a D-Path $P_{j,i}(t)$ exists for all $t > t_o$.*

Proof: Lemma 4.1 shows that $d_{j,i}(t)$ is non-increasing, and therefore satisfies the *Distance Increase Condition(DIC)* [34] which is sufficient to guarantee loop

freedom at all times for all nodes that selected i as the CN candidate. Therefore the successor assignment for j must reach node i in finite hops, which represents a valid D-Path.

Lemma 4.3 *Assume prior to time t_o, a H-Path $P_{j,i}(t_o)$ exists and all node in this path are in state TENTATIVE. If some changes occur on this path at time t_o, either some node enters FINAL_i or some node changes hop distances, then there exists a time $t_1 > t_o$ such that if j remains in TENTATIVE until t_1, then:*

I - *A new H-Path $P_{j,i}(t_1)$ exists such that all nodes are TENTATIVE, and*

II - *i remains in TENTATIVE and $P_{j,i}(t)$ exists $\forall t \in [t_o, t_1]$.*

Proof: As stated by the Lemma, there two possible changes that can occur to $P_{j,i}(t_o)$:

1. Some node, say n_k enters state FINAL_i at time t_o. Then it is clear that its predecessor n_{k+1}, since it did not enter FINAL_i, must have decreased its hop distance to k prior to time t_o so that condition B in the termination procedure can be satisfied. However, this leads to a contradiction because we assume at time t_o, node n_{k+1} is the predecessor of n_k. Therefore, such a change is impossible.

2. Some node, say n_{k+1} changes its hop distance. Lemma 4.1 tells us that the distance must be decreasing. Therefore $d_{n_{k+1},i}(t_o) = k$. By the time $t_o + \Delta t$, its previous successor n_k will be notified of this change. Since Lemma 4.2 guarantees that an acyclic D-Path exists at all time for any node that records finite hop distance to root i, then a new H-Path $P_{j,i}(t_o) =$

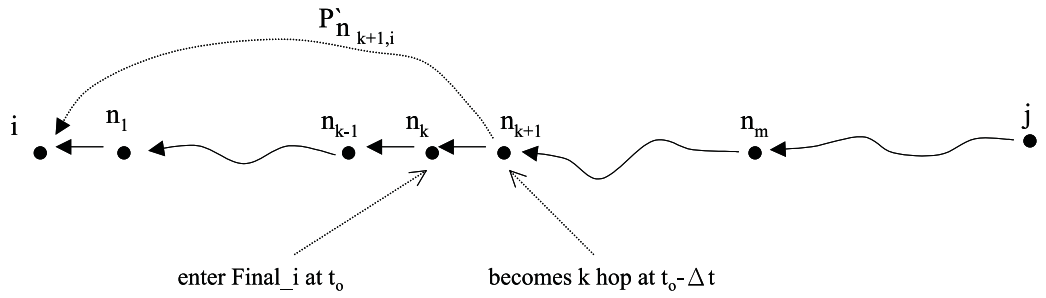

Figure 4.9: H-Path in Lemma 4.3

$P_{j,n_{k+1}} \cup P'_{n_{k+1},i}(t_o)$ is created. (See Figure 4.9) There are two cases that we need to consider:

(a) If no state change occurs to n_k, then the state of i is unaffected by this change, and at the latest, by time $t_o + k\Delta t$, $P'_{n_{k+1},i}$ can be guaranteed to be in state TENTATIVE.

(b) If n_k enters FINAL_i because condition B of the termination procedure is now valid, then it is *possible* for nodes $\{n_1, n_2, \cdots, n_{k-1}\}$ to enter FINAL_i in succession. Node n_1 can enter FINAL_i at time $t + k\Delta t$ at the earliest, and node i will be notified of n_1's state change at time $t + (k+1)\Delta t$. But node i will remains in state TENTATIVE because $P'_{n_{k+1},i}$ is already in state TENTATIVE at time $t_o + k\Delta t$.

Letting $t_1 = t_o + k\Delta t$, then we can see that both I and II are true. This proves Lemma 4.3.

Lemma 4.4 *Assuming node i will be the winning CN, it cannot reach state FINAL_i before all nodes select it as the CN.*

Proof: Suppose that prior to time t', there exists a node j that remains in state TENTATIVE. If we look at the state of the network at the beginning, at time $t = 0$, $\exists$ H-Path $P_{j,i}(0)$ such that all nodes on it are in state TENTATIVE. By Lemma 4.3, we see that if a change occurs on this path at later time t_o, then $\exists$ time t_1 such that node i remains in TENTATIVE during $[t_o, t_1]$, and $\exists P_{j,i}(t_o)$ in state TENTATIVE as well. We can see that during $[0, t_o) \cup [t_o, t_1]$, i remain in TENTATIVE. If changes occurs to the $P_{j,i}(t_1)$ at time t_2, then Lemma 4.3 tells us that there exists a time t_3 so that node i remains in TENTATIVE. Using the same argument repeatedly, we see that node i will remain in TENTATIVE in $[0, t_m] = [0, t_o] \cup [t_o, t_1] \cup \cdots \cup [t_{m-1}, t_m]$ as long as $t_m < t'$.

Let m^* be such that $t_{m^*+1} \geq t' > t_{m^*}$, and note that the times denoted by $\{0, t_o, t_1, t_2, \cdots, t_m < t'\}$ correspond to the times either some change occurs to $P_{j,i}$, or the time when all nodes in $P_{j,i}$ have returned to state TENTATIVE. However, it is still unclear what happens during $[t_{m^*}, t']$ if node j changes state at time t'. There are two possibilities:

$P_{j,i}(t')$ **in state TENTATIVE** - The back propagation of FINAL messages due to j's entrance to state FINAL_i will reach i at $t' + d(j,i)\Delta t$, at the earliest.

$P_{j,i}(t')$ **not in state TENTATIVE** - This implies at time t_{m^*} a change has occured but TENTATIVE state has not been fully restored to $P_{j,i}$. However, by examining the argument in the proof of Lemma 4.3, we see that node i will remain in TENTATIVE as long as the back-propagation of **elect_final** messages caused by node j's entance to state FINAL_i did not reach node i. We see that, if the **elect_final** message did reach node i, the earliest time node i can enter FINAL_i is time $t' + d(j,i)\Delta t > t'$.

We can see that in both cases, node i cannot enter FINAL_i before t'. This proves Lemma 4.4, and established an very important fact:

> For a CN to reach state FINAL, it must be preceded by the completion of SWE and ST algorithm.

