

Ad Hoc Network Formation Using Bluetooth Scatternets

by

Amar Kailash Mehta

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Electrical Engineering and Computer Science

and

Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

June 2001

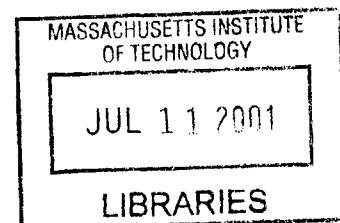
© MMI Massachusetts Institute of Technology. All rights reserved.

Author
Department of Electrical Engineering and Computer Science
May 23, 2001

Certified by
Kai-Yeung Siu
Associate Professor of Mechanical Engineering
Thesis Supervisor

Accepted by
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

BARKER



Ad Hoc Network Formation Using Bluetooth Scatternets

by
Amar Kailash Mehta

Submitted to the
Department of Electrical Engineering and Computer Science

May 23, 2001

In Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical Engineering and Computer Science
and Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis presents and evaluates an ad hoc network formation algorithm. The algorithm is designed to function under the specifications of Bluetooth, a short-range radio technology with a host of wireless applications. Since Bluetooth is a fairly new technology, the current specification does not explicitly detail how Bluetooth-enabled devices discover one another and optimally group together, forming a scatternet. Properly formed scatternets allow communication between any two Bluetooth-enabled devices. The algorithm presented is evaluated on several performance metrics based on both the final scatternet configuration as well as the time and packet complexity in forming the scatternet. Lower-level link control protocols in conjunction with the higher-level scatternet formation algorithm are simulated to yield an accurate assessment. The simulation results validate existing theoretical bounds.

Thesis Supervisor: Kai-Yeung Siu
Title: Associate Professor

Acknowledgements

First, I would like to thank my thesis supervisor, Professor Sunny Siu for giving me the opportunity to join the research team at the Auto-ID Center. During the course of my nine months with the center, Professor Siu has provided me with excellent guidance and support during discussions about my progress. He allowed a great deal of freedom in choosing a research topic, focusing more on my research interests and building skills that will allow me to be successful as I join the work force next year. I also want to thank Professor Siu for offering me funding during my two semesters with him.

My research would have been impossible had it not been for the efforts of Ching Law, a PhD student also working in the lab. Ching created the algorithm discussed in this thesis that gave me the opportunity to perform my simulations. On a more personal level, I consider Ching to be my mentor during my time with the lab. He was always available to answer questions and critique results. It has truly been my pleasure to work alongside Ching.

I would also like to thank two of my friends, Ashutosh and Paritosh Somani for all of their encouragement and motivation as the three of us completed our theses. Ash, Par, and I are fraternity brothers and have collaborated together on numerous projects since we were freshman. I will miss working with them next year.

Last and most of all, I would like to thank Mom and Dad for all their love and support.

Contents

1	Introduction	6
1.1	MIT Auto-ID Center.....	6
1.2	Connection Between Bluetooth-Enabled Devices.....	8
1.2.1	Masters and Slaves.....	8
1.2.2	Frequency Hopping.....	9
1.2.3	Piconets.....	10
1.2.4	Scatternets.....	11
1.3	Related Work.....	12
1.3.1	Scatternet Performance.....	12
1.3.2	Scatternet Size.....	13
1.3.3	Scatternet Formation.....	13
1.3.4	Evaluation.....	14
2	Scatternet Formation Algorithm	15
2.1	Performance Metrics.....	16
2.2	Message Types.....	18
2.3	The Leader.....	19
2.4	The Message-Passing Algorithm.....	20
2.4.1	Connection.....	21
2.4.2	Relocation.....	27
2.5	Scatternet Formation Example.....	35
3	Link Control Protocols	39
3.1	Inquiry.....	39
3.2	Page.....	44
3.3	Interpiconet.....	46
3.4	Frequency Hopping.....	48
3.5	Clock Drift.....	51
4	Simulations and Results	53
4.1	Simulation Tool.....	54
4.2	Scatternet Formation Simulation and Results.....	55
4.3	Link Control Simulation and Results.....	61
4.4	Complete Scatternet Formation Results.....	70

5	Conclusion and Future Work	72
6	Appendix A: Scatternet Formation Pseudocode	74
7	References	80

Chapter 1

Introduction

The Bluetooth initiative began in 1994 with Ericsson looking for alternatives in the way their accessories could be connected to their mobile phones [1]. Looking for an alternative to cables, they began considering the use of radio links. Radio links were more advantageous compared to infrared links because they do not require direct line of sight. They can also handle both data and voice communications. The end result is Bluetooth, a low cost, low power, short-range radio technology.

Starting off simply as a wireless alternative to cables, Bluetooth has developed into a very versatile technology. The technology is quite reliable and yet has the ability to cope with errors. With it comes a standardization of communication between any set of electrical devices. Personal area networks can be created that use Bluetooth technology. This wireless network has a host of applications that will revolutionize information technology.

1.1 MIT Auto-ID Center

The MIT Auto-ID Center eventually aims to use Bluetooth in its development of a revolutionary tagging system that will replace the current bar coding system. This tagging system centers on embedding electronic identification on everyday objects [2].

Objects ranging from packs of gum to computer monitors will have these tags embedded in them for identification. The tag is created at the time of production and is expected to last for the lifetime of the object.

This tag is not an ordinary tag. It is meant to be extremely low cost, eventually one to two cents, holding only a 96-bit identification number of the product in which it is embedded. These 96 bits contain both class and instance identification. For example, the identification tag on a pack of gum will identify it as gum (class information) and separate it from other otherwise identical packs (instance information). The tag is a passive electronic tag with the ability to transmit its identification number if triggered.

If all objects contain these types of tags, readers can be used to determine which objects are within a given vicinity. The reader queries for objects at the correct frequency causing the inexpensive tag to resonate and send back its 96-bit identification. Once the reader has the identification number of an object, it can perform a query to see pertinent information about the object such as what it is, where it was made, and when it expires. This pertinent information will be placed on the Internet and be available remotely.

With this technology, supply chain management becomes significantly easier. It gives the ability to track any particular object and figure out where that object has been in the past. Every time an object arrives at a new location, its information remotely available on the Internet will be updated. No longer will large amounts of inventory be lost. With inventory calculation capable of being done on the fly, stores will instantly know when to restock shelves. There are numerous other applications to this technology.

The center envisions Bluetooth eventually be used on the readers that query for objects. Different Bluetooth-enabled readers can communicate with one another as to

what objects they have found. This gives readers the ability to sort through results as a whole and come up with one set of final results. Connections to the Internet will also use Bluetooth technology allowing readers to query the Internet in order to find out more information about any particular object. The advantage of using Bluetooth is that the whole process is done wirelessly. Bluetooth will play a large role in the development of this technology.

1.2 Connection Between Bluetooth-Enabled Devices

Bluetooth-enabled devices operate in the Industrial, Scientific, and Medical (ISM) band at 2.4 GHz. This band is a license-free band that is globally available. Given that wide ranges of applications make use of this band, Bluetooth has to be robust in using this shared spectrum. To this end, Bluetooth-enabled devices take on the roles of masters and/or slaves in Bluetooth topologies. Frequency hopping is used to avoid interference from other devices and applications.

1.2.1 Masters and Slaves

When two Bluetooth-enabled devices form a connection, one of the devices acts as the master while the other device acts as the slave. Any Bluetooth-enabled device can take on either role. If a Bluetooth-enabled device is participating in multiple communication connections, it can be the master in one connection while the slave in another. The only restriction is that it can only be the master only once.

Masters and slaves have the same privileges. The role of master does not give a Bluetooth-enabled device any extra authority. These roles are made primarily for two reasons:

- Masters and slaves search in different ways when they are trying to establish connections. This is done so that they can find each other easily.
- Masters maintain the frequency hopping pattern and phase shift within the pattern. All slaves that are connected to the master use this hopping pattern in unison with the master to communicate.

The reason that a Bluetooth-enabled device can only be the master for one group of connections is because otherwise multiple groups of slaves would be using the hopping pattern of the master. More pertinent for this discussion concerning the connection between Bluetooth-enabled devices, however, is the differentiation of masters and slaves in terms of how this determines the connected topology that results.

From this point forward, when referring to Bluetooth-enabled devices, the term “node” will be used for simplicity. This will also allow us to think of these devices as nodes that are attempting to form bi-directional communication links analogous to a bi-directional graph. Figure 1-1 can be used to identify the role taken on by a node at a given point in time.

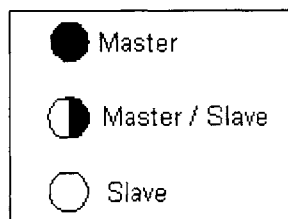


Figure 1-1: Node Identification Key

1.2.2 Frequency Hopping

There are many other devices that also operate in the 2.4 GHz band, including home appliances as well as wireless LANs used in the office. Frequency hopping and the

use of spread spectrum help to reduce the interference caused by these devices as well as interference from other Bluetooth nodes. Instead of communicating on a fixed frequency, nodes hop very quickly among a set of frequencies. They do so in the following manner:

- During inquiry and page (attempting to establish a connection), nodes hop 3,200 times per second among 32 predefined frequencies.
- During connection (link has already been made), nodes hop 1,600 times per second among 79 frequencies.

The frequency hopping pattern is determined by the Bluetooth address of the master during connection and by each individual node during inquiry. This clock is used to determine the phase shift within the sequence. Both the address and the clock are 48-bit numbers.

Using frequency hopping also provides increased security. By switching between frequencies at such a high rate, it is difficult for large amounts of data to be intercepted. This is because it would be extremely difficult to follow the hop sequence without knowing the address and clock of the master node. Coupled with encryption, authentication, and limited range capabilities, Bluetooth is quite secure.

1.2.3 Piconets

A group of slaves operating together under a common master is referred to as a piconet. All the slaves in a piconet follow the frequency hopping sequence of the master by obtaining the master's address and an accurate estimate of the master's clock. The slaves in a piconet do not have direct links between them. Connections only occur between the master and each slave. The Bluetooth specification limits the number of slaves in a piconet to seven, allowing a total of at most eight nodes in a piconet. Figure 1-2 shows an example of a piconet with five slaves and a master.

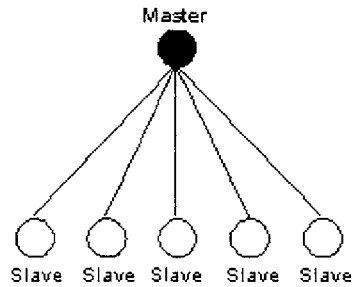


Figure 1-2: A Piconet

1.2.4 Scatternets

With a limit of eight nodes in a piconet, there needs to be some means of connecting a larger group of nodes. A larger number of nodes can be part of a network by linking piconets together to form what is called a scatternet. In a scatternet, some nodes are a part of more than one piconet. When a node is part of multiple piconets, it must time-share, spending slots of time in each piconet. Figure 1-3 shows two types of scatternets.

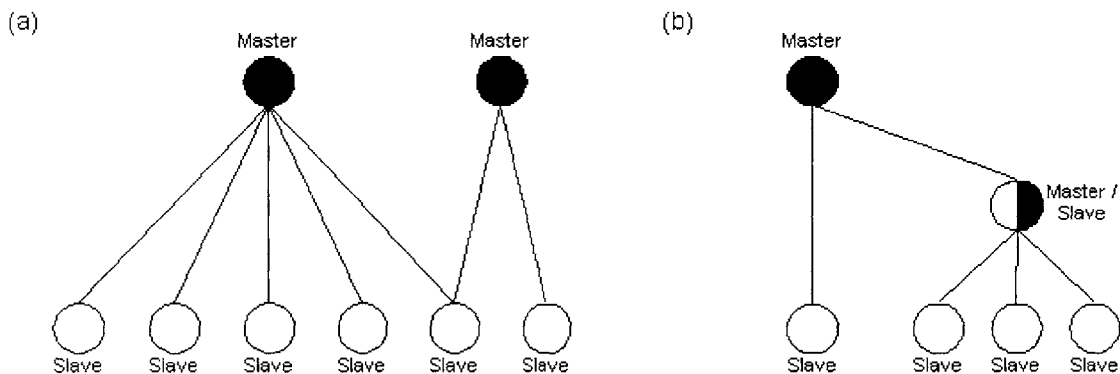


Figure 1-3: Scatternet Diagram

(a) A scatternet with a shared slave (b) A scatternet with a node that is a master and a slave

In the first scatternet (Figure 1-3 (a)), a slave is a member of two piconets and has two masters. The second scatternet (Figure 1-3 (b)), in contrast, shows that a node that is a slave in one piconet can be a master of another.

1.3 Related Work

There has been widespread research already done in the Bluetooth arena concerning scatternets. The areas of previous research highlighted in this section aim to illustrate the viability of Bluetooth scatternets as well as show current proposed techniques of scatternet formation. In the last section of this chapter, the related work will be assessed. This evaluation will lay the foundation for future chapters presented.

1.3.1 Scatternet Performance

The formation of scatternets is relevant only if scatternets indeed yield acceptable performance once they are created. There are many different metrics that could be used to determine performance of scatternets. Two of the most important are robustness under high network usage and the ability to integrate with other wireless environments. These two issues play a prominent role in the acceptance of Bluetooth in the wireless community.

“Short Range Radio Based Ad-hoc Networking: Performance and Properties” studied the stress associated with a mix of data and voice traffic on Bluetooth piconets by means of simulation. The types of conditions used included high load and bursty traffic. It concluded that Bluetooth was a very versatile system capable of sharing both voice and data traffic concurrently even when the traffic load was high and bursty. Bluetooth makes short distance connectivity very possible. [3]

An architecture in which a Bluetooth personal area network can be embedded in a wireless LAN is presented in “Architecture and Performance of an Embedded IP Bluetooth Personal Area Network”. This architecture acts as a bridge between the personal area network and wide-area IP networks allowing information exchange with

corporate servers in a mobile fashion. The study concluded that in some cases acceptable throughput with minimal error could be achieved. [4]

1.3.2 Scatternet Size

Another important quality of scatternets in any particular environment is the number of nodes that can be handled with acceptable performance. “Radio Network Performance of Bluetooth” attempts to simulate a Bluetooth environment with heavy unidirectional WWW traffic and bi-directional voice traffic. The results showed that hundreds of nodes could be supported in the same area without significant performance degradation due to interference. Also, the aggregate capacity for Bluetooth is very high, in particular for packet data traffic. [5]

1.3.3 Scatternet Formation

The manner by which scatternets are formed is not fully specified in the Bluetooth Specification Version 1.1. There is expected to be a more fully quantified formation algorithm for future versions of the specification. Various research groups have done some work on algorithms and techniques that can be used in the formation of scatternets.

The IBM India Research Laboratory has come up with a clustering algorithm for the formation of scatternets in “Clustering Algorithms for Wireless Ad Hoc Networks.” The algorithm they have proposed is a two stage distributed, randomized algorithm that finds the minimum number of star shaped clusters that are of maximum size. The performance of the algorithm is evaluated by simulation. [6]

In “Proximity Awareness and Fast Connection Establishment in Bluetooth”, techniques are discussed for fast connection of a single master and slave. The study discusses the issues surrounding symmetric connection between nodes using a protocol

based on a random schedule. It shows that connection delay between a pair of nodes can be reduced if appropriate decisions are made in parameters. [7]

1.3.4 Evaluation

Bluetooth is indeed a viable technology. The discussions concerning scatternet performance and size give quite positive indications. Bluetooth has been shown, through simulation, that it can handle various types of traffic at high load levels with burstiness. Many nodes can function in this manner together at the same time without a large reduction in performance. Furthermore there are means for Bluetooth to integrate with other wireless networks such as the wireless LAN. These are all positive signs for Bluetooth as a short- range wireless communication technology.

Concerning scatternet formation, however, research is quite underdeveloped. Clustering does not seem feasible in the manner that it was described. The algorithm presented simply partitions the nodes in the environment into independent piconets. In the next stage, a super-master is elected that is knowledgeable about each piconet. This in fact is not a scatternet as described because the individual piconets are not interconnected. Furthermore, because the super-master has to collect information from every piconet, the algorithm has a time complexity of $\Omega(n)$. The proximity awareness discussion provides low level results regarding scatternet formation but gives no clear algorithm by which to make a scatternet. It, in fact, regards itself as an introductory piece.

Scatternet formation is a very important issue to consider. An algorithm must be developed that forms scatternets efficiently.

