

5-2019

## Data Gathering in Cognitive Radio Ad Hoc and Sensor Wireless Networks

Kimberly A. Brown

Follow this and additional works at: [https://csuepress.columbusstate.edu/theses\\_dissertations](https://csuepress.columbusstate.edu/theses_dissertations)



Part of the [Computer Sciences Commons](#)

---

### Recommended Citation

Brown, Kimberly A., "Data Gathering in Cognitive Radio Ad Hoc and Sensor Wireless Networks" (2019). *Theses and Dissertations*. 349.

[https://csuepress.columbusstate.edu/theses\\_dissertations/349](https://csuepress.columbusstate.edu/theses_dissertations/349)

This Thesis is brought to you for free and open access by the Student Publications at CSU ePress. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of CSU ePress.



DATA GATHERING IN COGNITIVE RADIO AD HOC AND SENSOR  
WIRELESS NETWORKS

Kimberly A. Brown  
2019



COLUMBUS STATE UNIVERSITY

DATA GATHERING IN COGNITIVE RADIO AD HOC AND SENSOR WIRELESS  
NETWORKS

A THESIS SUBMITTED TO  
THE D. ABBOTT TURNER COLLEGE OF BUSINESS  
IN PARTIAL FULFILLMENT OF  
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BY

KIMBERLY A. BROWN

COLUMBUS, GEORGIA

2019

DATA GATHERING IN COGNITIVE RADIO AD HOC AND SENSOR WIRELESS NETWORKS

Kimberly A. Brown

Committee Chair

Copyright © 2019 Kimberly A. Brown

All Rights Reserved.

Committee Members

Dr. Jianbin Yang

Dr. Fuh-Lai

Columbus State University

May 2019



DATA GATHERING IN COGNITIVE RADIO AD HOC AND SENSOR WIRELESS  
NETWORKS

By

Kimberly A. Brown

Committee Chair:

Dr. Lixin Wang

Committee Members:

Dr. Jianhua Yang

Dr. Suk Lee

Columbus State University

May 2019

## ABSTRACT

Data gathering is a network communication task in which all of the network's nodes send their individual messages to a distinguished sink node. In cognitive radio ad hoc and sensor wireless networks (CR-AHSWNs), unlicensed secondary users (SUs) opportunistically use channels when the licensed primary users are not using them. Therefore, the channels available to each SU vary with time and location, which makes the development of data gathering algorithms for CR-AHSWNs challenging.

In this thesis, a data gathering protocol for CR-AHSWNs is proposed. The protocol consists of several distributed SU action selection and channel selection algorithms. An algorithm that can reduce the data gathering delay by selecting message forwarding SUs is also proposed. Finally, an algorithm that calculates an estimate of the successful data gathering ratio (SDGR) is proposed. The SDGR is affected by each SU's channel availability and network collisions, and the exact value is extremely challenging to calculate.

**INDEX WORDS:** cognitive radio, ad hoc and sensor wireless networks, data gathering, non-uniform channel availability, channel hopping



## ACKNOWLEDGEMENTS

This work would not have been possible without the guidance and patience of my advisor, Dr. Lixin Wang. I would also like to thank my committee members, Dr. Jianhua Yang and Dr. Suk Lee, for their advice. Thank you to Dr. Shamim Khan, who encouraged me to pursue a thesis. Finally, thanks to my family for their support.

<b>CHAPTER 1: BACKGROUND</b>	<b>1</b>
1.1 Ad Hoc and Sensor Wireless Networks	2
1.2 Cognitive Radio	3
1.3 Data Gathering	3
1.4 Prior Work	3
1.5 Network Model and Assumptions	5
<b>CHAPTER 2: DATA GATHERING PROTOCOL</b>	<b>8</b>
2.1 Overview	8
2.2 Initialization	9
2.3 Action Selection	10
2.3.1 Assumptions	13
2.3.2 One Radio	14
2.3.3 Two Radios	26
2.4 Channel Selection	28
2.4.1 Assumptions	30
2.4.2 Random Channel Selection	30
2.4.3 Guaranteed Channel Match Algorithms	31

CHAPTER 3: SELECTION OF	TABLE OF CONTENTS	51
3.1 Assumptions		53
ACKNOWLEDGEMENTS		iv
LIST OF TABLES		vii
LIST OF FIGURES		ix
LIST OF ABBREVIATIONS		xi
<b>CHAPTER 1: BACKGROUND</b>	<b>-----</b>	<b>1</b>
1.1 Ad Hoc and Sensor Wireless Networks		1
1.2 Cognitive Radio		1
1.3 Data Gathering		3
1.4 Prior Work		3
1.5 Network Model and Assumptions		5
<b>CHAPTER 2: DATA GATHERING PROTOCOL</b>	<b>-----</b>	<b>8</b>
2.1 Overview		8
2.2 Initialization		9
2.3 Action Selection		10
2.3.1 Assumptions		13
2.3.2 One Radio		14
2.3.3 Two Radios		28
2.4 Channel Selection		38
2.4.1 Assumptions		39
2.4.2 Random Channel Selection		39
2.4.3 Guaranteed Channel Match Algorithms		40



<b>CHAPTER 3: SELECTION OF FORWARDING SU SETS</b> -----	<b>51</b>
3.1 Assumptions.....	53
3.2 Network Topology is Known by All SUs.....	54
3.2.1 Description.....	54
3.2.2 Algorithms.....	57
3.2.3 Example.....	60
3.2.4 Analysis.....	66
3.3 Network Topology is Determined by the Sink SU.....	67
3.3.1 Description.....	67
3.3.2 Algorithms.....	68
3.3.3 Example.....	70
3.3.4 Analysis.....	71
<b>CHAPTER 4: SUCCESSFUL DATA GATHERING RATIO ALGORITHM</b> -----	<b>72</b>
4.1 Assumptions.....	74
4.2 SDGR Estimate Calculation Algorithm.....	75
4.3 SDGR Estimate Algorithm Example.....	78
4.4 Single Hop Probability Calculations.....	81
4.4.1 Random Channel Selection.....	82
4.4.2 Guaranteed Channel Match Channel Selection.....	82
4.5 SDGR Algorithm Evaluation.....	83
<b>CHAPTER 5: CONCLUSIONS</b> -----	<b>86</b>
5.1 Summary.....	86
5.2 Future Work.....	86



Table 22: Layer 2 messages and forwarding SU sets after first iteration .....	63
Table 23: Messages received by the sink SU in the CR-AHSWN in Figure 29 .....	70
Table 1: Notation for the action selection algorithms.....	15
Table 2: Differences between the sending and listening SU distances.....	28
Table 3: Differences between the sending and sending/listening SU distances .....	37
Table 4: Differences between the sending/listening and listening SU distances.....	38
Table 5: Random channel selection example for the CR-AHSWN in Figure 20 .....	40
Table 6: Notation for the GCM channel sequence algorithms .....	41
Table 7: Example of a GCM sending channel sequence .....	42
Table 8: Example of a GCM listening channel sequence .....	44
Table 9: Example of GCM channel sequences .....	45
Table 10: Example of GCM two radio channel sequence for SUs with even distances.....	49
Table 11: Example of GCM two radio channel sequence for SUs with odd distances .....	49
Table 12: Silent time slots when SUs with even distances send to SUs with odd distances .....	50
Table 13: Silent time slots when SUs with odd distances send to SUs with even distances .....	50
Table 14: Notation for the forwarding SU sets selection algorithms.....	57
Table 15: Layers array for the CR-AHSWN in Figure 23 .....	61
Table 16: Messages values for layer 2 .....	62
Table 17: Layer 1 message values after first iteration .....	63
Table 18: Layer 2 messages and forwarding SU sets after first iteration .....	63
Table 19: Layer 1 message values after second iteration .....	64
Table 20: Layer 2 messages and forwarding SU sets after second iteration .....	64
Table 21: Layer 1 message values after third iteration .....	65



Table 22: Layer 2 messages and forwarding SU sets after third iteration .....	65
Table 23: Messages received by the sink SU in the CR-AHSWN in Figure 29.....	70
Table 24: Notation for the SDGR algorithm.....	75
Table 25: Layers array for the CR-AHSWN in Figure 31 .....	79
Table 26: Layer 2 probability array after the first recursion of the SDGR algorithm .....	80
Table 27: Layer 2 probability array after the second recursion of the SDGR algorithm.....	81
Table 28: SDGR values determined using simulations and the SDGR algorithm .....	84
Figure 6: Example of action selection for one radio - time 4 .....	20
Figure 7: Example of action selection for one radio - time 8 .....	21
Figure 8: Example of action selection for one radio - time 12 .....	22
Figure 9: Example of action selection for one radio - time 16 .....	23
Figure 10: Example of action selection for one radio - time 20 .....	24
Figure 11: Example of action selection for one radio - time 24 .....	25
Figure 12: Example of action selection for one radio - time 28 .....	26
Figure 13: Action cycle for CR-AHSWNs using devices with two radios .....	29
Figure 14: Example of action selection for two radios - time 0.....	31
Figure 15: Example of action selection for two radios - time 4.....	32
Figure 16: Example of action selection for two radios - time 8.....	33
Figure 17: Example of action selection for two radios - time 12.....	34
Figure 18: Example of action selection for two radios - time 16.....	35
Figure 19: Example of action selection for two radios - time 20.....	36
Figure 20: CR-AHSWN - random channel selection example .....	39
Figure 21: CR-AHSWN - GCM channel selection example.....	44

Figure 22: CR-AHSWN - increase in LIST OF FIGURES	52
Figure 23: CR-AHSWN - Forwarding SU Set Selection algorithm example	60
Figure 1: CR-AHSWN - message transmission between two SUs	2
Figure 2: The data gathering operation	3
Figure 3: Action cycle for CR-AHSWNs using devices with one radio	14
Figure 4: Example of action selection for one radio - initialization	18
Figure 5: Example of action selection for one radio - time 0	19
Figure 6: Example of action selection for one radio - time 4	20
Figure 7: Example of action selection for one radio - time 8	21
Figure 8: Example of action selection for one radio - time 12	22
Figure 9: Example of action selection for one radio - time 16	23
Figure 10: Example of action selection for one radio - time 20	24
Figure 11: Example of action selection for one radio - time 24	25
Figure 12: Example of action selection for one radio - time 28	26
Figure 13: Action cycle for CR-AHSWNs using devices with two radios	29
Figure 14: Example of action selection for two radios - time 0	31
Figure 15: Example of action selection for two radios - time 4	32
Figure 16: Example of action selection for two radios - time 8	33
Figure 17: Example of action selection for two radios - time 12	34
Figure 18: Example of action selection for two radios - time 16	35
Figure 19: Example of action selection for two radios - time 20	36
Figure 20: CR-AHSWN - random channel selection example	39
Figure 21: CR-AHSWN - GCM channel selection example	44



Figure 22: CR-AHSWN - increase in the data gathering delay .....	52
Figure 23: CR-AHSWN - Forwarding SU Set Selection algorithm example .....	60
Figure 24: Bipartite graph for layer 3 .....	61
Figure 25: Bipartite graph for layer 2 .....	62
Figure 26: Bipartite graph for layer 2's first semi-matching .....	62
Figure 27: Bipartite graph for layer 2's second semi-matching .....	64
Figure 28: Bipartite graph for layer 2's third semi-matching .....	65
Figure 29: CR-AHSWN - construction of the network topology example .....	70
Figure 30: Network topology after processing the first four messages .....	71
Figure 31: CR-AHSWN - SDGR algorithm example.....	79
Figure 32: CR-AHSWN - calculation of the layer 2 SDGR, first iteration .....	80
Figure 33: CR-AHSWN - calculation of the layer 2 SDGR, second iteration .....	80
Figure 34: CR-AHSWN - evaluation of the SDGR algorithm .....	84

This chapter provides background information on the three basic concepts covered in this thesis:

AHSWN	Ad Hoc and Sensor Wireless Network
CR	Cognitive Radio
CR-AHSWN	Cognitive Radio Ad Hoc and Sensor Wireless Network
GCM	Guaranteed Channel Match (a channel selection algorithm)
SDGR	Successful Data Gathering Ratio

### 1.1 Ad Hoc and Sensor Wireless Networks

Ad hoc and sensor wireless networks (AHSWNs) are wireless networks that operate without any type of infrastructure or centralized control [1]. Message transmissions in an AHSWN can be either single hop, when the intended destination node is within the transmission radius of the sending node, or multi-hop, when the intended destination node is not within the transmission radius of the sending node. When multi-hop routing is required for message transmission, intermediate nodes must forward the message until it reaches its destination [1].

AHSWNs are flexible, low-cost, and can be quickly deployed [2]. These factors make them useful in applications in many areas, including military, healthcare, industrial monitoring, and environmental monitoring.

### 1.2 Cognitive Radio

In the United States and many other countries, government agencies issue licenses for the exclusive use of radio spectrum frequencies. The demand for radio spectrum is increasing, but



## Chapter 1: Background

This chapter provides background information on the three basic concepts covered in this thesis: ad hoc and sensor wireless networks, cognitive radio, and the data gathering operation for wireless networks. In addition, this chapter describes prior research on these concepts and its applicability to the specific area of data gathering in cognitive radio ad hoc and sensor wireless networks (CR-AHSWNs). Finally, the network model and assumptions that are used throughout the rest of the work are provided.

### 1.1 Ad Hoc and Sensor Wireless Networks

Ad hoc and sensor wireless networks (AHSWNs) are wireless networks that operate without any type of infrastructure or centralized control [1]. Message transmissions in an AHSWN can be either single hop, when the intended destination node is within the transmission radius of the sending node, or multi-hop, when the intended destination node is not within the transmission radius of the sending node. When multi-hop routing is required for message transmission, intermediate nodes must forward the message until it reaches its destination [1].

AHSWNs are flexible, low-cost, and can be quickly deployed [2]. These factors make them useful in applications in many areas, including military, healthcare, industrial monitoring, and environmental monitoring.

### 1.2 Cognitive Radio

In the United States and many other countries, government agencies issue licenses for the exclusive use of radio spectrum frequencies. The demand for radio spectrum is increasing, but

unlicensed frequencies are becoming scarce [3]. Studies have found that some licensed radio spectrum is underutilized [4], and cognitive radio (CR) technology has been proposed as a method for efficiently using those underutilized frequencies [5]. Specifically, CR allows unlicensed Secondary Users (SUs) to opportunistically use licensed frequencies when the licensed Primary Users (PUs) are not using them [6].

A PU always has priority on the use of its licensed frequency, so an SU must vacate a frequency when a PU begins using it. Therefore, the SUs must sense their environment to determine which frequencies are available for use and adjust their available frequencies, or channels, accordingly. Since the SUs must be able to switch channels, SUs in CR networks are multi-channel, and the specific channels available to individual SUs vary with location and time [6]. The dynamic nature of the CR networks makes the design of network protocols for CR networks difficult.

Figure 1 illustrates this difficulty:

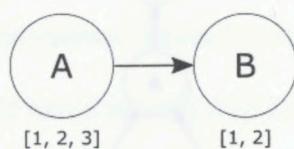


Figure 1: CR-AHSWN - message transmission between two SUs

SU *A* needs to send a message to SU *B*. Each SU has its own set of available channels; *A*'s available channel set is  $\{1, 2, 3\}$ , and *B*'s available channel set is  $\{1, 2\}$ . In order to successfully transmit the message, both *A* and *B* will need to select the same channel at the same time.

However, neither SU knows which channel the other is currently using. In addition, neither has any knowledge of the other's available channel set.



Cognitive radio devices can be used in AHSWNs, creating cognitive radio ad hoc and sensor wireless networks (CR-AHSWNs).

### 1.3 Data Gathering

Data gathering is a primitive networking operation in which all nodes in the network must send their individual messages to a distinguished sink node [7]. Therefore, each node in the network must send its own message and each message that it receives towards the sink node. In the network shown in Figure 2, the sink node for the network is node *S*. Node *B* sends its message to *A* since *A* is on the path between *B* and *S*. Node *C* also sends its message to *A*. Consequently, node *A* must send three messages to *S*: its own message, the message from *B*, and the message from *C*. Every message is sent separately towards the sink node without any data combination or aggregation.

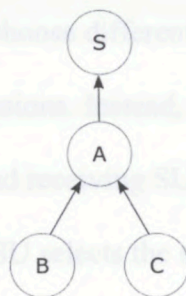


Figure 2: The data gathering operation

### 1.4 Prior Work

Prior work on data gathering has been focused on traditional, single-channel AHSWNs. The main difference between traditional AHSWNs and CR-AHSWNs is channel availability. In traditional AHSWNs, there is usually only a single channel available on the network, and communication between nodes simply uses that channel. In CR-AHSWNs, the channel

availability for every SU is dynamic. Not only can an SU have a different available channel set than its neighbors, its available channel set can change every time a PU uses or vacates a channel. In practice, SUs typically do not have any way to discover the currently available channels of their neighboring SUs.

Data gathering for traditional AHSWNs usually requires using efficient link scheduling algorithms that guarantee collision-free message transmissions with low delays [7]. However, SUs in CR-AHSWNs typically do not have any knowledge of the network topology, so link schedules cannot be created. In previous work on CR-AHSWN operations, the SUs use time-slotted channel hopping [8] [9]. In time-slotted channel hopping, a channel is chosen for each time slot over a predetermined number of time slots, and a message transmission is attempted in each time slot. Since every message transmission is attempted multiple times, and SUs that are sending to a common receiving SU can choose different channels, a collision free protocol is not required for successful message transmissions. Instead, successful message transmission depends on the probability that the sending SU and receiving SU will select the same channel in at least one time slot, and that no other sending SU selects the same channel in that time slot.

A significant amount of research has been focused on the broadcast operation in CR networks [8] [9] [10] [11] [12]. Some of this work can also be applied to the data gathering operation. Specifically, the channel selection techniques described in [8] and [9] can be used to develop a channel selection algorithm for data gathering that can guarantee that pairs of sending and receiving SUs select the same channel, as demonstrated in Section 2.4.3.



In [10], an analytical model for evaluating the success ratio and delay of the broadcast operation on CR networks is presented. In the broadcast operation, a source SU sends a single message that must be propagated to every other SU in the network [7]. In data gathering, every SU sends a message that must be received by the sink SU [7]. Because of these differences, the analytical model presented in [10] cannot be used for the evaluation of data gathering performance.

Therefore, a novel algorithm for calculating the success ratio for the data gathering operation on CR networks is proposed in chapter 4.

### 1.5 Network Model and Assumptions

Each device in the CR-AHSWN is equipped with an omnidirectional antenna, which transmits and receives messages in all directions. Each device has either one radio or two radios. In the one radio device, the single radio is dual function; it can switch between transmitting and receiving messages and can only perform one function at a time. The two radio device has one transmitter and one receiver, and these radios can operate at the same time. Separate algorithms for the one and two radio devices are provided in the data gathering protocol described in chapter 2.

Every device in the CR-AHSWN has the same circular transmission radius, and all SUs within an SU's transmission radius are considered neighbors of the SU. If two or more devices use the same channel to send to an SU that is within both of their respective transmission radii, a collision will occur between the messages, and none of the messages will be successfully transmitted. Every device has access to the same set of channels, but the channels available to each SU differ based on activity of their neighboring PUs.