Lemma 4.5 *ST algorithm will converge in finite time after the completion of SWE algorithm.*

Proof: Note that after the completion of SWE, the formation process is completely controlled by the ST algorithm, whose convergence can be demonstrated by the following argument. Let node i be the winning CN and t_c be the completion time of the SWE algorithm. Then Lemma 4.1 shows us that $d_{j,i}(t)$ is non-increasing for all j and $t \geq t_c$. If we examing the ST algorithm, we see that each hop distance decrease must either:

1. cause its *predeccessor* to decrease hop distance after Δt time units, or
2. cause no further changes.

Therefore, we can conclude that as long as the ST algorithm did not converge, we can observe at least one instance of hop-distance decrease within any window of observation of length Δt. This being so, since we know that after t_c there are only a finite number of nodes each with finite hop-distance value to the CN, then this chain of hop-distance decreases cannot persist longer than $\Delta t \sum_j d_{j,i}(t_c)$ after SWE is completed. So we see that the ST algorithm must stop in finite time.

Theorem 4.1 *If node i is the winning CN, then after the formation process begins, (1) SWE and ST algorithm will converge in finite time, and (2) node i will reach state FINAL in finite time after the completion of SWE algorithm.*

Proof: Let d be the diameter of the network, and assume that at time $t = 0$, some node k in the network initiated the formation process by declaring itself as a CN candidate. There are two possibilities:

1. If $i = k$, then the propagation of **Elect** messages for i will spread throughout the network before $d\Delta t$

2. If $i \neq k$, then node i will declare itself as a candidate either on its own or after receiving **Elect** messages from its neighbors. One of these will happen at or before time $d_{k,i}\Delta t$. Then since i will be the winning CN, its **Elect** message will propagate throughout the network before time $(d_{k,i} + d)\Delta t$.

Since $\forall k, d \geq \max_k d_{k,i}$, then we know the SWE algorithm will be completed at time $t_c \leq 2d\Delta t$. By Lemma 4.5, we know ST algorithm will terminate in finite time after SWE algorithm stops. This proves (1).

Lemma 4.4 guarantees that node i cannot reach FINAL before SWE completion. Therefore it is clear that if it does reach state FINAL, that must take place *after* SWE completion. To prove (2), we only have to produce an upperbound on the time required to reach state FINAL, since we know the ST algorithm will converge in finite time after SWE completion. Then we know that after ST convergence, the termination condition B and condition A are both satisfied for all nodes who have no higher hop-distance neighbors. The maximum hop-distance for these nodes, say $d^* \leq d$ is finite. When they reach FINAL, all nodes at hop-distance $d^* - 1$ will also reach FINAL after Δt time units. We can see that after $(d^* - 1)\Delta t$ time units, all 1-hop neighbors of CN will be at FINAL. Therefore at the latest, after $d^*\Delta t < d\Delta t$ time units, node i will reach FINAL. This proves (2).

4.2.3 Remark on the proof

At this point, I would like to make a few remarks regarding the above proof. We have proven that the termination procedure works properly for CN, i.e., that after election is initiated, the SWE and ST algorithm will converge first, then CN will reach FINAL in finite time. However, for any node, say node j, that is not the eventual CN, it has not been theoretically demonstrated that it will *never* reach FINAL_j. The difficulty we encountered is in Lemma 4.3, where node i is *required to be the CN selected by SWE*. If this requirement can be lifted, then Lemma 4.3 can guarantee that any node j that does not win the SWE can never reach FINAL_j. However, we know that it is the periphery of the network that will enter state FINAL first, due to condition B of the termination procedure. It is highly unlikely that any node j can reach FINAL_j without winning the election at the periphery of the network, which is usually quickly controlled by the winning CN who has a head start. Even if there is a temporary domination, it is highly unlikely that condition A or B are both satisfied everywhere in the network at some time instant. These intuitions are confirmed by simulation experiments where not one single exception is observed over tens of thousands of trials.

4.2.4 Failure Recovery

In the event that some link failure occurs, then it is possible that some nodes may not reach state FINAL because either condition A, condition B or both (see section 4.2) are violated. This creates a deadlock on the whole termination procedure. A simple fix can be implemented by requiring the MAC protocol to

report connection failures. Knowing such a failure occurs, condition A and/or B can be waived for those neighbors with whom connection has broken down. The result is that a portion of the local network, those nodes severed from the CN due to link failures, will not participate in the cooperative function.

If the MAC level protocol cannot provide a quick failure report, then a timer can be used to allow each node to return to normal, low-duty cycle operation, while still retaining the sensor data and routing information if memory allows. At the CN, when this timer expires, it can send a special message to trigger a re-declaration of membership. At this time, topological information of the local network will be corrected, and SWE can restart. However, one must consider the necessicity of such complex procedures according to the significance one attaches to the target. It might be satisfactory to simply reset the network after timer expiration and hope to detect the same target again, which is very likely the case if the target remains active within detection range.

4.3 Simulation Results

To illustrate the energy-delay trade-off and its dependence on the size of the local network, simulations were conducted using both NF and FF targets. The election criterion chosen is the SNR of the target signal, and the voluntary delay at each node is calculated by a simple inverse relation: $D \propto \frac{D_0}{SNR}$ time units, which allows high SNR nodes to start the election earlier than those with lower SNR. For NF targets, the target position is random, and the received SNR is modeled by an inverse square law on the distance between the target and each sensor; for a FF target, a Raleigh random variable is used to capture the effect

of multipath on signal strength. The average transmission delay, including frame latency and queuing is assumed to be 1 time unit. The non-integer network size is the result of averaging over many different scenarios.

Figure 4.10 shows the overhead-delay trade-offs. As expected, overhead decreases with longer delay for both FF and NF targets and reached a minimum quickly. The interesting thing to note is that while in the NF case, the most modest amount of delay ($\approx$ 20 units) seems to be sufficient to minimize overhead regardless of network size, the FF case would required much more delay(40 units or higher) especially for larger networks(size = 99). This confirms our speculation that for FF targets, the dominant CN candidates tend to be separated over longer hop-distances, and therefore longer initial delay is required for energy efficiency. In the NF cases, since deviations of SNR are predominantly controlled by physical distance, the dominant CN candidates tend to cluster together, regardless of the absolute signal strength of the NF target.

For NF targets, as long as some voluntary delay is used($D_o > 0$), for a 227% increase in network size(from 6.44 to 21.12), the overhead increase is at about 12%. While without imposing voluntary delay($D_o = 0$), the overhead increase is nearly 80%. For a FF target, when no delay is used, overhead increase is rougly 93% for a 1500% increase in network size, while when $D_o = 500$, the overhead is nearly unchanged. The simulation seems to suggest that our algorithm is acutally more efficient for FF targets, but that is not really the case. The seemingly lower overhead is caused by differences in SNR modeling, and the delay coefficient used.