Chapter 2

Scatternet Formation Algorithm

This chapter presents a new algorithm for the formation of scatternets. Ching Law, a PhD student at the Massachusetts Institute of Technology, originally wrote this algorithm in his work “A Bluetooth Scatternet Formation Algorithm” [8]. Through collaboration, this algorithm has been rewritten in a message-passing format to show specifically when each node sends a message to another node. The essentials of the algorithm, however, remain the same, including the evaluation metrics, the terminology used, as well as the intermediate steps and final results of the algorithm.

The description of this algorithm begins with an overview of the different parameters that constitute a well-formed scatternet. These parameters will be used to determine the quality of the scatternets formed using the algorithm. Since the algorithm is presented in a message-passing format, the different types of messages that can be sent between nodes will be introduced next. This is followed by a discussion of the various states nodes can be in during the execution of the algorithm. The algorithm itself is presented in two parts, the first part called connection and the second part called relocation. This chapter ends with an example of the algorithm as it is executed with twelve nodes in the environment.

2.1 Performance Metrics

In the development of this algorithm, two assumptions were made about the nodes in the environment in order to come up with suitable performance metrics. Initially, each node is an isolated node. This means that it is unaware of any other nodes in the environment and therefore has no connections prior to the start of the algorithm. As a result of this assumption, the formation algorithm has to be distributed. The other assumption is that all the nodes are within communication distance. This allows potentially any node to have a direct communication link with any other node. For Bluetooth, this communication distance is ten meters without amplification and up to one hundred meters with amplification.

In evaluating the performance of this algorithm, there are two types of measures. The first measure will be termed algorithm complexity and depends on issues surrounding the actual execution of the algorithm. The second measure, which will be called scatternet quality, deals with the end result of the algorithm. Scatternet quality evaluates the end-result scatternet based on several quality measures.

The measures of algorithm complexity are fairly straightforward. It deals with the following:

- the amount of time the algorithm takes to execute
- the number of messages sent during the execution

A smaller amount of time, as well as a fewer number of packets sent, indicate a more suitable algorithm.

It is quite a bit harder to quantify the measures of a good scatternet. This algorithm focuses on three measures of scatternets for quality.

- the number of piconets formed

This measures the efficiency of the scatternet. All piconets use the same set of 79 frequency channels in the 2.4 GHz ISM band. Therefore, as the number of piconets increases, the number of collisions will increase when messages are sent. Messages collide when they are sent on the same frequency and the same time. When they collide, none of the messages sent are received.

- the maximum degree of a node

The maximum degree of a node is the maximum number of piconets any one particular node belongs to. This is important because piconets communicate through shared slaves. If a node has a high degree, and therefore belongs to many piconets, then the slave could become the bottleneck for communications between piconets. The aim is to avoid overloading any particular node.

- the diameter of the network

The network diameter refers to the maximum number of hops between any two pairs of nodes. This is important because it the maximum number of times a message will have to be forwarded before it reaches its final destination.

A good balance among the quality measures is desirable. Consider for example, a star topology. In this topology, there is only one shared slave with this slave being a member of every piconet. Also assume that each piconet is full, containing the maximum of seven slaves. In this case, with the number of nodes as such, the number of piconets is minimized. However, this scatternet would likely not perform very well as the single shared slave will become overloaded, unless the number of piconets is small.

2.2 Message Types

There are several types of messages that are sent during communication between nodes. It is worth differentiating these messages because they each serve a different function and do not necessarily take the same amount of time. The three types of messages presented are inquiry, page, and interpiconet messages.

Inquiry messages are used to find other nodes. If a node that is a master is in search of an additional slave, that node will send an inquiry message. If another node, seeking to be a slave, happens to be listening in inquiry scan mode, it will respond to the master node with an inquiry response. This message type is used when both nodes are unaware of each other and are arbitrarily searching for others. Once an inquiry message is sent, and an inquiry response message is received, the nodes are now aware of each other and proceed to page mode to complete the connection.

Paging, analogous to inquiry, consists of page and page response messages. In this state, however, both nodes are aware of each other. They each know the address of the other node for identification. The aim in sending this message type is to complete the connection and synchronize the slave with the master. Page messages always come after successful a successful inquiry, but they may also occur in attempts to reorganize a piconet in which the nodes are already aware of each other.

The final message type is perhaps the simplest. It is the interpiconet message that allows a master to communicate with his slaves and vice versa. This message is the most trivial because the master and his slaves are already synchronized and awaiting messages from each other.

2.3 The Leader

The three possible states of nodes at any point in time were identified in Section 1.2.1 as master, master/slave, or slave. For the purpose of this algorithm, however, a new state must be introduced. This state is called the leader state. It exists only during the formation of the scatternet when the algorithm is executing. Once the algorithm terminates and the scatternet is formed, leader is no longer a possible node state. Figure 2-1 shows the extended version of the node identification key with the addition of the leader state.

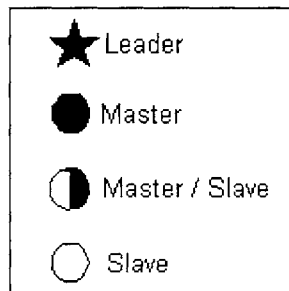


Figure 2-1: Extended Node Identification Key

At the beginning of the algorithm, all the nodes are isolated. As the algorithm progresses, nodes form connections and partition themselves into clusters. A cluster can be any set of interconnected devices. An isolated node is the only member of its cluster. Within each cluster, whether an isolated node, a piconet, or a scatternet, only one leader exists.

The leader plays a critical role during the execution of the algorithm. In the trivial case when a node is isolated, it is obviously the leader. If the cluster is a piconet, then the leader is the master of the piconet. One of the masters is termed the leader in a scatternet. In a piconet or a scatternet, the leader has the same functionality as a master in that it is

allowed to have slaves, but it also has the additional responsibilities associated with the algorithm. Only a leader is allowed to search for new slaves. Nodes that are just masters cannot and for the most part remain inactive. Similarly, only leaders can designate the node that is in search forming a connection as a slave. Again, nodes that are just masters are not given this authority.

2.4 The Message-Passing Algorithm

The message-passing algorithm operates in rounds. At the beginning of the simulation, all nodes are isolated and therefore all nodes are leaders. As the rounds proceed, the number of leaders decreases. The simulation terminates when only one leader remains. In each round, zero or more connections can be made. For every connection that is made, the number of leaders decreases by one. This means that each round at most the number of leaders can be halved.

The Bluetooth Specification states that a node can inquire for slaves until it receives a number of responses that it deems adequate. In general, that means that a node wishing to be master could continue to inquire until it received at most seven responses after which it would attempt to page each of the nodes in hopes of adding them as slaves. For this simulation, however, if a leader is attempting to search for new slaves, once it receives its first response, it will attempt to page the node and connect immediately. Never will a leader inquire for more than one node in any given round.

Leaders come in two forms in this algorithm. A leader can be isolated, meaning it has made no connections or a leader can be connected, meaning it has one or more connections. In some cases, it is important to differentiate between the two types of

leaders. This will be done by adding an isolated or connected label to the leader when depicted in figures. These types of designations may also be necessary for slaves. Slaves come in two forms, those that are shared between piconets and those that are unshared and exist only in one piconet. Similar labels will be used to designate slaves when necessary.

A unique property of this algorithm is that a connected leader will always have at least one unshared slave. This property is true at the start and end of each round for all connected leaders. The proof for this property can be found in “A Bluetooth Scatternet Formation Algorithm” [8].

The actual algorithm is presented in the following two sections. The first section, called connection, highlights how nodes search for each other and form their initial connection. In the next section, entitled relocation, shows how nodes regroup themselves once the initial connection is made. Connection and relocation occur once each round for each cluster of nodes.

2.4.1 Connection

At the beginning of each round, the leader of each cluster determines the state that the cluster will have for the remainder of the round. There are two possible states, the seek state and the scan state. If the cluster chooses the seek state, it is attempting to find other nodes to add to its cluster. If the scan state is chosen, then the cluster is attempting to be sought by a cluster in the seek state. The cluster is trying to be acquired into another cluster.

The state that a cluster chooses each round is determined in a probabilistic fashion. It is much like flipping a coin. At the beginning of each round, the leader of each cluster randomly decides which of the two states the cluster will take for the ensuing round. For the purposes of this algorithm, there is an equal likelihood of being in either state. Permissible probabilistic values, however, range from $1/3$ to $2/3$ for each state.

There are two possible types of connections that can be made. When in the seek state, it is the leader of the cluster that always participates. In the scan state, however, the leader scans only if it is an isolated node. If it is a connected node, then it designates one of its unshared slaves to be the scanner for the round. The simpler of the two cases is the cases when a leader that is seeking connects to an isolated leader that is scanning. This case will be discussed first, followed by a discussion of the changes that occur in the second type of connection.

Case 1: Seeking Leader Connects to a Scanning Isolated Leader

At the start of this procedure, neither the seeker nor the scanner is aware of each other. They have just probabilistically decided their roles for the round and are attempting to form a connection. Figure 2-2 illustrates the process by which the two nodes learn of each other in the first step of this part of the algorithm. An entity called the matcher is responsible for bringing as many sets of seekers and scanners together as possible during each round.

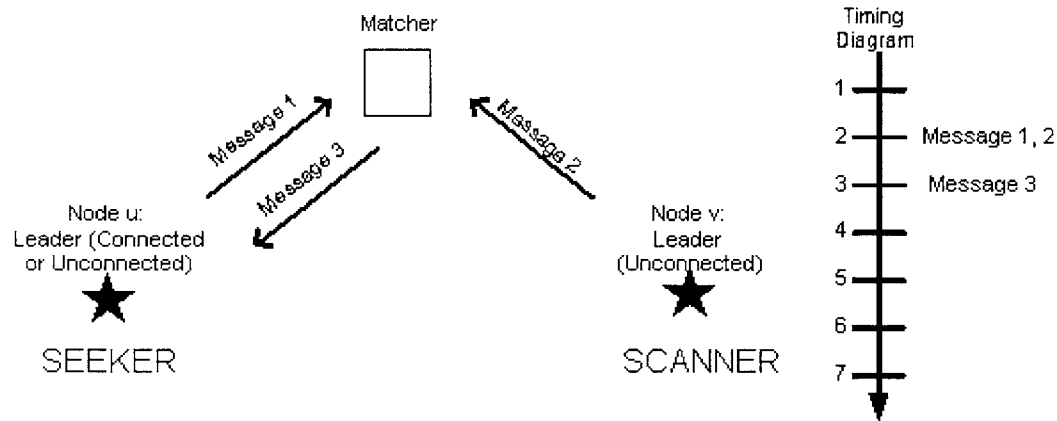


Figure 2-2: Part I of Connection between a Seeking Leader and a Scanning Leader

This process begins at time 2 with node u, the seeker, and node v, the scanner, sending messages to the matcher trying to make a connection. Node u sends a message telling the matcher that it is a seeker and that it would like to be matched with a scanner. Node v, as the scanner, does the opposite. The two messages are sent at the same time and count together as one inquiry message.

The matcher receives these messages at time 3. At the same time, the matcher also receives seek and scan messages from all the other seekers and scanners for this round also. It is the job of the matcher to arbitrarily pair together seekers and scanners. The number of matches that the matcher makes each round can be quantified as the min (the number of seek messages, the number of scan messages). This number is how many connections will be made the round. In Figure 2-2, the matcher has paired together node u and node v.

The matcher sends an inquiry response message to node u in the next time interval with the address of the scanning node that the seeker should connect to. Node u, the seeker, can now use this address to page node v and establish a connection. If node u

does not receive an acknowledgement from the matcher at time 3, it assumes that it was unsuccessful in finding a match for the round and its entire cluster retires until the next round.

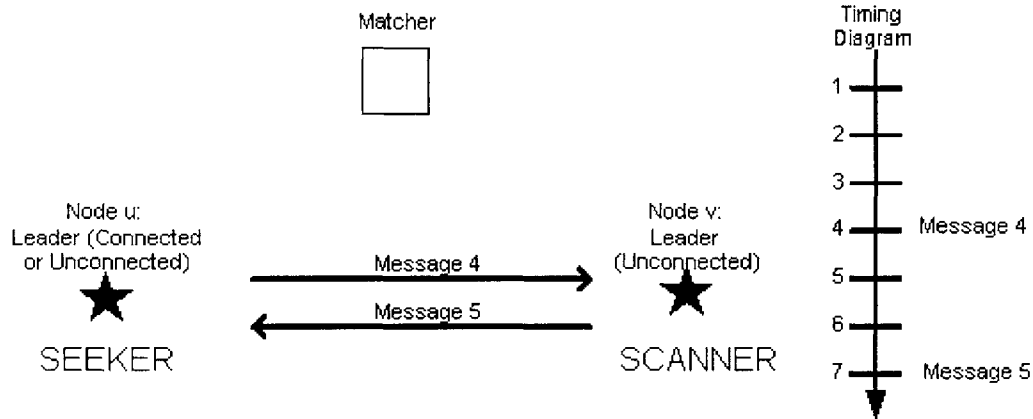


Figure 2-3: Part II of Connection between a Seeking Leader and a Scanning Leader

The paging process begins at time 4 when node u, the seeker, sends a page message to node v. Included in the page message is a size indication. The size indication is the number of slaves that are directly connected to node u prior to this new connection. This number will be useful to node v as it processes the page message. If node v does not receive a page response at this time, it assumes that a match was not made for it this round by the matcher. Node v will therefore retire for the rest of this round.

Upon receiving a page message from node u, node v uses the appended size indicator to decide the next course of action. This will determine the next action to be performed in the relocation portion of the algorithm. There are only two possible actions because the scanner is an isolated leader:

- a trivial relocation if node v will be the sixth slave or less for node u
- a shift relocation otherwise (node v will be the seventh slave for node u)

Using the size indicator from node u, node v determines the action for the relocation portion of the algorithm and sends back a page response message to node u. The page response message is sent at time 7 and has attached to it this action so that node u is also aware of the next course of action. Once the page response is sent back, the two nodes are connected.

Counting the total number of messages sent gives a good indication of complexity. For case one of the connection portion of the algorithm, where the scanning node is an isolated leader, there were a total of one inquiry message, one inquiry response message, one page message, one page response message, and zero intericonet messages.

Case 2: Seeking Leader Connects to a Scanning Unshared Slave

In the second case of the connection portion of the algorithm, the leader that probabilistically decides to be a scanner is no longer an isolated leader. The algorithm changes because the scanner already has some slaves. When the scanner is connected, it selects one of its unshared slaves to perform the scanning function for the round. Figure 2-4 shows the changes associated with connection from case one.

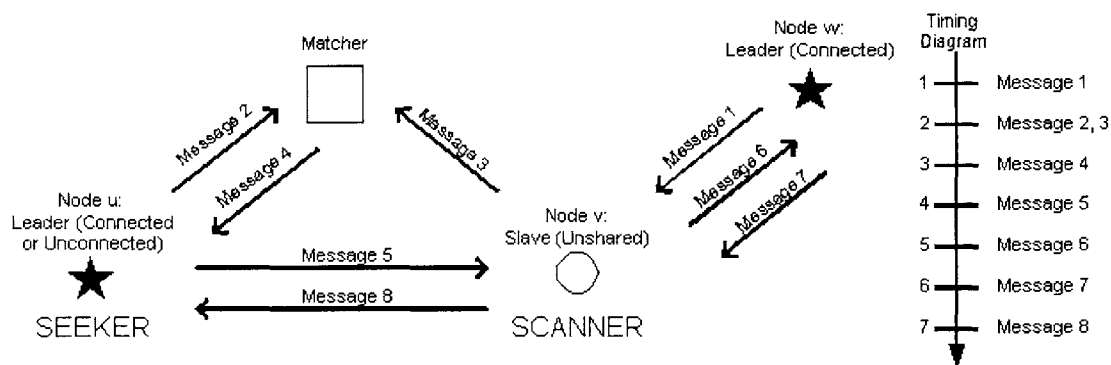


Figure 2-4: Connection between a Seeking Leader and a Scanning Slave

The basic premise of this case is the same as that of the first case. The idea is to connect the seeker and the scanner while making all nodes aware of the action that will occur in the relocation portion of the algorithm next. In this case, node w begins by selecting the unshared slave to act as the scanner by sending it an intericonet message. Messages two through five are identical to their corresponding messages in case one. In this case, however, node v does not know the full topology of its piconet and therefore cannot decide on the next course of action for relocation. As a result, node v sends an intericonet message with the address and size indication of the seeker, node u , to its current master, node w so that a decision can be made. If node w does not receive an intericonet message at time 5 from node v , then it assumes the matcher did not make a match for node v this round. Node w then retires until the next round.