During the data gathering operation, it is assumed that the SUs' available channel sets and locations will remain stable. Once a data gathering operation starts, the SUs will not lose or gain channels or move to a new location until after the sink SU has received the final data gathering message and the data gathering operation is complete.

The system times on all the devices in the CR-AHSWN are synchronized, and all SUs operate using time slots with the same predetermined length. The length of the time slot is long enough for the successful transmission of a message across the entire SU transmission radius. Therefore, there is an upper limit on the size of a single data gathering packet. That is, every data gathering packet must be able to be transmitted within the length of a time slot.

A CR-AHSWN can be represented as an undirected graph, with the SUs represented by the graph's vertices. Two SU vertices are directly connected in the graph if and only if they have at least one channel in common and their Euclidean distance is no more than the SU transmission radius.

Finally, the data gathering protocols described in chapter 2 are intended to work under realistic conditions for practical applications of CR-AHSWNs. The algorithms are distributed; each SU must make its own decisions on when to send and when to listen. Each SU must also determine which channels to use for these actions. There is no central control SU that determines how the data gathering operation should be performed. Each SU determines its own available channel set by sensing the current PU channel usage within its sensing radius. However, the SUs have no information about the available channel sets of their neighbors, and the SUs have very little



information about the network topology. These conditions are typically considered realistic for practical applications of CR-AHSWNs.

Data gathering is a challenging operation to implement in a CR-AHSWN for two reasons. First, as discussed in the previous chapter, the SUs in a CR-AHSWN typically have no knowledge of the available channel sets of neighboring SUs. This makes the implementation of the data-gathering operation difficult because, in a data-gathering operation, a sending SU must find a matching channel with a receiving SU for each individual message that it sends. Second, SUs in a CR-AHSWN typically do not have knowledge of the full network topology, so they cannot determine which of their neighbors are on a path to the sink SU.

In this chapter, a data-gathering protocol for CR-AHSWNs is presented. This protocol includes distributed algorithms that SUs use to determine when to send and when to listen for messages, without any knowledge of the network topology, as well as algorithms for channel selection that do not require knowledge of neighboring SUs' available channels.

The proposed data-gathering protocol is composed of three parts: an initialization step, the action-selection algorithm, and the channel-selection algorithm. In the initialization step, the SUs gather information that is necessary for the proposed algorithms. Each SU uses the action-selection algorithm to determine which action it should perform in each time slot. The action choices are to send, listen, stay silent (neither listen nor send), or stop. Each SU uses the channel-selection algorithm to determine which channel to use when sending or listening.

## Chapter 2: Data Gathering Protocol

### 2.1 Overview

Data gathering is a challenging operation to implement in a CR-AHSWN for two reasons. First, as discussed in the previous chapter, the SUs in a CR-AHSWN typically have no knowledge of the available channel sets of neighboring SUs. This makes the implementation of the data gathering operation difficult because, in a data gathering operation, a sending SU must find a matching channel with a receiving SU for each individual message that it sends. Second, SUs in a CR-AHSWN typically do not have knowledge of the full network topology, so they cannot determine which of their neighbors are on a path to the sink SU.

In this chapter, a data gathering protocol for CR-AHSWNs is presented. This protocol includes distributed algorithms that SUs use to determine when to send and when to listen for messages without any knowledge of the network topology, as well as algorithms for channel selection that do not require knowledge of neighboring SUs' available channels.

The proposed data gathering protocol is composed of three parts: an initialization step, the action selection algorithm, and the channel selection algorithm. In the initialization step, the SUs gather information that is necessary for the protocol algorithms. Each SU uses the action selection algorithm to determine which action it should perform in each time slot. The action choices are to *send*, *listen*, stay *silent* (neither listen nor send), or *stop*. Each SU uses the channel selection algorithms to determine which channel to use when sending or listening.



## 2.2 Initialization

The data gathering sink SU must establish itself as the sink SU for the network by sending a message to all of the other SUs in the network. This is accomplished using a broadcast operation, which is a network operation in which a single message is sent by a source node, and the message is then propagated through the entire network until every node in the network has received the message [7]. In the initialization step, the data gathering sink SU acts as the source SU for the broadcast operation.

Since the proposed data gathering protocol should work without the SUs having knowledge of the network topology, the broadcast operation must also work without the SUs having knowledge of the network topology. Two broadcast protocols that operate without the network topology are given in [8] and [9]. The chosen broadcast protocol for this initialization step should have a high success ratio, which is the probability that all SUs in the network receive the broadcast message. Only the SUs that receive the broadcast message will participate in the data gathering operation.

As the broadcast messages pass through the network, each SU will include its own best known hop distance to the sink SU in the broadcast message. In this context, an SU's best known hop distance to the sink SU is the number of hops the broadcast message took from the sink SU to reach the SU. The sink SU will include its best known hop distance, which is zero, in its broadcast message. As each SU receives the broadcast message, it will remove the sending SU's distance and increment it by one to determine its own best known hop distance. Then, it will

include its own best known hop distance in the broadcast message that it sends. Using this method, every SU will be able to determine its best known hop distance to the sink SU.

This initializing broadcast step should be run as frequently as necessary based on the needs of the specific CR-AHSWN. In the action selection algorithms described in the next section, the SU's best known hop distance is used to determine the SU's action. However, the activity of the PUs or movement of SUs could invalidate the previously established best known hop distances.

Therefore, it is important that the initializing broadcast operation be run frequently enough to maintain valid SU best known hop distances so that SUs can successfully send their data gathering messages to neighboring SUs. Stable systems with little PU activity, immobile SUs, and many available channels could run the initializing broadcast once, while systems with a significant amount of PU activity, mobile SUs, and smaller available channel sets might need to run the initializing broadcast on a more frequent basis.

### 2.3 Action Selection

Two action selection algorithms are developed in this section: one for devices with a single radio, and one for devices with two radios. The algorithms for these devices are very similar. The SUs determine their current action using their best known hop distance to the sink SU and the current time slot. The algorithms are designed so that the SUs send their data gathering messages to neighboring SUs that are one hop closer to the sink SU.

It is possible for messages to be sent from one SU to another that has a greater best known hop distance. This could occur in dense networks where the SUs are very close together. To prevent



these backwards message transmissions, each SU must include its own best known hop distance in each data gathering message that it sends. Receiving SUs will drop received messages when the distance in the message is less than its own distance.

In each of these algorithms, the SUs cycle through the actions until the stopping criteria are met. The SU performs the selected action for  $S_r$  consecutive time slots. The  $S_r$  consecutive time slots are collectively referred to as an action interval.

Each SU maintains a queue of the messages that it needs to send. The queue is initially populated with the SU's own message, and each message that is received is added to the queue. In the beginning of each send interval, the SU removes the message at the front of its send queue and attempts to send it for the duration of the send interval.

The stopping criteria for the action cycles are tracked with four flag variables: *collision*, *done*, *last*, and *listen*. The *collision* flag is set to true when an SU is listening and detects that a collision occurred between two or more messages during a listen interval.

The *done* flag is set to true when the SU is done listening because no additional messages will be received. The *done* flag is set to true when the following two conditions are met:

1) In the last listen interval, one of these conditions occurs:

- The SU received no messages.
- All of the *last* flags included with the received messages are true.

2) The SU's *collision* flag is false.

The *listen* flag is set to true when the SU has completed a listen interval. This flag ensures that SU has performed at least one listen interval before stopping.

Finally, the *last* flag is set to true when the following two conditions are met:

- 1) The SU's *done* flag is true.
- 2) The SU's message queue contains one message.

The sending SU includes its *last* flag with the messages that it sends, and this flag indicates to the receiving SU that this message is the last message that the sending SU will send.

An SU exits the action cycle and stops when it is done listening and sending; that is, when the SU's *done* flag is true and its send queue is empty. The data gathering operation for the network is complete when the sink SU has stopped.

The length of the action interval depends on two things: the length of the channel sequences produced by the specific channel selection algorithms and the performance requirements of the network. If the channel selection algorithm requires a specific number of time slots, such as the Guaranteed Channel Match channel selection algorithms described in Section 2.4.3, then the action interval length must equal the number of time slots required by the channel selection algorithm.

However, if the channel selection algorithm does not require a specific number of time slots, such as the random channel selection method described in Section 2.4.2, the value of  $Sr$  is



determined using the requirements of the network. The performance metrics for a network operation are typically the success ratio and the delay [10]. In the data gathering operation, the success ratio is the probability that the sink SU will receive messages from all of the other SUs in the network. The delay is the number of time slots required for the entire data gathering operation to complete successfully, from the time when the first data gathering message is sent to the time when the last message is received by the sink SU. Choosing a value of  $Sr$  is a trade-off between the success ratio and the delay performance metrics [9]. For the random channel selection algorithm, a larger value of  $Sr$  will result in both a greater success ratio and a greater delay than a smaller value of  $Sr$ .

### 2.3.1 Assumptions

For a successful data gathering to occur using the action selection algorithms described in this section, the channel selection must meet the following requirements:

- 1) At least one set of channel sequences for all SUs in the network must exist such that every sending SU in the network is able to select a channel that matches the channel of at least one of its receiving SUs in at least one time slot.
- 2) For at least one of the receiving SUs and at least one of the time slots in which the channels of the sending SU and the receiving SU match, no other SU sending to that receiving SU selects the same channel.

If these requirements are not met, the action selection algorithms cannot be used, because the data gathering operation will always fail.

## 2.3.2 One Radio

### 2.3.2.1 Description

In the action selection algorithm for devices with one radio, the SUs perform actions in the cycle shown in Figure 3. The SUs cycle through the actions until the stopping criteria described in the previous section are met. The SUs can start anywhere in this cycle, and, as described previously, each SU must perform at least one listen interval before stopping.

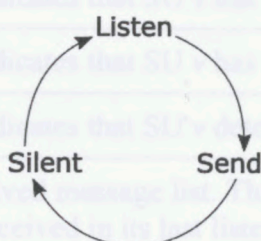


Figure 3: Action cycle for CR-AHSWNs using devices with one radio

### 2.3.2.2 Algorithm

The notations shown in Table 1 are used in both the one radio and two radio action selection algorithms shown in Sections 2.3.2.2 and 2.3.3.2 respectively.



Table 1: Notation for the action selection algorithms

<i>time</i>	The current time slot
<i>S<sub>r</sub></i>	The number of consecutive time slots in an action interval
<i>v.dist</i>	SU <i>v</i> 's best known hop distance to the sink SU
<i>v.action</i>	SU <i>v</i> 's action. The options are <i>Send</i> , <i>Listen</i> , <i>Silent</i> , and <i>Stopped</i> . The two radio device has an additional action option: <i>Send/Listen</i> .
<i>v.done</i>	Flag that indicates that SU <i>v</i> is done listening
<i>v.last</i>	Flag that indicates that SU <i>v</i> has sent its last message
<i>v.listen</i>	Flag that indicates that SU <i>v</i> has performed a listen interval
<i>v.collision</i>	Flag that indicates that SU <i>v</i> detected a collision
<i>v.RM</i>	SU <i>v</i> 's received message list. This list holds the messages that SU <i>v</i> received in its last listen interval.
<i>v.SQ</i>	SU <i>v</i> 's message send queue
$ v.SQ $	The number of messages in SU <i>v</i> 's send queue

---

**Algorithm 1:** Select an action (*Send, Listen, Silent, Stopped*) for SU  $v$ . The SUs in this network have one dual function radio.

---

Input:  $time, Sr, v.dist, v.action$

Output:  $v.action$

---

```

1  if (time % Sr) = 0 then
2    /* time to switch actions */
3    if v.done = True and v.last = True then
4      /* SU v is done listening and has sent its last message*/
5      v.action = Stopped
6
7    else
8      /* choose action using time and distance */
9      if (time / Sr) % 3) = 0 then
10       if (v.dist % 3) = 1 then
11         set_action_send()
12       else if (v.dist % 3) = 0 then
13         set_action_listen()
14       else
15         v.action = Silent
16
17     else if (time / Sr) % 3) = 1 then
18       if (v.dist % 3) = 0 then
19         set_action_send()
20       else if (v.dist % 3) = 2 then
21         set_action_listen()
22       else
23         v.action = Silent
24
25     else if (time / Sr) % 3) = 2 then
26       if (v.dist % 3) = 2 then
27         set_action_send()
28       else if (v.dist % 3) = 1 then
29         set_action_listen()
30       else
31         v.action = Silent
32
33 return v.action

```

---



---

**Algorithm 2:** This helper function is used when the SU sets its action to *Send*.

---

```

set_action_send() :
1   v.action = Send
2   if v.listen = True then
3   /* the SU must have listened at least once before setting the
4     other flag values */
5     if ((v.RM = [] or for all messages in v.RM last = True) and
6         v.collision = False) then
7         v.done = True
8         if v is the sink then
9             v.action = Stopped
10        else if |v.SQ| = 1 then
11            v.last = True
12        else if |v.SQ| = 0 then
13            v.action = Stopped

```

---

**Algorithm 3:** This helper function is used when the SU sets its action to *Listen*.

---

```

set_action_listen() :
1   v.action = Listen
2   v.listen = True
3   v.RM = []

```

### 2.3.2.3 Example

The action selection algorithm for one radio devices is demonstrated using the CR-AHSWN shown in Figure 4. SU *S* is the sink SU for the data gathering operation, and the length of the action interval,  $S_r$ , is four time slots. Therefore, the SUs' actions change every four time slots. All of the SUs, with the exception of the sink SU, initially populate their send queues with their own data gathering messages.

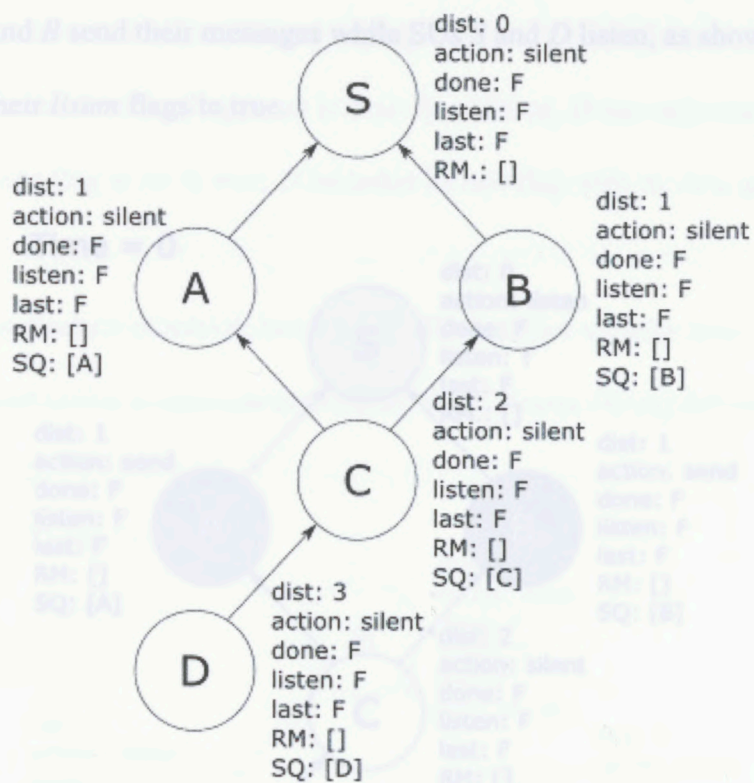


Figure 4: Example of action selection for one radio - initialization

In Figures 5 through 12, SUs that are listening are shaded light grey, SUs that are sending are shaded dark grey, and SUs that are silent or stopped are white. Throughout the example, all message transmission are assumed to be successful, and no collisions occur.



At time 0, SUs *A* and *B* send their messages while SUs *S* and *D* listen, as shown in Figure 5. The listening SUs set their *listen* flags to true.

Time = 0

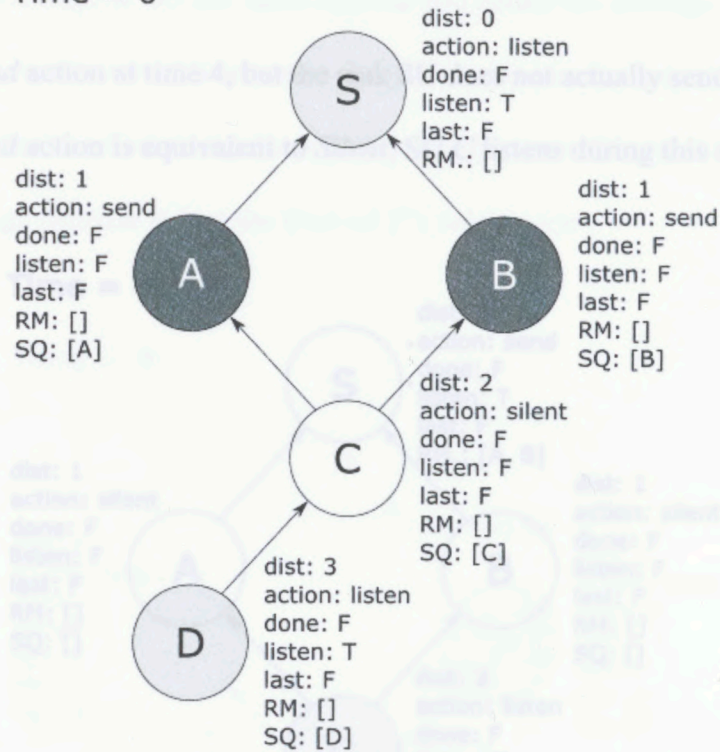


Figure 5: Example of action selection for one radio - time 0

At time 4, SU *D*'s action is *Send*, and it sets its flag values. Since *D* did not receive any messages in its last listen interval, its *done* flag is set to true. In addition, *D* has only one message in its send queue, so its *last* flag is set to true. *D* includes its *last* flag with its data gathering message.

SU *C* received *D*'s message in the last listen interval and added the message to its send queue.

SU *S* selects the *Send* action at time 4, but the sink SU does not actually send any messages. For the sink SU, the *Send* action is equivalent to *Silent*. SU *C* listens during this action interval.

SU *C* sends its own message because it is at the front of *C*'s send queue.