The most interesting behavioral differences to note is that for a NF target, the highest overhead efficiency is achieved at low latency regardless of network size. For FF targets, overhead efficiency is earned gradually; the more delay that

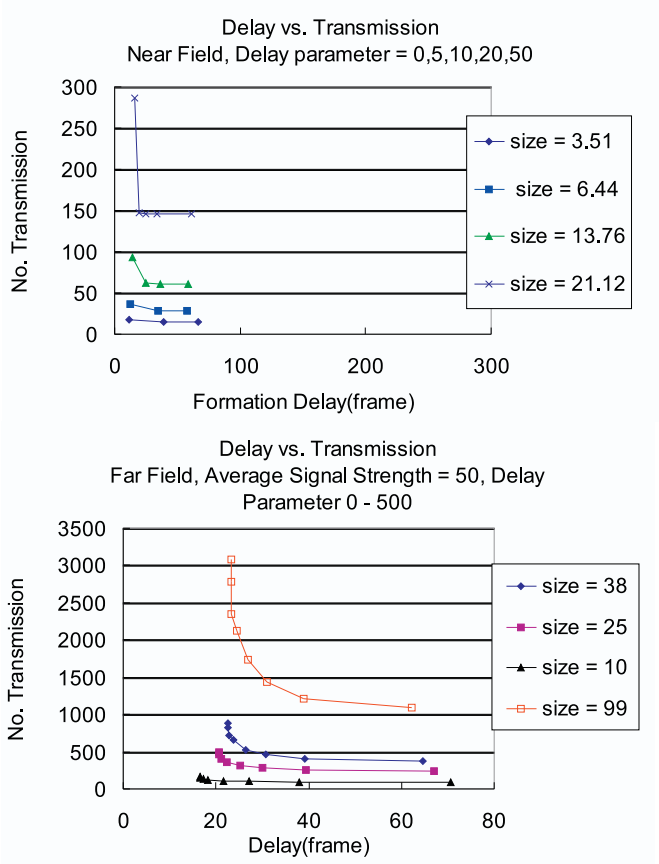

Figure 4.10: Overhead and Formation Delay

is tolerated, the higher the efficiency. Again the reason for this difference is that the dominant competing CN candidates are located very close to each other in the NF case regardless of network size, while for a FF target, the multi-path effect caused the dominant CN candidates to grow farther apart as the network size increases. In general, Figure 4.11 shows the per node overhead is concave, and for larger delay becomes nearly flat. This indicates that the algorithm is quite scalable, especially in NF cases.

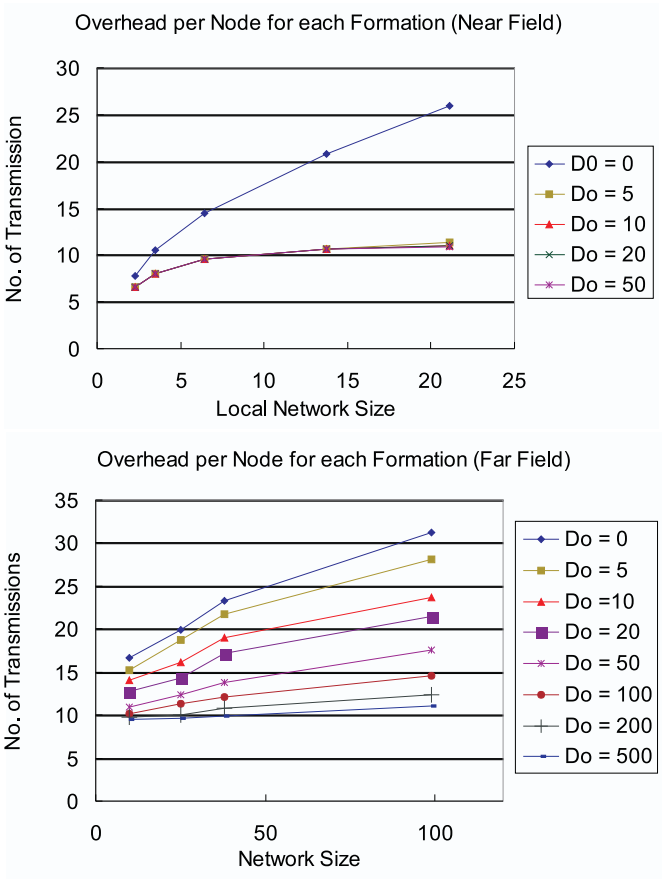

Figure 4.11: Overhead and Network Size

We also examined the relationship between network delay and size. Figure 4.12 shows that delay can both increase or decrease with network size. In the NF case, only when no voluntary delay is used will the delay increase with network size, otherwise, larger network actually seem to produce lower delay. In both the NF and the FF cases, the *election delay* will be roughly proportional to the round trip transmission delay from CN to the periphery of the network. When the network size increases, *election delay* will increase but the *initial delay* will actually decrease because the maximum SNR is higher when there are

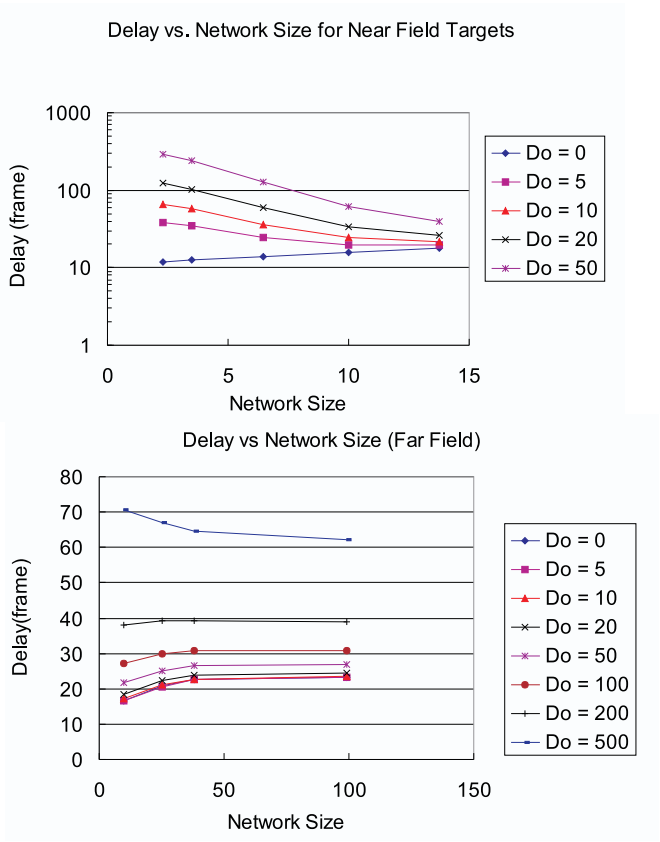

Figure 4.12: Formation Delay and Network Size

more nodes. Therefore when the *initial delay* component dominates the *formation delay*, the total delay will become inversely related to network size despite any increase in *election delay*.

