ENERGY-EFFICIENT BOUNDED-DIAMETER TREE SCATTERNETS FOR BLUETOOTH NETWORKS

By

JONATHAN THOMAS CAMPBELL

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

MAY 2005

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of JONATHAN THOMAS CAMPBELL find it satisfactory and recommend that it be accepted.

_____
Chair

_____

_____

ACKNOWLEDGEMENT

I would like to express my gratitude toward my advisor, Dr. Murali Medidi. His guidance during my research kept me on track and focused on the final results.

I would like to thank Dr. Sirisha Medidi and Dr. Jabulani Nyathi for serving as commmittee members and evaluating my thesis.

I would like to express gratitude to my family, especially my parents Gary and Katherine Campbell. The drive for excellence which they instilled in me served as a foundation for success throughout my life.

I would also like to thank Holly Hazard, who served as the much needed friend and motivator throughout my graduate work.

ENERGY-EFFICIENT BOUNDED-DIAMETER TREE SCATTERNETS

FOR BLUETOOTH NETWORKS

Abstract

by Jonathan Thomas Campbell, M.S.
Washington State University
May 2005

Chair: Murali Medidi

Bluetooth is a promising wireless technology that enables devices to form short-range multihop wireless ad-hoc networks, or personal area networks.

However, Bluetooth scatternet formation is one of the challenges that must be resolved since the performance of a Bluetooth network depends largely on the scatternet topology used. We first present a height-balanced binary tree, termed ACB-tree for almost-complete-binary tree, that allows two such trees to be combined to create a larger ACB-tree retaining the height-balance requirements. Next, we propose a ATSF, a distributed scatternet formation algorithm for creating ACB-trees. The generated scatternet is shown to minimize the number of piconets and provide a logarithmic-diameter in the multihop interconnection network. We also present simulations, conducted using Blueware simulator, to provide experimental results to study and compare the performance of the resulting scatternets.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER ONE

INTRODUCTION

Embedded processors have been used in consumer electronics for many years and the idea of the network connected refrigerator or coffee maker is not a new concept. Yet, the adoption of these devices has not been widespread. Major stumbling blocks to adoption include high costs of proprietary wireless equipment and the high overhead of current wireless standards such as IEEE 802.11x. The alternative to wireless solutions, most often an Ethernet network interface, requires existing structures to be retrofitted for wired connectivity.

The Bluetooth networking specification, standardized in 1999, was designed to fill the need for short range wireless communication between devices. The specification features connectivity with low power consumption and no line of sight requirements. Other technical features which make Bluetooth stand out among other wireless standards include low cost, low computational overhead and the use of frequency hopping at the physical layer.

Current applications of Bluetooth have focused primarily on cutting unnecessary wires that can clutter a typical desk. While wireless keyboards, mice, and other peripherals have been available for some time, each requires proprietary dongles for connectivity and could possibly interfere with each other. By using Bluetooth, a single receiver can be used for a variety of devices, such as those listed above, as well as printers, scanners or wireless headsets. However, while Bluetooth has been successfully utilized to minimize desk clutter, the potential for much greater integration is still not fulfilled.

The Bluetooth specification allows for the formation of piconets – networks of up to 8 active devices. For connectivity beyond 8 devices, the specification dictates that multiple piconets can be interconnected to form scatternets. Multi-hop scatternets can provide connectivity over a larger distance than a single piconet can cover, and frequency hopping allows for devices to communicate in spite of interference in the popular 2.4GHz ISM frequency band. While piconet formation is

well defined, the Bluetooth specification does not specifically dictate how these scatternets should be formed. Because of this, scatternet formation is an open problem which has become an active area for research in wireless networks.

The scatternet formation problem is more appropriately split into three major areas: topology formation, link scheduling, and routing. While common in other wireless scenarios, each of these problems present unique challenges when discussed in terms of Bluetooth. Due to the type of devices to which Bluetooth is targeted, the networks formed are highly dynamic, ad hoc, and can range from a few nodes to many hundred nodes.

Of the published research into scatternet formation, most of the papers to date have focused on the topology formation aspect. While each algorithm has its own focus and unique characteristics, they can generally be grouped by three primary characteristics. Some algorithms utilize a centralized approach, which can yield tighter control and a more optimal topology over the final scatternet, while others use a more robust decentralized approach. Handling the dynamic nature of a Bluetooth network is important, but some algorithms choose to focus on the case where all nodes participating in the scatternet are present throughout the run of the topology formation algorithm. Finally, algorithms can be differentiated by whether or not they handle some nodes being out of communication range of each other.

Very little research has been presented in the area of energy efficient scatternet formation. Routing methods considering energy efficiency as a goal have been presented [25], showing that the use of power control based on distance and switching of master-slave roles based on battery levels can result in significant gains in network lifetime. However, no link scheduling algorithms have considered energy efficiency and only the SF-DeviL algorithm has been presented in the category of energy efficient topology creation.

In this thesis, we present a new scatternet formation algorithm called Almost-Complete-Binary Tree Scatternet Formation (ATSF). The ATSF algorithm is designed to create an energy efficient multi-hop scatternet in a distributed fashion. The final scatternet is a tree structure, which provides

a natural routing concept and is able to cope with dynamic node arrival and departure.

The ATSF algorithm is primarily focused on creating a connected scatternet with a bounded diameter in a short period of time. Secondary goals include energy efficient selection of bridge nodes and minimizing the number of piconets. ACB-Trees are a variation on the classic binary tree, but with an extra node called the handle connected to the root of the tree. Utilizing this handle, two trees can be merged together into a single tree which has a height one greater than the height of the taller of the two initial trees. The new root of the tree is one of the handle nodes, while the other handle node is promoted to be the handle of the new tree. Energy efficiency is obtained by selecting the handle of the new tree to be the most energy capable node of the two handles, causing the most energy capable nodes to be found at the top of the tree and the least capable nodes as leafs. Dynamic node arrival is handled by allowing leaf nodes which have fewer than the maximum number of slaves to look for new nodes, then adding them as slaves when found. By utilizing the slaves, which are otherwise idle, node departures can be handled by selecting a slave node to replace the node which is departing.

The remainder of this thesis is organized as follows. Chapter 2 presents background information on Bluetooth in the form of a brief overview of the Bluetooth specification. Chapter 3 presents related work, including other scatternet formation algorithms and metrics for analyzing scatternet formation performance. Chapter 4 presents the ATSF algorithm in detail, including the algorithm and its features. Chapter 5 presents the performance evaluation of the ATSF algorithm, including simulation results for ATSF and other algorithms. Chapter 6 consists of the conclusions and a discussion of future work.

CHAPTER TWO

BLUETOOTH TECHNOLOGY

The Bluetooth 1.1 specification [1] establishes a standard for short-range wireless communication in the unlicensed 2.4-2.5GHz ISM (Industrial, Scientific, and Medical) radio frequency band. The main focus for this specification was to produce a method for connecting devices that was low cost, low power, and required low computational overhead. A fast Frequency Hopping Spread Spectrum (FHSS) technique, with 1600 hops per second, provides a reliable communications platform which is robust against interference and fading. Each hop corresponds to a slot of 625μs in length which can be used for either transmission or reception of data. The frequency band is split into 79 channels, and a device hops between the channels in a pseudo random fashion designed to prevent repetitive patterns which could hurt the reliability of Bluetooth. The Gaussian Frequency Shift Keying (GFSK) modulation method is used, allowing for a 1Mbps transfer rate. Bluetooth provides two main types of links - ACL and SCO.

ACL (Asynchronous Connection-Less) create a packet switched communication link between a master and a slave device. One ACL link is allowed between a given pair of devices, and it provides asymmetric throughput of 723.2 kbps upstream and 57.6 downstream, and symmetric throughput of 433.9 kbps.