There are three possible relocation actions in this case, each of which is different from those presented in the first case. Each is again based on the size indication that is passed from node u that indicates the number of slaves that are directly connected to it. The possible actions are:

- a merge relocation if the size indication of node u (which does not include node v) plus the size indicator of node v (which does include node v) plus two (one for each leader u and w) is less than eight
- a rearrange relocation if the size indication of node u is one (which does not include node v)
- a migrate relocation otherwise

Once the relocation action is determined, it is sent back to node v through an intericonet message and then forwarded in the next time interval to node u as an attachment in the page response message. Now all three nodes are aware of the relocation action and are ready to proceed. Node v is now connected to node u and node w as a shared slave.

With the addition of node w, there were more messages sent during the execution of case two of the connection portion of the algorithm. Just as the previous case, there was one message sent of each of the following: inquiry, inquiry response, page, and page response. In addition, three intericonet messages were sent.

2.4.2 Relocation

Once node v is initially connected to node u, a relocation process begins. This relocation occurs so that the new piconet or scatternet that results can be formed into one of good quality. Descriptions of what constitutes a piconet or scatternet of good quality can be found in Section 2.1. Relocation begins at time 8, as both case one and case two of connection end at this point, and ends at time 10. During these three time units, one of five possible relocation actions can take place: trivial, shift, merge, rearrange, or migrate. The first two possible actions result from case one of connection while the last three can happen after case two of connection.

Trivial Relocation

The trivial relocation is quite straightforward. No new messages are sent and topology of the scatternet also remains the same. Figure 2-5 shows the network topology of this case at the end of the configuration.

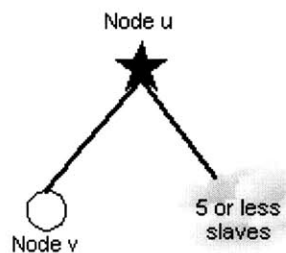


Figure 2-5: Trivial Relocation End Configuration

Since no new messages were sent, the total number of messages sent for the round when a trivial action is completed is the same as case one of connection. The total number of messages sent of each type is:

- 1 inquiry message
- 1 inquiry response message
- 1 page message
- 1 page response message
- 0 interpiconet messages

Shift Relocation

The other option with case one of connection is the shift relocation. In this case, node v has connected to node u and is the seventh slave in the piconet. If some type of relocation does not occur, node u is now saturated and cannot accommodate any more slaves even though it is a leader. Figure 2-6 shows the initial configuration.

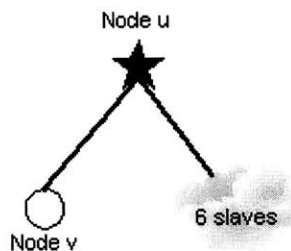


Figure 2-6: Shift Relocation Initial Configuration

One of the subtleties of this algorithm, however is that a leader always has an unshared slave. Displaying this unshared slave, another version of the initial configuration is

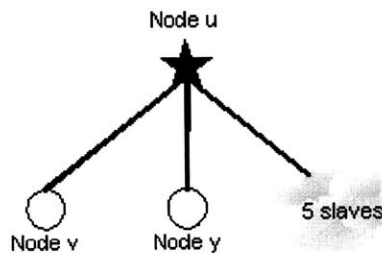


Figure 2-7: Shift Relocation Initial Configuration with Unshared Slave y Displayed

shown in Figure 2-7.

This relocation begins at time 8 with node u sending two interpiconet messages. The first message is sent to node y, directing node y to change masters to node v. The second interpiconet message is sent to node v, directing it become a leader and add node y as a slave. At this point node u and node v disconnect from each other. In the next time interval, at time 9, node v sends a page message to node u and to node y, attempting to add both of them into its piconet. Node u is expecting this page message, as it is aware of the action that is happening. Node y is also awaiting this message, as it was just directed to switch masters. At time 10, both node u and node y send page response messages to node v. Node v is the new leader and has two slaves. Node y is its unshared slave as required and node u is its other slave. Node u is still also a master of five other slaves. The final configuration is shown in Figure 2-8.

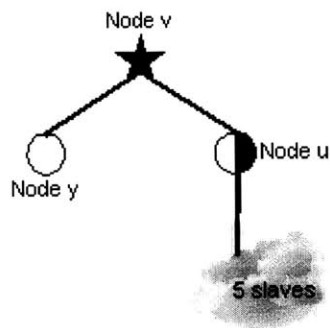


Figure 2-8: Shift Relocation Final Configuration

In this relocation case, the total number of messages sent are the messages sent in case one of connection plus two messages of each of the following types: interpiconet, page, and page response. The total number of messages sent of each type is:

- 1 inquiry message
- 1 inquiry response message

- 3 page messages
- 3 page response messages
- 2 intericonet messages

Merge Relocation

The merge relocation may occur after case two of connection. The aim of this relocation process is to turn a scatternet that consists of a shared slave linking together two piconets into a single piconet. The idea is to merge together the scatternet. This action is only done, however, if the merging gives the single piconet six or fewer slaves. If seven slaves resulted from a merge, then the piconet would be saturated and would not be able to accommodate any more slaves in future rounds. Figure 2-9 shows the initial configuration at the start of the merge relocation.

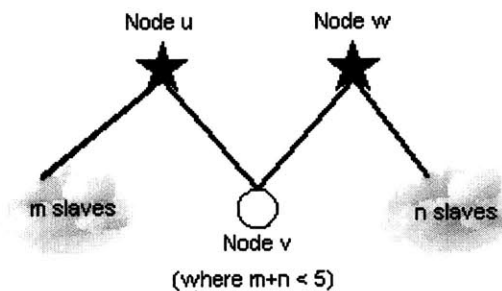


Figure 2-9: Merge Relocation Initial Configuration

The merge relocation begins at time 8 with node w sending several intericonet messages. It first sends n intericonet messages (one to each of its n slaves) directing each slaves to change masters to node u. It also sends an intericonet message to node u with the identification of each of its slaves so that node u can add each of them to its piconet. The message to node u is an intericonet message because during the connection portion of the algorithm, it is assumed that node w temporarily becomes aware of the synchronization pattern of node u. At this point, node w disconnects from

all of its slaves. In the next time period, node u pages each of the slaves that node w previously had. These slaves are awaiting the page message as they were just directed to change masters. It also pages node w as a slave to add. Node w is also expecting this message since it is aware that the action that is occurring is a merge. In the last time period, at time 10, a page response message is sent to node u from each of the nodes that were sent a page message. After this is complete, the nodes have integrated into a single piconet as shown in Figure 2-10.

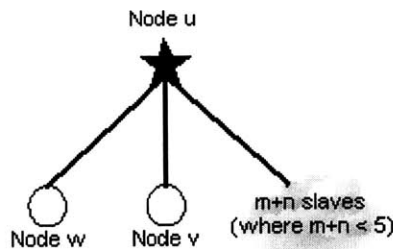


Figure 2-10: Merge Relocation Final Configuration

There are a variable number of total messages sent in the merge action. The number of messages sent depends on how many slaves node w has at the beginning of the action. Including case two of the connection, when a merge occurs, the total number of message sent of each type is:

- 1 inquiry message
- 1 inquiry response message
- (2 + n) page messages
- (2 + n) page response messages
- (4 + n) interpiconet messages

Rearrange Relocation

The next relocation action occurs in a very specific case. In this case, node u only has one slave in its piconet. This slave is node v and is shared with node w. Also, node w must have exactly six slaves including node v. If it has any less, a merge would have occurred instead. On the other hand, if node w had any more slaves, then the piconet would be saturated and would never be able to seek, just scan as it is now. The object of this relocation is to give node u an unshared slave for future rounds as well as to decrease the overall distance between nodes. Figure 2-11 shows the initial configuration.

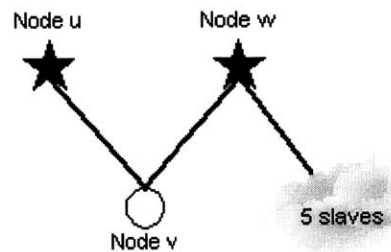


Figure 2-11: Rearrange Relocation Initial Configuration

The message passing in this action is quite simple. At time 8, node w and node v disconnect. This means that node v is now an unshared slave of node u. In the next time interval, node u sends a page message to node w. Node w is expecting this page message since it is aware of the relocation action that is taking place. At time 10, node w sends a page response message back to node u. The final configuration of this relocation action is shown in Figure 2-12.

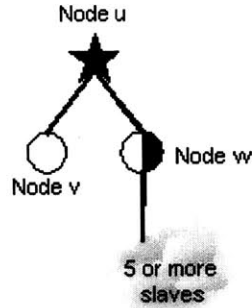


Figure 2-12: Rearrange Relocation Final Configuration

In this action, there was only one addition page and page response message sent.

This brings the total number of messages sent including connection to:

- 1 inquiry message
- 1 inquiry response message
- 2 page messages
- 2 page response messages
- 3 intericonet messages

Migrate

In the event that neither a merge nor a rearrange action occur, the migrate action is executed. At the start of this action, both node u and node w have their own slaves along with a common shared slave in node v. The object of this action is to move as many slaves as possible from node u to node v so that node u may remain leader and collect more slaves in future rounds. Node u has m slaves initially, of which one must be an unshared slave called node y. The initial configuration is shown in Figure 2-13.

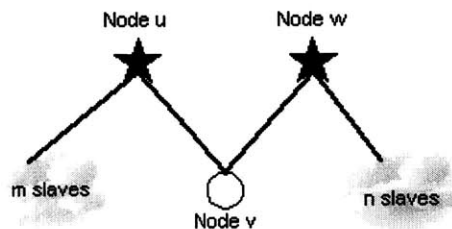


Figure 2-13: Migrate Relocation Initial Configuration

The number of slaves to be moved from node u 's piconet to that of node w depends on the number of slaves each piconet has. Node u will move as many slaves as possible to node w without exceeding the maximum of seven slaves in a piconet. This means that either node u will run out of slaves to migrate before node w 's piconet is completely full or node u will not be able to migrate any more slaves because there is no more room for additional slaves in node w 's piconet. During migration, since node u will remain leader after the action is complete, node u must retain one unshared slave of its m slaves for future rounds. The number of slaves to migrate x is defined as the $\min(m - 1, 6 - n)$.

This action, just as all the others, begins at time 8 with node u sending x interpiconet messages to its slaves, making sure to retain one that is unshared, directing each to change masters to node w . It also sends an interpiconet message to node w giving the identification of each node that it should add as a slave. In the next time unit, node w sends a page message to each of the x slaves that it is trying to add. The slaves are expecting this message as they were directed to change masters in the previous time unit. They respond at time 10 with a page response message to node w and complete the connection. Figure 2-14 shows the possible end configuration for this action depending on whether node u ran out of slaves to migrate to node w (Option A) or whether node u

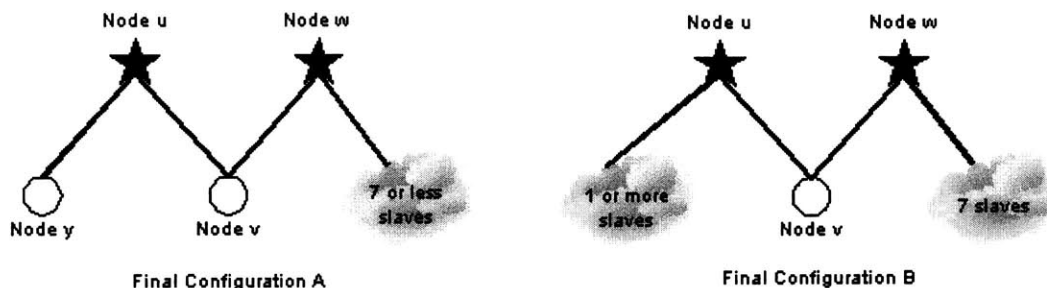


Figure 2-14: Migration Relocation End Configurations

was able to migrate enough slaves to give node w seven slaves (Option B).

In this case, there are again a variable number of messages sent depending on how many slaves are migrated. The total number of messages sent, including connection, is:

- 1 inquiry message
- 1 inquiry response message
- $(1 + x)$ page messages
- $(1 + x)$ page response messages
- $(4 + x)$ intericonet messages

2.5 Scatternet Formation Example

The following section illustrates the scatternet formation algorithm. For simplicity, in this example only twelve nodes will be used. Any number of nodes could have been used however. Each node initially begins isolated. Rounds continue, with the intention of forming connections, until a scatternet is made in which all twelve nodes are in some way, directly or indirectly, connected to all other nodes. In each round, first the probabilistic decision of each node will be shown (whether it is a seeker or a scanner). Next the pairings arbitrarily made by the matcher will be displayed. For each pairing, the relocation associated with the two clusters will also be listed. To end each round, a figure will show the configuration of all the nodes at that time.

Initial Configuration

As previously indicated, the twelve nodes begin isolated, with each node a leader.

This is shown below in Figure 2-15.



Figure 2-15: Initial Configuration of Twelve Nodes

Round 1

The seekers are nodes 1, 4, 7, 9, and 10. The scanners are nodes 2, 3, 5, 6, 8, 11, and 12. Since there are five seekers and seven scanners, there are five pairings made by the matcher. The pairings are as follow:

Node 1 connects to Node 5 using relocation action TRIVIAL
Node 7 connects to Node 6 using relocation action TRIVIAL
Node 10 connects to Node 12 using relocation action TRIVIAL
Node 9 connects to Node 11 using relocation action TRIVIAL
Node 4 connects to Node 2 using relocation action TRIVIAL

The configuration at the end of this round is displayed in Figure 2-16.

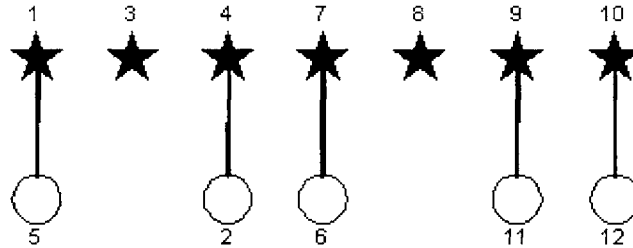


Figure 2-16: Configuration at the End of Round 1

Round 2

The seekers are nodes 3, 4, and 7. The scanners are nodes 5, 8, 11, and 12. Since there are three seekers and four scanners, there are three pairings made by the matcher. The pairings are as follow:

Node 4 connects to Node 11 using relocation action MERGE
Node 3 connects to Node 8 using relocation action TRIVIAL
Node 7 connects to Node 5 using relocation action MERGE

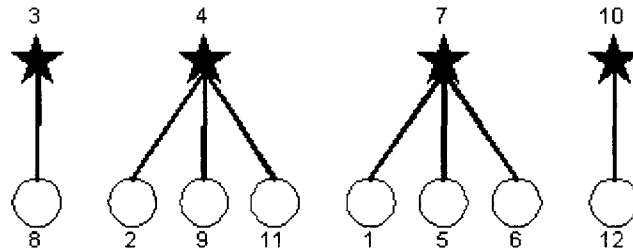


Figure 2-17: Configuration at the End of Round 2

The configuration at the end of this round is displayed in Figure 2-17.

Round 3

The seekers are nodes 3, 4, and 7. The scanner is node 12. Since there are three seekers and one scanner, there is one pairing made by the matcher. The pairing is as follows:

Node 3 connects to Node 12 using relocation action MERGE

The configuration at the end of this round is displayed in Figure 2-18.

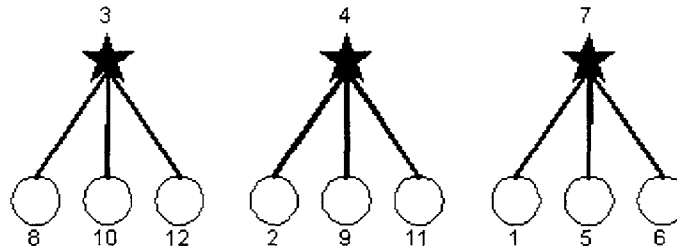


Figure 2-18: Configuration at the End of Round 3

Round 4

The seeker is node 4. The scanners are nodes 6 and 8. Since there is one seeker and two scanners, there is one pairing made by the matcher. The pairing is as follows:

Node 4 connects to Node 8 using relocation action MIGRATE

The configuration at the end of this round is displayed in Figure 2-19.