4.4 Summary

The simulation results basically confirm the overhead-delay trade-off that we expect to see, and we can see that the fundamental performance differences between NF and FF targets have their origin in the separation distances among the dominant CN candidates. Overhead and delay trade-off will present the network USER the choice between latency and energy efficiency. The degree to which one favors latency or efficiency will depend on the requirement of the applications.

Chapter 5

Extension of Adaptive Local Routing Algorithm for Coherent Cooperative Functions

In this chapter, we describe an extension of the network formation algorithm presented in Chapter 4 that is suitable for coherent cooperative signal processing. While the underlying challenge remains that of energy efficiency, we make note of the fundamental differences between the coherent and non-coherent cases and provide details on the modifications and additional complexity necessary for energy efficient operation.

5.1 Key Differences from the Non-Coherent Case

Cooperative signal processing can be viewed as a form of hierarchical information processing where raw sensor data is first collected and processed by individual

sensors, and then a parametric or compressed representation of the original data is gathered for a second round of processing at a central node (CN).

The advantage of hierarchical processing is that each level of processing is a "filter" that removes information irrelevent to the next level of processing. The overhead for information exchange is therefore minimized. The distinction between coherent and non-coherent processing lies in the degree to which "temporal" information is removed from the data waveform. One example would be the application of appropriate "compressive transformations" on the raw sensor data to produce a likelihood function that can be used for target detection purposes.

However, subtle features that are hidden in the high order statistics of the data set can be lost if too much pre-processing is done at the node level. In order to extract these subtle features that are necessary for the highest level of reliability in applications such as target classification, preservation of the sensor data in its raw form is required. This shifts the burden of processing from the individual sensors back to the central node, and the entire process becomes more "centralized." Being more "centralized," coherent signal processing requires higher communication overhead. The energy expenditure for uploading raw data from each sensor to the central node will be significantly higher than that needed for network formation. The focal point of attention for protocol design now shifts away from algorithmic simplicity to the minimization of energy consumption *during* the execution phase of the cooperative function. Our procedure thus applies to any local cooperative function for which traffic loading is dominated by data transfer rather than overhead in network formation.

5.2 Network Formation Process

We now consider computation of the minimum energy required for the data transfer in coherent cooperation. Let S denote the set of source nodes (SN), which will eventually provide the raw-data used in cooperative computation. For each node $i \in S$, there are ρ_i packets of data available for uploading to the CN. Let $P_{i,j}$ be the set of paths from node i to j and l_k's denote the energy consumption across one single link. Then the minimum energy path $p^*_{i,j} \in P_{i,j}$ is the one such that:

$$\sum_{l_k \in p^*_{i,j}} l_k = \min_{p \in P_{i,j}} \sum_{l_k \in p} l_k = m^*_{i,j} \tag{5.1}$$

Here $m^*_{i,j}$ represents the minimum energy required to send one packet from node i to node j. The energy required to upload data from all SNs to node j will be:

$$E_j = \sum_{i \in S} \rho_i m^*_{i,j} \tag{5.2}$$

The best CN is the j that will minimize Eq 5.2. Therefore the minimum energy will be:

$$E^* = E_{CN} = \min_j E_j = \min_j \sum_{i \in S} \rho_i m^*_{i,j} \tag{5.3}$$

The network formation process will have to compute the quantities in Eq 5.1, 5.2, and 5.3 in order to optimize energy efficiency of the cooperative function.

Figure 5.1 shows the general network formation process. The first graph shows the topology of the local network after target detection and membership declaration. Then through a multi-winner election (MWE) procedure, a subset of nodes are elected as source nodes (SN), based on some criteria internal to the cooperative function. During this process, the loading of each SN, $\{\rho_i, i \in S\}$, is also broadcasted to the network. By piggybacking transmission power on each

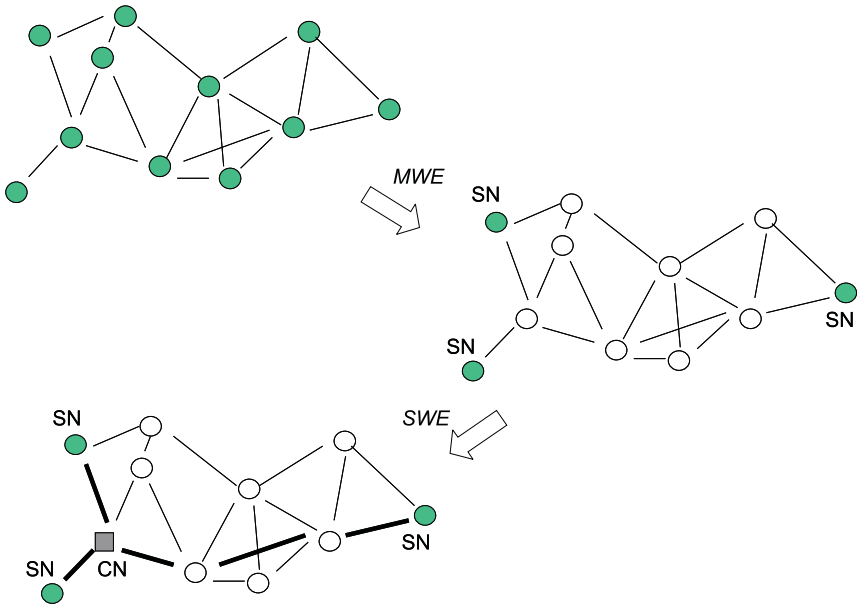

Figure 5.1: Network Formation for Coherent Cooperative Function

link in the *Elect* messages, each node j can compute $\{m^*_{i,j} \forall i \in S\}$, and E_j at the end of the process. Finally, a SWE election process will select a node that optimizes E_j as CN and generate the set of optimal paths that will achieve the minimum energy E^*.

5.2.1 Multi-Winner Election & CN Election

The Multi-Winner Election (MWE) is basically an extension of the SWE. It is designed to select a subset of nodes from the network based on some criteria. It helps to limit the number of participating nodes and the traffic loading, especially when a target with strong signal strength is present. The basic approach in MWE is the same as the SWE with the following differences:

Keeping multiple candidates in registry - The SWE algorithm, as it filters through each **Elect** message it receives, only keeps the record of the best candidate; the MWE algorithm will attempt to keep record for as many as $n(>1)$ candidates, where n is the the maximum number of SNs allowed.

Minimum spanning tree uses reverse link power metric - In the coherent case, a simple minimum hop spanning tree is built centered at the CN. During the SN election, the calculation of $\{m^*_{i,j}, p^*_{i,j} \forall i \in S, j \in V\}$ are required, therefore the spanning tree calculation uses a power metric rather than hop distance. In CN election, the spanning tree is created for traffic that flows *toward* CN; in SN election, the spanning tree is created for traffic that flows *away* from SN. So the link power metrics used are those in the *reverse* direction.