SCO (Synchronous Connection-Oriented) provide a symmetric point-to-point link between a master and a specified slave. The master sends SCO packets at regular time intervals, but these packets are not retransmitted. SCO is a circuit switched connection, and ideal for time-delay sensitive traffic such as video, voice traffic, etc. Unlike ACL links, which only allow a single link between a master and a slave, SCO slaves can support up to three links from the same master.

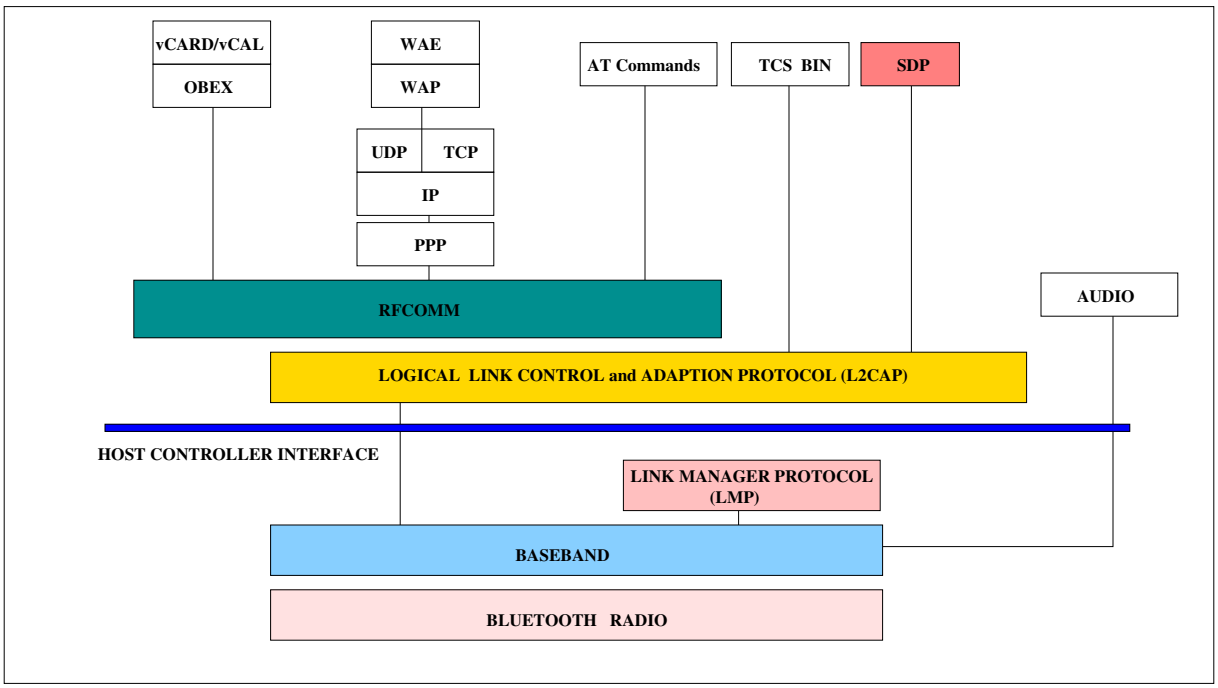


Figure 2.1: Bluetooth Protocol Stack

2.1 Bluetooth Protocol Stack

The Bluetooth specification standardizes the Bluetooth protocol stack, which is shown in Figure 2.1.

2.1.1 Bluetooth Radio

The Bluetooth radio transceiver operates in the 2.4-2.5GHz ISM (Industrial, Scientific, and Medical) frequency band. The specification dictates certain radio characteristics to ensure compatibility between devices, as well as ensuring overall quality of the radio links.

Bluetooth radios are split into three major classes (Table 2.1), most easily differentiated by the maximum power output. Class 1 radios are the most powerful, allowing for 100mW (20dBm) of power and a range of 100 meters, class 2 uses up to 2.5mW (4dBm) with a range of 20 meters, and class 3 is 1mW (0dBm) with a 10 meter range. The transceiver allows for power control via the Receiver Signal Strength Indicator (RSSI), which measures signal strength and determines whether

the power output of the other radio on the link should be increased or decreased. Class 1 devices must incorporate this power control, but it is optional for class 2 and 3 devices.

Table 2.1: Bluetooth Radio Classes

Class	Power	Range
1	100mW(20dBm)	100m
2	2.5mW(4dBm)	20m
3	1mW(0dBm)	10m

The frequency band is split into 79 channels which have a channel spacing of 1MHz, with the top and bottom of the spectrum unused as a buffer against interference. In situations where special algorithms or special frequency regulations are required, only 23 channels are available. The radio uses a Gaussian Frequency Shift Keying (GFSK) modulation method which provides a raw transfer rate of 1Mbps.

2.1.2 Bluetooth Baseband

The Bluetooth Baseband, sitting on top of the radio is the physical layer of the Bluetooth protocol stack. The Baseband manages physical channels and links, outside of the tasks of error correction, data whitening, hop selection, and security. It is implemented as a Link Controller which works with the Link Manager for performing link level routines such as link connection and power control. The Baseband manages both asynchronous and synchronous links, handles packets sent and received, and does the task of inquiry and paging to find and connect to other nearby Bluetooth devices. Each device has a Bluetooth Device Address (BD_ADDR), a 48-bit IEEE MAC address. This address is used as a seed for some operations, such as determining the hop sequence.

2.1.3 Link Manager Protocol

The Link Manager Protocol (LMP), in cooperation with the services provided by the Link Controller, performs link setup, authorization, link configuration, and other protocols. The LMP consists of a number of Protocol Data Units (PDUs) which are sent between devices based on the

Active Member Address (AM_ADDR). The LMP specifies a number of modes which a device can be in at a given time.

- *Active* - a device which is participating in the piconet by listening for packets containing its AM_ADDR. The device's AM_ADDR is unique in a given piconet.
- *Sniff* - A device in *sniff* mode operates in a similar fashion to a device in active mode. When entering into sniff mode, the master and slave decide on a *sniff interval*. The *sniff interval* dictates the period between two time slots where the slave will listen for packets.
- *Hold* - A device enters the *hold* state for a specified *hold time*. During this time, the device cannot support ACL packets, but SCO packets can still be transmitted.
- *Park* - A device in park gives up it's AM_ADDR but remains synchronized with a piconet. It is issued a Parked Member Address (PM_ADDR) which the master can use for un-parking the device. An Access Request Address (AR_ADDR) is also issued, which the slave can use to request that it be un-parked. Un-parking is only allowed at certain points in time which have a constant period, called the beacon interval.

2.1.4 Host Controller Interface

The Host Controller Interface (HCI) provides a standardized command interface to the Baseband controller. The interface provides access to hardware status information, control registers, the Link Manager and the Link Controller. The HCI exists across the host, transport layer, and host controller sections of the Bluetooth protocol stack.

2.1.5 Logical Link Control and Adaptation Protocol

The Logical Link Control and Adaptation Protocol (L2CAP) is layered over the Baseband protocol and resides in the data link layer. It provides both connection-oriented and connectionless data services to upper layer protocols with multiplexing capability, as well as segmentation and

reassembly operation. The L2CAP allows higher level protocols and applications to transmit and receive L2CAP packets up to 64KB in length.

2.1.6 RFCOMM

The RFCOMM protocol is based on a subset of the ETSI standard TS07.10. It provides emulation of serial ports over the L2CAP, as well as transport data stream and control channels over the L2CAP channels.

2.1.7 Service Discovery Protocol

The Service Discovery Protocol (SDP) provides a means for applications to discover available services and their characteristics. The SDP is a necessity for Bluetooth due to the dynamic nature of the network formed by the types of mobile devices to which Bluetooth is targeted. The SDP is designed specifically to meet the requirements of Bluetooth.