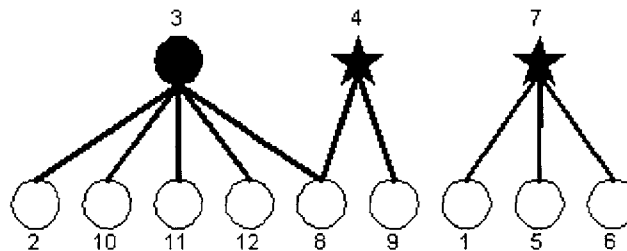


Figure 2-19: Configuration at the End of Round 4

Round 5

There are no seeker nodes. The scanners are nodes 5 and 9. Since there are no seekers and two scanners, no pairings are made by the matcher. At the end of this round, the configuration has not changed from the configuration in the previous round.

Round 6

The seeker is node 7. The scanner is node 9. Since there is one seeker and one scanner, there is one pairing made by the matcher. The pairing is as follows:

Node 7 connects to Node 9 using relocation action MERGE

The configuration at the end of this round is displayed in Figure 2-20.

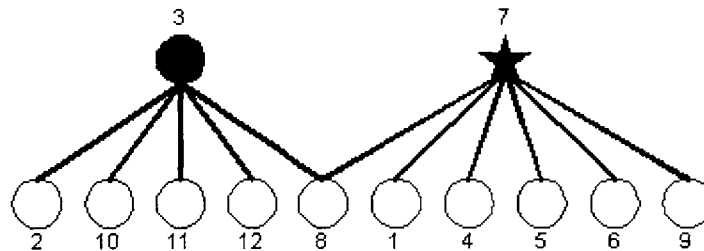


Figure 2-20: Configuration at the End of Round 6

Final Configuration

Since there is only one leader left, the algorithm has completed. A scatternet is formed in which all nodes are somehow connected. Returning to the notation without leaders as described in Figure 1-1, the fully formed scatternet is displayed in Figure 2-21.

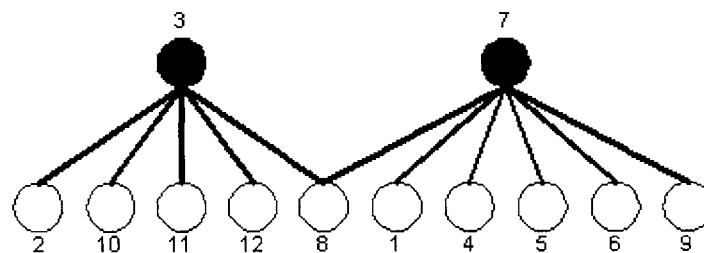


Figure 2-21: Final Configuration of Twelve Nodes

Chapter 3

Link Control Protocols

The scatternet formation algorithm described in Section 2.4 requires the transmission of three different types of messages. These messages were abstracted away under the classification inquiry, page, and interpiconet messages. In reality, however, sending a single packet of information cannot always complete the tasks that each message is attempting. Far more packets may be necessary. This chapter explores the intricacies associated with the inquiry and page procedures along with communication between members of piconets. It will show how inquiry and page are actually performed and describe in detail the use of frequency hopping and clock drift. This introduction of the lower level protocols of Bluetooth will allow a more accurate analysis of the algorithm complexity metrics, total time taken and total messages sent.

3.1 Inquiry

In the inquiry state, nodes search for other nodes in their vicinity. When the scatternet formation algorithm was introduced, this search was accomplished through the use of a centralized matcher. The matcher collected information about which nodes were inquiring and which nodes wanted to be inquired. It then made matches among as many nodes as possible. The algorithm in reality, however, is decentralized and has no

matcher. Nodes perform the search for other nodes, called inquiry, and the attempt to be found, called inquiry scan, on their own using a specific inquiring procedure.

Attempting to gather information about other nodes in vicinity during inquiry involves obtaining two main things. The two pieces of information that need to be collected when searching are the Bluetooth address and clock of nodes that respond to inquiry attempts. This information exchange occurs when potential masters enter the inquiry substate and potential slaves enter the inquiry scan substate. Potential masters in the inquiry substate send inquiry packets that hopefully will be received by potential slaves. If a potential slave does receive an inquiry packet while in inquiry scan mode, it will send an inquiry response packet back to the potential master that includes its Bluetooth address and clock.

In each of the substates described (inquiry, inquiry scan, and inquiry response) each node utilizes a frequency hopping pattern to determine which frequency to send packets. The intricacies of the frequency hopping pattern are described further in Section 3.3. In general, however, potential masters transmit inquiry packets changing their transmission frequency at a very rapid rate in hopes of finding a potential slave as quickly as possible. By changing frequencies quickly, they query more different frequency channels and increase the probability of finding a potential slave listening on any one of those channels. Potential slaves, on the other hand, change their listening frequency at a very slow rate, waiting for a potential master to find their listening frequency. Because potential masters are unaware of exactly when potential slaves are listening, a potential master will repeat its inquiry transmission many times. Once an inquiry packet that is sent is actually received by a potential slave, an inquiry response packet is returned to the

potential master. The remainder of this section will focus on the low level details of this inquiry process.

In the inquiry substate, nodes in inquiry and inquiry scan use only 32 frequencies as opposed to the 79 frequencies used once connected. The 32 frequencies are the same for all nodes and are derived from the Bluetooth general inquiry access code (GIAC). This address is specified to the frequency hopping finite state machine described in Section 3.3 along with the current clock of the node. The GIAC reserved address is '0x9E8B33'. The 32 frequencies that result from the GIAC are divided into two sets of 16 frequencies. Each set of 16 frequencies is termed a train. While in the inquiry state, potential masters have two possible trains, train A and train B. Determination of which frequencies fall in which train is dependent upon the clock or phase shift of the potential master. All potential masters, however, use the 32 frequencies in the same order; differences only occur when frequencies shift from one train to another.

Using these two frequency trains, potential masters search for possible slaves. Figure 3-1 illustrates the use of the two trains by potential masters as they switch

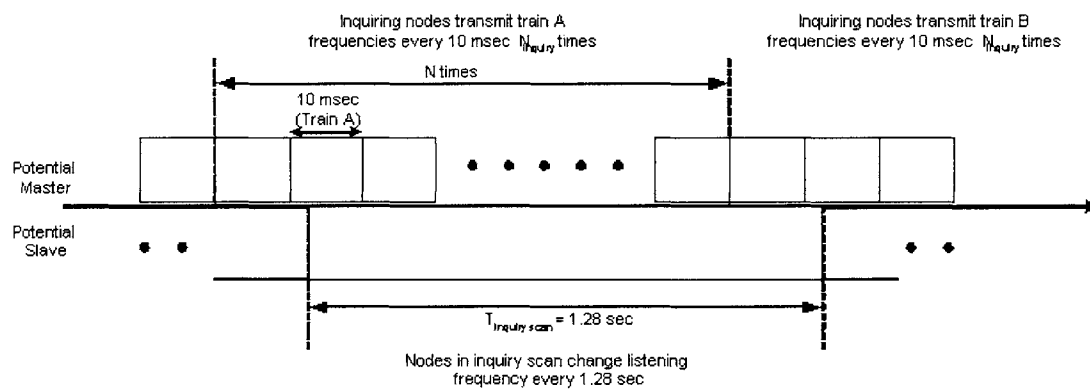


Figure 3-1: Changing of Transmission Frequencies by Potential Masters and Slaves [9]

frequencies at a faster rate than potential slaves. Notice that a master will transmit inquiry packets using the same train N times. N is generally specified to be 256. The time taken to transmit one entire train is 10 milliseconds. This means that 2.56 seconds elapse before a potential master switches trains attempting to collect slaves. The other interesting item to take away from Figure 3-1 is the difference in how quickly potential masters change frequency versus potential slaves. Potential slaves stay on the same frequency listening for 1.28 seconds. Only after this amount of time will they switch to the next frequency as specified by the GIAC. This contrasts the actions of potential masters as they send inquiry packets to an entire train of 16 frequencies in only 10 milliseconds. The reason for this difference is so that potential masters can find slaves as quickly as possible.

During the course of a potential master sending along one frequency train, there are 8 transmit slots and 8 receive slots. Each transmit slot lasts 625 microseconds with transmissions along two frequencies occurring in the slot. Figure 3-2 illustrates each inquiry packet transmission by a master, showing the times during which slaves may

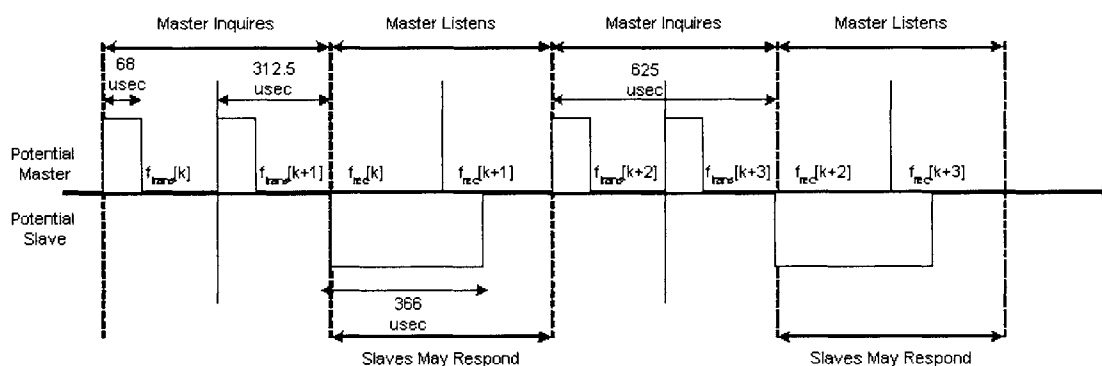


Figure 3-2: Transmission and Response Times by Potential Masters and Slaves [9]

respond. A potential master begins by transmitting at the next two frequencies in the current frequency train being used. It takes about 88 microseconds to transmit a single inquiry packet followed by a period in which the master waits to send the next one. After both inquiry packets are sent, each on a different frequency, the potential master listens for a response in the event a potential slave was listening at either frequency. This makes the hop rate 3,200 hops per second.

A potential slave does not respond as soon as it gets an inquiry packet. Although it is unlikely that multiple slaves will respond to the same inquiry packet, Bluetooth uses an anti-collision mechanism in case this happens. The anti-collision mechanism is a back-off mechanism. Once a potential slave receives an inquiry packet from a potential master, it does not respond immediately. Instead, it enters the inquiry response substate and waits an arbitrary number of slots. The allowable number of slots that a potential slave can wait is between 0 and 1023. Once this waiting period is over, the potential slave will again attempt to receive an inquiry packet from a potential master. When this second inquiry packet is received, it will respond to the potential master.

The actual inquiry response packet sent to the potential master must also be sent at the appropriate time. Figure 3-2 shows that there is a 366 microsecond interval during which potential slaves may respond. Potential masters, however, must be listening on the correct frequency in order to receive the inquiry response packet. Therefore potential slaves wait until the corresponding portion of the next slot before they send a response packet. This is when potential masters expect to receive a response if there is to be one for the frequency transmitted one slot before.

Once a master has collected enough inquiry response packets, it will attempt to complete the connection with each of the nodes by going into the page state. This adds and synchronizes the node into the piconet of the master.

3.2 Page

A potential master pages a potential slave giving it an invitation to join the master's piconet. In order to page, the potential master must know the Bluetooth address of the potential slave along with an estimation of the potential slave's clock. This information can be obtained either after a success inquiry operation or if previous knowledge of the slave already exists. The page state consists of several substates. Potential masters can be in two states, page or master response. On the other hand, potential slaves can either be in page scan or page response.

Paging initially operates in a manner similar to the inquiry operation. A potential master will send page packets on a frequency as determined by the frequency hopping pattern that results from the Bluetooth address information about the potential slave. As with the inquiry operation, potential masters initially hop at a very fast rate of 3,200 hops per second utilizing two trains. The first train, train A, uses 16 frequencies centered around the hopping frequency that the potential master has predicted for the potential slave using the clock as an estimation of phase shift. Train B, the second train consists of the remaining frequencies. Potential slaves, in contrast, hop at a very slow rate. This difference in hop rate is used again to facilitate eventually communicating on the same frequency. Replacing all references to inquiry and inquiry scan with page and page scan, Figures 3-1 and 3-2 show how paging works. Although the operation is fairly similar to

inquiry, some of the parameters are slightly different. T_{pageScan} remains 1.28 seconds however N is now 128 times instead of 256 times. The primary difference between this initial portion of the page and inquiry operation is that when paging, the potential master has some knowledge about where the potential slave is listening. The information comes in the form of the clock estimation that allows a prediction of phase within the hop pattern of the node being paged. This means that the frequencies that page packets are being sent on by potential masters are not completely arbitrary. As a result, the speed by which communication with the potential slave is established is increased when compared against inquiry.

After the potential slave receives an initial page packet, several packets are sent back and forth to complete the connection before moving into the connected state. In

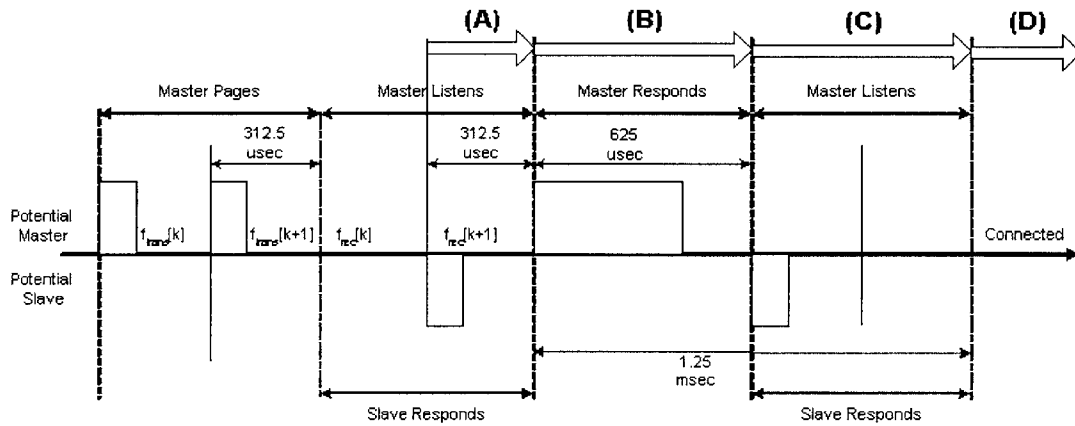


Figure 3-3: The Paging Process from Reception of Initial Page Packet Forward [9]

Figure 3-3, the packets sent before connection are shown. After a page packet is received from a potential master, the potential slave waits for the corresponding time period in the next slot to respond. This is illustrated in (A) of Figure 3-3 as the potential slave responds on its listening frequency of $f[k+1]$. This packet is sent as an acknowledgement

to the potential master. In this time block, the potential slave becomes half-synchronized with the potential master. This is because the potential slave is unaware of whether the page packet sent by the potential master was sent in the first or second half of the transmit slot. Full synchronization occurs in (B) of Figure 3-3 with a packet sent by the master that contains fundamental elements of the piconet. These elements allow the potential slave to join and participate in communications in the piconet. The potential slave lastly responds with one more acknowledgement packet in part (C) and enters the connected state fully becoming a slave in part (D).

The amount of time it takes to page a potential slave is variable. It depends on how long it takes for the potential master to transmit a packet on the frequency upon which the potential slave is listening. However, once this occurs, the process to connection is very straightforward. After the potential slave acknowledges the page with a page response packet, the page process takes roughly a constant time if there are no collisions. The time it takes to become connected after responding to a page packet is 1.25 milliseconds as shown in Figure 3-3.

3.3 Interpiconet

Once connected, interpiconet packets are sent to convey information between a master and its slaves. Slaves act as responsive nodes, only responding when their master sends them a packet. The slaves are fully synchronized with the master in control of communication. The master decides what should be transmitted and whom it should be sent to. Slaves, on the other hand, listen during the slots when packets may be sent. In

the event a packet gets sent with its own address, the slave receives the packet and will respond to it in the next slot.

If a master sends a packet to a slave and a response packet is not received from the slave in the following time slot, some type of collision has occurred. Either the packet sent by the master or the response packet sent by the slave collided with another packet sent by a different piconet. In this case, the master simply retransmits the packet in the next slot.

Interpiconet communication follows a set timing scheme. All nodes in the piconet hop at 1,600 hops per second. This means that a packet can be sent every 625 microseconds. Most of the packets that are sent are single slot packets but packets may last for one, three, or five slots. In each slot, either the master will be in transmit mode or receive mode. Slaves are always in the mode opposite of the master so as to enable communication. Figure 3-4 shows the timing associated with interpiconet communication.