No termination procedure - The end of MWE process is detected by measuring the frequency of **Elect** message propagations. The termination criterion is simple: a node will determine that the MWE process has ended globally when its *waiting_time*, which is the idle period elapsed from the last reception of an **Elect** message, exceeds a certain length. There are two reasons why a termination procedure similar to that described in Section 4.2 is not used here:

1. Complexity - Each winning SN, after reaching state FINAL, cannot determine whether other SNs have reached state FINAL as well. Additional coordination is required so that each SN can be fully informed of the state of other SNs.

2. Transiton from SN to CN election - When a CN election ends, the next phase of operation is the execution of the cooperative function which is a centralized operation. In this case, it makes sense that the CN knows the end of the election process with certainty, so that it can assume control at the proper time. The termination procedure described in Section 4.2 serves as a transition mechanism from a **distributed process** to a **centralized process.** However to make the transition from SN to CN elections - from a **distributed process** to another **distributed process** - such a termination procedure is not suitable.

The MWE is a distributed computational process, and its efficiency can likewise be improved by "sequentializing" the process with voluntary delay. The computational complexity is approximately n times that of the SWE process if n is small compared to local network size. At the end of the MWE process, each node j in the network will have the following information:

- S - the set of SNs selected.
- E_j - the minimum energy required to download data from all SNs to itself.
- $\{p^*_{i,j}, i \in S\}$ - the set of minimum energy paths.

CN election

The CN election differs from that in the non-coherent case in two ways:

1. Finding the *minimum* rather than the maximum of an election criterion.
2. Not requiring spanning tree computation since the optimal paths are already found by MWE.

These differences do not require additional complexity or change to the SWE algorithm. Election criteria can be mapped by an inverse relationship, turning the minimum to the maximum, and in fact, having the optimal path already computed, SWE can even be simplified by removing the ST component. (See the flowchart in Figure 4.4) Also note that in CN election, the same termination procedure as for the non-coherent case will be used.

5.2.2 Latency & Convergence

Latency in the coherent network formation process has more components than the non-coherent case. In Chapter 4, we showed that algorithmic efficiency is achieved by imposing a delay difference between the winning and losing candidates. Under the SNR criterion, the *initial delay* is directly related to the hop distance between contending candidates, which, due to different propagation modes, is larger for FF targets. If the SNs in the coherent case are also elected based on SNR, then they will have the same spatial separation as the better CN candidates in the non-coherent case. To minimize energy consumption, we can expect to find the best CN candidates inside the convex hull of the SNs. Therefore we can also say that *initial delay* required during the CN and SN election will not be significantly higher in the coherent case.

However, the most significant source of delay comes from the termination procedure of the SN election. Since SN election is terminated individually without considering the state of neighbors, there can be large differences between termination times. Theoretically, the discrepency between termination times00 should be bounded by the *maximum one-trip delay*, which is the time it takes for **Elect** message to diffuse across the network. However, termination delay is also

influenced by the *waiting_time* duration. Longer *waiting_time* prevents premature termination but causes longer delay; shorter *waiting_time* raises sensitivity so that the end of SN election can be detected quickly but such decisions are likely to be premature in large networks. Lacking global knowledge on the size of the local network, a longer *waiting_time* duration should be used to ensure the correct execution of the algorithm in most situations. However, once fixed at a sufficiently large value, the *waiting_time* becomes a constant component in the overall delay because it is independent of other scenario parameters. Therefore it will not interfere with the delay-size-energy dynamics in the formation process.

Another issue is the convergence of the spanning tree computation. While we have proved the convergence of the ST algorithm when calculating minimum hop routes, the real necessary condition for convergence is Lemma 4.1 in Chapter 4. There we see that *as long as the link power metric is unchanged during the network formation process*, then the argument in the proof remains valid. Since link power metric, as well as other routing parameters, are updated only at the beginning of a *computational epoch*, we are guaranteed that spanning tree calculation during local network formation will converge.

5.3 Simulation Results

Although algorithmic efficiency is no longer the primary design objective, the path optimality has already been shown by the fact that the ST algorithm and CN election converge. Therefore, our simulation effort is still focused on understanding the efficiency and delay performance of the algorithm. We choose the number of SN to be less than or equal to 5 for all target types and local network

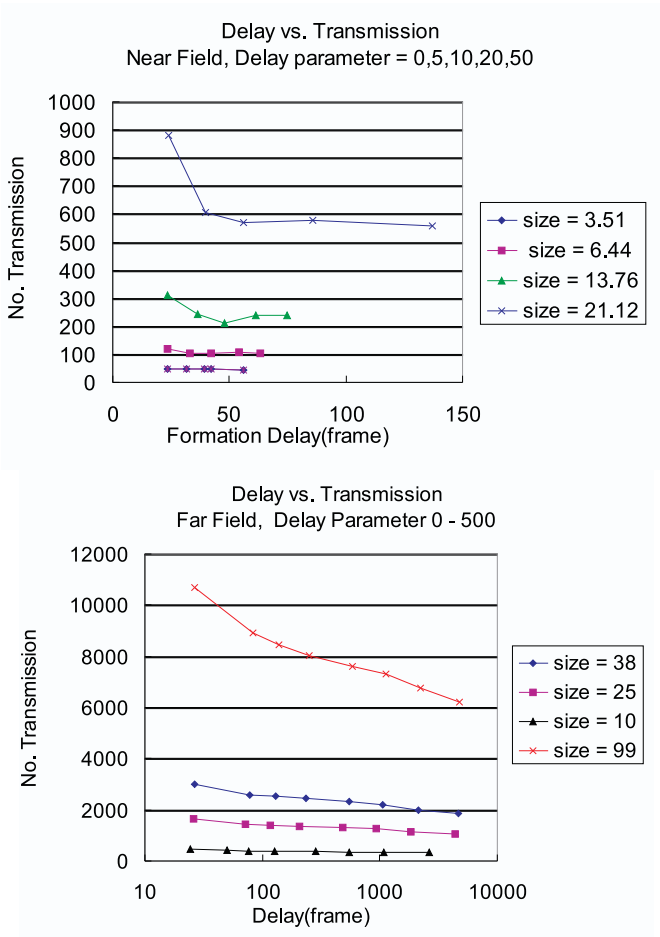

Figure 5.2: Overhead and Formation Delay

sizes, and assume that SNR is the SN election criterion. The data length is assumed to be proportional to the SNR at each SN. We used a fixed *waiting_time* for all simulation runs.