The SDP is designed around a client-server model, with each server storing a list of *Service Records*. A service record contains information on the characteristics of a given service. One SDP server instance exists on any device which provides services, regardless of the number of services available. Similar to the SDP server, a single SDP client instance can service queries from multiple applications. Any device may operate both as an SDP client and an SDP server simultaneously.

2.2 Network Structure

Beyond a single device, the most basic network structure in Bluetooth is the *piconet*. A piconet is constituted by one master and 1 to k active slaves, where k is set to 7 in the Bluetooth 1.1 specification. The master of the piconet polls each of its slaves in a Deficit Round Robin fashion, with only one device allowed to transmit during a given slot. Slave devices may not transmit unless they receive a transmission from their master, and can only start transmitting during an odd numbered time slot. In contrast, the master is allowed to start transmitting data in any even numbered slot.

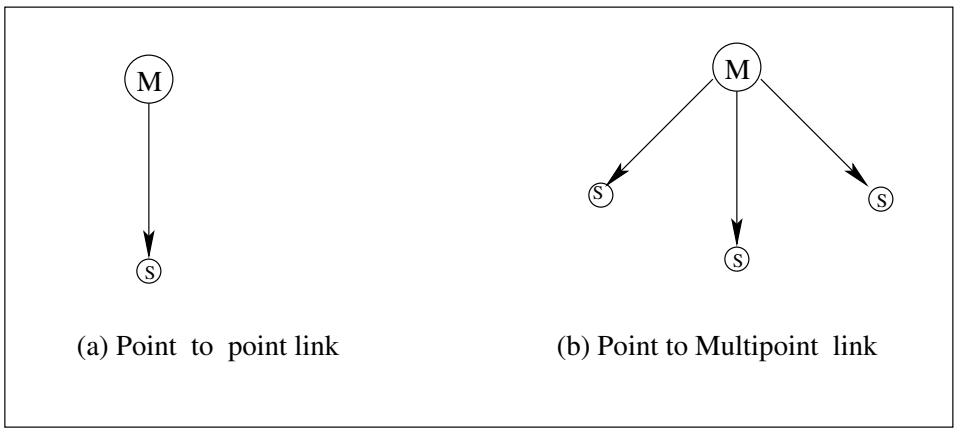


Figure 2.2: Bluetooth Link Types.

2.2.1 Piconet

Each piconet has a unique pseudo random frequency hopping sequence which is set by the master. Slaves calculate the hop sequence based on the master's BD_ADDR and clock information, which is obtained during connection setup. Each active slave in the piconet is assigned an Active Member Address (AM_ADDR), which is unique among the slaves of a given piconet.

2.2.2 Scatternet

Due to the availability of 79 distinct channels and the unique hop sequence of each piconet, multiple piconets can coexist in the same physical area with little or no interference. Multiple piconets can be connected into larger networks called a scatternet. Scatternets are formed when a device is shared between multiple piconets, acting as a bridge node.

Bridge nodes can either be a slave shared between two piconets or a node which is a master of one piconet but slave in another. Bridge nodes can be shared between more than two piconets, but a node can only be the master of one piconet. This limitation exists because the master is the node which dictates the hop sequence of its piconet, and having a node act as a master in two or more piconets would result in those piconets having matching hop sequences. Bridge nodes are able to participate in multiple piconets by performing time division multiplexing.

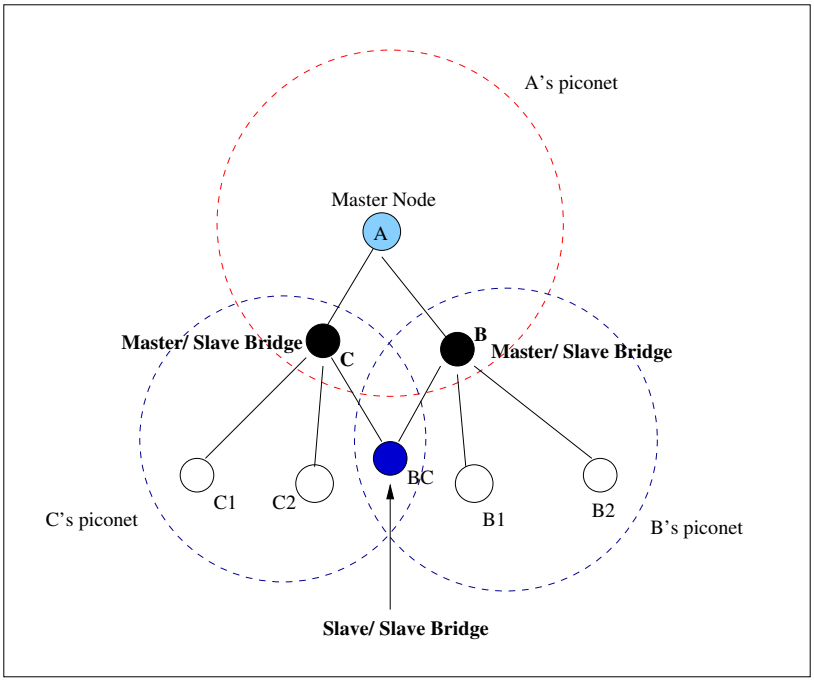


Figure 2.3: Example Bluetooth Scatternet.

2.3 Bluetooth Device States

A Bluetooth device must exist in one of the following states at any given point in time.

- *Standby* - This is the default state and is a low power mode. In this state, the radio transceiver is turned off.

- *Inquiry* - The inquiry mode is used to allow a device to discover other Bluetooth devices which are operating in the corresponding mode: Inquiry Scan. During the inquiry mode the device is continuously transmitting inquiry messages at different hop frequencies. Between inquiry transmissions, the device listens for inquiry responses. Two inquiry messages are sent in each transmission slot, and the device looks for FHS packets in the reception time slot. If a response is received, it is only acknowledged after a predetermined number of responses.

  This process is run until another device is contacted and enough responses are received, or

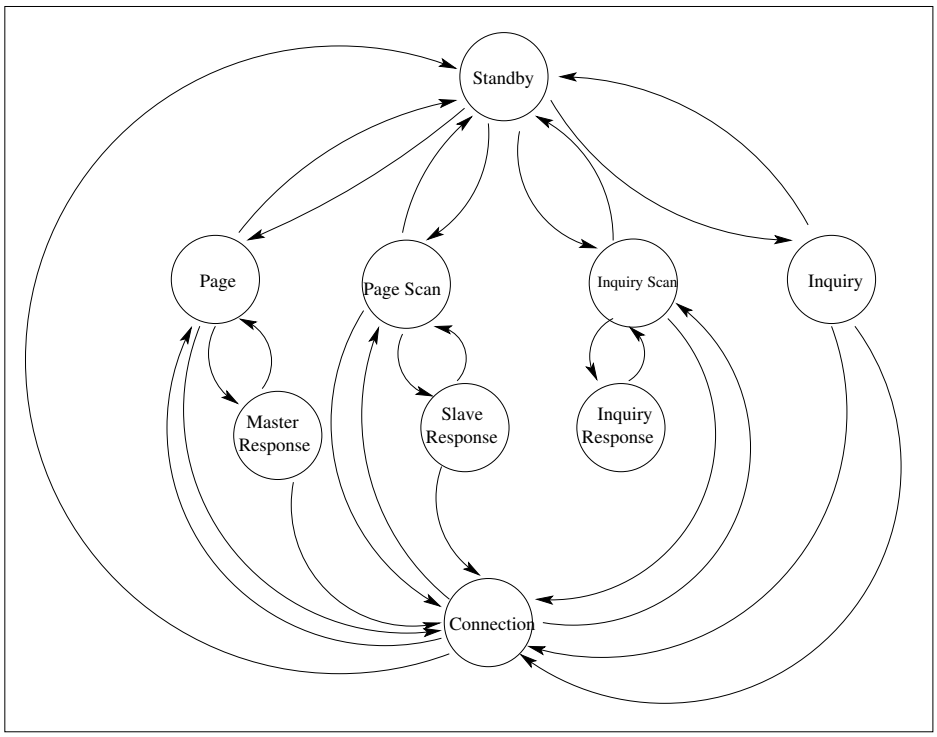


Figure 2.4: Transition diagram of Bluetooth device states. From [13].

until the inquiry time out has elapsed. Inquiry messages contain no information about the source node, but the FHS packets which are received contain essential information necessary for creating a connection with the sending device. During the inquiry process, a *General Inquiry Access Code* (GIAC) can be used to look for any device in communication range, or any one of a number of *Dedicated Inquiry Access Codes* (DIACs) can be used to look for a specific type of device.