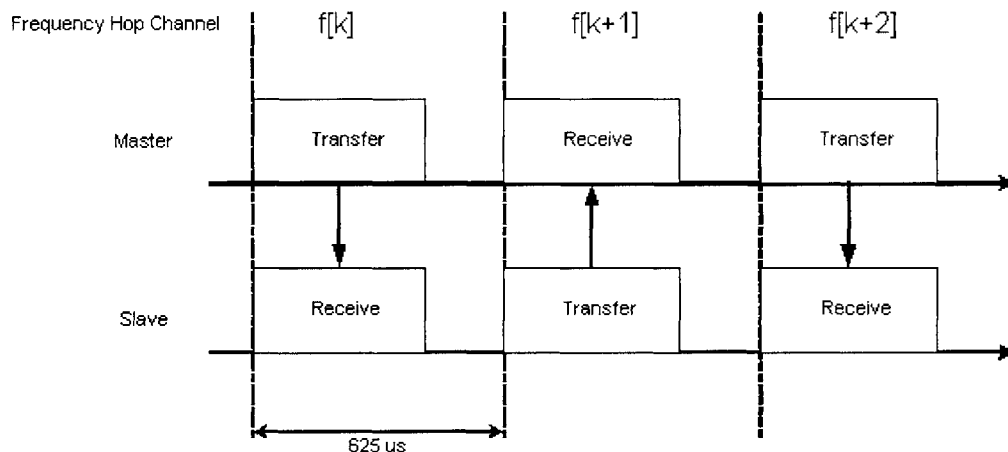


Figure 3-4: Timing for Single Slot Packet Transfer [10]

3.4 Frequency Hopping

The frequency hopping pattern used by the nodes when sending inquiry, page, and intericonet communication packets is quite complex. For this reason, the descriptions of each type of packet transfer described in Sections 3.1 through 3.3 simply use the notation $f[k]$ where k refers to a combination of an address and a clock to show the current frequency channel. This section attempts to describe the actual calculation that occurs in order to arrive at the frequency channel on which to transmit or listen. Chapter 11 of the Bluetooth Specification details the algorithm for hop selection. This algorithm will be described in the rest of this section. The design decisions pertaining to the development of the algorithm, however, are beyond the scope of this discussion.

At the most basic level, the frequency hopping selection kernel can be described as a black box. Figure 3-5 shows the address, clock, and mode as inputs with the output

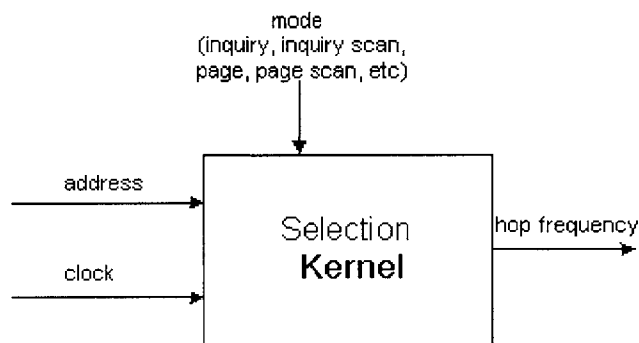


Figure 3-5: General Block Diagram of Hop Selection [11]

the hop frequency. This selection kernel, however, is composed of a variety of operations between bits in the address and clock. These operations include bit addition, exclusive or between bits, and permutation of bits. Figure 3-6 shows an expanded version of the selection kernel.

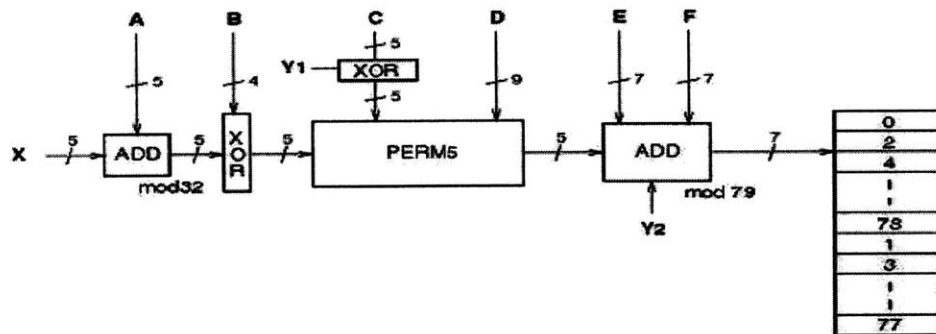


Figure 3-6: Expanded Block Diagram for Hop Selection Kernel [11]

In Figure 3-6, inputs A, B, C, D, E, and F are all selected bits of the address. Inputs X, Y1, and Y2 are derived from the clock. The line across each arrow indicates the number of bits expected for that input or output. For each of these inputs, Figure 3-7 shows the bits of the address and clock that are required. The letter A in the diagram

	Page scan/ Inquiry scan	Page/Inquiry
X	$CLKN_{16-12}$	$Xp_{4-0}^{79} / Xi_{4-0}^{79}$
Y1	0	$CLKE_1 / CLKN_1$
Y2	0	$32 \times CLKE_1 /$ $32 \times CLKN_1$
A	A_{27-23}	A_{27-23}
B	A_{22-19}	A_{22-19}
C	$A_{8,6,4,2,0}$	$A_{8,6,4,2,0}$
D	A_{18-10}	A_{18-10}
E	$A_{13,11,9,7,5,3,1}$	$A_{13,11,9,7,5,3,1}$
F	0	0

Figure 3-7: Bits Used as Input in Frequency Selection Kernel [11]

denotes addresses. The bits to be used of the address follow this designation. Counting begins with the lowest order bits as zero. For the clock inputs, CLK_N refers to the clock of the node itself while CLKE refers to an estimation of another nodes clock. In the calculation for X in page and inquiry mode, the following two equations are needed to derive the input value for X:

$$X_p = [\text{CLKE}_{16-12} + k_{\text{offset}} + (\text{CLKE}_{4-2, 0} - \text{CLKE}_{16-12}) \bmod 16] \bmod 32$$

$$X_i = [\text{CLKN}_{16-12} + k_{\text{offset}} + (\text{CLKN}_{4-2, 0} - \text{CLKN}_{16-12}) \bmod 16] \bmod 32$$

where $k_{\text{offset}} = 24$ for train A, 8 for train B.

With the information from Figure 3-7 and the associated equations for X_p and X_i , all input values have been determined.

Using the input values, three main operations occur as shown in Figure 3-6. The operations are bit addition, exclusive or between bits, and a permutation function that swaps a number of bits. The addition operation is fairly straightforward, simply adding the two or more input numbers and keeping only the specified number of output bits starting with the lower order of the result. The exclusive or function is also quite basic. It requires that each bit in the first input have an exclusive or performed with the corresponding bit in the second input. If the inputs do not have the same number of bits, then some of the higher order bits of one of the inputs will have no exclusive or operation performed. The permutation function, however, is rather complex.

The permutation function acts as a swapping mechanism, switching various bits within a single input. It could, for example, switch the second and fifth bits. There are fourteen possible switches that can occur using this function. Whether a switch occurs is determined by the two inputs above the permutation box in Figure 3-6. If a particular bit

is one, then the corresponding switch will occur. Since there are fourteen total bits above the permutation box, at most fourteen switches could occur. Figure 3-8 outlines the possible switches that could occur if triggered by the input bits above the permutation box. In this figure, Z refers to the input on the left side of the permutation box that is being swapped from Figure 3-6. Q refers to the input that results from the exclusive or between C and Y1. Input D will not be renamed as it is directly inputted into the permutation box. The output after the permutation operation is completed is a five-bit number.

Swap Determination	Q4	Q3	Q2	Q1	Q0	D8	D7	D6	D5	D4	D3	D2	D1	D0
Swap Bits	Z1, Z2	Z0, Z3	Z1, Z3	Z2, Z4	Z0, Z3	Z1, Z4	Z3, Z4	Z0, Z2	Z1, Z3	Z0, Z4	Z3, Z4	Z1, Z2	Z2, Z3	Z0, Z1

Figure 3-8: Permutation Determination with Bits to be Swapped [11]

After all the operations are complete, the last step is to perform a modulo 79 on the result. This will yield a number between 0 and 78. In order to determine the frequency upon which to transfer, this number is used as an index on a list of frequencies. The list of frequencies is ordered with all the even frequencies first, starting at zero, followed by all the odd frequencies, starting at one. Using the index, the corresponding frequency is the transmission or listening frequency.

3.5 Clock Drift

It is important that the master and each of its slaves remain synchronized. Synchronization occurs so long as the estimated master clock used by each slave of a piconet remains the same as the exact clock maintained by the master. The Bluetooth

Specification states that clock estimations drift at a maximum of 20 parts per million relative to the 625 microsecond slot interval. Correction of clocks can occur by slaves by comparing the expected receive time of a packet with the actual receive time. The packet does not even need to be sent to that particular slave since all slave are aware of the frequency hopping pattern. A slave could arbitrarily listen for any packet in order to resynchronize. This scheme ensures all slaves stay synchronized with the master and his associated hopping pattern. As a result, intericonet packets will be delivered assuming there are no collisions.

Chapter 4

Simulations and Results

Two main simulations were run in order to assess the performance of the scatternet formation algorithm. The first simulation, termed the scatternet formation simulation, executed the message-passing algorithm as described in Chapter 2, attempting to find quantitative results for the performance metrics. The scatternet formation simulation accurately found results for the quality of a scatternet, namely the maximum degree of a node, the network diameter, and the number of piconets formed. It also kept track of information concerning the total number of inquiry, page, and interpiconet messages sent. This simulation, however, abstracted away each message identically. In reality, sending some message types takes a longer amount of time than sending others. This is because each message type sends a different number of packets before the actual message is received.

To accurately measure time and packet complexity of the algorithm, a lower level simulation had to be performed. The link control simulation performed the inquiry and page functions defined in Chapter 3 to determine how many packets constituted each message type for various numbers of masters and slaves. The combination of the results from this simulation and the scatternet formation simulation yield a determination of the algorithm complexity metrics for the scatternet formation algorithm.

4.1 Simulation Tool

Both the formation and link control simulation were implemented in Java. This allowed the use of object oriented programming making each node its own entity, running on a separate thread. A Java based discrete event simulation tool called Simjava was used to simulate the sending and receiving of messages between nodes during discrete time intervals.

Simjava is quite a versatile tool. It is used to simulate an environment in which many objects, nodes in this case, frequently interact by sending messages to one another. Nodes are linked together by ports. These ports allow messages to be sent between connected nodes. The nodes in the environment derive from an entity class specified by Simjava. Deriving from the entity class makes each node have its own thread. A central system class is responsible for controlling the threads. This centralized system class also controls simulation time and delivers messages as appropriate to each node. It is aware of each node and can signal termination of the simulation.

The behavior of each node is implemented within the class that derives from the entity class. Simjava provides many useful actions that allow ease in implementation. Simply executing through a loop each time interval, each node performs actions and changes state as necessary. Nodes can effortlessly halt for a specified amount of simulation time, check for messages waiting, and send messages instantaneously or with a delay. This gives an excellent framework for communication between nodes. In combination with the implemented node behavior, this framework allowed the simulation of the scatternet formation algorithm as well as the link control protocols.

4.2 Scatternet Formation Simulation and Results

The scatternet formation simulation acted as a model of the corresponding algorithm presented in Chapter 2. This simulation needs only one parameter, the number of nodes with which to form the scatternet. At termination, the simulation reports the scatternet quality performance metrics as well as information about the maximum and total number of messages sent of each type. In a more detailed view, the simulation can also return the scanners and seekers for each round along with the connections formed and corresponding relocation actions. This detailed view also shows the final scatternet configuration with where each node resides within the scatternet.

The simulation contains three Java classes: Node, Matcher, and Message. The majority of the behavior of this simulation resides in the Node class including node states, roles during relocation actions, and timing of messages sent and received to ensure communication. The Matcher class, an abstraction of lower level link control protocols, is solely responsible for collecting seekers and scanners at the beginning of each round and pairing as many together as possible, returning pairings to the appropriate nodes. Message is the class used as a wrapper for information sent between nodes as well as for information between the matcher and a node. The complete pseudocode for this simulation can be found in Appendix A. It shows the methods called by nodes and the matcher as well as the timing of when to send and receive messages.

The simulation was run for 50 trials with a varying number of isolated nodes initially. The number of nodes used ranged from 2 to 128. After running with 2, 4, and 8 nodes, the number of nodes was increased by 8 incrementally until 128 was reached. This large sample space allowed linear as well as logarithmic analysis based on the

number of nodes. In many cases, the algorithm had theoretical expectations for the results. These expectations are not derived here; a derivation can be found in “A Bluetooth Scatternet Formation Algorithm” [8]. Frequent reference, however, will be made to these expectations in an attempt to analyze the scatternet formation algorithm.

By design, in the scatternet formation algorithm, the maximum degree on any one node within the final scatternet is two. This means that any node at most time-shares between two unique piconets. This was indeed true of the algorithm as depicted by Figure 4-1.

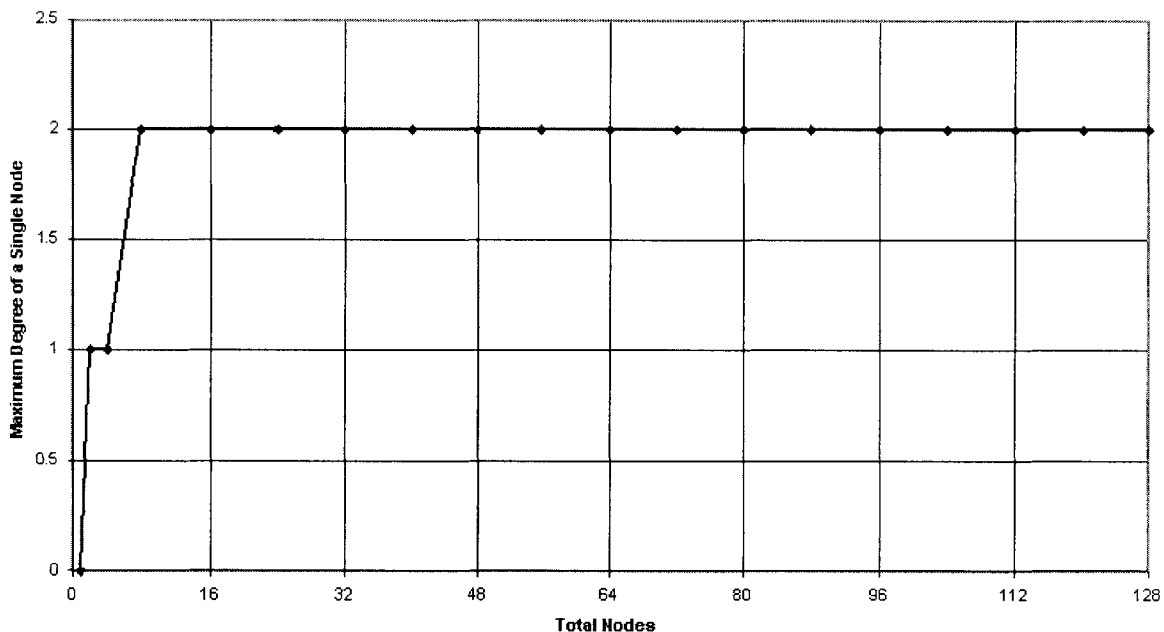


Figure 4-1: Maximum Degree of a Single Node

This result is quite important because it means that any one slave will not be overwhelmed because it has to listen to communication in several different piconets. For total nodes in excess of 8, in fact, the maximum degree of a single node is minimized at two because more than one piconet is needed. For node values of 8 or less, since only a single piconet is formed, the maximum degree of a single node is one. The algorithm

performs optimally in all cases except for exactly 8 nodes. Eight nodes can be grouped into one piconet yielding a maximum degree of one. The scatternet formation algorithm, however, breaks the 8 nodes into two piconets yielding a maximum degree of two. This is done, however, to allow other nodes to join the configuration at a later time.

The number of piconets formed in the final configuration is important because each piconet maintains its own hopping sequence for communication. An excessive number of piconets would flood the communication channel causing collisions at many different frequencies. Figure 4-2 shows the average number of piconets that resulted using the scatternet formation algorithm for various node topologies.