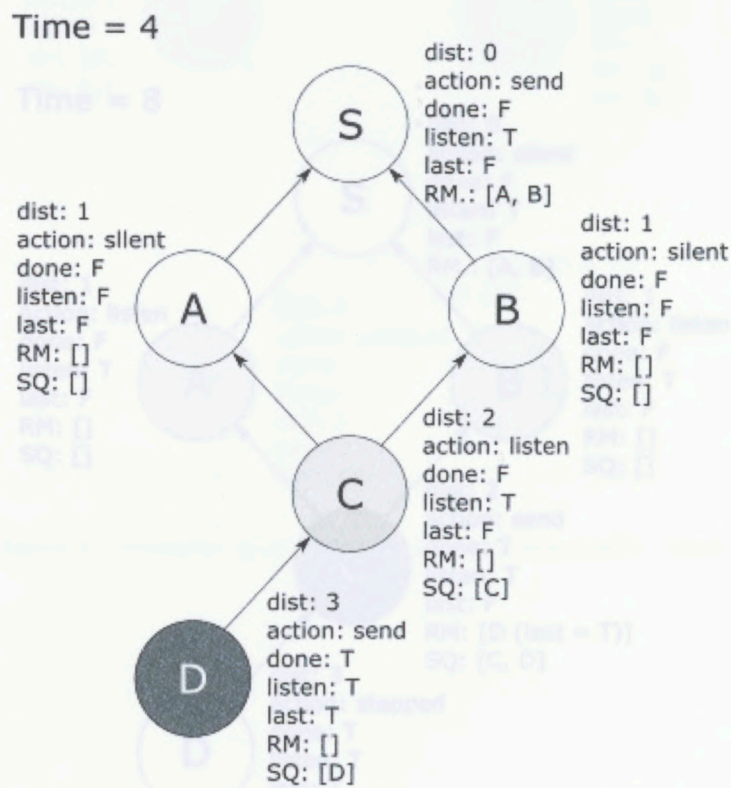


Figure 6: Example of action selection for one radio - time 4

Figure 7: Example of action selection for one radio - time 8



At time 8, SU *D*'s *done* and *last* flags are both true, so *D*'s action changes to *Stopped*. SUs *A* and *B* listen during this action interval.

SU *C* received *D*'s message in the last listen interval and added the message to its send queue.

Since the *last* flag included with *D*'s message was true and it was the only message that *C* received during the last listen interval, *C* sets its *done* flag to true. During this action interval, *C* sends its own message because it is at the front of *C*'s send queue.

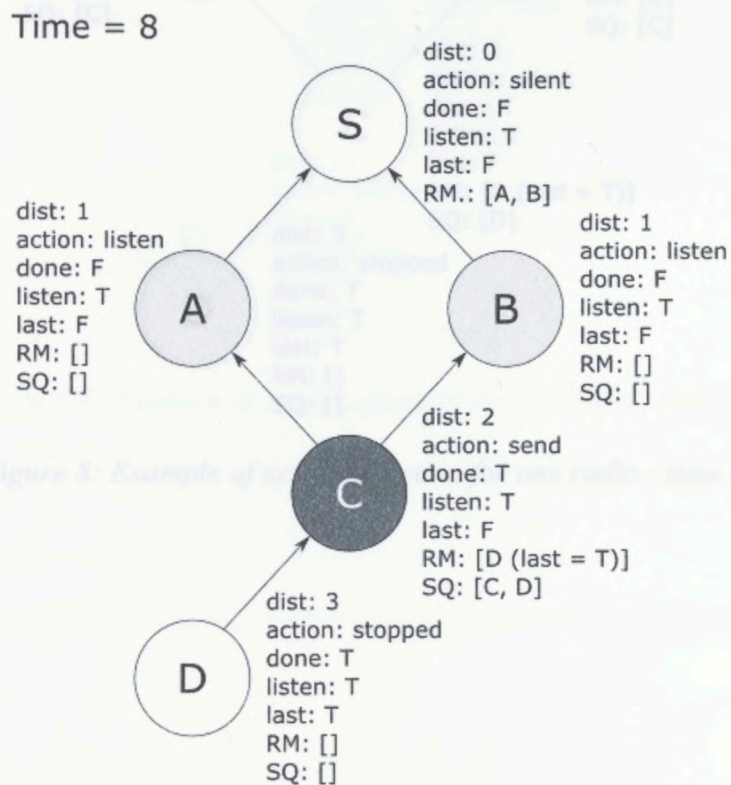


Figure 7: Example of action selection for one radio - time 8

At time 12, SUs *A* and *B* have both received *C*'s message. During this action interval, both *A* and *B* are attempting to send *C*'s message to SU *S*, which is listening.

Time = 12

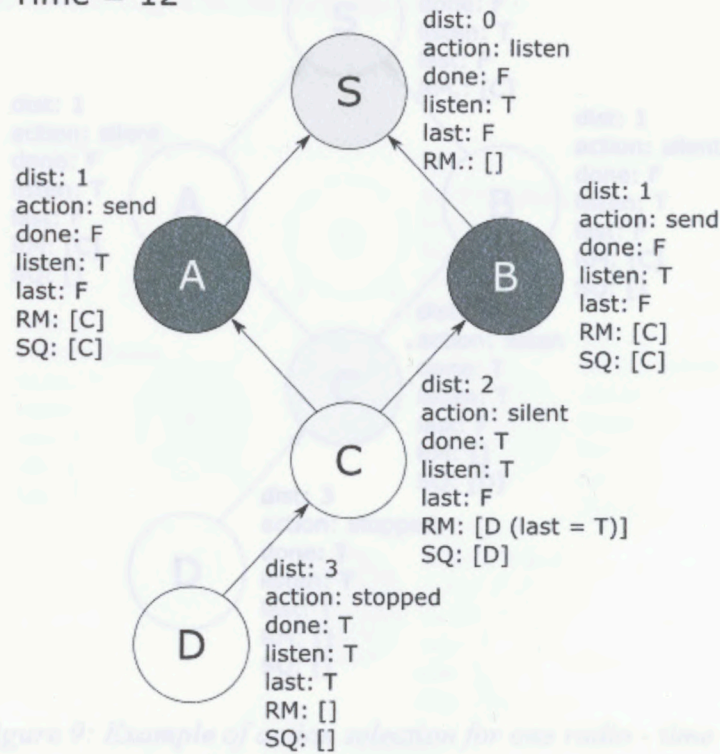


Figure 8: Example of action selection for one radio - time 12



At time 16, SU C listens while all of the other SUs are silent or stopped.

During this time interval, so its done flag is set to true. In addition, since C has only one message in its send queue, its last flag is set to true. During this time interval, C sends its last message which is the message it received from D, and includes its last flag with the message.

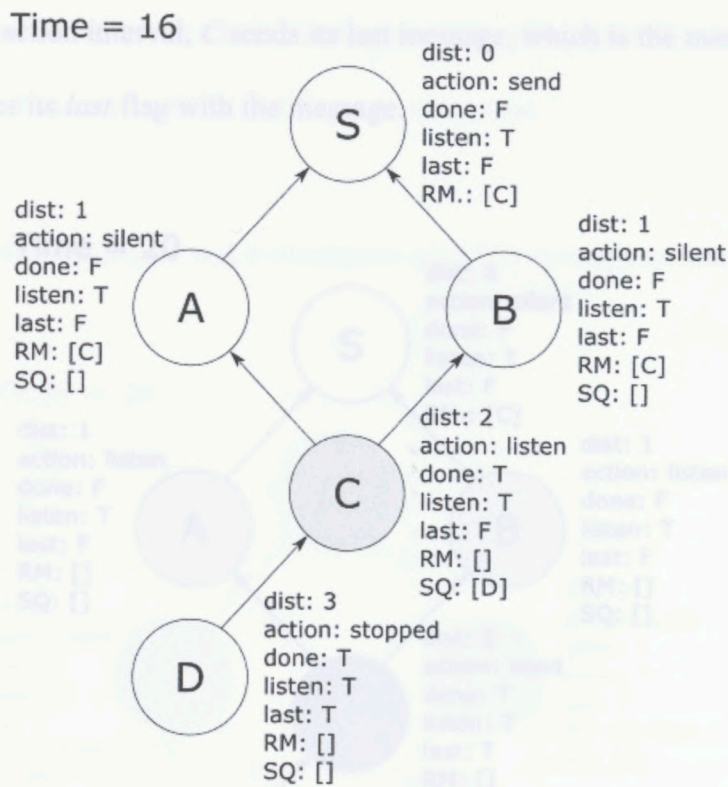


Figure 9: Example of action selection for one radio - time 16

Figure 10: Example of action selection for one radio - time 20

At time 20, SU *C* detects that it did not receive any messages in its last listen interval, so its *done* flag is set to true. In addition, since *C* has only one message in its send queue, its *last* flag is set to true. During this action interval, *C* sends its last message, which is the message it received from *D*, and includes its *last* flag with the message.

Time = 20

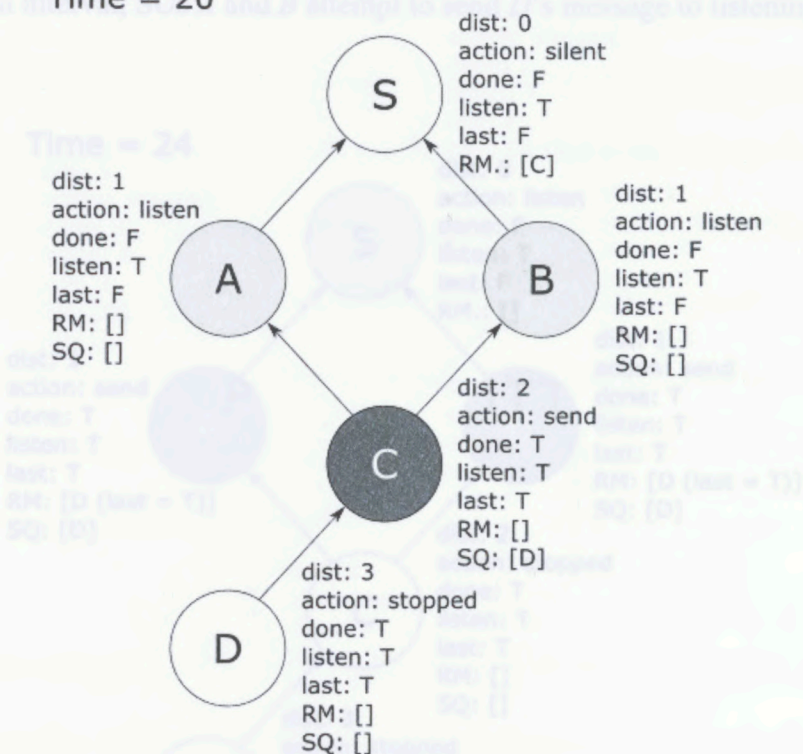


Figure 10: Example of action selection for one radio - time 20

Figure 11: Example of action selection for one radio - time 24



At time 24, SU *C*'s *done* and *last* flags are true, so *C* stops. SUs *A* and *B* have both received *D*'s message from *C*, along with *C*'s *last* flag. This is the only message that *A* and *B* received in their last listen interval, so both SUs set their *done* flags to true. Also, both SUs have only one message in their send queues, so their *last* flags are set to true.

During this action interval, SUs *A* and *B* attempt to send *D*'s message to listening SU *S*.

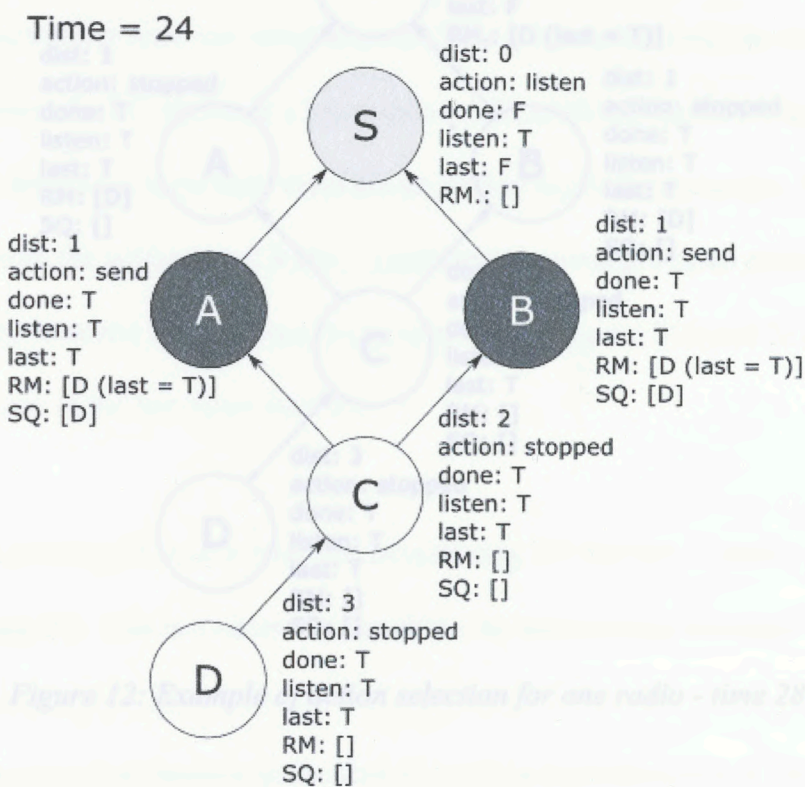


Figure 12: Example of action selection for one radio - time 24

3.3.2.4 Analysis Figure 11: Example of action selection for one radio - time 24

*Theorem 1:* The proposed action selection algorithm will allow messages from all SUs in the network to reach the sink SU, assuming that a successful data gathering is possible. Section 2.3.1 describes the requirements that must be met for a successful data gathering to be possible.

*Proof:* Two conditions must be analyzed for this proof. First, the data gathering operation cannot terminate while messages remain unrecd in the network. This is controlled by the use of the SUs'

At time 28, SUs *A* and *B* stop. SU *S* has received *D*'s message and the included *last* flags from SUs *A* and *B*. SU *S* sets its *done* flag to true, and since *S* is the sink SU, it changes its action to *Stopped*. The data gathering operation is complete, and the sink SU, *S*, has received messages from all of the SUs in the network.

Time = 28

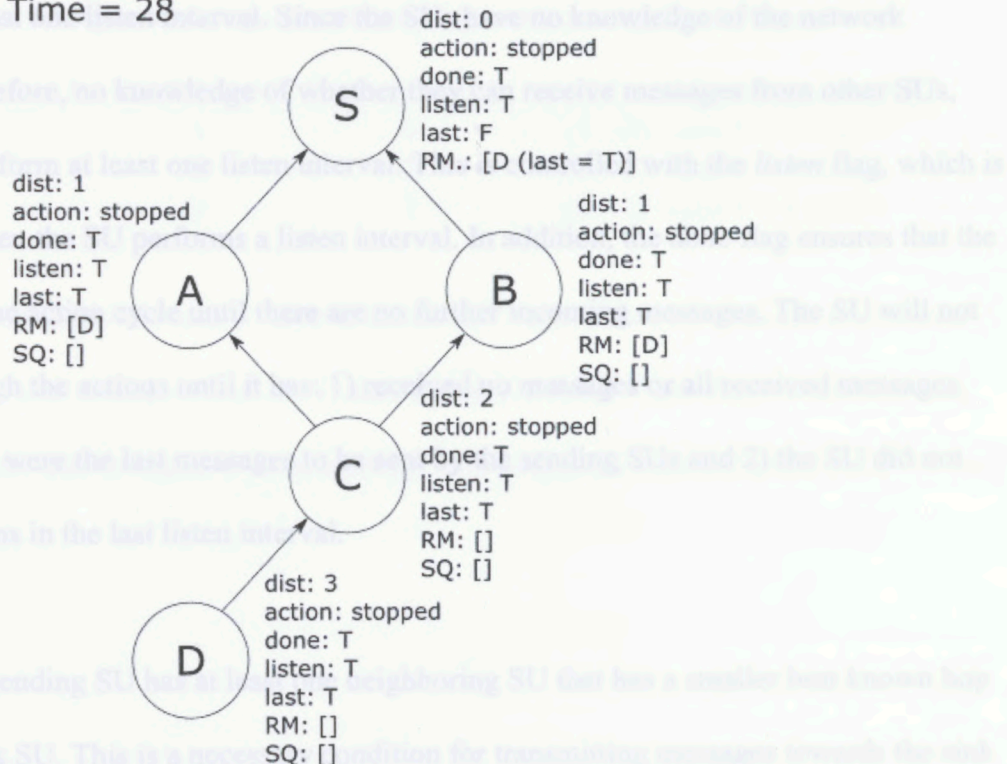


Figure 12: Example of action selection for one radio - time 28

#### 2.3.2.4 Analysis

*Theorem 1:* The proposed action selection algorithm will allow messages from all SUs in the network to reach the sink SU, assuming that a successful data gathering is possible. Section 2.3.1 describes the requirements that must be met for a successful data gathering to be possible.

*Proof:* Two conditions must be analyzed for this proof. First, the data gathering operation cannot terminate while messages remain unsent in the network. This is controlled by the use of the SUs'



send queues and the *last* flag. An SU will not exit the action cycle until one of the following conditions has been met: 1) the send queue is empty or 2) the SU's *last* flag has been set to true, which indicates that the last message in the send queue was sent in the last send interval.

Second, the data gathering operation cannot terminate before all SUs have attempted to receive messages for at least one listen interval. Since the SUs have no knowledge of the network topology, and therefore, no knowledge of whether they can receive messages from other SUs, every SU must perform at least one listen interval. This is controlled with the *listen* flag, which is only set to true when the SU performs a listen interval. In addition, the *done* flag ensures that the SU does not exit the action cycle until there are no further incoming messages. The SU will not stop cycling through the actions until it has: 1) received no messages or all received messages indicated that they were the last messages to be sent by the sending SUs and 2) the SU did not detect any collisions in the last listen interval.

*Theorem 2:* Each sending SU has at least one neighboring SU that has a smaller best known hop distance to the sink SU. This is a necessary condition for transmitting messages towards the sink SU.

*Proof:* The best known hop distance to the sink SU will be represented by  $d$ . The action selections ensure that when SUs with distance  $d$  are sending, SUs with distance  $d-1$  are listening. First, since each SU determined its distance during the initialization step when it received the broadcast message from an SU with distance  $d-1$ , at least one neighboring SU with distance  $d-1$  must exist, assuming that the SUs still share a common channel and have not moved.

Three cases must be analyzed to demonstrate that the action selection algorithm allows SUs to select actions so that the messages are sent from SUs with distance  $d$  to neighboring SUs with distance  $d-1$ . In Table 2, the differences between the sending and listening SU distances are shown for each of the three cases. The sending SU's distance is denoted by  $send.d$ , and the listening SU's distance is denoted by  $listen.d$ .

Table 2: Differences between the sending and listening SU distances

$(time / Sr) \bmod 3$	$send.d \bmod 3$	$listen.d \bmod 3$	$send.d \bmod 3 - listen.d \bmod 3$
0	1	0	$(1 - 0) \bmod 3 = 1 \bmod 3$
1	0	2	$(0 - 2) \bmod 3 = 1 \bmod 3$
2	2	1	$(2 - 1) \bmod 3 = 1 \bmod 3$

In each case, the difference is  $1 \bmod 3$ . This indicates that when SUs with distance  $send.d \bmod 3$  select to send, SUs with distance  $(send.d - 1) \bmod 3$  select to listen. Therefore, when SUs with  $send.d$  values greater than or equal to 1 are sending, their neighbors with distance  $send.d - 1$  will be listening.

### 2.3.3 Two Radios

#### 2.3.3.1 Description

The action selection algorithm for devices with two radios is similar to the algorithm for devices with one radio, with the most important difference being that the two radio action cycle has four actions as shown in Figure 13. The additional action is *Send/Listen*, which indicates that the SU will send and listen at the same time.



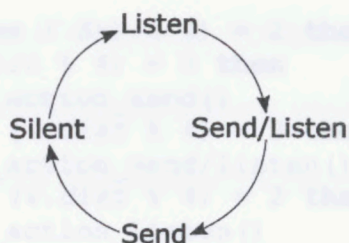


Figure 13: Action cycle for CR-AHSWNs using devices with two radios

### 2.3.3.2 Algorithm

The notations for the algorithms shown in this section can be found in Table 1 in Section 2.3.2.2.