- *Inquiry Scan* - Inquiry scan is the corresponding mode to inquiry, in which a device waits to be found by other devices. When the device receives a valid inquiry packet, it enters the *inquiry response* sub-state, where it responds to the inquiry packets with its own FHS packet. This packet contains essential connection setup information, such as the address and clock of the device. Both inquiry and inquiry scan utilize a special frequency hopping sequence to reduce the time required to obtain a frequency match. Though the inquiry frequency hopping

sequence is fast, the inquiry scan sequence is necessarily slow to allow for the devices to obtain synchronization.

- *Page* - The device which will become the master of the connection enters the page state. In this state, the device transmits paging messages to the desired device. Once the recipient acknowledges the packet, the master enters the *master response* sub-state and sends its own FHS packet.

- *Page Scan* - The page scan mode is the corollary to the page mode. In this mode, the device seeks to become the slave of the connection to the corresponding device in the page mode. Once the device receives the page message from the other device, it acknowledges the packet and enters the *slave response* mode. The device waits for the FHS packet from the soon-to-be master, and upon reception it updates its own timing to match the new master before moving to the connection state.

- *Connection* - In the connection state, the slave switches to the master's clock. By doing so, the slave assumes the frequency hopping and timing sequence of the master. After this has occurred, the master and slave can communicate.

2.4 Link Formation

The link formation process for a pair of Bluetooth devices is illustrated in Figure 2.5. Every Bluetooth device initially begins as the master of its own piconet with zero slaves. Larger piconets, and eventually scatternets, are constructed by performing the neighbor discovery process. This process is divided into the two phases of *inquiry* and *page*. The inquiry process is used to exchange information between nodes so that connection establishment can occur. Upon successful completion of the inquiry phase, the devices move to the page phase, where the information gathered during the inquiry phase is used to establish the connection.

During the inquiry phase, the device seeking to become the master is in the inquiry mode, while the device seeking to become the slave is in the inquiry scan mode. The device in inquiry mode rapidly alternates between transmission small identification packets which contain *Inquiry Access Codes* (IACs) and listening for inquiry response messages. Due to the phase difference between the two devices, the Bluetooth specification dictates that the device in inquiry mode hops at a faster rate than the device in inquiry scan mode.

After the device in inquiry scan receives a valid ID packet, it backs off for a random period of time to avoid contention between two listeners who might respond simultaneously. After the backoff time has elapsed, the device in inquiry scan listens for ID packets on the same frequency which it previously detected an ID packet. If a second ID packet is received from the same host the device responds with a *Frequency Hop Synchronization* (FHS) packet containing address and clock information. The address information aids the master in determining the DAC of the slave, as well as the page hopping sequence that the node will use to page the slave. The clock is used to determine the phase of the slave, therefore eliminating the *Frequency Synchronization* (FS) delay during the paging process.

Once the inquiry process is complete, there is an asymmetric knowledge shared between two neighboring devices. The device in inquiry mode knows the address and clock of the node in inquiry scan upon receipt of the FHS packet. However, the node in inquiry scan does not know the identity of the inquiring node. The information available at the end of the inquiry process is the minimal amount of information needed by the node in inquiry mode to estimate the hop sequence used by the slave. By using this information, the node in inquiry mode can invite the other node to join its piconet via the paging procedure.

When the node in inquiry receives the FHS packet from the node in inquiry scan, it moves into the page mode. The node in page mode then sends a DAC packet on the frequency which the other node, now in page scan, is listening. As stated in the Bluetooth specification, the device in page mode is defined to be the *master*, while the device in the page scan mode is called the *slave*.

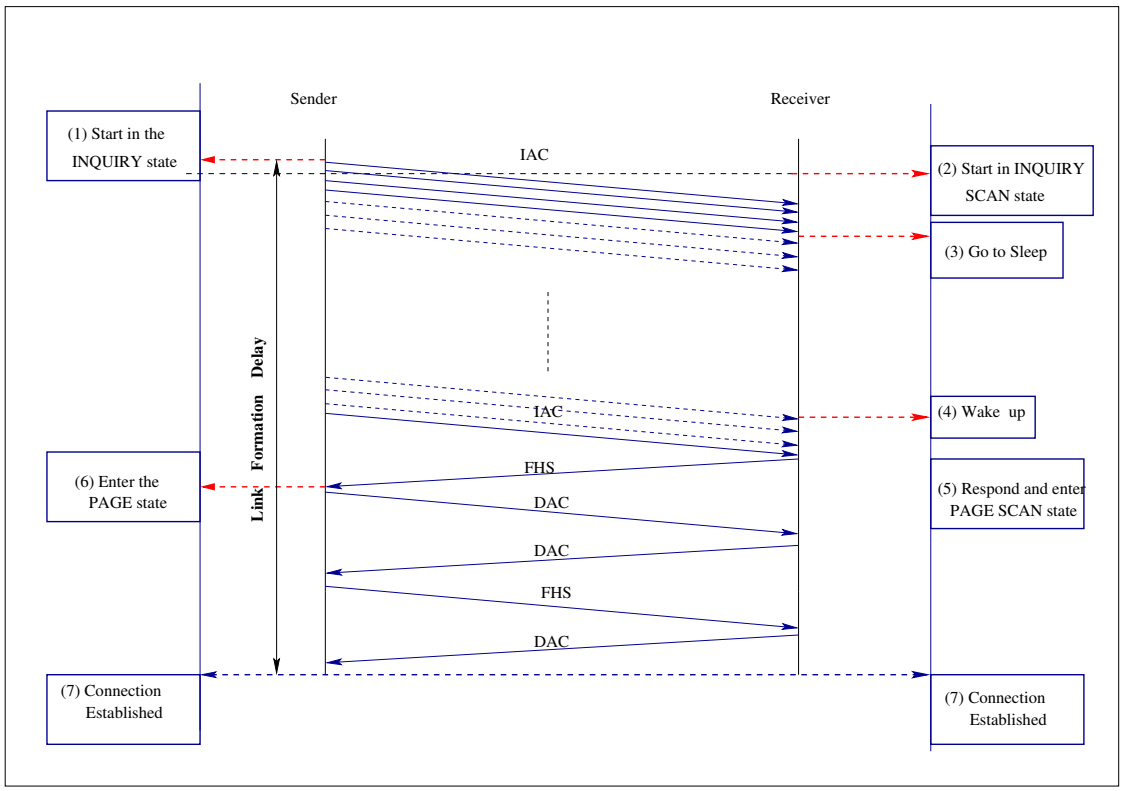


Figure 2.5: Bluetooth Link Formation.

The slave node will respond to the DAC sent by the master with a DAC packet of its own. The master then responds with an FHS packet containing its address and clock. At this point, there is now symmetrical knowledge between the master and slave. The slave then uses the information from the FHS packet sent by the master to synchronize itself with the master's frequency hopping sequence and phase. Once this has been completed, it sends a final DAC packet to the master over the newly established connection, letting the master know that the connection has been fully established.