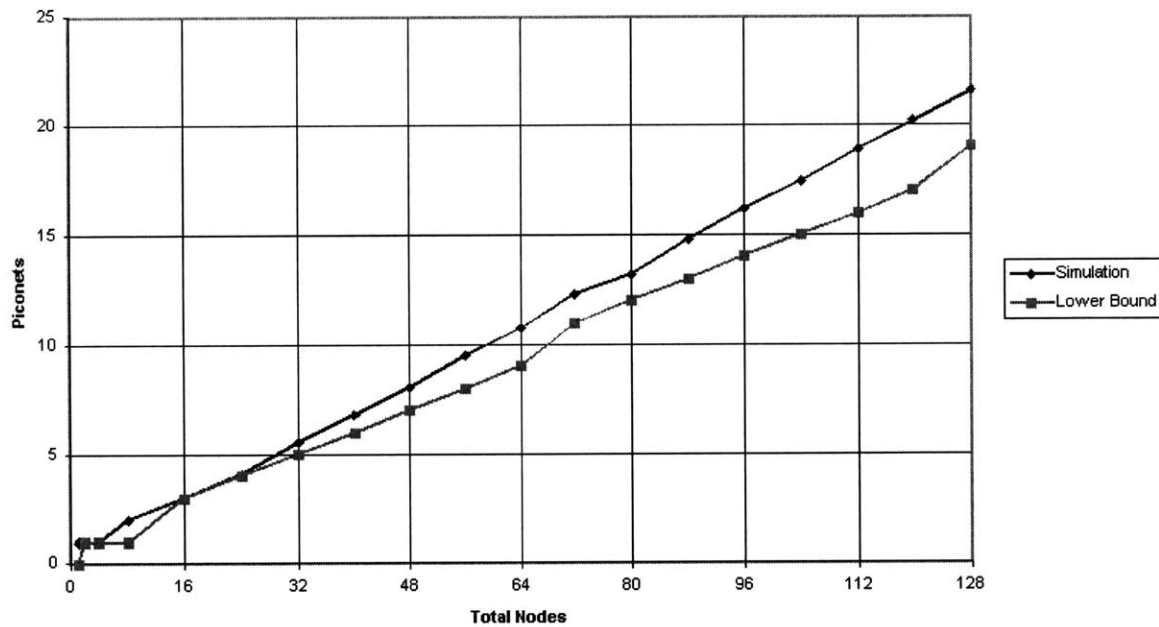


Figure 4-2: Number of Piconets Formed

The figure is shown in a linear scale on both axes, as the theoretical expectation for the number of piconets is also linear. The theoretical lower bound on the number of is defined as the following: number of piconets = $\lceil (n-1) / 7 \rceil$ where n is the number of

nodes. The number of piconets formed only slightly above the theoretical lower bound for up to 80 nodes with only a slighter larger deviation for more than 80 nodes. Even in the case of the largest deviation, at 128 nodes, the deviation is only about 2 or 3 piconets with the theoretical lower bound 18 and the average number of piconets from the simulation 21.5. This means that the number of piconets formed is near optimal and the communication channel will not be flooded.

The maximum diameter of the scatternet formed is also important as it determines the largest number of hops a node will have to make in order to communicate with another node in the scatternet. There were no theoretical bounds for this performance metric. The results, however, seem sufficient. The diameter of a scatternet varies with $\log(n)$ where n is the total number of nodes in the scatternet. Figure 4-3 illustrates this relationship.

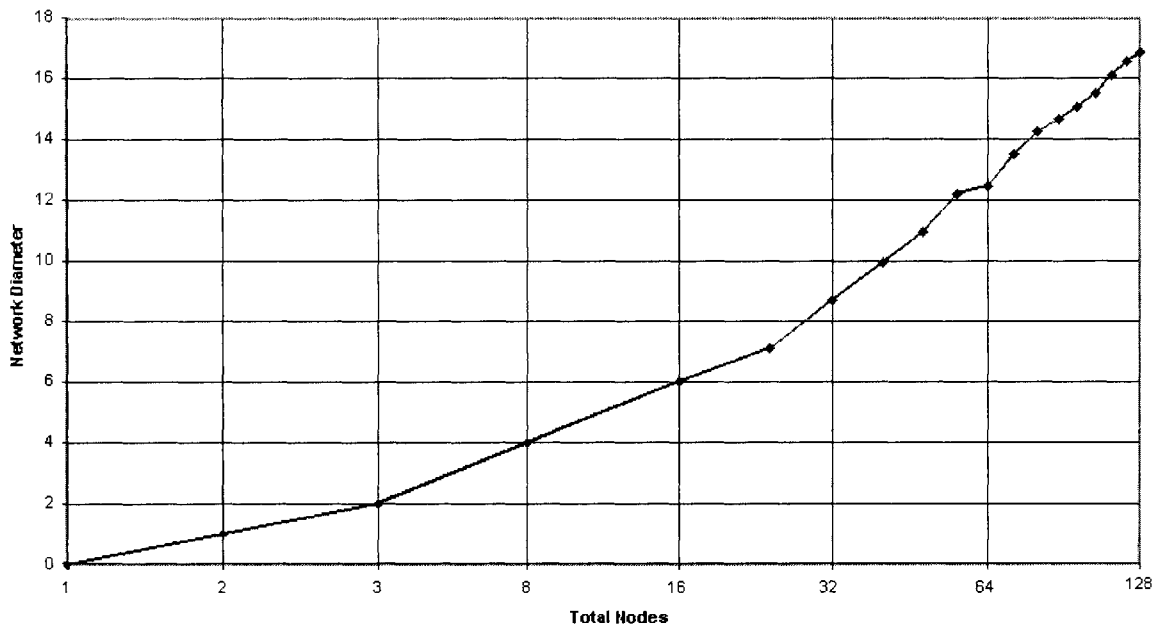


Figure 4-3: Network Diameter

The maximum number of hops could perhaps have been minimized even further, but at the expense of degrading other performance metrics.

The number of rounds required to complete the algorithm varies since the algorithm is randomized. Leaders choose to seek or scan each round in a probabilistic fashion. Figure 4-4 shows the average number of rounds to completion for different node topologies in a logarithmic scale.

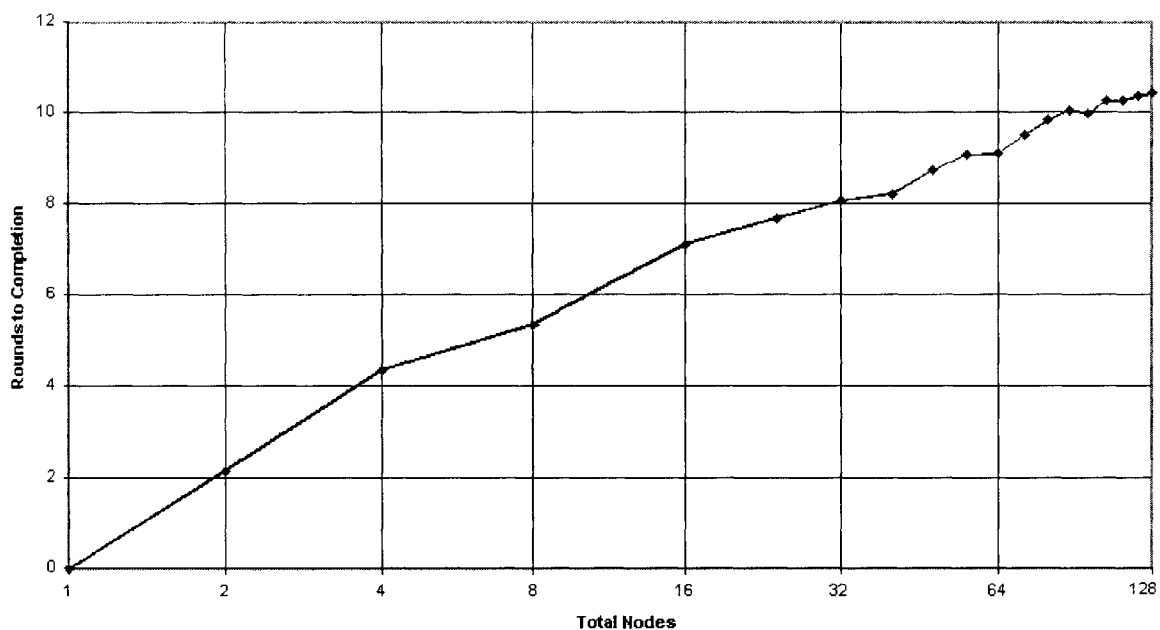


Figure 4-4: The Number of Rounds to Completion

The theoretical expectation for the number of rounds to completion is $\log(n)$ where n is the number of nodes in the topology. This means that a logarithmic (total nodes) versus linear (rounds to completion) graph should produce a linear result to match expectations. Figure 4-1 roughly approximates a line with only slight discontinuities at 64 and 96 nodes. These discontinuities can be attributed to a possible need for more trials to be run. Initially, this simulation was run with only 20 trials, yielding a graph with larger

discontinuities at more total node values in the linear approximation. With 50 trials, the figure smoothed significantly and running more trials would have smoothed the line even further.

The total number of inquiry, page, and intericonet messages sent was expected to increase linearly with the number of nodes in the topology. Figure 4-5 shows the actual results through simulation. Each message type was quantified separately since each takes a different number of packets to complete based on the link control protocols.

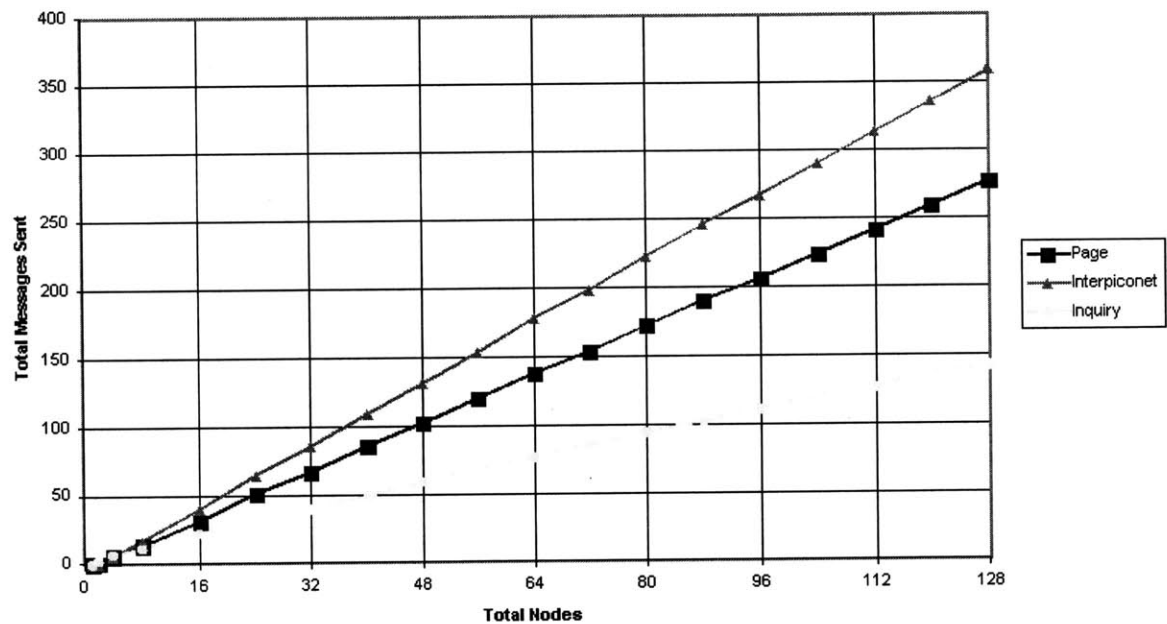


Figure 4-5: Total Inquiry, Page, and Intericonet Message Sent Through Completion

The results matched expectations quite well as depicted in the linear (total nodes) versus linear (total messages sent) scale on Figure 4-5. There were, in fact, not even slight deviations from the linear approximations for any node topologies in each of the three message types.

Using results from the link control simulation in combination with the number of rounds to completion and the total number of messages sent, algorithm performance metrics of the scatternet formation algorithm can be analyzed.

4.3 Link Control Simulation and Results

The link control simulation attempts to measure the actual number of packets, and the corresponding amount of time associated with each message sent. Only inquiry and page messages were simulated, as the time and number of packets sent for interpiconet messages are clearly defined within the Bluetooth Specification. For interpiconet messages, slots are alternated every 625 microseconds between possible master transmission and slave response. A master transmits one packet, perhaps over multiple slots, expecting a response from a slave in the next slot unless a collision has occurred. Of more interest are inquiry and page messages. Inquiry messages do not use a centralized matcher, as defined in the scatternet formation algorithm and simulation, to find other nodes. Instead, a node queries for other nodes in their vicinity sending packets using the protocols defined in Section 3.1. As a result, the time for an inquiry message to be received and responded to depends on the number of masters and slaves in the environment and the number of resulting collisions. Paging is also interesting because in this case nodes are aware of each others address and clock as opposed the case in inquiry. This information allows the node performing the page to use the hop sequence associated with the node to be paged, almost at exactly the correct phase shift.

In modeling the inquiry and page protocols, the simulation ran with two input variables, the number of masters and the number of slaves. Although inquiries allow

masters to collect as many slaves as desired before switching to page mode to complete the connection, this simulation allowed masters to collect only a single slave inquiry mode before paging for that node. The reason for this strict definition in the simulation is because in the scatternet formation algorithm, seekers collect at most only one node. This allows for proper evaluation of the scatternet formation algorithm. After a successful inquiry, the master then proceeds to page the potential slave.

A master can begin paging the potential slave that it is collecting anytime after the inquiry process is completed. For the purposes of the simulation, an attempt was made to separate inquiry and page so as to obtain distinct results for each, without one process inhibiting the other. As a result, masters began paging for slaves after a specified amount of time, T_{switch} . After T_{switch} , either all masters have collected one potential slave or the slaves in the vicinity for collection had been exhausted. Masters that have collected potential slaves then begin paging to add a collected slave to their piconet.

At the completion of the simulation, each master that has collected a slave outputs the total amount of time taken to inquire for the slave as well as the time taken to complete the connection by paging for the slave. Also shown is the total number of packets sent by all the nodes together along with how many of those packets collided with other packets sent on the same frequency at the same time.

The simulation is implemented using the Simjava package along with four other classes: Node, Air, Message, and Hopper. The Node class contains the current state of the node (master or slave) and determines when the node should transmit and listen in the inquiry and page states. Masters also keep track of responses to inquiries with the address and clock of the slave to be collected. Clock drift was not implemented in this

simulation so the clock estimation used by the master in paging a potential slave is actually exactly the clock of the slave. This was primarily because introducing clock drift requires an extra 20 bits of resolution on the clock, slowing simulation time significantly. Furthermore, since only one node was being paged relatively quickly after the initial inquiry response, a clock that reflected the drifted value would not affect page time significantly. There was not a large delay between inquiry and page so clock drift was deemed negligible. Air was used to check for collisions when messages were sent between nodes. Only non-collided messages were available after checked by Air. Messages were used to communicate between nodes. Hopper implemented the inquiry and page hopping as illustrated in Section 3.4. It was implemented exactly as described in the Bluetooth Specification. For correctness, the hopping pattern was checked against results produced by the System Design Group at Seoul National University [12]. The hop patterns are identical. BlueHoc, an open source Bluetooth simulation tool made by IBM, was not used because this simulation was initiated prior to its release [13].

This simulation was performed with approximately ten trials with various numbers of nodes. The number of nodes increased exponentially by 2^n where n is all integers between zero and seven inclusive. For each node topology, the simulation determined inquiry time, page time, as well as the number of messages sent that collided. For each topology (except the one with two nodes), the simulation found results in the case that 25%, 50%, and 75% of the nodes were masters with the remaining percentage as slaves. This was done because in the scatternet formation algorithm, although the expected number of seekers and scanners is 50%, there is a range in which masters and slaves are likely to distribute themselves. This range is created using the approximation

that nodes are likely (~95% of the time) to distribute themselves within two standard deviations from the expected value of masters and slaves. The range of masters within two standard deviations for various numbers of nodes using an even probability of seeking and scanning is shown in Figure 4-6.

Nodes	Probability	Expected Value	Two Standard Deviations	Master Lower Bound	Master Upper Bound
4	0.5	2	2.0	0	4
8	0.5	4	2.8	1	7
16	0.5	8	4.0	4	12
32	0.5	16	5.6	10	22
64	0.5	32	8.0	24	40
128	0.5	64	11.3	52	76

Figure 4-6: Upper and Lower Bound for Two Standard Deviations

Notice that in most cases, especially for larger node topologies (greater than eight nodes), the range of masters used (25% - 75%) easily encompasses this two standard deviation benchmark. For smaller node topologies, this was not as much of an issue since inquiry and page time would be quick regardless given the small number of nodes in the environment.