**Algorithm 4:** Select an action (*Send*, *Listen*, *Send/Listen*, *Silent*, *Stopped*) for SU  $v$ . The SUs in this network have two single function radios.

Input:  $time$ ,  $Sr$ ,  $v.dist$ ,  $v.action$

Output:  $v.action$

```

1  if (time % Sr) = 0 then
2  /* time to switch actions */
3  if v.done = True and v.last = True then
4  /* SU v is done listening and has sent its last message */
5  v.action = Stopped
6  else
7  /* choose action using time and distance */
8  if (time / Sr) % 4) = 0 then
9  if (v.dist % 4) = 2 then
10 set_action_send()
11 else if (v.dist % 4) = 1 then
12 set_action_send/listen()
13 else if (v.dist % 4) = 0 then
14 set_action_listen()
15 else
16 v.action = Silent
17
18 else if (time / Sr) % 4) = 1 then
19 if (v.dist % 4) = 1 then
20 set_action_send()
21 else if (v.dist % 4) = 0 then
22 set_action_send/listen()
23 else if (v.dist % 4) = 3 then
24 set_action_listen()
25 else
26 v.action = Silent
27

```

```

28     else if (time / Sr) % 4) = 2 then
29         if (v.dist % 4) = 0 then
30             set_action_send()
31         else if (v.dist % 4) = 3 then
32             set_action_send/listen()
33         else if (v.dist % 4) = 2 then
34             set_action_listen()
35         else
36             v.action = Silent
37
38     else if (time / Sr) % 4) = 3 then
39         if (v.dist % 4) = 3 then
40             set_action_send()
41         else if (v.dist % 4) = 2 then
42             set_action_send/listen()
43         else if (v.dist % 4) = 1 then
44             set_action_listen()
45         else
46             v.action = Silent
47
48     return action

```

---

**Algorithm 5:** This helper function is used when the SU sets its action to *Send/Listen*.

---

```

set_action_send/listen():
1     v.action = Send/Listen
2     if v.listen = True then
3         /* the SU must have listened at least once before setting the
4            other flag values */
5         if ((v.RM = [] or for all messages in v.RM last = True) and
6            v.collision = False) then
7             v.done = True
8             if v is the sink then
9                 v.action = Stopped
10            else if |v.SQ| = 1 then
11                v.last = True
12            else if |v.SQ| = 0 then
13                v.action = Stopped
14     v.listen = True
15     v.RM = []

```

### 2.3.3.3 Example

The CR-AHSWN shown in Figure 4 in section 2.3.2.3 is used to demonstrate the action selection algorithm for devices with two radios. As before, the SUs initially populate their send queues



with their own messages. In Figures 14 through 19, SUs that are shaded light grey are listening, those that are shaded dark grey are sending and listening simultaneously, and those that are shaded black are sending. SUs that are shaded white are silent or stopped.

At time 0, SU *C* attempts to send its message to SUs *A* and *B*, as shown in Figure 14. SUs *A* and *B* are simultaneously listening and sending their messages to listening SU *S*. The SUs that are listening or both listening and sending set their *listen* flags to true.

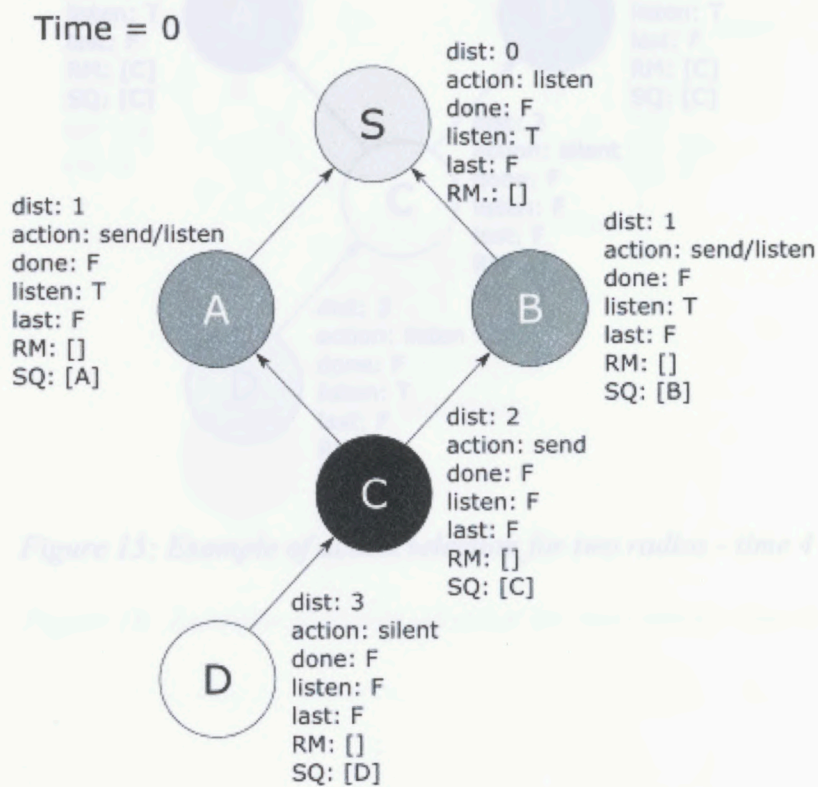


Figure 14: Example of action selection for two radios - time 0

At time 4, SUs *A* and *B* have received *C*'s message, which they add to their send queues. Since this is the only message in their send queues, they each attempt to send *C*'s message to SU *S* during this action interval.

Time = 4

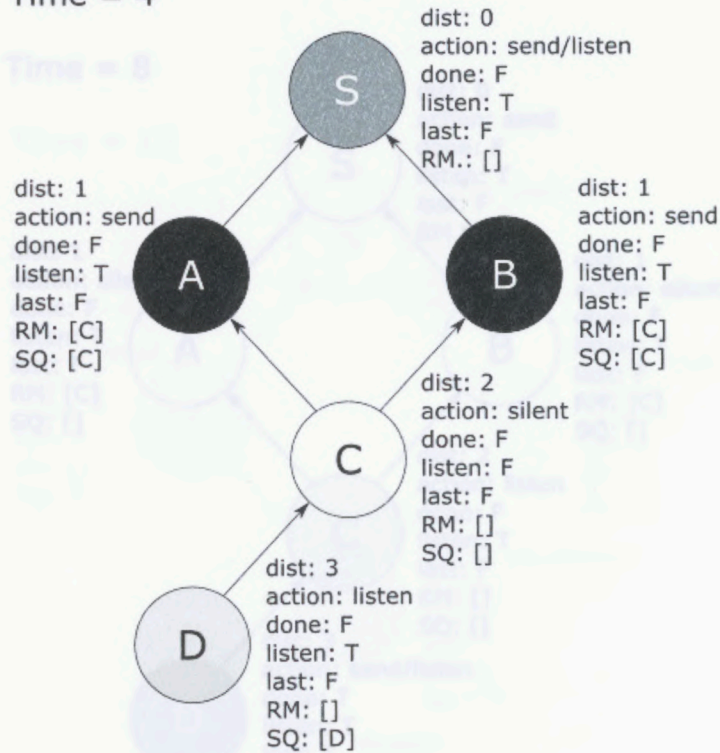


Figure 15: Example of action selection for two radios - time 4

Figure 16: Example of action selection for two radios - time 3



At time 8, SU *D* sets its *done* flag to true because it did not receive any messages in the previous listen interval. In addition, it sets its *last* flag to true because it has only one message in its send queue. During this action interval, SU *D* listens and simultaneously sends its message, which includes its *last* flag, to SU *C*.

Time = 8

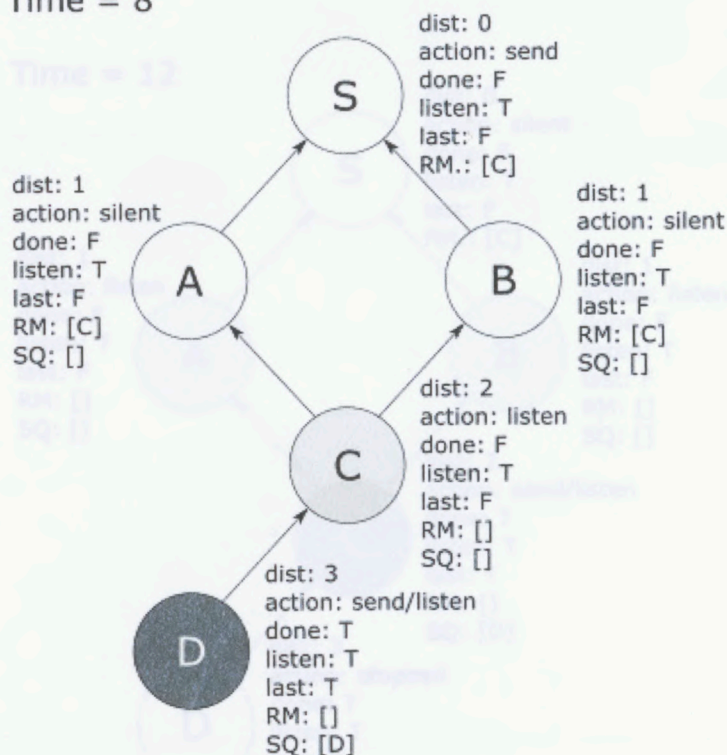


Figure 16: Example of action selection for two radios - time 8

Figure 17: Example of action selection for two radios - time 12

At time 12, SU *D* has stopped because both its *done* and *last* flags were true. SU *C* has received *D*'s message, which was the only message that SU *C* received in its last listen interval. The *last* flag included with *D*'s message was true, so *C* sets its *done* flag to true. In addition, *C*'s *last* flag is set to true. In this action interval, *C* attempts to send *D*'s message along with its own *last* flag to SUs *A* and *B*.

Time = 12

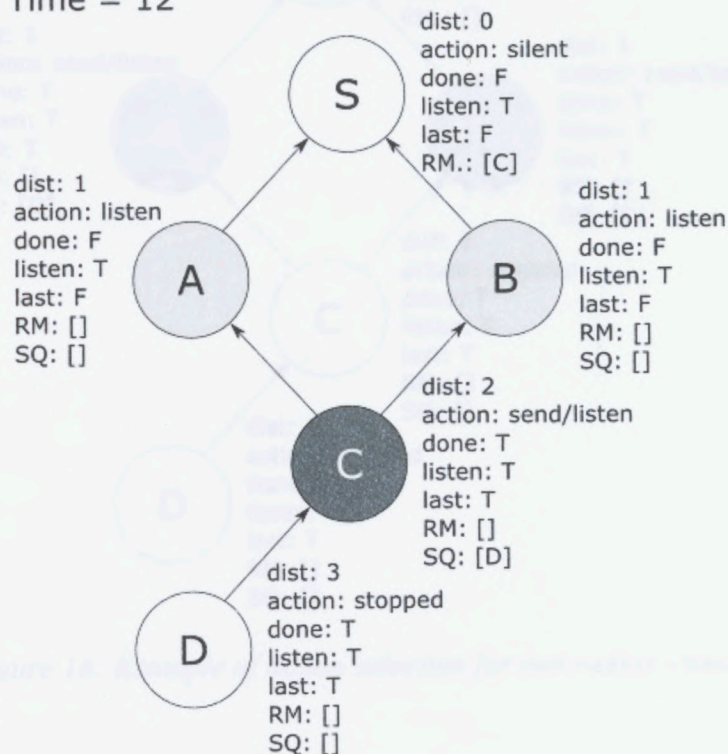


Figure 17: Example of action selection for two radios - time 12



At time 16, SU *C* has stopped. SUs *A* and *B* have received *D*'s message from *C*, along with the *last* flag. *A* and *B* set their own *done* and *last* flags to true. In this action interval, SUs *A* and *B* will attempt to send *D*'s message, along with their own *last* flags, to SU *S*.

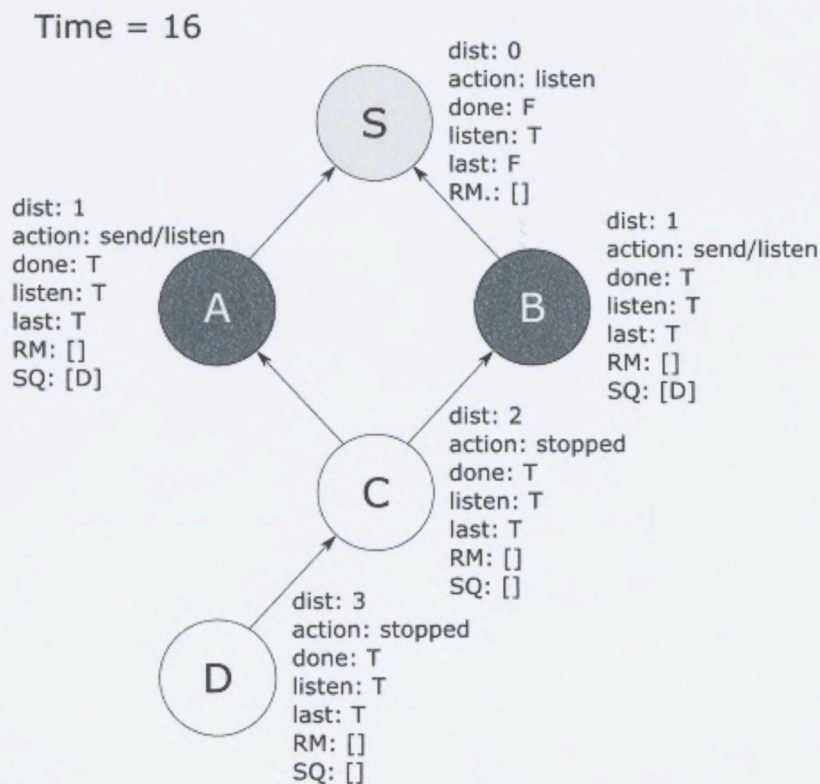


Figure 18: Example of action selection for two radios - time 16

At time 20, SU *S* has received *D*'s message from SUs *A* and *B*. The *last* flag included with these messages is true, so *S* sets its *done* flag to true and changes its action to *Stopped*. The data gathering operation is complete. The sink SU, *S*, has received messages from all of the SUs in the network.

Time = 20

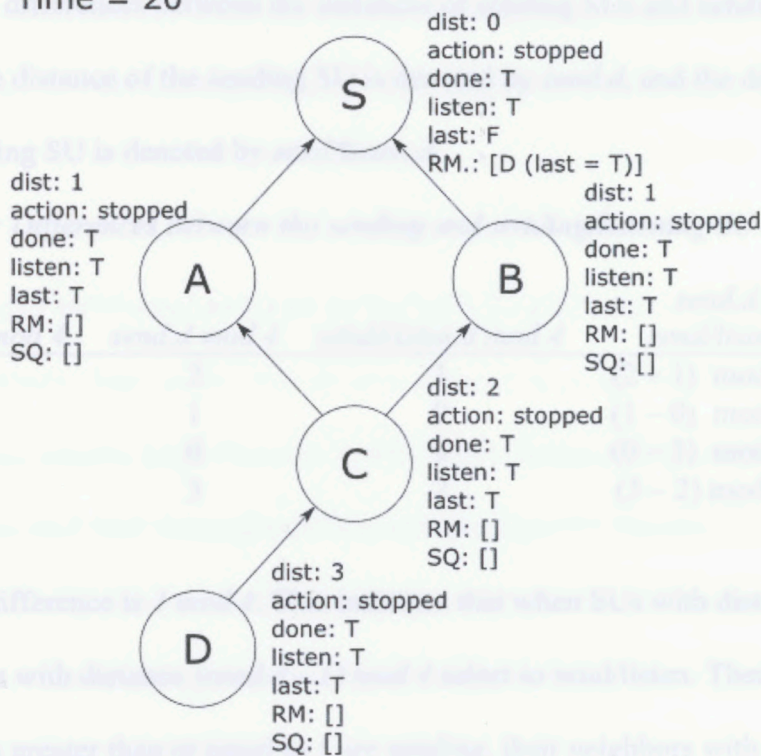


Figure 19: Example of action selection for two radios - time 20

### 2.3.3.4 Analysis

*Theorem 1* described in Section 2.3.2.4 also applies to this algorithm.

*Theorem 3:* This theorem is the same as *Theorem 2* described in Section 2.3.2.4: each sending SU has at least one neighboring SU that has a smaller best known hop distance to the sink SU.

This is a necessary condition for transmitting messages towards the sink SU.



*Proof:* The proof for this theorem generally follows that of *Theorem 2* with additional conditions and cases that must be analyzed. First, there are two sending actions that SUs can select in this algorithm: *Send* and *Send/Listen*. Each of these conditions must be analyzed with each of the cases.

Table 3 shows the differences between the distances of sending SUs and sending/listening SUs for four cases. The distance of the sending SU is denoted by  $send.d$ , and the distance of the sending and listening SU is denoted by  $send/listen.d$ .

Table 3: Differences between the sending and sending/listening SU distances

$(time / Sr) \bmod 4$	$send.d \bmod 4$	$send/listen.d \bmod 4$	$send.d \bmod 4 - send/listen.d \bmod 4$
0	2	1	$(2 - 1) \bmod 4 = 1 \bmod 4$
1	1	0	$(1 - 0) \bmod 4 = 1 \bmod 4$
2	0	3	$(0 - 3) \bmod 4 = 1 \bmod 4$
3	3	2	$(3 - 2) \bmod 4 = 1 \bmod 4$

In each case, the difference is  $1 \bmod 4$ . This indicates that when SUs with distance  $send.d \bmod 4$  select to send, SUs with distance  $(send.d - 1) \bmod 4$  select to send/listen. Therefore, when SUs with  $send.d$  values greater than or equal to 1 are sending, their neighbors with distance  $send.d - 1$  will be sending/listening.

Similarly, Table 4 shows the differences between the distances of sending/listening SUs and listening SUs for all four cases.

Table 4: Differences between the sending/listening and listening SU distances

$(\text{time} / S_r) \bmod 4$	$\text{send}/\text{listen}.d \bmod 4$	$\text{listen}.d \bmod 4$	$\frac{\text{send}/\text{listen}.d \bmod 4 - \text{listen}.d \bmod 4}{\text{listen}.d \bmod 4}$
0	1	0	$(1 - 0) \bmod 4 = 1 \bmod 4$
1	0	3	$(0 - 3) \bmod 4 = 1 \bmod 4$
2	3	2	$(3 - 2) \bmod 4 = 1 \bmod 4$
3	2	1	$(2 - 1) \bmod 4 = 1 \bmod 4$

As in Table 3, the difference for each case is  $1 \bmod 4$ . When SUs with  $\text{send}/\text{listen}.d$  values greater than or equal to 1 are sending/listening, their neighbors with distance  $\text{send}/\text{listen}.d - 1$  will be listening.

*Theorem 4:* Messages are transmitted from an SU with an odd distance to an SU with an even distance, or, conversely, from an SU with an even distance to an SU with an odd distance.