After a link has been established, a situation could arise where the roles of master and slave need to be reversed. An example would be where node u is the master of a piconet with a number of slaves, with v assuming the role of master of its own piconet and slave to u. If all of the slaves, other than v, in u's piconet disappeared, then it would be more efficient for u to switch to being the

slave of the u-v link, allowing v to serve only the role of master. The benefit for v is that it is no longer required to split its time between being a slave in one piconet and a master in another. This role switch procedure is specifically addressed by the Bluetooth specification via the exchange of a specific LMP packet. The packet exchanged indicates that the devices should switch to the master's frequency hopping sequence.

2.5 Scheduling

Bluetooth scheduling can be separated into two types: intra-piconet scheduling and inter-piconet scheduling. Intra-piconet scheduling is used to coordinate communications between the master and each of its slaves. In order to provide fairness for each of the slaves, the master uses a round robin polling scheme. While the master can start to transmit in any even numbered slot, slaves are only allowed to begin transmitting in a slot immediately following the reception of a packet from the master. Because of this, the master controls which slave is allowed to transmit in any given slot.

Inter-piconet scheduling addresses the problem of time sharing which occurs at bridge nodes, or nodes which are shared among multiple piconets. These bridge nodes must effectively switch between piconets by synchronizing with the master of each piconet for a given time period, called the switching time. In addition to the switching time, the bridge node must also wait for its turn to transmit by waiting for polling from the master. As a result, the total overhead for the switch between piconets is the time spent idle during the switching process.

2.6 Open Problems

While piconet formation and communication are well defined in the Bluetooth 1.1 specification, three major problems remain open in regards to scatternet formation. These problems can be considered to be topology formation, inter-piconet scheduling, and routing.

The Bluetooth 1.1 specification describes the ideas of a scatternet, as well as the basic tools

with which they can be built. However, the specification does not describe the shape of the scatternet which should be produced. As noted in Section 3.3, the network topology can be optimized based on a number of major factors, such as network diameter, number of piconets, etc. Each of these factors can have a result on the scatternet's ability to handle various traffic loads, latency requirements, and healing ability. Because some of the metrics are contradictory in regards to each other, it is only possible to optimize for some of these properties.

Once a scatternet topology has been established, it becomes necessary to determine the inter-piconet scheduling procedure. This allows the bridge nodes, essential to the construction of scatternets, to participate in each of the piconets in which they are members. Because a bridge node must synchronize with each master to which it is a slave, there is an associated overhead for the time spent switching and then waiting to be polled by the master. Fewer switches between piconets yields less overhead due to switching, but results in longer routing delays. Because of this, inter-piconet scheduling is tied to both routing and scatternet topology.

Finally, a routing protocol must be established for the network. While generic algorithms, such as link-state or distance-vector, can perform the basic tasks of routing, they can suffer significant overhead due to the unique aspects of a given scatternet topology and Bluetooth scatternets in general. Additionally, a routing algorithm should be designed to take advantage of topology of the scatternet, and to minimize the negative impact of the overhead introduced due to inter-piconet scheduling.

2.7 Simulation Tools

Due to the complexity of communications networks, simulation tools are often used to analyze the performance of networking related algorithms. In the case of Bluetooth, a number of simulation tools have been developed or utilized for the purpose of performance evaluation. These include, but are not limited to, SimJava[6], BlueHoc [3], Blueware [4], GloMoSim[5], and UCBT [7].

SimJava is a process based discrete event simulation package for Java [6]. Unlike BlueHoc,

Blueware, and UCBT, which are all extensions to the network simulator NS2 [2], SimJava is a more generic discrete time simulation package. Beyond network simulations, SimJava has been used to implement a task farm and a simple memory cache. The simulation framework has been tested in a variety of simulations, but does not have the rigorous pedigree which has been established for NS2 in the area of discrete time network simulation. Among the scatternet formation algorithms implemented using SimJava, the most notable is BTCP [27].

GloMoSim [5], or Global Mobile Information Systems Simulation Library, is an open source scalable simulation environment for wired and wireless networks. Built around an approach similar to the OSI layered network model, GloMoSim uses a standardized API to communicate between various layers. GloMoSim was designed by UCLA, and has been used to implements scatternet formation algorithms such as BTSpin [15].

BlueHoc[3], created by researchers at IBM, is an open source extension to the NS2 discrete time event simulator. BlueHoc implements a basic subset of the Bluetooth 1.0 specification, including the L2CAP, LMP, radio and Baseband. The implementation is able to simulate a piconet containing a master and several slaves, including a ”realistic” packet loss model. Only ACL links are supported, but the connection establishment process closely follows the procedures dictated in the Bluetooth specification. Parking of nodes is also not available in BlueHoc. Basic scatternet formation support is available via BlueScat, which is part of BlueHoc versions 2.0 and higher. BlueHoc/BlueScat has been used to implement a number of algorithms, such as a genetic [29] and a loop [38] scatternet formation algorithm.

Blueware is a Bluetooth simulation module for NS2 created at MIT, and is designed as an extension to the BlueHoc framework. The most significant difference between Blueware and BlueHoc is the addition of an improved infrastructure for building scatternet formation and link scheduling algorithms. The standard Blueware package includes an example scatternet formation scheme, TSF, and an implementation of the LCS link scheduling algorithm. As was the case with BlueHoc, Blueware does not support SCO links or the parking of nodes. Algorithms implemented on

Blueware include TSF [33], BlueMesh & BlueCube [19].

University of Cincinnati Bluetooth Simulator (UCBT) [7] is another module for NS2, and is designed to implement the full Bluetooth stack defined in the Bluetooth 1.1 specification and is partially version 1.2 compliant. Unlike BlueHoc and Blueware, which have not been actively developed since 2001 and 2002 respectively, UCBT is under active development and works well with current versions of NS2. UCBT handles SCO connections as well as ACL connections and parking of nodes. UCBT does not support general scatternet formation out of the box at this time , but the author of the module did publish simulation results [35] using the module. The currently published version provides limited support for scatternets with slave/slave bridge nodes, where every bridge in the network only has two roles. Scatternets can be constructed by manually dictating which nodes should enter page/page scan, bypassing the inquiry/inquiry scan based discovery process.

CHAPTER THREE

RELATED WORK

3.1 General Overview

The Bluetooth 1.1 specification [1], while specifying the procedure for creating piconets, does not include any standard method for creating larger scatternets. As a result, scatternet formation has become an active area for research. Scatternet formation consists of three main components: topology construction, link scheduling and routing. This thesis focuses on the topology construction of a scatternet, the most active research area (based on the number of papers published) in Bluetooth scatternet formation. Link scheduling and routing are very important aspects of scatternet formation; however, the implementation of these aspects can only occur once a topology formation scheme has been created. Though generic methods for link scheduling and routing are available, specialized methods designed for a specific topology can yield better performance and avoid some of the caveats of the generic methods.

Among the various scatternet formation schemes, major groupings can be established based on the characteristics and assumptions of the algorithms. Ghosh *et al.* [15] chose to separate algorithms into categories based on three main characteristics. The first major separation is based on whether or not an algorithm is distributed. A centralized algorithm has the potential to produce the most efficient scatternet based on a number of characteristics but it requires a significant amount of overhead to discover each node in the scenario. An overall leader must be somehow elected before actual formation takes place. Because of these drawbacks, as well as the inherent distributed nature of the ad hoc networks which Bluetooth devices are designed to participate, most recent research has focused on distributed algorithms. These algorithms allow each node to operate independently until larger groupings have been formed. Each grouping makes independent decisions on how to combine with other groupings.