The inquiry time measured was the time taken until the last possible inquiry response was received. This means that if in the topology the number of masters exceeds the number of slaves, the time measured is how long it takes the last slave to send an inquiry response that is received by a potential master and proceed into the page state. If, on the other hand, the number of slaves exceeds the number of masters, the time

measured determines how long it takes for the last master to obtain a potential slave through inquiry and proceed to paging. In the case of an equal number of masters and slaves in the topology, both of the previous conditions mentioned are met. This is because each master is exhausted by finding a slave to page and similarly, each slave is exhausted by obtaining a potential master. This time, therefore, gives an accurate measurement of how long it takes the entire topology to complete the inquiry process with each master collecting at most one slave. The results are shown in Figure 4-7 with the various distributions of masters and slaves within each the seven initial topologies (2, 4, 8, 16, 32, 64, 128 nodes).

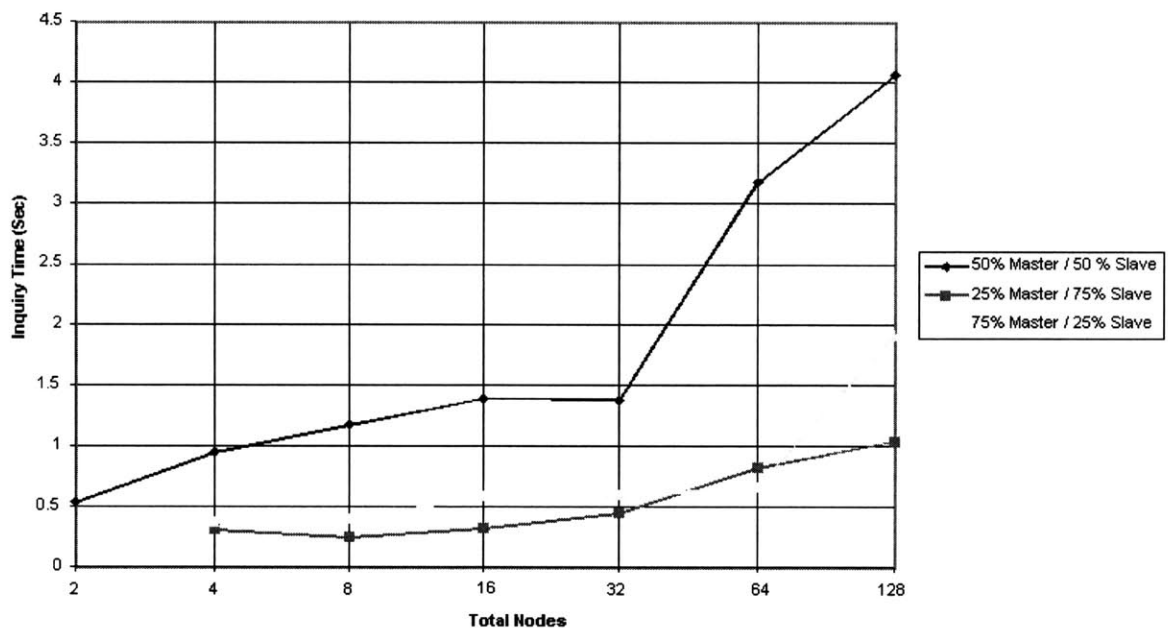


Figure 4-7: Inquiry Time for Various Node Topologies and Distributions

The first observation to make with regard to inquiry time is that it always takes longer for an even distribution of masters and slaves to complete the inquiry process as compared to skewed distributions. This also intuitively seems correct because in the case of the even distribution, every master must complete the inquiry process before

proceeding to paging. In each of the skewed cases, some masters do not either receive a slave or vice versa, some slaves do not find a master. This means fewer total inquiries need to complete before paging begins. With fewer inquiries to complete, the time taken is obviously going to be shorter.

In the case of the two skewed distributions, both need to make the same number of connections. The difference is one case has excess masters while the other has excess slaves. This is also evident from Figure 4-7. Notice that both take roughly the same amount of time for smaller topology sizes, but there begins to be a discrepancy at 64 nodes that is clearly visible at 128 nodes. This discrepancy occurs because masters transmit far greater packets than slaves do in inquiry. Slaves only respond when an inquiry packet is received, while masters flood channels with two inquiry packets every transmit slot. In the skewed case with more masters at 128 nodes, 96 masters are inquiring at the same time. This causes many collisions with few packets actually available to slave for reception. Having 32 masters with 96 slaves, in contrast, does not saturate the channel and therefore yields a much faster completion time.

The combination of a large number of masters and having to complete the maximum number of inquiries is evident in the symmetric case at 64 and 128 nodes. There is indeed a large jump in inquiry completion time that results from the combination of these two factors. The time to completion roughly doubles in the 64 node case from that of the 32 node case.

The final interesting point to take away from the inquiry process is that for node topologies between 8 and 32 nodes, roughly the same amount of time was taken for the different number of nodes within the same distribution. For the skewed cases, this can

again be attributed to excess nodes with fewer inquiries needed to complete. In the symmetric case, however, this can be attributed to the use of trains in the Bluetooth link control protocol. A train is used 256 times in a row before switching to the opposite train; that is 2.56 seconds. As a result, if only one master and slave is left in the inquiry process that are not on coinciding trains, the final inquiry will not complete until after the master switches trains. In these three cases, at 8, 16, and 32 nodes, the channel is not saturated so collisions occur at about the same rate. Since all three cases require the final nodes to complete, their times to completion are in close proximity.

Paging time measures the amount of time for a node to complete paging within the topology once inquiry is complete. This time is much shorter than inquiry because it involves two nodes, a master and a slave, that are specifically trying to connect together. The master has the knowledge of the address and clock of the potential slave. Figure 4-8 illustrates paging times.

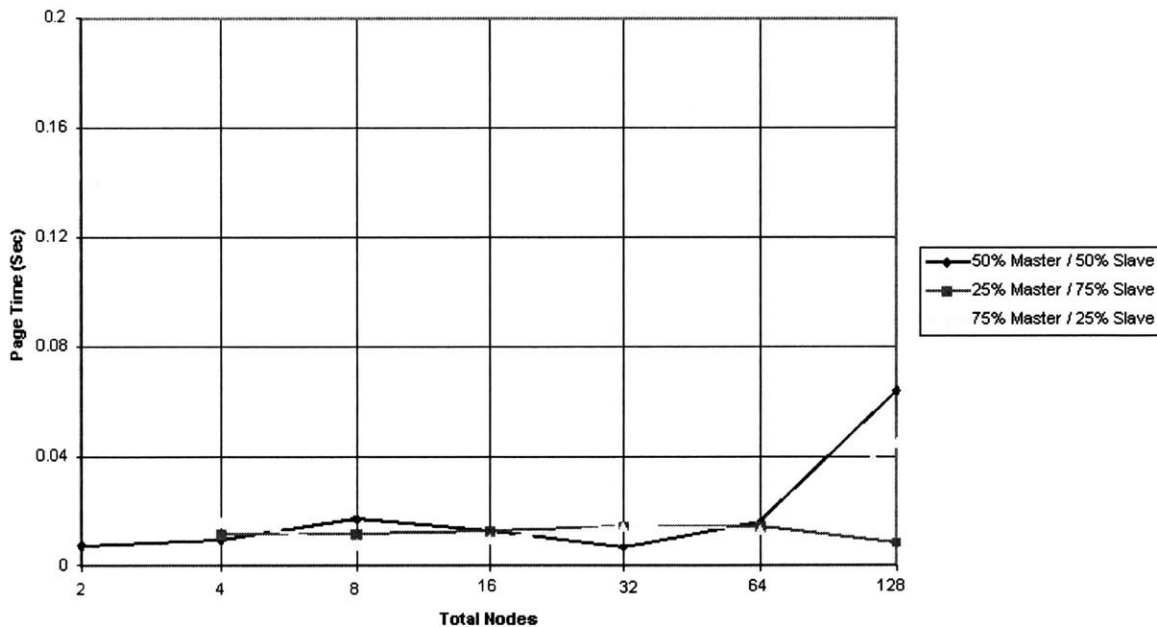


Figure 4-8: Page Time for Various Node Topologies and Distributions

Notice that paging time is quite fast, in fact more than an order of magnitude faster than corresponding inquiry times. Also, paging time is roughly constant for all node topologies and distributions. It only deviates in the 128 node case with 64 and 96 masters. This can be attributed to the channel being saturated with too many packets and as a result a significant number of collisions. Even with these collisions, however, the page is still fast with respect to inquiry.

Collisions are a major cause in the delay of completing the inquiry and page process. It is important to see how large of scatternet can be formed with initially isolated nodes without excessive messages being lost due to collision. Figure 4-9 shows the percentage of packets that collide based on the symmetric node case. The packets shown in the figure indicate the number of packets sent through the completion of inquiry and page.

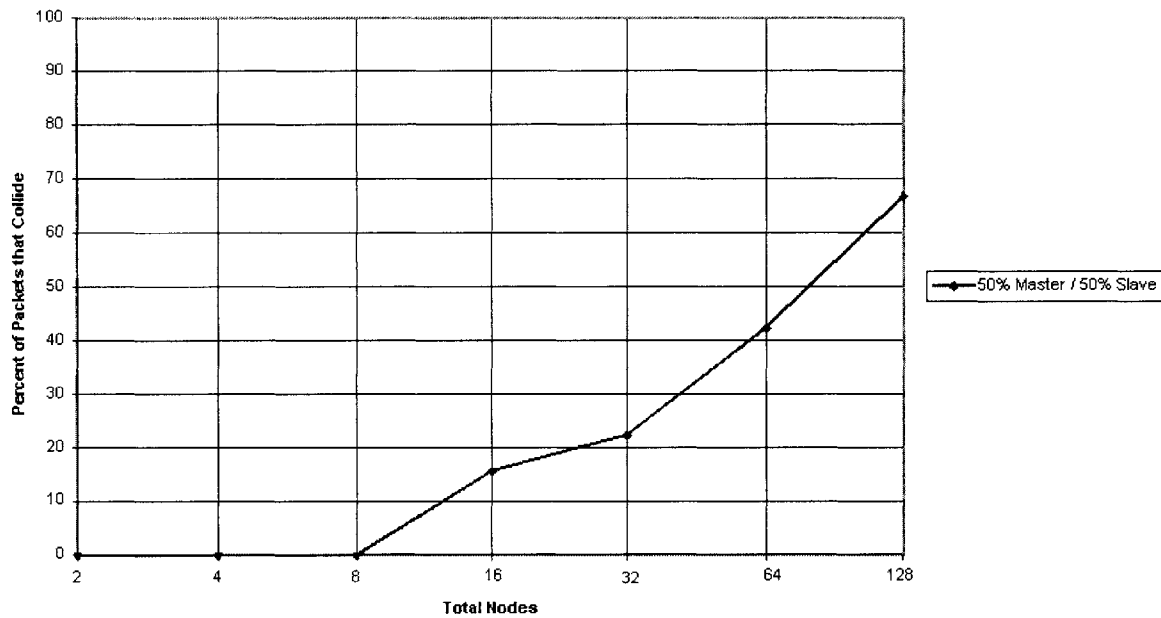


Figure 4-9: Percentage of Packets that Collide Through Inquiry and Page

Significant channel degradation occurs at 64 nodes with about 45% of packets sent collided with others. The channel becomes much worse with 128 nodes, with more than 65% of packets sent not capable of being received. Collision percents in other node topologies are acceptable.

The total number of packets sent increases as more nodes are in the topology because more inquiries and pages have to be made. There are also significantly more collisions. Figure 4-10 shows the total number of packets sent in order to complete both inquiry and page for the symmetric case with equal numbers of masters and slaves.

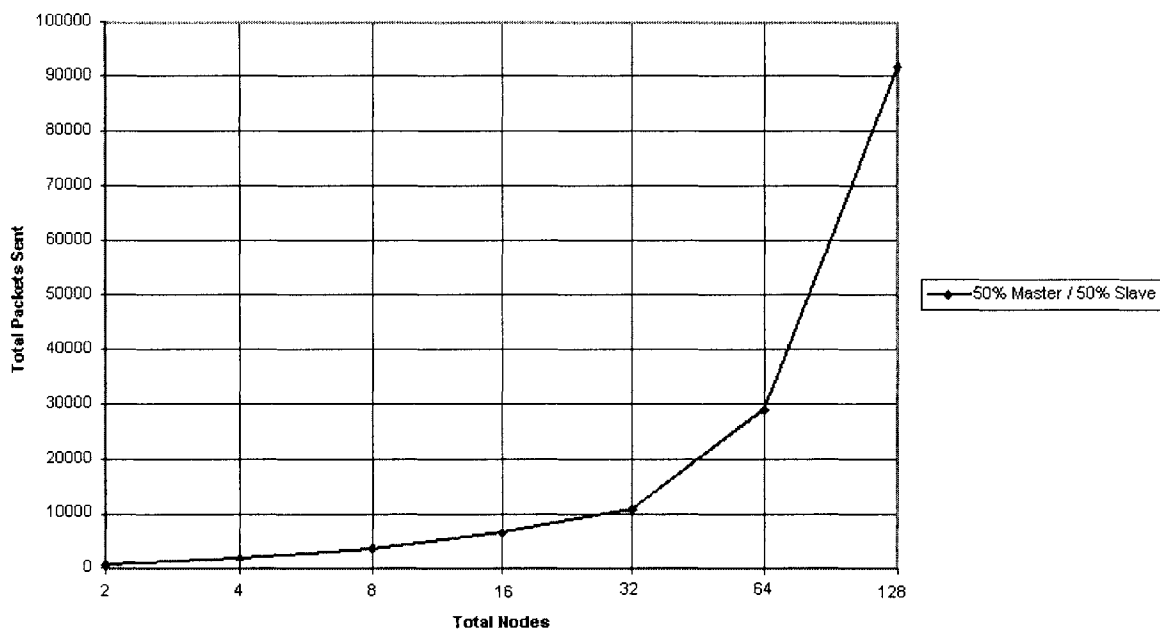


Figure 4-10: Total Packets Sent Through Inquiry and Page

Notice the sharp increase at 64 and 128 nodes. This corresponds very well to the sharp increase in the percentage of packets that collide in Figure 4-9.

4.4 Complete Scatternet Formation Results

This section aims to bring together the results presented in Sections 4.2 and 4.3 to give quantitative results for the algorithmic metrics. It attempts to find the worst-case amount of time to completion and associated number of messages sent.

Time to completion can be modeled quite simply. It is just the number of rounds multiplied by the time per round. Each round has a maximum amount of time associated with it based on the characteristics derived from the number of nodes in the topology. Figure 4-11 shows the typical actions that can happen during a round in a timing diagram. The timing diagram correlates to the one presented in Chapter 2 when the scatternet formation algorithm was introduced.

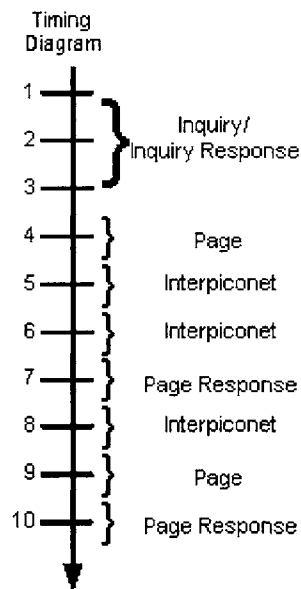


Figure 4-11: Timing Diagram for a Round in the Scatternet Formation Algorithm

The inquiry process, however, takes far longer than the sending page messages and interpiconet messages. Since there is a difference by orders of magnitude, the time taken to complete a round can be approximated to be the inquiry time for that topology.

The total number of packets sent can also be approximated since inquiry and page send far more packets together than simple intericonet communication. Using the worst-case assumption that nodes are divided exactly evenly as masters and slaves each round, an upper bound can be found on the packets sent by multiplying the number of rounds by the packets sent each round. This does not take into account the reduction in nodes querying each round, as fewer masters remain active as leaders. As a result, the approximation is an upper bound. In actuality, significantly fewer packets would be sent because there are not as many nodes querying in the later rounds.

The results of the performance metrics are outline in Figure 4-12. The figure contains completion time and packets sent calculated as described. It also includes the scatternet quality performance metric results from Section 4.2. The results are shown for small scatternets to large scatternets, with the number of nodes ranging from 2 to 128.