*Proof:* This theorem follows from *Theorem 3*. SUs with a distance  $d$  that select the *Send* or *Send/Listen* actions send their messages to SUs with distance  $d-1$ , therefore SUs with an odd distance send to SUs with an even distance, and SUs with an even distance send to SUs with an odd distance. This property is important for the GCM channel selection algorithm for two radio devices described in section 2.4.3.2.

## 2.4 Channel Selection

When an SU has determined that it will either listen or send in a time slot, it must select a channel to use for that action from its available channel set. In this section, two channel selection methods are described: random channel selection and Guaranteed Channel Match (GCM) channel selection.

Figure 26: CR-ANISWN - random channel selection example



### 2.4.1 Assumptions

Each sending SU must always have at least one channel in common with at least one of its receiving SUs. If this assumption is not met, a successful data gathering is not possible.

A/Rx	1	1	2	2	1	2	1
C/Rx	3	2	2	3	2	3	2

### 2.4.2 Random Channel Selection

In the random channel selection method, each SU randomly chooses a channel from its available channel set for each time slot in the action interval [10]. When the channels selected by a sending and receiving SU pair match, and a collision does not occur, the message can be transmitted. A collision occurs when more than one sending neighbor of the receiving SU selects the same channel as the receiving SU. The messages from the sending SUs collide, and the transmissions are all unsuccessful.

In Figure 20, two SUs,  $B$  and  $C$ , are attempting to send their messages to SU  $A$ . The available channel lists for each SU are  $A$ :  $\{1, 2, 3\}$ ,  $B$ :  $\{1, 2\}$ , and  $C$ :  $\{2, 3\}$ , and the length of the action interval is 9. Table 5 shows an example of sending and listening channel schedules for these SUs. At time slot 1, SU  $B$  successfully sends its message to SU  $A$ . SU  $B$  also sends its message successfully in later time slots, such as 2 and 5, but only one successful message transmission is necessary. At time 6, SU  $C$  successfully sends its message to SU  $A$ . At time 4, all SUs have selected channel 2 and a collision occurs, so neither of the messages are transmitted.

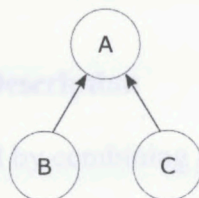


Figure 20: CR-AHSWN - random channel selection example

Table 5: Random channel selection example for the CR-AHSWN in Figure 20

	0	1	2	3	4	5	6	7	8
A (Rx)	2	3	1	2	2	1	3	2	1
B (Tx)	2	1	1	1	2	1	2	1	1
C (Tx)	3	2	2	3	2	2	3	2	3

### 2.4.3 Guaranteed Channel Match Algorithms

The random channel selection method does not provide any guarantees that a sending and receiving pair of SUs will select the same channel in a time slot. The Guaranteed Channel Match Algorithms described in this section guarantee that a sending and receiving pair of SUs with at least one channel in common will select the same channel in at least one time slot.

These channel selection algorithms are largely based on the broadcast channel sequence algorithms presented in [8], with some modifications made for the data gathering operation. Two separate channel sequences are required by the SUs: a sending channel sequence and a listening channel sequence. Therefore, two channel sequence algorithms are proposed in this section.

#### 2.4.3.1 One Radio

This section presents channel sequence algorithms for devices with one dual function radio. The radio can switch between sending and listening, and it can perform only one function at a time.

##### 2.4.3.1.1 Sending Channel Sequence Description

The sending channel sequence is created by combining  $M$  permutations of the SU's available channel set, where  $M$  is the total number of channels available for devices in the CR-AHSWN.

Each permutation must itself have a length of  $M$ . In cases where the number of available



channels for an SU is less than  $M$ , the difference is made up by randomly selecting channels from the SU's available channel set. The length of the sending channel sequence is then  $M^2$  time slots.

#### 2.4.3.1.2 Sending Channel Sequence Algorithm

The notations shown in Table 6 are used in all of the GCM channel sequence algorithms shown in Sections 2.4.3.1 and 2.4.3.2.

Table 6: Notation for the GCM channel sequence algorithms

$v.C$	Available channels for SU $v$
$ v.C $	The number of available channels for SU $v$
$v.listen\_seq$	Listening channel sequence for SU $v$
$v.send\_seq$	Sending channel sequence for SU $v$
$v.dist$	Best known hop distance between SU $v$ and the sink SU
$M$	Total number of channels available in the CR-AHSWN

#### 2.4.3.1.3 Sending Channel Sequence Example

In the following algorithms, the order of the channels is randomized before they are added to the sending and listening sequences. This process is intended to prevent collisions. If two SUs are attempting to send to the same receiving SU, and both sending SUs have the same available channel set, their sending sequences would be the same if the order of the channels is not randomized. This scenario would cause collisions in every time slot. Randomizing the order of the channels while constructing the sending and listening sequences will help prevent collisions.

Algorithm 6 is based on a sending channel sequence algorithm presented in [8] for the broadcast operation.

**Algorithm 6:** Create a sending channel sequence of length  $M^2$  for SU  $v$ . The SUs in this network have one dual function radio.

Input:  $v.C, M$

Output:  $v.send\_seq$

```

1  i = 0
2  v.send_seq = [] /*initialize sending channel sequence */
3  while i < M do
4      send_rand_c = copy v.C
5      if |send_rand_c| < M then
6          /* create list of M channels by adding randomly
7             selected channels to send_rand_c */
8              addtl_chans = randomly choose (M - |v.C|) channels
9                  from v.C
10             append addtl_chans to send_rand_c
11             randomize the order of send_rand_c
12             j = 0
13             while j < M do /* add send_rand_c to v.send_seq */
14                 v.send_seq[(i * M) + j] = send_rand_c[j]
15                 j = j + 1 /* repeat M times */
16             i = i + 1 /* repeat M times */
17  return v.send_seq

```

### 2.4.3.1.3 Sending Channel Sequence Example

SU  $v$  has the available channels set  $v.C = \{1, 3\}$ .  $M$ , the total number of channels available for this CR-AHSWN, is 3. Therefore, the sequence length will be 9 time slots. Since the length of  $v.C$  is 2, which is less than  $M$ , a random channel will be added to each of the  $M$  permutations of the available channel set. An example sending channel sequence for SU  $v$  is shown in Table 7.

Table 7: Example of a GCM sending channel sequence

0	1	2	3	4	5	6	7	8
1	3	3	1	1	3	1	3	1



#### 2.4.3.1.4 Listening Channel Sequence Description

The listening channel sequence is created by repeating each channel in the SU's available channel set  $M$  times. If the number of available channels is less than  $M$ , randomly selected channels are added to the end of the listen sequence to create a total sequence length of  $M^2$  time slots.

An example listening channel schedule for SU  $v$  is shown in Table X.

#### 2.4.3.1.5 Listening Channel Sequence Algorithm

Algorithm 7 is based on a receiving channel sequence algorithm presented in [8] for the broadcast operation.

#### 2.4.3.1.7 Message Transmission Example

**Algorithm 7:** Create a listening channel sequence of length  $M^2$  for SU  $v$ . The SUs in this network have one dual function radio.

Input:  $v.C, M$

Output:  $v.listen\_seq$

---

```

1  v.listen_seq = [] /* initialize listen channel sequence */
2  listen_rand_c = copy v.C
3  randomize the order of listen_rand_c
4  i = 0
5  while i < |listen_rand_c| do
6      j = 0
7      while j < M do
8          v.listen_seq[(i * M) + j] = listen_rand_c[i]
9          j = j + 1 /* repeat each channel M times */
10         i = i + 1 /* repeat for every channel */
11
12     if length(v.listen_seq) < M2 then:
13         addtl_chans = randomly select (M2 - length(v.listen_seq))
14             channels from v.C
15         append addtl_chans to v.listen_seq
16     return v.listen_seq

```

---

### 2.4.3.1.6 Listening Channel Sequence Example

An SU,  $v$ , has the available channels list  $v.C = \{1, 3\}$ .  $M$ , the total number of channels available for this CR-AHSWN, is 3. The length of  $v.C$  is 2 and each channel is repeated  $M$  times, so the length of the listen schedule is 6 time slots, which is less the required length of 9 time slots.

Therefore, 3 random channels are added to the end of the listen schedule. An example listening channel schedule for SU  $v$  is shown in Table 8.

Table 8: Example of a GCM listening channel sequence

0	1	2	3	4	5	6	7	8
1	1	1	3	3	3	3	3	1

### 2.4.3.1.7 Message Transmission Example

In Figure 21, two SUs,  $B$  and  $C$ , are attempting to send their messages to SU  $A$ . The available channel lists for each SU are:  $A: \{1, 2, 3\}$ ,  $B: \{1, 2\}$ , and  $C: \{2, 3\}$ . For this CR-AHSWN,  $M$  is 3.

Table 9 shows sending and listening channel schedules for these SUs. At time 1, SU  $B$  successfully sends its message to SU  $A$ . At time 4, SU  $C$  successfully sends its message to SU  $A$ .

At time 6, all SUs have selected channel 2, and a collision occurs.

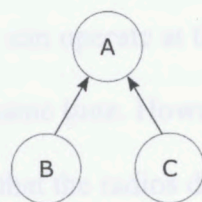


Figure 21: CR-AHSWN - GCM channel selection example



Table 9: Example of GCM channel sequences

	0	1	2	3	4	5	6	7	8
A (Rx)	1	1	1	3	3	3	2	2	2
B (Tx)	2	1	2	1	2	2	2	1	1
C (Tx)	3	3	2	2	3	2	2	3	3

#### 2.4.3.1.8 Algorithm Analysis

*Theorem 5:* A pair of sending and receiving SUs with at least one available channel in common is guaranteed to select the same channel at least once in  $M^2$  time slots.

*Proof:* Algorithms 6 and 7 are based on the broadcasting channel sequences in [8], so the proof of the channel match guarantee follows the proof provided in [8]. In each set of  $M$  consecutive time slots, the sending SU selects each of its available channels at least once. The listening SU stays on each of its available channels for  $M$  consecutive time slots. Therefore, there must be at least one time slot within the  $M^2$  time slots in which both SUs select the same channel.

#### 2.4.3.2 Two Radios

This section presents channel sequence algorithms for devices with two single function radios. One of the radios is a transmitter and can only send messages. The other radio is a receiver and can only listen for messages. The radios can operate at the same time, so this device can listen for messages and send messages at the same time. However, when performing both functions simultaneously, the device must ensure that the radios do not select the same channel, since the message being sent by the transmitting radio would collide with any incoming messages at the receiving radio.

#### 2.4.3.2.1 Sending and Listening Channel Sequence Description

The sending and listening channel sequence algorithms are very similar to the channel sequence algorithms for one radio devices described in Section 2.4.3.1, with two significant differences.

First, the algorithms ensure that the sending and listening channel sequences do not place the same channel in the same time slot, so a single algorithm creates the two sequences. Second, the lengths of the channel sequences are  $M \times (M+1)$ . The channel sequences are created the same way as the one radio channel sequences with a silent time slot added to each set of  $M$  time slots

The listening channel sequence repeats a channel,  $c$ , for  $M$  time slots. Therefore, in that set of  $M$  time slots, the sending channel cannot be channel  $c$ . The channel sequence algorithm adds a silent time slot before or after each set of  $M$  time slots in the listening sequence, and the sending channel sequence uses channel  $c$  in those additional time slots. This method prevents the sending and listening sequences from using the same channel in the same time slot.

Since no messages can be transmitted during the listening channel sequence's silent time slots, the sending channel sequences of the sending SUs should include matching silent time slots. The channel sequence algorithm also adds a silent time slot to each set of  $M$  time slots in the sending channel sequence. Each SU creates its own listening and sending channel sequences with the added silent time slots, and pairs of sending and receiving SUs must place the silent time slots in the same locations.

This problem is solved by using the SU's best known hop distance to the sink SU to determine whether to place the silent time slots at the beginning or the end of each set of  $M+1$  time slots.



SUs with an odd distance place the silent time slot at the beginning of each set of  $M+I$  time slots in the listen channel sequence and at the end of each set of  $M+I$  time slots in the sending channel sequence. Conversely, SUs with an even distance place the silent time slot at the end of each set of  $M+I$  time slots in the listen channel sequence and at the beginning of each set of  $M+I$  time slots in the sending channel sequence. These sequences allow pairs of sending and receiving SUs to match the locations of their respective silent time slots.

#### 2.4.3.2.2 Sending and Listening Channel Sequence Algorithm

The notation table for this algorithm can be found in Table 6 in Section 2.4.3.1.2. The sending and listening channel sequences created by Algorithm 8 are based on the sending and receiving channel sequence algorithms presented in [8] for the broadcast operation.

```

16   j = 0
17   k = 0
18   while j < N + 1 do
19     if j = 0 and (v.dist % 2) = 1 then
20       v.listen_seq[1 * (M + 1) + j] = None
21       v.send_seq[1 * (M + 1) + j] = listen_rand_v[i]
22     else if j = 0 and (v.dist % 2) = 0 then
23       v.listen_seq[1 * (M + 1) + j] = listen_rand_v[i]
24       v.send_seq[1 * (M + 1) + j] = None
25     else if j = N and (v.dist % 2) = 1 then
26       v.listen_seq[1 * (M + 1) + j] = listen_rand_v[i]
27       v.send_seq[1 * (M + 1) + j] = None
28     else if j = N and (v.dist % 2) = 0 then
29       v.listen_seq[1 * (M + 1) + j] = None
30       v.send_seq[1 * (M + 1) + j] = listen_rand_v[i]
31     else
32       v.listen_seq[1 * (M + 1) + j] = listen_rand_v[i]
33       v.send_seq[1 * (M + 1) + j] = send_rand_u[k]
34       k = k + 1
35     j = j + 2
36   j = j + 1
37   return v.listen_seq, v.send_seq

```

**Algorithm 8:** Create sending and listening channel sequences for SU  $v$ . The SUs in this network have two single function radios.

Input:  $v.C$ ,  $v.dist$ ,  $M$

Output:  $v.listen\_seq$ ,  $v.send\_seq$

```

1   $v.listen\_seq = []$ ,  $v.send\_seq = []$ 
2   $listen\_rand\_c = \text{copy } v.C$ 
3  if  $|v.C| < M$ :
4       $addtl\_chans = \text{randomly choose } (M - |v.C|) \text{ channels from}$ 
            $v.C$ 
5      append  $addtl\_chans$  to  $listen\_rand\_c$ 
6      randomize the order of  $listen\_rand\_c$ 
7
8       $i = 0$ 
9      while  $i < M$  do
10          $send\_rand\_c = \text{copy } v.C / listen\_rand\_c[i]$ 
11         if  $|v.C| < M$  then
12              $addtl\_chans = \text{randomly choose } (M - |v.C|) \text{ channels}$ 
                   from  $send\_rand\_c$ 
13             append  $addtl\_chans$  to  $send\_rand\_c$ 
14             randomize the order of  $send\_rand\_c$ 
15
16              $j = 0$ 
17              $k = 0$ 
18             while  $j < M + 1$  do
19                 if  $j = 0$  and  $(v.dist \% 2) = 1$  then
20                      $v.listen\_seq[i * (M + 1) + j] = \text{None}$ 
21                      $v.send\_seq[i * (M + 1) + j] = listen\_rand\_c[i]$ 
22                 else if  $j = 0$  and  $(v.dist \% 2) = 0$  then
23                      $v.listen\_seq[i * (M + 1) + j] = listen\_rand\_c[i]$ 
24                      $v.send\_seq[i * (M + 1) + j] = 0$ 
25                 else if  $j = M$  and  $(v.dist \% 2) = 1$  then
26                      $v.listen\_seq[i * (M + 1) + j] = listen\_rand\_c[i]$ 
27                      $v.send\_seq[i * (M + 1) + j] = \text{None}$ 
28                 else if  $j = M$  and  $(v.dist \% 2) = 0$  then
29                      $v.listen\_seq[i * (M + 1) + j] = \text{None}$ 
30                      $v.send\_seq[i * (M + 1) + j] = listen\_rand\_c[i]$ 
31                 else
32                      $v.listen\_seq[i * (M + 1) + j] = listen\_rand\_c[i]$ 
33                      $v.send\_seq[i * (M + 1) + j] = send\_rand\_c[k]$ 
34                      $k = k + 1$ 
35                  $j = j + 1$ 
36              $i = i + 1$ 
37         return  $v.listen\_seq$ ,  $v.send\_seq$ 

```



### 2.4.3.2.3 Sending and Listening Channel Sequence Example

SU  $v$  has the available channels set  $v.C = \{1, 2, 3\}$ , distance  $v.dist = 4$ , and  $M = 4$ , so the channel sequence length will be 20 time slots. Table 10 shows the listening and sending channel sequences produced by Algorithm 8. The silent time slots are denoted with '-'.  
*Theorem 5: In the proof of Theorem 6 it*

Table 10: Example of GCM two radio channel sequence for SUs with even distances

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Listen	3	3	3	3	-	2	2	2	2	-	1	1	1	1	-	2	2	2	2	-
Send	-	1	2	2	3	-	3	1	3	2	-	3	3	2	1	-	1	1	3	2

*same slot at the beginning or end of each set of  $M+1$  time slots. Tables 12 and 13 show the silent*

Table 11 shows the same example as above with an odd distance,  $v.dist = 5$ .  
*for each pair of*

Table 11: Example of GCM two radio channel sequence for SUs with odd distances

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Listen	-	3	3	3	3	-	2	2	2	2	-	1	1	1	1	-	2	2	2	2
Send	3	1	2	2	-	2	3	1	3	-	1	3	3	2	-	2	1	1	3	-

*Odd Listening Beginning*

Inspection of the silent time slots in the above tables shows that when an SU with an odd distance is sending to an SU with an even distance, the silent time slots are located at the end of each set of  $M+1$  time slots. When an SU with an even distance is sending to an SU with an odd distance, the silent time slots are located at the beginning of each set of  $M+1$  time slots.

### 2.4.3.2.4 Sending and Listening Channel Sequence Algorithm Analysis

*Theorem 6:* A pair of sending and receiving SUs with at least one available channel in common is guaranteed to select the same channel in at least one time slot in  $M \times (M+1)$  time slots.

*Proof:* Algorithm 8 is simply an extension of Algorithms 6 and 7, so the proof of *Theorem 6* is the same as that of *Theorem 5* as long as the silent time slots occur at the same time, which is analyzed in *Theorem 7*.

*Theorem 7:* The silent time slots occur at the same time for sending and listening SU pairs.

*Proof:* In any pair of sending and listening SUs, one SU must have an odd distance and one must have an even distance, as proven in *Theorem 4* in Section 2.3.3.4. Algorithm 8 places a silent time slot at the beginning or end of each set of  $M+1$  time slots. Tables 12 and 13 show the silent time slot locations. As shown in the tables, the location of the silent time slots for each pair of sending and listening SUs match.

*Table 12: Silent time slots when SUs with even distances send to SUs with odd distances*

Distance	Action	Silent Time Slot Location
Even	Sending	Beginning
Odd	Listening	Beginning

*Table 13: Silent time slots when SUs with odd distances send to SUs with even distances*

Distance	Action	Silent Time Slot Location
Odd	Sending	End
Even	Listening	End



### Chapter 3: Selection of Forwarding SU Sets

In this chapter, a method for decreasing the data gathering delay is proposed. The delay is the total number of time slots required for the entire data gathering operation to be completed. The data gathering delay can be decreased by intelligently selecting SUs to forward received messages to the next layer.