The next major grouping of algorithms is based on whether or not all of the nodes are in communication range of each other. The most general case of scatternet formation must consider the situation where some nodes are not in range of each other. However, due to the lack of link scheduling and routing algorithms, the ability to simulate and implement these methods is somewhat limited. Again, a majority of the research has focused on the formation of scatternets where all devices are in range of each other. These algorithms are particularly interesting because they allow for more focus on controlling characteristics of the scatternet to improve certain metrics, whereas the most important focus for out-of-range algorithms is obtaining a single connected scatternet.

Finally, algorithms are further divided by whether or not they handle dynamic node arrival and departure. The simplest case to analyze is where every node is present at time the algorithm starts and each node remains throughout the entire scenario. However, the more common real world case would incorporate en masse arrival as well as dynamic node arrival and departure.

3.2 Scatternet Formation Algorithms

3.2.1 Formation Algorithms for In-Range Devices

Aggarwal *et al.* [8] propose a scatternet formation protocol utilizing a 'Super master' that knows about all nodes. In a follow-up publication [26], Ramachandran *et al.* redefined the problem of scatternet formation as the problem of clustering. With this novel perspective, they devised a new randomized clustering algorithm specifically for Bluetooth networks which treated each piconet as a separate cluster. As with other scatternet formation algorithms, the clustering algorithm seeks to form a single cluster, which is the same as a connected scatternet. This method can be fairly flexible due to the use of a single node with global knowledge. However, the use of a central node creates a bottleneck and a single point of failure. The basic process of the algorithm is split into two phases: information gathering and role assignment. In the first phase, a 'Super master' is selected and all nodes exchange information. Once the master is chosen, it determines the minimum number of

clusters needed. In the second phase the roles are assigned to each node. However, the scheme presented does not produce a true scatternet. A separate phase of reorganization is required to connect the scatternet together and no indication of how these bridge connections between piconets is provided. As such, the scatternet resulting from the algorithm (as presented) is not connected.

Salonidis *et al.* [27] proposed Bluetooth Topology Construction Protocol (BTCP). BTCP uses three phases for construction of the scatternet. In the first phase, a coordinator is elected which has complete knowledge of all devices in range. Phase two continues by having the coordinator tell all the other masters how the scatternet should be formed. Finally, in phase three, the scatternet is formed according to the instructions. Based on the design of BTCP, a maximum of 36 devices can be connected. As in [8], a single node is used to determine the scatternet topology and notify the devices involved. In addition, BTCP has no means to handle dynamic node arrival or departure.

Tan *et al.* [33] proposed a Tree Scatternet Formation (TSF) protocol which uses a distributed tree algorithm for construction. The protocol keeps loose control of diameter and has fast formation time, but does not minimize piconets. As with all tree protocols, there is a unique path between any two nodes. This algorithm also handles dynamic node arrival and departure via reorganization of the tree structure. The fast formation time is obtained by allowing multiple merges to occur simultaneously. The tradeoff for obtaining such fast formation is the lack of control over the final diameter or the number of piconets in which a node must participate.

Law *et al.* [18] suggest a distributed scheme which guarantees minimum piconets and prove some of the desirable theoretical properties. The algorithm produces piconets with $k - 1$ slaves (except for one) and the degree of every node is either one or two. They also prove that the scatternet will form using a linear number of messages, and will reduce the number of leaders by a constant fraction with a constant probability during each round. This algorithm has no control over network diameter and inherently will result in a scatternet that is similar to a path.

Medidi and Daptardar [13, 19] present a distributed algorithm for formation of mesh and cube structured scatternets, BlueMesh and BlueCube respectively. The algorithm is divided into three

phases: piconet gathering, structure growth and a final cleanup phase. In the first phase, nodes are grouped into small piconets with up to $k - n$ nodes, where n is 3 for meshes and 4 for cubes. Phase one concludes when a piconet gathers $k - n$ slaves, or when a timeout is reached. Phase two consists of mesh or cube building, where equally sized structures are connected to form larger structures. As with phase one, phase two ends with a timeout. In phase three the size restrictions are relaxed to allow a single scatternet to be formed. An 'orphan' list is kept throughout phases two and three, which is used to match up similarly sized structures which have not contacted each other directly. Scatternet diameter is controlled and the resulting scatternet has a natural routing scheme. Due to the tight restrictions on what smaller meshes can combine to make larger meshes, there is a significant number of connections and disconnections associated with failed attempts to merge miss-matched meshes.

Baatz *et al.* [9] proposed a formation algorithm based on *1-factors*. The scatternet is modeled by a directed graph where Bluetooth devices are nodes and edges in the graph are master-slave links. The *1-factors* are chosen in such a way as to attempt a perfect *1-factorization*, but different scatternet topologies can be obtained based on the selection of specific *1-factors*. The authors chose to distribute the roles of master and slave equally because the traffic pattern of the network is unknown at formation time.

Chen *et al.* [11] presented a centralized method for creating scatternets which can have arbitrary topologies. Focusing on what they consider the 'conference' scenario, the algorithm is not designed for a large network. They dictate three parameters which can be controlled in order to form various scatternet topologies: S_{max} – the maximum number of slaves in a piconet, M_{max} – the number of piconets in which a bridge node can participate, L – the number of loops allowed in a scatternet. Additionally, two other constraints always observed are that a scatternet must have a route between any two nodes and the fewest possible number of bridges should be used. For example, a ring scatternet can be formed by using $\{ L = 1, S_{max} = 2, M_{max} = 2 \}$, while a star/tree topology can be created with $\{ L = 0, S_{max} \geq 2, M_{max} \geq 2 \}$. The algorithm was

implemented, with results shown for networks of up to 8 devices. This algorithm is one of the few actually implemented and tested on real devices.

Chong and Chaing [12] presented Bluering, a formation scheme which creates a ring structure. The ring topology provides two unique paths between any two nodes, which increases reliability and provides a natural routing and scheduling methodology. However, the scatternet can become partitioned due to a single node seeking to join the network. In the algorithm, each node starts as a line of length 1. As nodes connect to each other they form lines. Once all lines have joined into a single line the line is formed into a ring by connecting the two end points together. The resulting scatternet has each node acting as a master and slave, with exactly two links.

Zhang, Hou and Sha [38] proposed an algorithm for loop scatternets, comprised of two phases. During phase one, individual nodes are gathered into piconets with up to $k-1$ slaves. The remaining slave slot is saved for use in phase two. This optimization phase is where each piconet selects a slave to share with other piconets. The slave is shared with another piconet which is $\sqrt{P}$ hops away, which reduces the diameter of the network and minimizes node contention.

Zhen, Park and Kim [39] developed a two phase algorithm which first creates 'blue-star islands', then bridges them together in a second phase using route request messages. The use of route request messages is a technique utilized by on-demand routing protocols. The major downside of these messages is that, with a large network, the network can become flooded with the route request messages.

Pamuk and Karasan [21] proposed SF-DeviL, a scatternet formation scheme which uses energy efficiency as an additional criteria during scatternet formation. Each device in the scenario has a device grade, which is derived from the class of device and the battery level. This class is defined by grouping together devices which have similar battery levels. For example, desktop computers and servers are considered to have infinite battery life, laptops have more battery capacity than cellular phones, which in turn have more capacity than Bluetooth headsets. In addition to the device grade, each device assigns a received signal strength grade for each of its neighbors indicating the

received signal as weak, medium, strong or very strong. By using these two values, a 'best master' is selected among a piconet. The scatternet is formed in a basic tree fashion and then reorganized such that the 'best master' among each piconet is promoted to be the master. For example, if a cellular phone is initially the master of a piconet and many or all slaves are laptops, one of the laptops would be selected as the best master to reduce the load on the cellular phone. At the end of the formation process, the final topology consists of a single spanning tree with the most energy capable device as the root and the least capable devices as leaves. The simulations results presented in the paper indicated that this scheme has a high formation delay in comparison to other scatternet formation algorithms. However, none of the other algorithms considered energy efficiency as a metric.