Number of Nodes	2	4	8	16	32	64	128
Time to Completion	1.14	4.10	6.28	9.85	11.15	28.89	42.37
Messages Sent	1856	6625	10196	15913	17926	46461	68855
Piconets	1	1	2	3	5.54	10.76	21.58
Network Diameter	1	1	4	6	8.72	12.44	16.84
Maximum Degree	1	1	2	2	2	2	2

Figure 4-12: Scatternet Performance Metrics

Chapter 5

Conclusion and Future Work

This thesis presents and evaluates a message-passing scatternet formation algorithm for Bluetooth. The evaluation is completed in two stages, through a high-level scatternet formation simulation supplemented with a link control simulation. The link control simulation is used to aid in finding upper bounds on the time to completion as well as the number of packets sent.

An analysis of this scatternet formation algorithm shows that it indeed forms scatternets of good quality. The number of piconets and maximum degree of a single node are close to maximization. The network diameter seems viable given the number of nodes in the topology. The completion time and total number of packets sent are both found to have acceptable upper bounds.

Some performance metrics could have been better evaluated with one integrated simulation instead of integrating the results of two separate simulations. With a single simulation, a much tighter bound on the total number of packets sent would be available. This is because as rounds progress, fewer leaders exist. As a result fewer messages are sent in comparison to the worst case in which all nodes are leaders every round – half as masters and half as slaves. For the upper bound analysis, it was assumed that this worst-case existed each round until completion.

An upper bound for the time to completion for the scatternet formation algorithm was also established. As presented, the algorithm is a synchronous algorithm that operates in rounds. Each round is assumed to run for a specified amount of time, primarily based on how long the inquiry process takes in the worst-case. This worst-case occurs with an equal number of masters and slaves when all nodes are leaders. A more accurate bound on time to completion can be obtained taking into account the actual number of active nodes inquiring each round, rather than having nodes wait for the worst case timeout before proceeding to complete the round. This would entail having each node knowing exactly when that last inquiry completed for the current round, eliminating the idle time waiting for the worst-case timeout to occur. With a synchronous algorithm, this is quite difficult.

Performing this algorithm in an asynchronous fashion has many benefits. The completion time of the algorithm will be significantly faster as nodes do not have to wait for the worst-case inquiry time to guarantee that all nodes have completed inquiry. Instead, as nodes find one another, they can perform the proper relocation action and quickly return to inquire for other nodes that may be searching. There is no need for any worst-case assumptions that involve the number of nodes in the network – an arbitrary number of nodes can exist. Future investigation of the scatternet formation will proceed in evaluating this type algorithm.

Appendix A: Scatternet Formation Pseudocode

This appendix contains the pseudocode for the Scatternet Formation Algorithm as defined in Section 2.4. The pseudocode is written as a message-passing algorithm; each message sent is explicitly indicated. There are a couple of important facts concerning the messages that are sent:

- each message by default contains the ID of the sender and receiver
- messages that are sent can be received immediately, there is no delay

Figure A-1 illustrates the possible flows of execution of the various methods in each round of the algorithm. A line between two methods indicates that the higher of the two methods calls the lower. This figure was made to ease in the readability of the pseudocode.

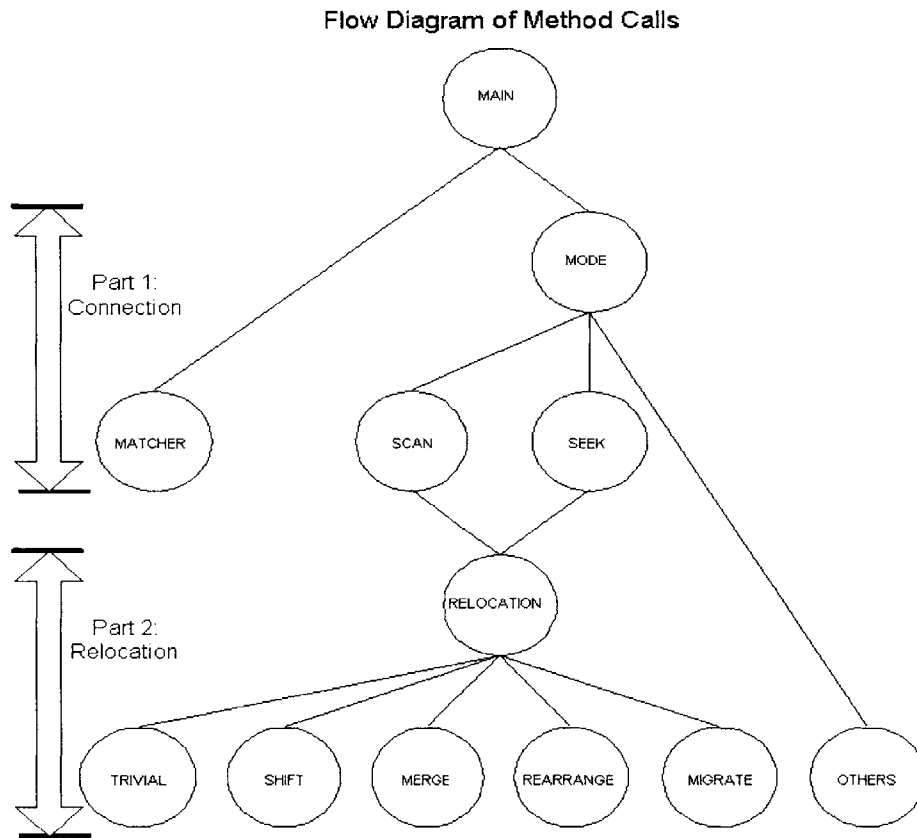


Figure A-1: Method Execution Flow in Pseudocode

```

MAIN () {
1   Z ← { all nodes }
2   each node in Z is a leader
3   m ← a matcher
4   round ← 0
5   clock ← 0 //clock goes until 10 each round
6   while (number of leaders in Z > 1)
7       increment round
8       reset and start clock
9       MATCH (m)
10      for each node in Z
11          n ← current node
12          MODE (n, m)

```

```

MODE (node n, matcher m)
1   if (n is a leader)
2       x ← a random number in [0, 1)
3       if (x < p) where (1/3 < p < 2/3)
4           SEEK (n, m)
5       else
6           SCAN (n, m)
7   else if (n is an unshared slave)
8       SCAN (n, m)
9   else if (n is a shared slave or master or master/slave)
10      OTHERS (n)

```

```

MATCH (matcher m)
1   m holds until clock is 2
2   m obtains inquiry messages from seekers and scanners at time 2
3   connect ← min (the number of seek messages, the number of scan messages)
4   pairings ← 0 //connection made so far
5   while (pairings < connect)
6       increment pairings
7       a ← an arbitrary node ID that sent a seek message
8       b ← an arbitrary node ID that sent a scan message
9       m sends an inquiry response message to a with the ID of b at time 3
10      m no longer uses messages sent by a or b
11      m holds until clock is 10

```

```

SEEK (node n, matcher m) //n must be a leader
1   n holds until clock is 2
2   n sends an inquiry message to m with seek at time 2
3   n holds until clock is 3
4   if (m returns an inquiry response message at time 3)
5       v ← the slave to be added as directed by m
6       n sends a page message to v with piconet size at time 4
7       n holds until clock is 7
8       n obtains a page response from v with relocation action at time 7
9       r ← the relocation action with participating node IDs
10      n holds until clock is 8
11      RELOCATION (r)
12  else //no seek match made this round
13      n holds until clock is 10

```

SCAN (node n, matcher m)

```

1      if (n is an unconnected leader)
2          n holds until clock is 2
3          n sends an inquiry scan message to m with scan at time 2
4          n holds until clock is 4
5          if (a page message is returned at time 4)
6              u ← the ID of the leader sending the page message
7              size ← the size of the piconet in which u is leader
8              if (size + 1 < k where k is max piconet size = 8)
9                  // size + 1 to include the node scanning
10                 r ← the relocation action trivial with IDs (u, n)
11             else //this slave being added is the seventh
12                 r ← the relocation action shift with the IDs (u, n)
13             n holds until clock is 7
14             n sends a page response message to u with relocation action r
15             n holds until clock is 8
16             RELOCATION (r)
17         else //no scan match made this round
18             n holds until 10
19     else if (n is a connected leader)
20         n holds until clock is 1
21         v ← an unshared slave of n
22         n sends an interpiconet message to v with scanner at time 1
23         n holds until clock is 5
24         if (an interpiconet message is returned with size from v at time 5)
25             u ← the ID of the leader sending the page message to v
26             size ← the size of the piconet in which u is leader
27             if (size + n's piconet size < k where k is max piconet size = 8)
28                 r ← the relocation action merge with the IDs (u, v, n)
29             else if (u is an unconnected leader)
30                 r ← the relocation action rearrange with the IDs (u, v, n)
31             else
32                 i ← min(k – the size of n's piconet, the size of u's piconet – 2)
33                 r ← the relocation action migrate with the IDs (u, v, n) and migration number i
34             n holds until clock is 6
35             n sends an interpiconet message to v with the relocation action r
36             n holds until clock is 8
37             RELOCATION (r)
38         else //no scan match made this round
39             n holds until clock is 10
40     else //n is an unshared slave
41         n holds until clock is 1
42         w ← the leader of n
43         if (w returns an interpiconet message with scanner at time 1)
44             n holds until clock is 2
45             n sends an inquiry scan message to m with scan at time 2
46             n holds until clock is 4
47             if (a page message with size is returned at time 4)
48                 u ← the ID of the leader sending the page message
49                 size ← the size of the piconet in which u is leader
50                 n holds until clock is 5
51                 n sends an interpiconet message to w with size and u to w at time 5
52                 n holds until clock is 6
53                 n obtains an interpiconet message from w with relocation action at time 6
54                 r ← the relocation action with participating node IDs
55                 n holds until clock is 7
56                 n sends u a page response message with relocation action r
57                 n holds until clock is 8
58                 RELOCATION (r)
59             else //no scan match made this round
60                 n holds until clock is 8
61         else //n is not a scanner this round
62             n holds until clock is 8

```

RELOCATION (relocation action r)

```

1      if (r is of type trivial)
2          u ← seeking leader extracted from r
3          v ← scanning leader extracted from r
4          TRIVIAL (u, v)
5      else if (r is of type shift)
6          u ← seeking leader extracted from r
7          v ← scanning leader extracted from r
8          SHIFT (u, v)
9      else if (r is of type merge)
10         u ← seeking leader extracted from r
11         v ← scanning slave extracted from r
12         w ← leader of v extracted from r
13         MERGE(u, v, w)
14     else if (r is of type rearrange)
15         u ← seeking leader extracted from r
16         v ← scanning slave extracted from r
17         w ← leader of v extracted from r
18         REARRANGE(u, v, w)
19     else //r is of type migrate
20         u ← seeking leader extracted from r
21         v ← scanning slave extracted from r
22         w ← leader of v extracted from r
23         s ← migration number extracted from r
24         MIGRATE (u, v, w, size)

```

TRIVIAL (node u, node v)

```

1      if (this is node u)
2          u makes v an unshared slave
3          u stays leader
4      else //this is node v
5          v becomes an unshared slave of u
6      hold until clock is 10

```

SHIFT (node u, node v)

```

1      if (this is node u)
2          y ← an unshared slave of u
3          u sends an interpiconet message to y with change master to v at time 8
4          u sends an interpiconet message to v with add unshared slave y at time 8
5          u disconnects v from the piconet
6          u holds until clock is 9
7          u obtains a page message from v at time 9
8          u holds until clock is 10
9          u sends a page response message to v at time 10
10         u becomes a master/slave (master of remaining nodes in piconet, slave of v)
11     else //this is node v
12         v obtains an interpiconet message from u with add unshared slave at time 8
13         y ← unshared slave to add
14         v disconnects from u's piconet
15         v holds until clock is 9
16         v sends page messages to y and u at time 9
17         v holds until clock is 10
18         v obtains page response messages from y and u at time 10
19         v makes y and u slaves
20         v becomes leader

```

MERGE (node u, node v, node w)

```
1      if (this is node u)
2          u obtains an interpiconet message from w with IDs from w's piconet at time 8
3           $N \leftarrow \{\text{IDs from w's piconet} / v\}$ 
4          u holds until clock is 9
5          u sends pages messages to each N at time 9
6          u holds until clock is 10
7          u obtains page response from each N at time 10
8          u makes each N an unshared slave
9          u stays leader
10     else if (this is node v)
11         v disconnects from w
12         v holds until clock is 10
13         v becomes an unshared slave of u
14     else //this is node w
15         w sends an interpiconet message to u with piconet IDs at time 8
16          $S \leftarrow \{\text{slaves from w's piconet} / v\}$ 
17         w sends an interpiconet message to each S with change master to u at time 8
18         w disconnects from all slaves
19         w holds until clock is 9
20         w obtains a page message from u at time 9
21         w holds until clock is 10
22         w sends a page response message to u at time 10
23         w becomes an unshared slave of u
```

REARRANGE (node u, node v, node w)

```
1      if (this is node u)
2          u holds until clock is 9
3          u sends page message to w at time 9
4          u holds until clock is 10
5          u obtains a page response message from w at time 10
6          u makes v and w slaves
7          u stays leader
8     else if (this is node v)
9         v disconnects from w
10        v holds clock until clock is 10
11        v becomes unshared slave of u
12    else //this is node w
13        w disconnects v from piconet
14        w holds until clock is 9
15        w obtains a page message from u at time 9
16        w holds until clock is 10
17        w sends a page response message to u at time 10
18        w becomes a master/slave (master of remaining nodes in piconet, slave of u)
```

MIGRATE (node u, node v, node w, int s)

```
1      if (this is node u)
2          S ← {s slaves, exhausting shared slaves first}
3          u sends an intericonet message to each S with change master to w at time 8
4          u sends an intericonet message to w with ID of each S and add slave at time 8
5          u disconnects from each S
6          u holds until clock is 10
7          u makes v a shared slave
8          u stays leader
9      else if (this is node v)
10         v holds until clock is 10
11         v becomes a shared slave between u and w
12     else //this is node w
13         w obtains intericonet message from u with IDs of slaves and add slave at time 8
14         S ← {IDs of slaves set by u}
15         w holds until time 9
16         w sends a page message to each S at time 9
17         w holds until time 10
18         w obtains a page response from each S at time 10
19         w makes each S a slave
20         w becomes a master
```

OTHERS (node n)

```
1      if (n receives an intericonet message with change master from n's master)
2          n holds until 9
3          n obtains a page message
4          k ← id that sent the page message
5          n holds until clock is 10
6          n sends a page response message to k at time 10
7          n becomes a slave of k
8      else //no master change this round
9          n holds until clock is 10
```

References

- [1] Nathan Muller. Bluetooth Demystified. McGraw Hill. 2001.
- [2] MIT Auto-ID Center. Research Conducted.
[<http://auto-id.mit.edu/research/index.html>]
- [3] Per Johansson, Niklas Johansson, Ulf Korner, Johannes Elg, and Goran Sennarp. "Short Range Radio Based Ad-hoc Networking: Performance and Properties." IEEE 1999. Pages 1414-1420.
- [4] David Farnolari and Prathima Agrawal. "Architecture and Performance of an Embedded IP Bluetooth Personal Area Network." IEEE 2000. Pages 75-79.
- [5] Stefan Zurbes, Wolfgang Stahl, Kirsten Matheus, and Jaap Haartsen. "Radio Network Performance of Bluetooth." IEEE 2000. Pages 1563-1567.
- [6] Lakshmi Ramachandran, Manika Kapoor, Abhinanda Sarkar, and Alok Aggarwal. "Clustering Algorithms for Wireless Ad Hoc Networks." IBM India Research Laboratory. 2000.
- [7] Theodoros Salondis, Pravin Bhagwat, and Leandros Tassiulas. "Proximity Awareness and Fast Connection Establishment in Bluetooth." IEEE 2000. Pages 141-142.
- [8] Ching Law and Kai-Yeung Siu. "A Bluetooth Scatternet Formation Algorithm." Massachusetts Institute of Technology. 2001.
- [9] Brent Miller and Chatschik Bisdikian. Bluetooth Revealed. Prentice Hall. 2001.
- [10] Jennifer Bray and Charles Sturman. Bluetooth: Connect Without Cables. Prentice Hall. 2001.
- [11] Bluetooth Special Interest Group. "The Bluetooth Specification."
[<http://www.bluetooth.com/developer/specification/specification.asp>]
- [12] System Design Group, Seoul National University. A Bluetooth Presentation.
[http://sdgroup.snu.ac.kr/class/multi/project/present/ingu_01.pdf]
- [13] IBM Research. The BlueHoc Simulator.
[<http://oss.software.ibm.com/developerworks/projects/bluehoc>]