Consider a pair of SU layers: layer  $A$  sends to layer  $B$ . Recall that a send interval is a consecutive set of time slots in which the SUs in a sending layer of SUs send a message. Define the number of send intervals needed for layer  $A$  to send its messages to layer  $B$  as  $n$ . The number of send intervals needed for layer  $B$  to send the messages it received from layer  $A$  must be at least  $n$ . Depending on the network topology, the receiving SUs could experience an accumulation of waiting messages in their message queues, which results in a larger delay. This would result in the number of send intervals needed for layer  $B$  to send the messages it received from layer  $A$  to be greater than  $n$ .

The CR-AHSWN shown in Figure 22 demonstrates how waiting messages can accumulate. In this network, SU  $C$  has four messages and will attempt to send them to SUs  $A$  and  $B$ . SU  $D$  has two messages and will attempt to send them to SU  $B$ . Four send intervals are required to send  $C$  and  $D$ 's messages. If all of the messages are successfully transmitted to  $A$  and  $B$ , SU  $B$  will have received six messages: four messages from  $C$  and two messages from  $D$ . Therefore, six send intervals are required for  $B$  to transmit the messages it received. After SUs  $C$  and  $D$  have stopped sending messages, SU  $B$  still has two messages in its message queue that must be sent.

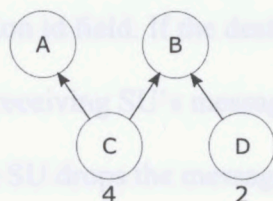


Figure 22: CR-AHSWN - increase in the data gathering delay

This accumulation of waiting messages at SU  $B$  is not necessary. SU  $C$ 's messages could be directed exclusively to SU  $A$ . When SU  $B$  receives messages from  $C$ ,  $B$  could just drop the messages rather than adding them to its message queue. Then, there would be no accumulation of waiting messages in  $B$ 's message queue. Both  $A$  and  $B$  would send each received message in the next action interval.

The goal of the algorithms presented in this chapter is to prevent unnecessary accumulations of messages in the receiving SU's message queues. In cases where a sending SU can send its messages to multiple receiving SUs, the sending SU should intelligently select one of the receiving SUs as the forwarding SU for each message. Multiple SUs could simultaneously send their messages to the same receiving SUs. Therefore, the forwarding SUs must be selected on a layer-by-layer basis in a manner that balances the number of waiting messages in the receiving layer's message queues.

The forwarding SU set selection process results in a set of forwarding SUs for each sending SU. Each forwarding SU in the set is the destination for a message sent by the sending SU, and the forwarding SUs must be used in the order that they were selected. Consider a sending SU with the forwarding SU set  $[A, B, A]$ , where  $A$  and  $B$  are receiving SUs. The sending SU will address its first message to  $A$ , its second message to  $B$ , and its third message to  $A$ . When an SU receives a



message, it will examine the destination id field. If the destination field contains the receiving SU's id, the message is added to the receiving SU's message queue. If the destination field does not contain the receiving SU's id, the SU drops the message.

Selecting SUs to forward the data gathering messages requires knowledge of the network topology. In this chapter, two selection methods are described. In the first, each SU knows the entire network topology and can determine its own forwarding SU set. This is a distributed algorithm, and each SU independently selects the forwarding SU for each of its messages. In the second method, the SUs do not have knowledge of the network topology. Instead, this method uses a series of broadcast and data gathering operations to gather the required information at the sink SU. Then, the sink SU can broadcast the network topology or the forwarding SU set selections for all of the SUs to the network.

### 3.1 Assumptions

The following assumptions are required for the algorithms described in this chapter:

- 1) The sending SUs and each of their selected forwarding SUs always have at least one channel in common.
- 2) The load-balanced semi-matching algorithm used in Algorithm 10 must iterate through the vertices in a consistent order when creating the semi-matching. Each SU must create the same forwarding SU sets, so every SU must iterate through the SUs in the same order.

## 3.2 Network Topology is Known by All SUs

### 3.2.1 Description

If all of the SUs in the network have knowledge of the entire network topology, each SU can determine its own forwarding SU set using Algorithm 9. First, the layers of the network are determined, where a layer is a set of SUs with the same graph hop distance from the sink SU. Then, the forwarding SU sets are determined in layer-by-layer manner, starting with the layer that is furthest from the sink SU, using Algorithm 10.

In Algorithm 10, each pair of sending and receiving layers is modeled as a bipartite graph. Each pair consists of a sending layer with distance  $d$  from the sink SU and its associated receiving layer with distance  $d-1$ . In a bipartite graph, the nodes are separated into two partitions, and each edge of the graph is adjacent to a node in each partition [13]. The nodes in the sending layer are placed in partition  $X$ , and the nodes in the receiving layer are placed in partition  $Y$ .

The forwarding SU sets for the sending layers are determined by iteratively constructing semi-matchings using a load-balanced semi-matching algorithm from [14] until there are no messages remaining at any of the SUs in the sending layer. Consider a bipartite graph with vertex partitions  $X$  and  $Y$ , and edges  $E$ . In a semi-matching, each vertex in partition  $X$  is incident on exactly one edge in  $E$  [14]. The vertices in partition  $Y$  may be incident on any number of edges. In the context of the forwarding SU selections, each sending SU will send a message to exactly one receiving SU. The receiving SUs can receive any number of messages.



Each iteration of the while-loop in Algorithm 10 models a send interval as described in Chapter 2. In a send interval, each sending SU sends one message from its message queue. The receiving SUs receive the messages and store them in their message queues. This process repeats until the sending SUs have sent all of their messages.

In order to decrease the data gathering delay, the number of messages in the receiving SUs' message queues must be balanced. The size of a receiving SU's message queue is affected by two values:

- 1) The number of messages received that are addressed to the receiving SU.
- 2) The number of messages that are currently waiting in the receiving SU's message queue.

The Forwarding SU Set Selection algorithm needs to take both of these values into account in order to balance the size of the message queues. This is accomplished in Algorithm 11 by adding vertices that represent each waiting message to the bipartite graph. Recall that the bipartite graph models a pair of sending and receiving layers; the sending SUs are in one partition, and the receiving SUs are in the other partition. For each message waiting in a receiving SU's message queue, a vertex is added to the sending partition of the bipartite graph. An edge is added between each "waiting message" vertex and the receiving SU where the message is waiting. These "waiting message" vertices represent the load that already exists on the receiving SU before any messages are sent in the current send interval. After the "waiting message" vertices and edges have been added to the bipartite graph, the load-balanced semi-matching is created.

The semi-matching consists of the edges selected to connect every vertex in the sending partition to a vertex in the receiving partition. Each sending SU is incident on one edge in the semi-matching. The receiving SU that is adjacent to a sending SU is the selected forwarding SU for the sending SU's message.

The while-loop in Algorithm 10 continues until no messages are left at the SUs in the sending layer. In each iteration of the while-loop, the "waiting message" vertices are added to the bipartite graph, the semi-matching is created, and the selected forwarding SUs are added to each sending SU's forwarding SU set. At the end of this process, each sending SU will have a forwarding SU set, with the forwarding SUs in the order in which they will be used. Then, when performing a data gathering, the sending SUs will use the forwarding SU sets to determine the destination id for each of their messages.



### 3.2.2 Algorithms

Table 14 shows the notations used in the algorithms shown in Sections 3.2.2 and 3.3.2.

*Table 14: Notation for the forwarding SU sets selection algorithms*

$G(V, E)$	CR-AHSWN $G$ with vertices $V$ and edges $E$
$u.dist$	Graph hop distance between SU $u$ and the sink SU
$u.num\_messages$	The number of messages received by SU $u$ .
$u.curr\_messages$	The number of messages received by SU $u$ in the current semi-matching iteration
$u.waiting\_messages$	The number of waiting messages in SU $u$ 's message queue
$BG(X, Y, E')$	A bipartite graph. $X$ is the set of sending SUs, $Y$ is the set of receiving SUs, $E'$ is the set of edges.
$e(u, v)$	A edge between SU $u$ and SU $v$ .
$E(u)$	The set of edges in $E$ that are incident to SU $u$ .

**Algorithm 9:** Construct the forwarding SU set for SU  $s$ Input:  $G(V, E)$ , SU  $s$ Output: set of forwarding SUs for all messages sent by SU  $s$ 


---

```

1  Perform a Breadth-First Search to set the graph hop distance,
    $u.dist$ , for each SU  $u$  in  $V$ 
2
3   $layers = []$  /* initialize layers array to hold set of SUs in
   each layer */
4
5  for each SU  $u$  in  $V$  do /* fill in layers array and initialize
   num_messages for each SU */
6   $layers[u.dist].add(u)$ 
7   $u.num\_messages = 1$ 
8
9   $forwarding\_selections = \{\}$  /* initialize hash table to hold the
   set of forwarding SU selections for
   each SU in the network */
10
11 for  $i$  in  $size(layers)-1$  to  $s.dist$  do
12  $layer\_forwarding\_selections =$ 
13    $create\_forwarding\_selections(G(V, E), layers, i)$ 
14
15   for each set in  $layer\_forwarding\_selections$  do
16      $forwarding\_selections.add(set)$ 
17
18 return  $forwarding\_selections[s]$ 

```

---

```

19   if  $x$  is in  $X$  then /*  $x$  is a sending SU */
20      $layer\_forwarding\_selections.add(x)$ 
21
22     /* message sent from  $x$ , so decrement num_messages */
23      $x.num\_messages = x.num\_messages - 1$ 
24
25     if  $x.num\_messages = 0$  then /*  $x$  is done sending */
26       remove  $x$  from  $X$ 
27       remove  $U(x)$  from  $E$ 
28
29     /* message received by  $y$ , so increment curr_messages */
30      $y.curr\_messages = y.curr\_messages + 1$ 
31
32   for each SU  $y$  in  $V$  do
33      $y.num\_messages = y.num\_messages +$ 
34        $(y.curr\_messages - y.waiting\_messages)$ 
35
36     /*  $y$  sends a message before listening for more messages,
37     so decrement and set the waiting_messages value */
38      $y.waiting\_messages = y.curr\_messages - 1$ 
39      $y.curr\_messages = 0$  /* reset for the next iteration */
40
41 return  $layer\_forwarding\_selections$ 

```



---

**Algorithm 10:** Create the forwarding SU set selections for a layer of graph  $G$

---

Input:  $G(V, E)$ , *layers* array, current *layer*

Output: *layer\_forwarding\_selections*, an array of forwarding SU sets for each SU in the layer

---

**create\_forwarding\_selections( $G(V, E)$ , *layers*, *layer*):**

```

1  X = layers[layer] /* sending SUs */
2  Y = layers[layer-1] /* receiving SUs */
3  layer_forwarding_selections = {} /* initialize hash table to hold
   the set of forwarding SUs
   for each SU in the layer */
4  /* get edges between X and Y */
5  for SU x in X do
6      for e(x, y) in E(x) do
7          if y.dist < x.dist then
8              E'.add(e(x, y))
9
10     for SU y in Y do
11         y.curr_messages = 0
12         y.waiting_messages = y.num_messages
13
14     while X is not empty do
15         BG(X', Y, E'') = create_bipartite_graph(X, Y, E')
16         M = compute a semi-matching of X to Y on BG(X', Y, E'') using
           a load-balanced semi-matching algorithm described in [14]
17
18         for each edge m(x, y) in M do
19             if x is in X then /* x is a sending SU */
20                 layer_forwarding_selections[x].add(y)
21
22                 /* message sent from x, so decrement num_messages */
23                 x.num_messages = x.num_messages - 1
24
25                 if x.num_messages = 0 then /* x is done sending */
26                     remove x from X
27                     remove E'(x) from E'
28
29                 /* message received by y, so increment curr_messages */
30                 y.curr_messages = y.curr_messages + 1
31
32         for each SU y in Y do
33             y.num_messages = y.num_messages +
               (y.curr_messages - y.waiting_messages)
34
35             /* y sends a message before listening for more messages,
           so decrement and set the waiting_messages value*/
36             y.waiting_messages = y.curr_messages - 1
37             y.curr_messages = 0 /* reset for the next iteration */
38
39     return layer_forwarding_selections

```

---

**Algorithm 11:** Create a bipartite graph for two adjacent layers of graph  $G$

---

Input: the sending SU partition  $X$ , the listening SU partition  $Y$ , and the edges between  $X$  and  $Y$ ,  $E'$   
 Output: a bipartite graph  $BG(X', Y, E'')$  with the waiting messages for each SU in  $Y$  represented by vertices added to  $X$

---

**create\_bipartite\_graph( $X, Y, E'$ ):**

```

1  X' = X
2  E'' = E'  /* initial value of a new_messages is set to one, which represents the SU's own
3
4  /* add vertices for all messages waiting in queues in Y */
5  for SU y in Y do
6      for i in y.waiting_messages do
7          create vertex y(i)
8          X'.add(y(i))
9          E''.add(e(y(i), y))
10
11 return BG(X', Y, E'')
```

### 3.2.3 Example

The Forwarding SU Set Selection Algorithm is demonstrated using the CR-AHSWN shown in Figure 23.

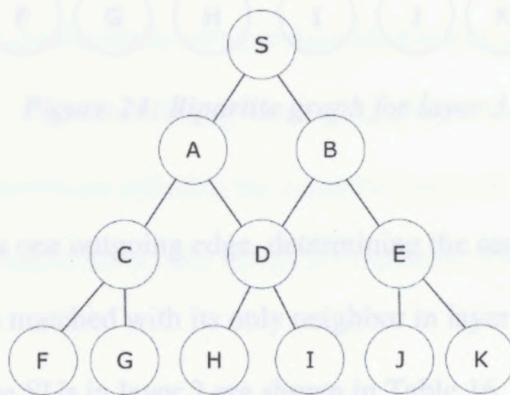


Figure 23: CR-AHSWN - Forwarding SU Set Selection algorithm example

First, the distance of each SU is determined, and the SUs are placed in the appropriate layer, as shown in Table 15.



Table 15: Layers array for the CR-AHSWN in Figure 23

Layer	SUs
0	[ S ]
1	[ A, B ]
2	[ C, D, E ]
3	[ F, G, H, I, J, K ]

The bipartite graph for layer 2 and its receiving layer, layer 1, is shown in Figure 25. The  $num\_messages$  value for each SU is also shown in the figure. Each of the sending SUs, C, D, and E, must send three messages. Each of the receiving SUs has one message waiting in its message queue.

The forwarding SU sets are determined in a layer-by-layer manner, starting with the outermost layer. The bipartite graph for layer 3 is shown in Figure 24. The bipartite graph consists of the SUs in layer 3, the SUs in layer 2, and the edges between the SUs in layers 2 and 3.

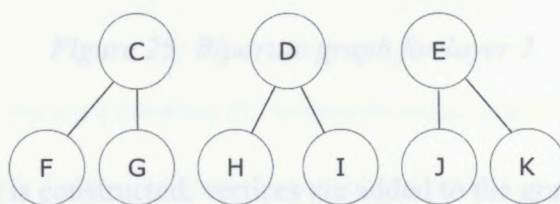


Figure 24: Bipartite graph for layer 3

Since each SU in layer 3 has one outgoing edge, determining the semi-matching for this graph is trivial; each SU in layer 3 is matched with its only neighbor in layer 2. The current  $num\_messages$  values for the SUs in layer 2 are shown in Table 16.

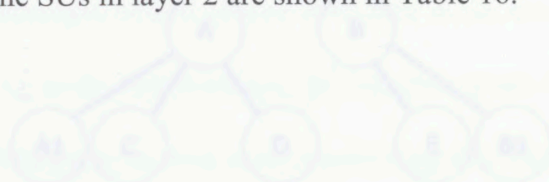


Figure 26: Bipartite graph for layer 2's first semi-matching

Table 16: Messages values for layer 2

SU	num_messages
C	3
D	3
E	3

The bipartite graph for layer 2 and its receiving layer, layer 1, is shown in Figure 25. The *num\_messages* value for each SU is also shown in the figure. Each of the sending SUs, C, D, and E, must send three messages. Each of the receiving SUs has one message waiting in its message queue.

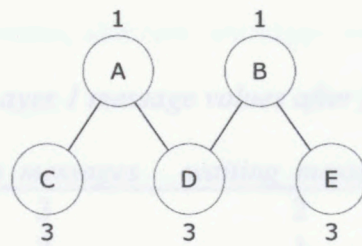


Figure 25: Bipartite graph for layer 2

Before the semi-matching is constructed, vertices are added to the graph for each message waiting at the receiving SUs. For each waiting message, a new vertex and an edge between the new vertex and the receiving SU are added to the bipartite graph. In Figure 26, vertex *A1* represents the message waiting at SU *A*, and vertex *B1* represents the message waiting at vertex *B*.

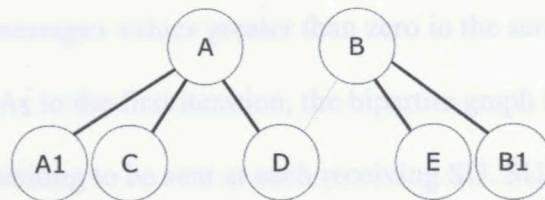


Figure 26: Bipartite graph for layer 2's first semi-matching



The semi-matching is constructed using a load-balanced semi-matching algorithm described in [14], and the edges selected for the semi-matching are shown in bold in Figure 26. Three messages are directed to SU  $A$ , including  $A$ 's waiting message,  $A1$ . Two messages are directed to SU  $B$ , including  $B$ 's waiting message,  $B1$ .

At the end of the first iteration, the  $num\_messages$ ,  $waiting\_messages$ , and  $curr\_messages$  values for each receiving SU are updated. The updated values are shown in Table 17. In all of the layer 1 message tables in this section, the values shown are the values before  $curr\_messages$  is reset to zero in line 37 of Algorithm 10.

Table 17: Layer 1 message values after first iteration

Receiving SU	$num\_messages$	$waiting\_messages$	$curr\_messages$
$A$	3	2	3
$B$	2	1	2

The  $num\_messages$  value for each sending SU is decremented, and the forwarding SU set for each sending SU is updated. Table 18 shows these updated values.

Table 18: Layer 2 messages and forwarding SU sets after first iteration

Sending SU	$num\_messages$	Forwarding SU Set
$C$	2	[ $A$ ]
$D$	2	[ $A$ ]
$E$	2	[ $B$ ]

There are SUs with  $num\_messages$  values greater than zero in the sending partition, so another semi-matching is created. As in the first iteration, the bipartite graph is constructed by adding vertices for the messages waiting to be sent at each receiving SU. SU  $A$  has two messages waiting, so vertices  $A1$  and  $A2$  are added to the graph. Edges between the new vertices and SU  $A$  are also added. SU  $B$  has one message waiting, so vertex  $B1$  is added to the graph, along with an

edge between the new vertex and SU  $B$ . The bipartite graph for the second semi-matching is shown in Figure 27, and the edges selected for the load-balanced semi-matching are shown in bold.