3.2.2 Formation Algorithms for Out-of-Range Devices

Unlike the algorithms presented in the previous section, each of the algorithms presented in this section consider the case where some of the devices are more than one hop away from each other. Because of this, their first and primary goal is to create a connected scatternet by managing appropriate bridge nodes. Only after a connected scatternet can be guaranteed can any other optimizations be attempted.

Petrioli, Basagni and Chlamtac [23] proposed a method for creating a mesh-like star topology. The three phase algorithm begins by performing initial device discovery, where the devices exchange information about their neighbors by creating temporary piconets. A weight is computed locally then used to determine if a node chooses to become a master or a slave. In phase two, the nodes assume their chosen roles when connecting to each other, forming Bluestars. In phase three all Bluestars are connected by selecting gateway nodes between pairs of piconets, with the final result being a BlueConstellations. Because the algorithm does not inherently limit the number of slaves in a given piconet to k, additional overhead can result due to overhead associated with parking and unparking.

Petrioli and Basagni [24] later proposed Bluemesh, a two phase algorithm designed to produce a degree-constrained scatternet. In phase one, each node discovers all of its one-hop and two-hop neighbors. In order to discover the two-hop neighbors, a piconet is established between every pair of one-hop neighbors. The second phase then establishes a scatternet based on the information gathered during the first phase. This algorithm is not expected to scale well, since the number of temporary piconets increases significantly as the number of nodes in the scatternet increases. This is due to the asymmetric nature of the Bluetooth discovery process, which causes the discovery aspect of this algorithm to dominate the formation delay.

Zaruba *et al.* [37] presented two protocols for scatternet formation. The first protocol selects a node, called the blueroot, which builds a spanning tree with itself as the root. In this method, the master-slave relationship between Bluetooth nodes is represented by the parent-child relationship in the tree. The second algorithm, called BGB, is split into two phases. In the first phase, a number of the devices are selected as blueroots, each building a bluetree with themselves as the root. In the second phase each of the bluetrees from phase one is mapped to a node in a virtual graph, which in turn has the algorithm from phase one ran on it. The number of slaves in a bluetree is limited to k and it is claimed that there exist blueroots which are close to the shortest available path between two nodes.

Dong and Wu [14] proposed three variations on Bluetree topologies by utilizing the location of a neighbor and/or the neighbor set of a nodes neighbor. These variations use this additional information to determine which connections should be made. Among the variations, the first uses knowledge of a neighbor's neighbors, the second uses a relative distance computation derived from received signal strength, and the third uses exact position data obtained using GPS coordinates. Simulation results provided in the paper showed that the modifications reduced the piconet count and the average shortest path when compared to the standard BGB algorithm.

Ghosh *et al.* [15] presented BTSpin, a single phase distributed algorithm. The authors define a spin as a period of time spent in Inquiry mode and then Inquiry Scan mode. They further define a

round as the sequence of having the master perform a spin followed by each of its slaves performs a spin. While it is claimed that the algorithm uses a 'mesh-based' approach to connect the piconets into a single scatternet, the final topology is much closer to a tree structure than a true mesh. This algorithm handles the dynamic node arrival and departure by utilizing a strategy for selecting backup gateways during the formation process. One of the main goals of BTSpin was to save energy by eliminating the temporary piconets created by other scatternet formation algorithms. These temporary piconets do not serve a purpose in the final scatternet, but are used in other algorithms to exchange information or occur when two smaller structures cannot merge.

3.3 Metrics

In order to compare various scatternet formation algorithms, we must first determine a set of metrics which are applicable to most or all implementations. Miklos *et al.* [20] presented a variety of metrics tailored to the specific needs of Bluetooth scatternets.

- *Visibility degree of node* i - The number of nodes in radio range of node i. The mean visibility degree is the average visibility degree for all nodes. This figure plays a key factor in the schemes focusing on out of range nodes. If the mean visibility degree is low, then a connected scatternet will be forced to utilize more bridge nodes and has fewer options for redundancy. Conversely, a high average visibility allows for more flexibility in choosing bridge nodes. For scatternet formations schemes where all nodes are in range of each other, this value is equal to the number of nodes in the scenario.

- *Connectivity degree of node* i - The number of Bluetooth links which node i has at a given point in time. The mean connectivity degree is the average connectivity for all nodes. A high degree of connectivity can indicate a dense scatternet, while a low degree of connectivity can indicate a sparse scatternet. Lower connectivity also allows for nodes to spend more time communicating with each node to which it is connected and can have an effect on the amount

of energy needed to service each of those nodes.

- *Number of masters* - The total number of nodes acting as masters in the scatternet, which can be equated to the number of piconets in the scatternet. Minimizing the number of masters has a number of benefits, including an increase in the scatternet density and a reduction in interference. Also, master nodes carry more data in the scatternet and therefore will consume more energy and other resources. Serving as a master of a connection carries a certain amount of overhead beyond the role of being a slave. The other two aspects will be covered below.

- *Piconet size* - The number of nodes, including the master, participating in a given piconet. Maximizing the piconet size will decrease the piconet density by decreasing the number of piconets in the scatternet. The benefits to minimizing the piconet size are the same as minimizing the piconet density.

- *Number of roles per device* - The number of roles which a node serves, which is equal to the number of piconets in which a given node is a member. A node in multiple piconets must split its time between each, increasing the load on that node. However, bridge nodes are inherently involved in two or more piconets. As such, this metric can be a more direct indicator of the number of bridge nodes.

- *Scatternet Formation Delay* - The total time required to create a complete scatternet. Intuitively, this should be kept as low as possible to ensure the fastest topology formation, which allows useful communication to occur in the shortest amount of time.

In addition to those metrics identified in [20], there are a variety of other metrics which have been defined. These metrics are also quite important to any scatternet formation algorithm, but are definitely secondary to many of the metrics mentioned above.

- *Network Diameter* - The minimum number of hops between any pair of nodes in the scatternet. Minimizing the network diameter results in fewer nodes participating in the route, which reduces overall network load. The network diameter has a direct effect on the end-to-end delay between two nodes.

- *Maximum Node Contention [38]* - The number of node pairs which use a given node as a relay for communication. High node contention for a particular node will yield a performance bottleneck and will also result in higher battery consumption due to the increased traffic at the bridge node. Node contention should be minimized in general.

- *Path count* - The number of paths which exist between any pair of nodes. As with any network, multiple paths can increase reliability and decrease load on a given bridge node via load balancing. However, multiple paths require more bridge nodes and therefore more piconets. Also, multiple paths create the need for more complex routing schemes and link scheduling methodologies. Because of these tradeoffs, each scheme must choose whether to maximize or minimize the number of paths.

- *Dynamic Node Arrival and/or Departure* -The Bluetooth specification dictates that a master node leaving a piconet should designate a replacement, generally selected from among its slave nodes. However, not all algorithms address this issue and many also neglect the case of nodes joining a scatternet after the initial topology is established. Handling these dynamic events places a higher overhead on the creation and maintenance of the scatternet topology. Due to the nature of the devices targeted by the Bluetooth specification, the scatternets formed by Bluetooth devices are inherently mobile and ad-hoc. Therefore, any general purpose algorithm should address these dynamic cases.

- *Energy Efficiency* - The Bluetooth specification was specially targeted toward mobile devices which have a limited battery capacity. As such, energy efficiency is very important to the

sustainability of a scatternet. A scatternet should be formed such that the most energy capable devices are assigned roles which incur the highest overhead. In Bluetooth scatternets, the nodes with the highest overhead are bridge nodes and master nodes. While energy efficiency is important, it is always secondary to creating and maintaining a connected scatternet.