From figure 5.2, we see the delay-overhead trade-off is similar to the non-coherent case. The NF case has the same behavior, yielding good overhead for a relatively small sacrifice in delay. For a network with 21 nodes, the overhead is reduced by roughly 33.3%(from about 900 down to 600 packets). In the

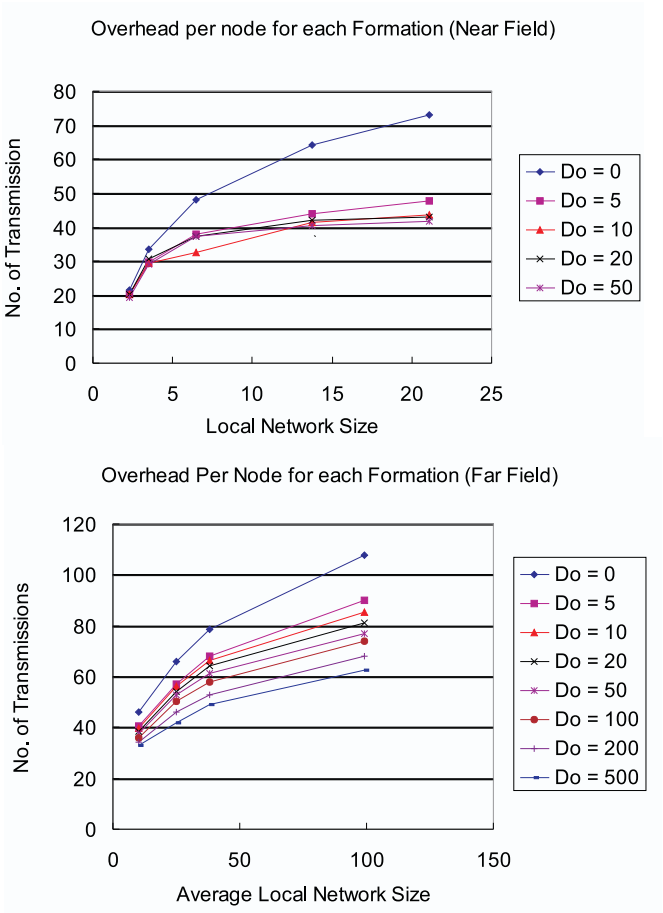

Figure 5.3: Overhead and Network Size

non-coherent case, the reduction is larger: 50%(from 290 down to 145 packets). Figure 5.3 shows that the algorithm has fairly good scalability in NF cases. However, for the FF case, it is less so. Using large delay parameters, $D_o = 500$ there is a 200% increase in overhead when the network size goes up by one order of magnitude compared to only a 10% increase in the non-coherent case. Overall, there is a general increase in overhead and delay, which is the result of adding the SN election process.

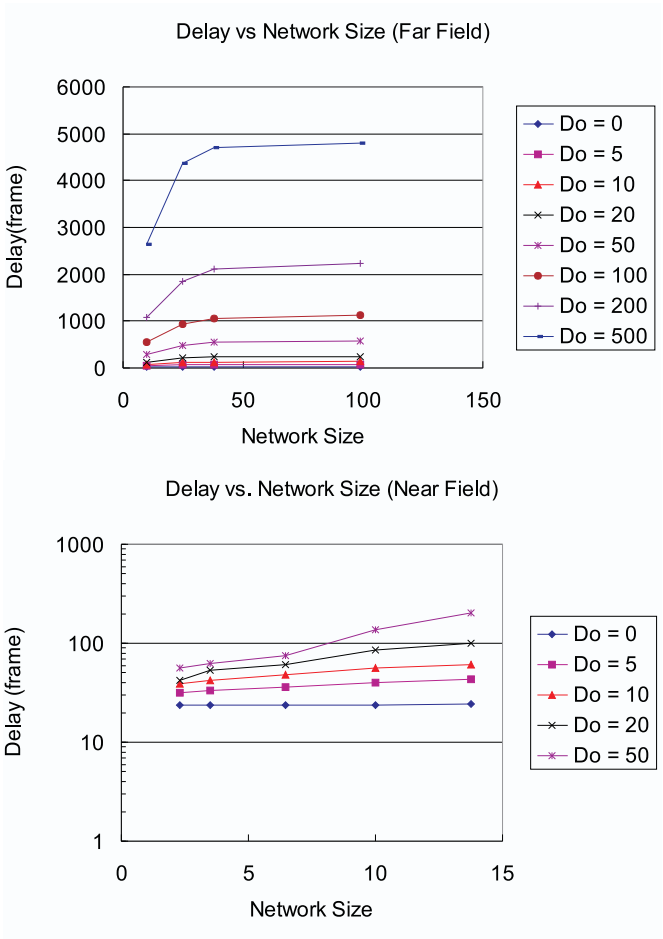

Figure 5.4: Formation Delay and Network Size

We also study the relationship between delay and network sizes. In both NF and FF cases, delay increases with network size. In the non-coherent case, increasing network size may actually shrink the initial delay of the winning candidate because the highest SNR value is larger. However, in the coherent case, CN election is based on finding the *minimum* energy $E^* = \min_j E_j$, larger network tends to make all E_j's larger, which increases the *initial delay* for all nodes because now the delay is directly proportional to E_j for each node j. The overall

delay is also influenced by the *waiting_time* duration used for SN election termination. However, we believed that since it delays each node for a fixed length of time after processing the last received **Elect** message, it does not influence the dynamics in algorithmic behavior if it is sufficiently long to separate the SN and CN elections in time.

5.4 Summary

In general, our algorithm is fairly scalable in NF cases when some delay can be tolerated, but FF target presents a much tougher challenge. However, we need not view this loss in scalability as a shortcoming. If good energy efficient paths are created for the data transfer between SNs and CN, then the energy required during setup is well spent. If we factor the potential energy saving accrued during data upload into the overall energy consumption measurement, the scalability of the overall process - network formation as well as the execution of the cooperative function - should be higher.

Chapter 6

Summary & Direction for Future Research

6.1 Summary of the Work Done

We have seen significant advances in both the sensing and signal processing fields in recent years. As sensor sensitivity approaches their fundamental physical limits, they gradually come to rely on signal processing techniques to enhance their performance. The use of microphone arrays is one such example. If we take this notion of a "sensor array" and simply expand its physical dimension, then we can get a sense of what a "sensor network" is capable of. However, to tap into the full potential of sensor networks, we have to deal with the communication problem. The best long term solution will be based on an ad-hoc network architecture because of its ability to operate without infrastructural support. However, the stringent constraint on energy resources has become the most significant challenge in sensor network design.