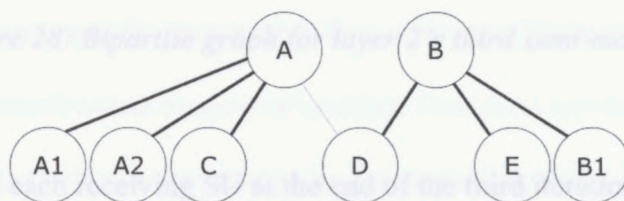


Figure 27: Bipartite graph for layer 2's second semi-matching

The *num\_messages*, *waiting\_messages*, and *curr\_messages* values for each receiving SU after the second iteration are shown in Table 19.

Table 19: Layer 1 message values after second iteration

Receiving SU	<i>num_messages</i>	<i>waiting_messages</i>	<i>curr_messages</i>
A	4	2	3
B	4	2	3

The updated *num\_messages* values and the forwarding SU sets for the sending SUs after the second iteration are shown in Table 20.

Table 20: Layer 2 messages and forwarding SU sets after second iteration

Sending SU	<i>num_messages</i>	Forwarding SU Set
C	1	[ A, A ]
D	1	[ A, B ]
E	1	[ B, B ]

Sending SUs  $C$ ,  $D$ , and  $E$  each have one remaining message to send, so another iteration is required. Each receiving SU has two waiting messages, so four vertices,  $A1$ ,  $A2$ ,  $B1$ , and  $B2$ , are added to the graph, as shown in Figure 28. The edges selected for the semi-matching are shown in bold.



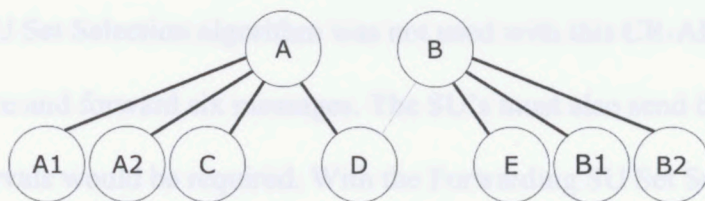


Figure 28: Bipartite graph for layer 2's third semi-matching

The message values for each receiving SU at the end of the third iteration are shown in Table 21.

Table 21: Layer 1 message values after third iteration

Receiving SU	<i>num_messages</i>	<i>waiting_messages</i>	<i>curr_messages</i>
A	6	3	4
B	5	2	3

The sending SUs' updated *num\_messages* values and the forwarding SU sets after the third iteration are shown in Table 22.

Table 22: Layer 2 messages and forwarding SU sets after third iteration

Sending SU	<i>num_messages</i>	Forwarding SU Set
C	0	[ A, A, A ]
D	0	[ A, B, A ]
E	0	[ B, B, B ]

After the third iteration, the value of *num\_messages* for each sending SU is zero, so all three sending SUs are removed from the bipartite graph. The sending partition is now empty, and the selection of the forwarding SU sets for layer 2 is complete. The right column of Table 22 shows the forwarding SU sets for each message sent by the SUs in layer 2.

The SUs in layer 1, A and B, both send their messages to the sink SU, S. Therefore, forwarding SU sets are not needed for the SUs in layer 1.

If the Forwarding SU Set Selection algorithm was not used with this CR-AHSWN, each SU in layer 1 would receive and forward six messages. The SU's must also send their own messages, so seven action intervals would be required. With the Forwarding SU Set Selection algorithm,  $A$  needs to send six messages, and  $B$  needs to send five messages. While this may seem like a small decrease, recall that a send interval consists of multiple time slots and that both a silent interval and a listen interval occur between send intervals. Preventing an unnecessary send interval could result in a delay reduction of many time slots.

### 3.2.4 Analysis *Topology is Determined by the Sink SU*

*Theorem 8:* In each iteration of the while-loop in Algorithm 10, a load-balanced semi-matching is constructed, where the load is defined as the number of messages in a receiving SU's message queue.

*Proof:* At the end of a listen interval, the number of messages in a receiving SU's message queue is equal to the sum of two values:

- 1) The number of messages that were already waiting in the queue at the beginning of the listen interval.
- 2) The number of messages that were received and added to the queue during the listen interval.

Each iteration of the while-loop in Algorithm 10 represents an action interval. In each iteration of the while-loop, vertices and edges are added to the bipartite graph. Initially, the bipartite graph consists of a sending SU partition and a receiving SU partition. A vertex is added to the sending partition for each message that is currently waiting in a receiving SU's message queue. An edge



is added between each new “waiting message” vertex and its associated receiving SU. The semi-matching is then created using a load-balancing semi-matching algorithm described in [14]; the proofs for the load balancing property of these semi-matching algorithms are provided in [14]. The semi-matching algorithm provides a load-balanced semi-matching of the bipartite graph that includes vertices for both the waiting messages and the received messages. Therefore, the load that is being balanced is the number of messages that will be in the receiving SUs’ queues at the end of the listen interval.

### 3.3 Network Topology is Determined by the Sink SU

#### 3.3.1 Description

If the network topology is not known by all SUs in the network, the sink SU must gather messages from all of the SUs and, using these messages, construct the network topology. Then, the sink SU can send the network topology to the other SUs in the network. Alternatively, the sink SU can select the forwarding SU sets for all of the SUs in the network using Algorithm 13 and broadcast the forwarding SU sets selections to the other SUs in the network.

The following steps are performed to construct the network topology at the sink SU:

- 1) Conduct an initialization broadcast operation as described in Section 2.2.
- 2) Conduct a data gathering operation using the action selection algorithm and a channel selection algorithm described in Chapter 2. Each data gathering message must include a *first\_recipient* field that stores the id of the first SU to receive the message. The messages must also include a *source* field that stores the id of the source SU.

- 3) After the data gathering operation is complete, the sink SU constructs the network topology using Algorithm 12. The network topology created by Algorithm 12 will include only the edges that represent a message transmission that occurs in the data gathering operation. Therefore, the resulting network topology will not include any lateral edges between SUs in the same layer.
- 4) The sink SU sends a broadcast message to the other SUs in the network. This message contains one of the following:
  - The constructed network topology. Each SU then uses Algorithm 9 to determine its own forwarding SU set, as described in the previous section.
  - The selected forwarding SU sets for all nodes in the network. The forwarding SU sets for all SUs in the network can be determined by the sink SU using Algorithm 13.

### 3.3.2 Algorithms

The notations for these algorithms are shown in Section 3.2.2.

---

**Algorithm 12:** Create the network topology at the sink SU using the received data gathering messages.

Input: *received\_messages*, the set of all received data gathering messages

Output:  $G(V, E)$ , a graph with nodes  $V$  and edges  $E$

---

```

1  for each message in received_messages do
2    if message.source not in  $V$  then
3       $V.add(message.source)$ 
4    if message.first_recipient not in  $V$  then
5       $V.add(message.first_recipient)$ 
6
7       $E.add(e(message.source, message.first_recipient))$ 
8
9  return  $G(V, E)$ 

```



### 3.3.3 Example

Algorithm 13 is nearly identical to Algorithm 9 in Section 3.2.2. The sink SU uses this algorithm to determine forwarding SU sets for all of the SUs in the network. Therefore, this algorithm iterates through all the layers in the network and returns the forwarding SU sets for all of the SUs in the network.

---

**Algorithm 13:** Select the forwarding SU sets for all SUs in the network

---

Input:  $G(V, E)$ ,

Output: *forwarding\_selections*, the sets of forwarding SUs for all messages sent by all SUs in the network

---

```

1   Perform a Breadth-First Search to set the graph hop distance,
    u.dist, for each SU u in V
2
3   layers = [] /* initialize layers array to hold set of SUs in
                each layer */
4
5   for each SU u in V do /* fill in layers array and initialize
                            num_messages for each SU */
6       layers[u.dist].add(u)
7       u.num_messages = 1
8
9   forwarding_selections = {} /* initialize hash table to hold the
                                set of forwarding SU selections for
                                each SU in the network */
10
11  for i in size(layers)-1 to 1 do
12      layer_forwarding_selections =
          create_forwarding_selections(G(V, E), layers, i)
13
14  for each set in layer_forwarding_selections do
15      forwarding_selections.add(set)
16
17
18  return forwarding_selections

```

after the first four messages are processed is shown in Figure 10.

### 3.3.3 Example

The network topology of the CR-AHSWN shown in Figure 29 will be determined using Algorithm 12.

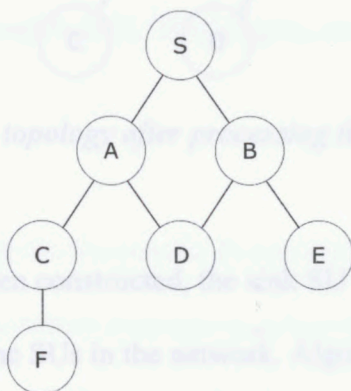


Figure 29: CR-AHSWN - construction of the network topology example

The messages shown in Table 23 are received by the sink SU.

Table 23: Messages received by the sink SU in the CR-AHSWN in Figure 29

Message id	message.source	message.first_recipient
1	A	S
2	B	S
3	C	A
4	D	B
5	D	A
6	E	B
7	F	C

The sink SU uses Algorithm 12 to produce the network topology. The algorithm iterates through each message, adding the source and first recipient SUs to the graph's node set and adding the edge between the source SU and first recipient SU to the graph's edge set. The state of the graph after the first four messages are processed is shown in Figure 30.



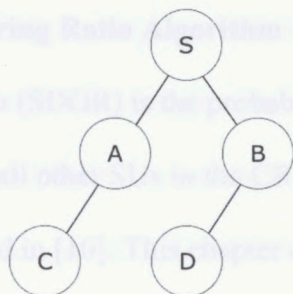


Figure 30: Network topology after processing the first four messages

After the network topology has been constructed, the sink SU can use Algorithm 13 to determine the forwarding SU sets for all of the SUs in the network. Algorithm 13 is nearly identical to Algorithm 9. An example of the use of Algorithm 9 is provided in Section 3.2.3.

### 3.3.4 Analysis

Algorithms 9 and 13 are nearly identical; Algorithm 9 returns the forwarding SU set for a single SU while Algorithm 13 returns the forwarding SU sets for all of the SUs in the network. Therefore, Theorem 8, shown in Section 3.2.4, also applies to Algorithm 13.

#### Chapter 4: Successful Data Gathering Ratio Algorithm

The Successful Data Gathering Ratio (SDGR) is the probability that the sink SU will successfully receive messages from all other SUs in the CR-AHSWN. This is analogous to the Successful Broadcast Ratio described in [10]. This chapter develops an algorithm that calculates an estimate of the SDGR.

In [10], the difficulty of calculating the Successful Broadcast Ratio is described. The calculation requires identifying all possible broadcast message propagation scenarios, which is an extremely challenging task. The task becomes nearly impossible for even small networks, such as a 3x3 grid network [10]. An algorithm that simplifies the calculation of the Successful Broadcast Ratio is proposed in [10], and the evaluation provided in [10] shows that the algorithm's estimates were reasonably close to the results obtained from simulations.

1) Remove all SUs that are not involved in transmitting the messages from layer  $l$  to the sink.

The calculation of the SDGR suffers from the same challenges as the calculation of the Successful Broadcast Ratio. However, the success ratio algorithm proposed in [10] cannot be used to estimate the SDGR. In the broadcast operation, a single source SU sends a message that propagates through the network to every other SU. Therefore, an SU sends the broadcast message to its neighbors only once. In other words, if the network is modeled as a graph, the broadcast message needs to travel over an edge only once. The algorithm in [10] capitalizes on this property of the broadcast operation by decomposing the graph into smaller, simpler graphs. The graph is decomposed until the Successful Broadcast Ratio for the decomposed graph is easy to calculate.

ancestors.



In the data gathering operation, multiple messages use the same edges to get to the sink SU, so the graph cannot be decomposed into smaller, simpler graphs. Instead, the SDGR algorithm proposed in this section calculates the SDGR for each layer of the graph. A layer is a set of SUs with the same graph hop distance to the sink SU. The SDGR for the entire network is the product of the SDGRs of all layers in the network.

The calculation of the SDGR for the first layer, the set of SUs that send directly to the sink SU, is straightforward. It is the joint probability that all SUs in the first layer successfully send their messages to the sink SU within a send interval.

#### 4.1 Assumptions

For the remaining layers, a recursive algorithm is used to calculate the SDGR of each layer. The following steps are used to calculate the SDGR of layer  $i$ , where  $i$  is greater than one:

- 1) Remove all SUs that are not involved in transmitting the messages from layer  $i$  to the sink SU. Specifically, remove all leaf SUs in layers less than  $i$  since they are not on the paths between the SUs in layer  $i$  and the sink SU. Also, remove all SUs in layers greater than  $i$ .
- 2) Initialize arrays that will hold the list of parent SUs for each SU in layer  $i-1$ , where a parent SU is an SU that sends to a receiving SU. Initialize arrays that will hold the probabilities of reaching the sink SU for each SU in layer  $i$ .
- 3) For each SU  $u$  in layer  $i-1$ , choose a parent SU  $v$  from  $u$ 's parent list. Calculate the probability of parent  $v$ 's message reaching the sink SU via  $u$ . Remove  $v$  from  $u$ 's parent list, and add the calculated probability to  $v$ 's probability list.
- 4) Remove all SUs in layer  $i-1$  that have empty parent lists. Also, remove their leaf ancestors.

- 5) Repeat from step 3 until no SUs are left in layer  $i-1$ .
- 6) Use the probability array for the SUs in layer  $i$  to calculate the SDGR for layer  $i$ .

Determining the probability of a message from SU  $v$  reaching the sink SU and the calculation of the joint probability for the first layer require single hop probabilities. The single hop probability is the probability that a message successfully transmits between two SUs without collision. These probabilities differ based on the specific channel selection method used. Section 4.5 describes the single hop probabilities for the random and GCM channel selection methods.

#### 4.1 Assumptions

The SDGR algorithm can be used to evaluate data gathering protocols that have the following properties:

- 1) The data gathering protocol must work in a layer-by-layer fashion as described in Section 2.3.
- 2) The channel selection method must allow the probability of a single-hop message transmission between two SUs to be calculated.

The input to the SDGR algorithm is the network graph,  $G(V, E)$ . Because messages are sent between layers, graph  $G(V, E)$  must not have any lateral edges between SUs with the same distance. Any lateral edges must be removed from the input graph.



## 4.2 SDGR Estimate Calculation Algorithm

Table 24 shows the notations used in the SDGR algorithm.

Table 24: Notation for the SDGR algorithm

$G(V, E)$	CR-AHSWN $G$ with vertices $V$ and edges $E$
$G(V, E).sink$	Sink SU for CR-AHSWN $G$
$SDGR$	SDGR for CR-AHSWN $G$
$v.dist$	Graph hop distance between SU $v$ and the sink SU
$layer.SR$	SDGR for a single layer in network $G$
$layer.E$	Set of edges for a single layer
$layer.V$	Set of vertices for a single layer
$layer.probabilities$	Array of probabilities for each SU in a layer
$layer.parents$	Array of parent SUs for each SU in a layer. A parent SU is an SU that sends to another SU.
$E(v)$	Set of edges in $G$ that are incident to SU $v$
$e(u, v)$	Edge between SU $u$ and SU $v$
$P(u, v)_G$	Probability of a successful single hop message transmission from SU $u$ to SU $v$ in network $G$

**Algorithm 14:** Calculate the SDGR estimate for the input network

Input:  $G(V, E)$   
 Output: SDGR estimate

```

1  SDGR = 1 /* initialize value */
2  layers = [] /* initialize array to hold set of SUs for each
                layer */
3  for v in V do
4      layers[v.dist].add(v) /* add each SU to the correct layer
                             based on its distance */
5
6  for i in range 1 to length(layers)-1 do
7      if i = 1 then
8          layer.SR = joint probability of all layer 1 SUs
9      else
10         /* initialize layer's data structures */
11         layer.probabilities = {} /* initialize hash table to hold
                                     probabilities for each SU in
                                     layer i */
12         layer.parents = {} /* initialize hash table to hold
                               parents for each SU in layer i */
13         layer.V = V; layer.E = E
14
15         /* remove SUs that are further from the sink than i */
16         for j in range i+1 to length(layers)-1 do
17             for v in layers[j] do
18                 layer.V = layer.V - v
19                 layer.E = layer.E - E(v)
20
21         /* initialize probabilities for SUs in layer i */
22         for v in layers[i] do
23             layer.probabilities[v] = [ ]
24
25         /* initialize parent sets for SUs in layer i-1 */
26         for v in layers[i-1] do
27             for e(v, u) in E(v) do
28                 if u.dist > v.dist then /* u is a parent of v */
29                     layer.parents[v].add(u)
30
31         layer.SR = SR(G(layer.V, layer.E), i,
                       layer.probabilities, layer.parents, layers)
32
33     SDGR = SDGR * layer.SR
34 return SDGR

```



---

**Algorithm 15:** Recursively calculate the success ratio for a specific layer in the input graph the sink SU via the specified receiving SU.

Input:  $G(V, E)$ , current layer, *layer.probabilities* hash table, *layer.parents* hash table, *layers* array  
 Output: success ratio for the current layer

---

**SR( $G(V, E)$ , layer, *layer.probabilities*, *layer.parents*, *layers*):**

```

1  for r in layers[layer-1] do
2  /* remove completed SUs */
3  if layer.parents[r] is empty then
4  else
5  remove r and its leaf ancestors from G and layers
6  return total_layer_SR(layer.parents[r], G(V, E))
7  if layers[layer-1] is empty then
8  return total_layer_SR(layer.probabilities)
9  else
10  for r in layers[layer-1] do
11  randomly choose SU s from layer.parents[r]
12  layer.probabilities[s].append(calculate_SU_P(s, r,
13  G(V, E))
14  layer.parents[r].remove(s)
15  return SR(G(V, E), layer, layer.probabilities, layer.parents,
16  layers)

```

---

**Algorithm 16:** Calculate the success ratio for a specific layer after all SU probabilities have been stored in the *layer.probabilities* hash table

Input: *layer.probabilities* hash table

Output: success ratio for the layer

---

**total\_layer\_SR(*layer.probabilities*):**

```

1  P = 1
2  for i in range 0 to length(layer.probabilities)-1 do
3  if length(layer.probabilities[i]) == 1 then
4  P *= layer.probabilities[i]
5  else
6  prob_fail = 1
7  for j in range 0 to length(layer.probabilities[i])-1 do
8  prob_fail *= (1 - layer.probabilities[i][j])
9  P *= (1 - prob_fail)
10 return P

```

---

---

**Algorithm 17:** Calculate the probability that the sending SU's message successfully reaches the sink SU via the specified receiving SU.