This dissertation has thus far presented two energy efficient protocols for multi-hop routing and the formation of a local network for cooperative signal processing. We anticipated a set of application level tasks and designed energy efficient algorithms to fulfill their communication needs. One of the application level tasks is the multi-hop communication between sensors and an information gathering entity called a USER. Such tasks typically involve lightly loaded traffic which predominately originates from the sensors. There is also the need to provide different QoS (delay, reliability, etc.) to individual packets according to the importance of their payloads. In chapter 3, a Sequential Assignment Routing (SAR) protocol was designed to fulfill this requirement based on a multipath, table-driven structure. The multipath component bring welcome attributes such as robustness and load balancing and lays the foundation for priority service. The table-driven approach provides efficiency under low mobility. The SAR algorithm is designed to make efficient use of limited energy resources. As its name would suggest, its objective is to raise the overall QoS on the entire "sequence" of service demand throughout the life time of the network, rather than optimize the individual communication sessions since the latter approach is only profitable without the energy constraint.

A cooperative signal processing function usually takes place among a subgroup of sensors that detected the signal generated by a common environmental phenomenon. The membership, location and size of this "group" can vary greatly. The specific signal processing techniques used can be distributed or centralized. However, regardless of the degree of computational distribution, a central entity needs to be selected to direct the computational process and make aggregate decisions. These cooperative signal processing functions must be supported by

several networking tasks such as (1) the identification of participating members, (2) the selection of central node according to appropriate criteria, and (3) the creation of communication routes between the controller and the individual members. In chapter 4, we described a network protocol that fulfills these three tasks for the general class of non-coherent cooperative functions that requires minimal exchange of parameters because most processing is done locally on each node. A single-winner election (SWE) is the used to facilitate the central node selection process and at the same time build a minimum hop spanning tree rooted from the central node to each group member. Since the election process is basically a distributed contention process, energy efficiency can be improved by suppressing the process on each node according to its estimated likelihood for winning the election. The suppression mechanism, however, will create a delay trade-off.

When the cooperative function is more centralized, which is typically the case for coherent processing, raw data transfered from each member to the central node will become the most significant source of energy consumption. As a result, the protocol design needed to refocus toward the minimization of energy consumption on each path. In chapter 5 we described an adaption of the original algorithm to this new design objective. First, a multi-winner election(MWE) algorithm, as an extention to SWE, is used to select a limited number of source nodes that will provide the raw data. During the MWE, each node will estimate the energy consumption required to connect to each source node. This information can be used in a SWE process to select the node that has the lowest total energy cost to be the gathering point for sensor data.

6.2 Future Research

There are several issues that have not been investigated due to their complexity and relative newness. One such issue is the development of a good model for target mobility and statistics. Our interest in this issue stems from the fact that the sensor network is operating mostly in response to its targets of interest. The movement and the frequency of appearance of a target naturally generates a different pattern of activities within the network. In both multi-hop and local adaptive network formation, the energy efficiency of our algorithm can be greatly affected by this pattern. Lacking good models, what we have done in the multi-hop case is to assume the worst case scenario where extreme non-uniformity occurs. This is modeled by putting all traffic on a small number of random locations for long duration, and testing the network's ability to adapt over time. However, a more accurate model can bring insights into how the network will behave under realistic situations, and help the designer find the actual area that needs work to make the system better. The same challenge lies in the accurate modeling of the characteristics of the transmission medium in the sensor domain, because they can influence the quality of sensor data and affect the behavior of the network formation process.

Preliminary work has already begun on designing a MAC level protocol capable of efficient mobility management for sensor networks, such as the EAR algorithm described in [7]. However, the effect of USER mobility on routing cost has not been addressed in this dissertation. While for low mobility systems, one would expect that a path recomputation is sufficient to keep the USER connected, we have not investigated the efficiency of such an approach. A crucial

question to ask is this: below what level of mobility is recomputation adequate? What steps would be taken if the USER mobility increases above that level?

The scalability of the coherent signal processing function is another area worth taking an in-depth look. Although using distributed processes and selective suppression, the scalability of the network formation protocol is fairly good given one is willing to tolerate delay, we believe there is room for improvement. One possibility is to partition the network into smaller subgroups running at low duty-cycle such that at any given time, only a very small but fixed number of subgroups are activated to observe target presence no matter how large the actual network is or how strong the target signal is. This eliminates the need to use the MWE process to limit the size of the cooperative group. However, mobile NF targets can penetrate the network undetected if the duty-cycle is too low, and an additional routing protocol has to join the subgroups together over long distances, especially in the coherent case where the inefficiency of the flooding technique is not tolerable. Another possiblity is to add an additional layer of signal processing functions on top of the network formation process to determine whether each target detected is NF or FF, and then suppress or activate subgroups in the network according to this decision. Whether the additional complexity will result in an overall energy saving remains to be seen.

6.3 Final Remarks

We believe that this dissertation has addressed some of the fundamental issues in sensor networks. Throughout the course of this investigation, we have to reorient our thinking toward energy efficiency, since it is one of the most significant

contraints in network design. It is our hope that the questions raised and the solutions proposed here will serve as a starting point for future research along this new direction.

Chapter 7

Appendix A

7.1 Proof of Lemma 7.1

Lemma 7.1 *Let* $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_M)$, $\beta = (\beta_1, \beta_2, \ldots, \beta_M)$. *Let* P *be the set of all permutation of* $\{1, 2, \ldots, M\}$ *and* $C(i_j) = \sum_{j=1}^{M} \alpha_j \beta_{i_j}$. *If* $\alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_M$ *and* $\beta_1 \leq \beta_2 \leq \ldots \leq \beta_M$, *then*

$$C^* = \min_{(i_1, i_2, \ldots, i_M) \in P} C(i_j) = \sum_{j=1}^{M} \alpha_j \beta_j$$

Proof: Consider any permutation i_j such that $\beta_{i_1}, \beta_{i_2}, \ldots \beta_{i_M}$ is not non-decreasing, $\exists l > k, n > m$ such that $\beta_{i_l} = \beta_m$ and $\beta_{i_k} = \beta_n$. Then

$$\begin{aligned}
C(i_j) &= \sum_{j=1}^{M} \alpha_j \beta_{i_j} \\
&> \sum_{j=1}^{M} \alpha_j \beta_{i_j} + \underbrace{(\alpha_k - \alpha_l)}_{>0} \underbrace{(\beta_m - \beta_n)}_{<0} \\
&= \sum_{j=1}^{M} \alpha_j \beta_{i_j} - \alpha_k \beta_n - \alpha_l \beta_m + \alpha_k \beta_m + \alpha_l \beta_n \\
&= \sum_{j=1}^{M} \alpha_j \beta_{i_j} - \alpha_k \beta_{i_k} - \alpha_l \beta_{i_l} + \alpha_k \beta_m + \alpha_l \beta_n
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{j \neq k,l} \alpha_j \beta_{i_j} + \alpha_k \beta_m + \alpha_l \beta_n \\
&= C(\tilde{i_j}), \text{where } \tilde{i_j} \in P
\end{aligned}
$$

This implies that $\forall$ permutation i_j such that β_{i_j} is not non-decreasing, a better permutation $\tilde{i_j}$ can be found. QED

Bibliography

[1] J. G. Proakis, *Digital Communications,* 2nd ed. McGraw-Hill, New York, 1989.

[2] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids,* Saunder College Publishing, Fort Worth, 1976.

[3] F. Harary, *Graph Theory,* Addison-Wesley, Massachusetts, 1969.

[4] J. D. Spragins, J. L. Hammond, and K. Pawlikowski, *Telecommunications: protocols and design,* Addison-Wesley, Massachusetts, 1991.

[5] E. M. Royerand C. Toh, "A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks," *IEEE Personal Communications*, April 1999, pp. 46.

[6] L. R. Ford Jr. and D. R. Fulkerson, *Flows in Networks,* Princeton University Press, 1962.

[7] K. Sohrabi, J. Gao, V. Ailawadhi and G. Pottie, "An Energy conscious Self-Organizing Wireless Sensor Network," *IEEE Personal Communications Magazine*, October 2000.

[8] M. Scott Corson, J. P. Macker, and G. H. Cirincione, "Internet-Based Mobile Ad Hoc Networking," *IEEE Internet Computing*, July-August 1999, pp. 63-70.

[9] I. Gitman, R. M. Van Slyke, and H. Frank, "Routing in Packet-Switching Broadcast Radio Networks," *IEEE Transactions on Communications*, August 1976, pp 26-930.

[10] M. Gerla and J. T. Tsai, "Multicluster, mobile, multimedia radio network", *Wireless Networks*, vol. 1, 1995, pp. 255-265.

[11] Thurber, K. J. "Tutorial: distributed processor communication architecture," IEEE, 1979.

[12] O. Lesser and R. Rom, "Routing by Controlled Flooding in Communication Networks," *INFOCOM 1990*, pp. 910-917.

[13] J. Jubin and J. D. Tornow, "The DARPA Packet Radio Network Protocols," *IEEE Proceedings*, vol. 75, no. 1, January 1987, pp. 21-32.

[14] E. W. Dijkstra and C.S. Scholten, "Termination Detection for Diffusing Computations," *Information Processing Letters*, vol. 11, no. 1, August 1980, pp.1-4.

[15] Ernest J.H. Chang, "Echo Algorithms: Depth Parallel Operations on General Graphs," *IEEE Transactions on Software Engineering*, vol. SE-8, no. 4, July 1982, pp. 391-401.

[16] Wai S. Lai, "Bifurcated Routing in Computer Networks," it Computer Communications Review, 1985, pp.28-49.

[17] Keith Scott, "Control and Routing in Self-Organizing Wireless Networks," Ph.D. Dissertation, Department of Electrical Engineering, UCLA, 1997, pp.61-76.

[18] N. F. Maxemchuk, "Dispersity Routing on ATM Networks," IEEE, 1993, pp.347-357.

[19] Philip M. Merlin and Adrian Segall, "A Failsafe Distributed Routing Protocol," *IEEE Transaction on Communications*, vol. COM-27, no. 9, Sept. 1979, pp.1280-1287.

[20] K. Sohrabi, J. Gao, V. Ailawadhi, G. Pottie, "A Self-organizing Wireless Sensor Network," *Proc. 39th Annual Allerton Conference on Communication, Control, and Computing*, Urbana, Illinois, October 1999.

[21] K. Sohrabi, B. Manriquez, and G. Pottie, "Near Ground Wideband Channel Measurement in 800MHz-1GHz," *Proceeding of IEEE Vehicular Technology Conference*, September 1999, Amsterdam, The Netherlands.

[22] K. Yao, R. E. Hudson, C. W. Reed, D. Chen, and F. Lorenzelli, "Blind Beamforming on a Randomly Distributed Sensor Array System," *IEEE Journal On Selected Areas in Communications*, vol. 16, no. 8, October 1998, pp.1555-1567.

[23] J. W. Suurballe, "Disjoint Paths in a Network," *Networks*, 4:125-145, 1974, John Wiley & Sons.

[24] D. Sidhu, R. Nair, and Shukri Abdallah, "Finding Disjoint Paths in Networks," *Computer Communication Review*, vol.21, no. 4, (SIGCOMM '91

Conference. Communications, Architectures and Protocols, Zurich, Switzerland, 3-6 Sept. 1991.) Sept. 1991, p.43-51.

[25] C. Cheng, S. P. R. Kumar, and J. J. Garcia-Luna-Aceves, " A distributed algorithm for finding K disjoint paths of minimum total length," *Proc. 28th Annual Allerton Conference on Communication, Control, and Computing*, Urbana, Illinois, October 1990.

[26] D. M. Topkis, "A k Shortest Path Algorithm for Adaptive Routing in Communications Networks," *IEEE Transactions on Communicatios*, vol. 36, no. 7, July 1988, pp. 855-859.

[27] Zheng Wang, "Analysis of shortest-path routing algorithms in a dynamic network environment," *Computer Communication Review*, vol.22, no. 2, April 1992, p.63-71.

[28] S. Sing, M. Woo, C. S. Raghavendra, "Power-Aware Routing in Mobile Ad Hoc Networks," MOBICOM'98, Dallas Texas. pp.181-190.

[29] T. J. Mcadams, T. L. Thomas, and R. Wade, "Communications System Considerations for Unattended Army Battlefield Munition and Sensor System," *IEEE*, 1997, pp.613-617.

[30] Gregory J. Pottie, "Hierarchical Information Processing in Distributed Sensor Networks," *ISIT*, Cambridge, MA, USA. August 1998, pp. 163.

[31] C. Derman, G. J. Lieberman, and S. M. Ross, "A Sequential Stochastic Assignment Problem," *Management Science*, vol. 18, no. 7, March 1972, pp.349-355.

[32] D. P. Kennedy, "Optimal Sequential Assignment," *Mathematics of Operations Research*, vol. 11, no. 4, November 1986, pp.619-626.

[33] Jacob Sharony, "An architecture for mobile radio networks with dynamically changing topology using virtual subnets," *Mobile Networks and Applications*, 1(1996)75-86.

[34] J. J. Garcia-Lunes-Aceves, "Loop-Free Routing Using Diffusing Computation," *IEEE/ACM Transactions on Networking*, vol. 1, no. 1, February 1993.

[35] M. Scott Corson and Anthony Ephremides, "A distributed routing algorithm for mobile wireless networks," *Wireless Networks*, 1(1995)61-81.

[36] Ko, Y.-B. and Vaidya, N.H. "Location-aided routing (LAR) in mobile ad hoc networks," *MobiCom'98 Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, Dallas, TX, USA, 25-30 Oct. 1998 p.66-75.