Input:  $send\_su, rec\_su, G(V, E)$

Output: probability that  $send\_su$ 's message reaches the sink SU via  $rec\_su$

---

**calculate\_SU\_P( $send\_su, rec\_su, G(V, E)$ ):**

```

1  if  $rec\_su = G(V, E).sink$  then
2      return  $P(send\_su, rec\_su)_G$ 
3  else
4      return  $P(send\_su, rec\_su)_G * calculate\_SU\_P2(rec\_su, G(V, E))$ 

```

---

**Algorithm 18:** Calculate the probability that the message received by  $rec\_su$  will reach the sink SU.

Input:  $rec\_su, G(V, E)$

Output: probability that the message received by  $rec\_su$  will reach the sink SU

---

**calculate\_SU\_P2( $rec\_su, G(V, E)$ ):**

```

1   $neighbors = []$ 
2  for  $e(u, r)$  in  $E(rec\_su)$  do
3      if  $u.dist < rec\_su.dist$  then
4           $neighbors.append(u)$ 
5   $fail = 1$ 
6  for  $n$  in  $neighbors$  do
7       $fail *= (1 - calculate\_SU\_P(rec\_su, n, G(V, E)))$ 
8  return  $1 - fail$ 

```

### 4.3 SDGR Estimate Algorithm Example

The SDGR algorithm is applied to network  $G$  shown in Figure 31. The single hop probabilities depend on the current graph, and the subscripts  $G$ ,  $G1$ , and  $G2$  indicate which graph is used in the probability calculations. As the algorithm proceeds, nodes and their edges will be removed from the graph, which changes the single hop probability values.



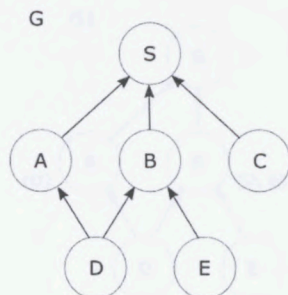


Figure 31: CR-AHSWN - SDGR algorithm example

First, the *layers* array is populated by placing each SU into the appropriate layer:

Table 25: Layers array for the CR-AHSWN in Figure 31

Layer	SUs
0	[ S ]
1	[ A, B, C ]
2	[ D, E ]

Next, the SDGR for each layer is calculated. For layer 1, the SDGR is simply the joint probability that messages from SUs  $A$ ,  $B$ , and  $C$  reach SU  $S$ , which is denoted by  $P([A, B, C], S)_G$ .

Layer 2 begins by initializing the probability arrays for the SUs in layer 2,  $D$  and  $E$ . The parent arrays for the SUs in layer 1 are also initialized. SU  $A$  has a single parent,  $D$ , and  $B$  has two parents,  $D$  and  $E$ . Since  $C$  has no parents, it is removed from the network, creating graph  $G_1$  shown in Figure 32. The parent lists for SUs  $A$  and  $B$  are shown next to the SUs.

Figure 32: CR-AHSWN - calculation of the Layer 2 SDGR, network reduction

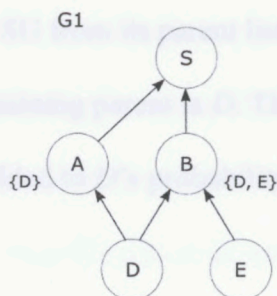


Figure 32: CR-AHSWN - calculation of the layer 2 SDGR, first iteration

For each SU in layer 1, an SU from its parent list is selected. For SU  $A$ ,  $D$  is chosen. The probability that  $D$ 's message reaches the sink SU via  $A$  is calculated and stored in  $D$ 's probability array.  $D$  is then removed from  $A$ 's parent list. Similarly, for  $B$ ,  $E$  is chosen, and the probability that  $E$ 's message reaches the sink via  $B$ , is calculated and stored. Then,  $E$  is removed from  $B$ 's parent list. The current probability arrays are shown in Table 26.

Table 26: Layer 2 probability array after the first recursion of the SDGR algorithm

SU	Probability Array
D	[ $P(D, A)_{G1} \times P(A, S)_{G1}$ ]
E	[ $P(E, B)_{G1} \times P(B, S)_{G1}$ ]

SU  $A$ 's parent list is now empty, so it is removed from the network, creating  $G2$ , which is shown in Figure 33.

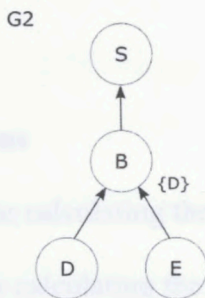


Figure 33: CR-AHSWN - calculation of the layer 2 SDGR, second iteration



As before, for each SU in layer 1, an SU from its parent list is selected. SU  $B$  is the only SU remaining in layer 1, and its only remaining parent is  $D$ . The probability of  $D$ 's message reaching the sink SU via  $B$  is calculated and added to  $D$ 's probability array.  $D$  is removed from  $B$ 's parent list.

The current probability arrays are shown in Table 27.

Table 27: Layer 2 probability array after the second recursion of the SDGR algorithm

SU	Probability Array
D	[ ( $P(D, A)_{G1} \times P(A, S)_{G1}$ ), ( $P(D, B)_{G2} \times P(B, S)_{G2}$ ) ]
E	[ ( $P(E, B)_{G1} \times P(B, S)_{G1}$ ) ]

The parent lists for all of the SUs in layer 1 are now empty, so the total success ratio for the layer is calculated using the *total\_layer\_SR(layer.probabilities)* function. This function returns the value shown in Equation 1.

$$SDGR_{layer\ 2} = (P(E, B)_{G1} \times P(B, S)_{G1}) \times \left[ 1 - \left( (1 - (P(D, A)_{G1} \times P(A, S)_{G1})) \times (1 - (P(D, B)_{G2} \times P(B, S)_{G2})) \right) \right] \quad [1]$$

Finally, the SDGR for the entire network is the product of the SDGRs for layer 1 and layer 2:

$$SDGR_G = P([A, B, C], S)_G \times SDGR_{layer\ 2} \quad [2]$$

#### 4.4 Single Hop Probability Calculations

The single hop probability is required for calculating the probability that an SU's message successfully reaches the sink SU and for calculating the joint probability that messages from all the SUs in the first layer will reach the sink SU.

#### 4.4.1 Random Channel Selection

The probability that a sending SU will select the same channel as the receiving SU is calculated as shown in Equation 3, which is obtained from [10]. In this equation,  $Z_{AS}$  is the number of channels that sending SU  $A$  and receiving SU  $S$  have in common.  $N_A$  is the number of available channels for SU  $A$ , while  $N_S$  is the number of available channels for SU  $S$  [10].

$$p(A, S)_{rand} = \frac{Z_{AS}}{N_A N_S} \quad [3]$$

The probability that sending SU  $A$ 's message will collide with the message sent by another SU at receiving SU  $S$  is shown in Equation 4.  $U$  is the set of SUs that send to SU  $S$ .  $U_{i,A}$  is the set of size  $i$  subsets of  $U$  that include SU  $A$ .  $V$  is an element of  $U_{i,A}$ , and  $v$  is an individual SU in  $V$ .  $Z_{VS}$  is the number of channels that the SUs in set  $V$  and SU  $S$  have in common.  $N_S$  is the number of available channels for SU  $S$ , while  $N_v$  is the number of available channels for SU  $v$ .

$$q(A, S)_{rand} = \sum_{i=2}^{|U|} (-1)^i \sum_{V \in U_{i,A}} \frac{Z_{VS}}{N_S \times \prod_{v \in V} N_v} \quad [4]$$

The probability that sending SU  $A$ 's message will be successfully transmitted to SU  $S$  within a send interval of  $Sr$  time slots is shown in Equation 5.

$$P(A, S)_{rand} = 1 - (1 - (p(A, S)_{rand} - q(A, S)_{rand}))^{Sr} \quad [5]$$

#### 4.4.2 Guaranteed Channel Match Channel Selection

In the GCM channel selection algorithms, a channel match between the sending SU  $A$  and receiving SU  $S$  is guaranteed to occur  $Z_{AS}$  times in  $M^2$  time slots, where  $Z_{AS}$  is the number of channels that SUs  $A$  and  $S$  have in common. However, the message transmission could still fail



due to a collision. The probability of a successful message transmission between sending SU  $A$  and receiving SU  $S$  is shown in Equation 6. In this equation,  $C_A$  and  $C_S$  represent the available channel sets for SUs  $A$  and  $S$ , respectively, while  $c$  is an element of the combined sets.  $U_{c, \neg A}$  is the set of SUs that send to SU  $S$  that have channel  $c$  in their available channel sets, excluding SU  $A$ .  $U_{i, c, \neg A}$  is the set of size  $i$  subsets of  $U_{c, \neg A}$ .  $V$  is an element of  $U_{i, c, \neg A}$ , and  $v$  is an individual SU in  $V$ .  $N_v$  is the number of available channels for SU  $v$ .

$$P(A, S)_{GCM} = 1 - \prod_{c \in (C_A \cap C_S)} \left( \sum_{i=1}^{|U_{c, \neg A}|} (-1)^{i-1} \sum_{V \in U_{i, c, \neg A}} \frac{1}{\prod_{v \in V} N_v} \right) \quad [6]$$

Table 28 shows the simulation and SDGR algorithm results for both the random and GCM

#### 4.5 SDGR Algorithm Evaluation

The performance of the SDGR algorithm was evaluated using simulations. These simulations were conducted with the following assumptions:

- 1) All SUs in the network have the same available channel sets and the channel sets do not change.
- 2) The SUs use the protocol described in chapter 2 to perform the data gathering operation.
- 3) When a pair of receiving and sending SUs select the same channel and no collision occurs, the message transmission is successful.

The simulations modeled only channel matching and collisions to determine if a message transmission was successful. Physical message transmissions were not simulated. PU activity was not simulated because calculation of the SDGR requires stable available channel sets.

random and GCM channel selection methods for this CR-NESWN. For random channel

This evaluation used the network shown in Figure 34. In this CR-AHSWN, the sink SU for the data gathering operation is SU S.

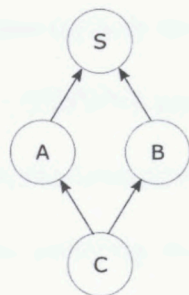


Figure 34: CR-AHSWN - evaluation of the SDGR algorithm

Table 28 shows the simulation and SDGR algorithm results for both the random and GCM channel selection methods. Four small channel set sizes were evaluated. In these evaluations, the length of the action interval is  $M^2$ , where  $M$  is the size of the available channel sets. The devices are assumed to have one radio, so the action selection and GCM channel selection algorithms for one radio were used. The simulation results are the average of 100,000 trials.

Table 28: SDGR values determined using simulations and the SDGR algorithm

Channel Set Size	Random Channel Selection			GCM Channel Selection		
	Sim.	SDGR Alg.	Percent Diff.	Sim.	SDGR Alg.	Percent Diff.
2	0.40	0.37	-7.5	0.56	0.53	-5.3
3	0.79	0.78	-1.3	0.93	0.93	0.0
4	0.93	0.93	0.0	0.99	0.99	0.0
5	0.97	0.97	0.0	1.0	1.0	0.0

These results demonstrate that the SDGR algorithm provides reasonable estimates for both the random and GCM channel selection methods for this CR-AHSWN. For random channel



selection the maximum difference between the simulation results and SDGR algorithm results is -7.5%, and for GCM channel selection the maximum difference is -5.3%.

In this thesis, several distributed algorithms for both SU access selection and channel selection were developed for data gathering in CR-AHSWNs operating under practical conditions. Through theoretical proofs and examples, the correctness of these algorithms was analyzed. An innovative algorithm that reduces the data gathering delay was also presented. This algorithm reduces the data gathering delay by intelligently selecting sets of forwarding SUs for each sending SU. Furthermore, another novel algorithm was presented which estimates the SDGR for data gathering operations that use the proposed access and channel selection protocols. The performance of the SDGR algorithm was evaluated by comparing its algorithm's results with simulation results. This performance analysis showed that the SDGR algorithm provided reasonable estimates of the SDGR, and the SDGR algorithm could be used to evaluate the performance of the data gathering operation.

### 5.2 Future Work

There are many opportunities for future research in the area of data gathering in CR-AHSWNs. The data gathering delay is an important performance metric for the data gathering operation. While a method for reducing the data gathering delay by intelligently selecting forwarding SUs was presented in chapter 3, other methods of reducing the delay should also be explored. In the GCM channel selection algorithms prepared in chapter 3, all of the channels available in the SUs were used. Some research has examined the use of downlink channel sets in the broadcasting operation with the goal of decreasing the delay with minimal effect on the success ratio [8] [9]. The use of downlink channel sets in the data gathering operation should also be explored.

## Chapter 5: Conclusions

### 5.1 Summary

In this thesis, several distributed algorithms for both SU action selection and channel selection were developed for data gathering in CR-AHSWNs operating under practical conditions. Through theoretical proofs and examples, the correctness of these algorithms was analyzed. An innovative algorithm that reduces the data gathering delay was also presented. This algorithm reduces the data gathering delay by intelligently selecting sets of forwarding SUs for each sending SU. Furthermore, another novel algorithm was presented which estimates the SDGR for data gathering operations that use the proposed action and channel selection protocols. The performance of the SDGR algorithm was evaluated by comparing the algorithm's results with simulation results. This performance analysis showed that the SDGR algorithm provided reasonable estimates of the SDGR, and the SDGR algorithm could be used to evaluate the performance of the data gathering operation.

### 5.2 Future Work

There are many opportunities for future research in the area of data gathering in CR-AHSWNs. The data gathering delay is an important performance metric for the data gathering operation. While a method for reducing the data gathering delay by intelligently selecting forwarding SUs was presented in chapter 3, other methods of reducing the delay should also be explored. In the GCM channel selection algorithms proposed in chapter 2, all of the channels available to the SUs were used. Some research has examined the use of downsized channel sets in the broadcasting operation with the goal of decreasing the delay with minimal effect on the success ratio [8] [9]. The use of downsized channel sets in the data gathering operation should also be explored.



## References

An analytical model that calculates estimates for both the broadcast success ratio and the delay is presented in [10]. An algorithm that calculates an estimate of the data gathering delay would be an important and useful tool. Along with the SDGR algorithm described in chapter 4, such an algorithm could be used for the evaluation of data gathering performance.

The data gathering protocol proposed in chapter 2, including initialization, action selection, and channel selection, should be demonstrated using hardware and software simulations. The simulations should include physical message transmissions and PU activity.

In addition, the results of the proposed SDGR algorithm should be compared to the results of additional simulations with varied network topologies and different available channel sets.

Comparison of the SDGR algorithm results to the results of additional simulations could provide further verification that the SDGR algorithm estimates are reasonable.

The security of the data gathering operation was outside the scope of this thesis, but it is a very important consideration for networking operations. Maintaining both the confidentiality and integrity of the information being transmitted through the network is already an important security topic in AHSWNS [15], and innovative approaches to secure the data gathering operation in CR-AHSWNS could be explored.

- [10] J. Wu and H. Li, "On calculating connected dominating set for efficient routing in ad hoc wireless networks," in *Proc. of the 3rd Int. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 1999.
- [15] J. Miola III, "Cognitive radio: An integrated agent architecture for software defined radio," in *IEEE Trans. Mobile Comput.*, vol. 14, no. 3, pp. 509-524, 2015.

## References

- [1] J. Wu and H. Li, "On calculating connected dominating set for efficient routing in ad hoc wireless networks," in *Proc. of the 3rd Int. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 1999.
- [2] P.-J. Wan, K. M. Alzoubi and O. Frieder, "Distributed construction of connected dominating set in wireless ad hoc networks," in *IEEE Infocom*, 2002.
- [3] FCC, "Promoting efficient use of spectrum through elimination of barriers to the development of secondary markets, WT Docket No. 00-230," 2002. [Online]. Available: <https://www.ntia.doc.gov/fcc-filing/2002/promoting-efficient-use-spectrum-through-elimination-barriers-development-secondary->.
- [4] I. F. Akyildiz, W.-Y. Lee, M. C. Vuran and S. Mohanty, "NeXt generation/dynamic spectrum access/cognitive radio wireless networks: a survey," *Computer networks*, vol. 50, pp. 2127-2159, 2006.
- [5] J. Mitola III, "Cognitive radio: An integrated agent architecture for software defined radio," Ph.D. dissertation, KTH Roy. Inst. Technol., Stockholm, Sweden, 2000.
- [6] I. F. Akyildiz, W.-Y. Lee and K. Chowdhury, "CRAHNS: Cognitive radio ad hoc networks," *Ad Hoc Networks*, vol. 7, no. 5, pp. 810-836, 2009.
- [7] P.-J. Wan, L. Wang and O. Frieder, "Fast group communications in multihop wireless networks subject to physical interference," in *IEEE MASS*, 2009.
- [8] Y. Song and J. Xie, "BRACER: A distributed broadcast protocol in multi-hop cognitive radio ad hoc networks with collision avoidance," *IEEE Trans. Mobile Comput.*, vol. 14, no. 3, pp. 509-524, 2015.



- [9] Y. Song and J. Xie, "A QoS-based broadcast protocol for multi-hop cognitive radio ad hoc networks under blind information," in *IEEE Global Telecommun. Conf.*, 2011.
- [10] Y. Song and J. Xie, "A novel unified analytical model for broadcast protocols in multi-hop cognitive radio ad hoc networks," *IEEE Trans. Mobile Comput.*, vol. 13, no. 8, pp. 1653-1667, 2014.
- [11] Y. Rondareddy and P. Agrawal, "Selective Broadcasting in Multi-Hop Cognitive Radio Networks," in *IEEE Sarnoff Symposium*, 2008.
- [12] C. J. L. Arachchige, S. Venkatesan, R. Chandrasekaran and N. Mittal, "Minimal time broadcasting in cognitive radio networks," in *Distributed Computing and Networking*, Springer, Berlin, Heidelberg, 2011, pp. 364-375.
- [13] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, 3rd ed., Cambridge, MA: The MIT Press, 2009.
- [14] N. J. Harvey, R. E. Ladner, L. Lovasz and T. Tamir, "Semi-matchings for bipartite graphs and load balancing," *Journal of Algorithms*, vol. 59, no. 1, pp. 53-78, 2006.
- [15] V. Kumar and S. Madria, "Performance Analysis of Secure Hierarchical Data Aggregation in Wireless Sensor Networks," in *Proceedings of the 4th Annusl ISC Resarch Symposium*, 2010.

DATA GATHERING IN COGNITIVE RADIO AD HOC AND SENSOR WIRELESS  
NETWORKS

A thesis submitted to the D. Abbott Turner College of Business in partial fulfillment of the  
requirements for the degree of

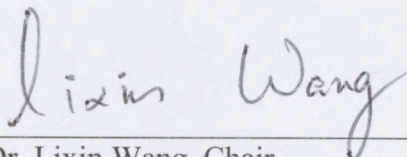
MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

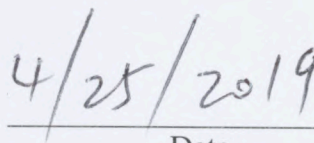
By

Kimberly A. Brown

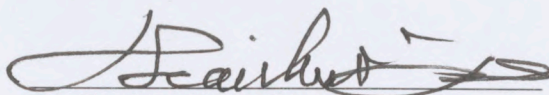
2019



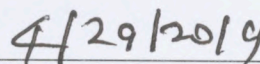
Dr. Lixin Wang, Chair



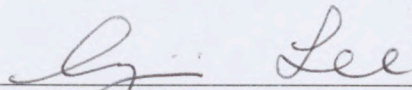
Date



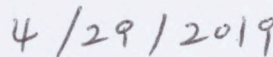
Dr. Jianhua Yang, Member



Date



Dr. Suk Lee, Member



Date