Inherently, there will always be engineering trade offs which occur when looking at broadly applicable metrics. Each formation algorithm must carefully select the metrics on which to focus. For example, optimizing for time alone neglects many other metrics.

The ATSF method presented in this thesis can be considered a decentralized dynamic algorithm for situations where all nodes are in-range of each other. As was shown in [38], all tree scatternets have at least one bottleneck node under the uniform traffic model. However, the tree structure provides the minimal number of links necessary to create a connected scatternet. A binary tree also provides a natural routing methodology, controlled diameter and degree and fast formation time. Due to the nature of the Bluetooth discovery process, the merge process of bringing two smaller trees together to create a larger tree is also a natural fit.

CHAPTER FOUR

ACB-TREE FORMATION

4.1 Desirable Scatternet Properties

When deciding on the topology used to form a scatternet it is important to keep performance aspects in mind. In order to analyze and compare performance, a variety of metrics have been defined (discussed in Section 3.3). [17] showed that the ideal topology is application dependant. However, a general purpose algorithm should achieve a reasonable balance between competing metrics while optimizing generally applicable metrics.

Obtaining the minimal scatternet formation delay is always ideal. This time is not necessarily an absolute, but rather the minimal time possible with respect to the other desired traits. Inherently, minimizing the formation delay will allow the scatternet to provide useful communications paths as soon as possible. However, controlling other properties of the scatternet usually comes at the cost of increasing the formation delay.

The time division multiplexing used by bridge nodes introduces undesirable overhead. This overhead creates complexity in link scheduling and routing and increases the end-to-end delay for a set of nodes. Minimizing the number of roles for a node will reduce the associated overhead and improve overall performance. The number of roles which a node performs also is an indicator of the load that node bears for the network. If a node has a high number of roles, it is responsible for connecting more network segments, and must route packets for each of those segments. The diameter of the network also has a direct effect on the end-to-end delay. However, minimizing network diameter generally increases the number of roles that a node performs.

Energy efficiency is a significant issue when forming networks with the small resource constrained devices to which Bluetooth is targeted. A scatternet formation algorithm should ensure that bridge nodes are strategically chosen such that devices which are more energy constrained,

such as cellular phones or personal digital assistants, are not chosen as bridge nodes when more capable nodes, possibly desktop or laptop computers, are available. These choices cannot prevent a connected scatternet from forming and should ensure that the formation delay and other metrics are within acceptable bounds.

If a given scatternet algorithm is intended for use outside of a laboratory environment it must address the issue of node mobility. In terms of scatternet formation, mobility is generally expressed via dynamic node arrival and departure. Once a scatternet topology has been established the network should be robust against failure due to node departures. The scatternet should also be able to accept new nodes into the network quickly while maintaining the general properties of the original scatternet.

Finally, and certainly most importantly, a scatternet topology should ensure that all available nodes are members of the resulting network. Regardless of whatever other properties which can be established, an unconnected network is not useful.

4.2 ACB-Trees

The Almost-Complete-Binary Tree (ACB-Tree) was designed with the goal of creating a simple distributed and recursive algorithm for forming bounded diameter scatternets quickly. The balance tree structure was selected for it's simple structure, logarithmic diameter and natural routing structure. The tree is defined by having the minimum number of links necessary to connect a given set of piconets, which in turn minimizes the number of roles that a node must perform. Trees also offer the benefit of having no fixed limit on the number of nodes which can participate, making them highly scalable.

In addition to meeting the basic goals for the scatternet, ACB-Trees offer a number of additional features that enhance the performance of the resulting network. Rather than making each device a node in the tree, as was the case in TSF, ACB-Trees treat each piconet as a node. This yields a lower piconet count overall and reduces the number of bridge nodes in the network.

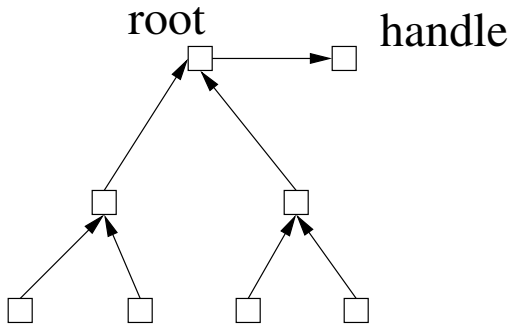


Figure 4.1: An example ACB-Tree.

Connections in Bluetooth networks are formed by connecting two devices together. Therefore, the natural choice for the tree type was a binary tree built using a *recursive doubling* technique illustrated in Figure 4.2. By utilizing this method, the standard device discovery process is used for merging two smaller trees into a larger tree. A binary tree structure also limits the number of roles played by a device. This prevents high fanout, which can unfairly tax certain nodes, by ensuring that all bridge nodes have a fixed number of roles.

Bridge nodes have the number of roles constrained between 1 and 3. In a binary tree roughly half of the nodes are leaves, meaning that half of the piconet masters in ACB-Trees serve only the role of master of their own piconet. The remaining piconet masters participate in three piconets (their own plus their two children) by serving a master/slave/slave role. The choice to use piconets as the nodes of the tree also yields the benefit of limiting the number of nodes which must participate in routing and limits the final diameter of the tree.

While loop, mesh and cube structures provide some reliability gains by providing redundant communications paths, there is inherent overhead associated with the setup and maintenance of those connections. Beyond the extra time required to establish these redundant connections, there is also energy and time overhead spent due to the time division multiplexing used by the bridge nodes to participate in multiple piconets. For example, while only half of ACB-Tree piconet master nodes must split their time between multiple piconets, nearly all of the piconet master nodes in BlueMesh and BlueCube must participate in multiple piconets. The lack of loops and redundant

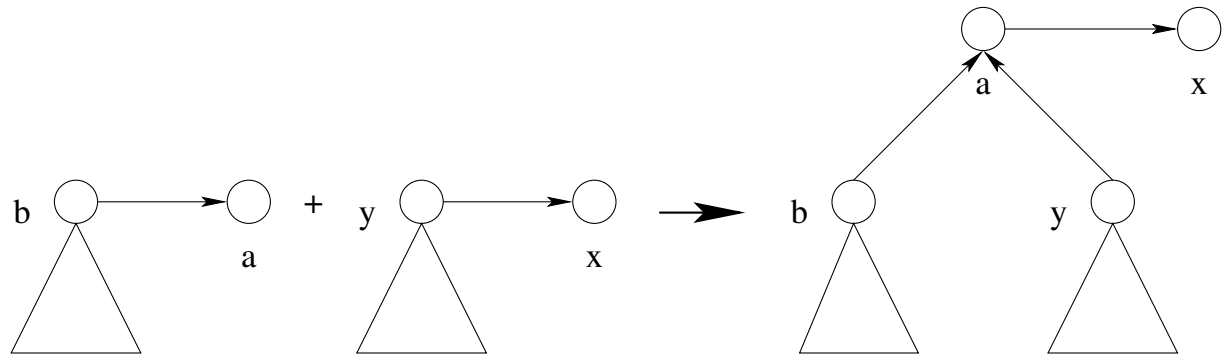


Figure 4.2: Two trees combined via recursive doubling.

connections also simplifies routing due to the existence of a unique path between any two nodes. While redundancy can add reliability, the unshared slave nodes of the piconets can be used to fill voids left when a node leaves the scatternet, resulting in only a temporary service interruption.

Under uniform traffic models, it has been shown [38] that nodes which are located closer to the root node of a tree will carry a higher burden than those located further from the root node. This yields a natural method for obtaining an energy efficient scatternet, where nodes which are more energy capable are placed further up the tree (closer to the root) and the least energy capable nodes are made to be leaf nodes.

4.3 Formation Algorithm

The ATSF algorithm is divided into three conceptual phases. In phase one, devices discover each other and form into piconets of up to $k - 1$ nodes (6 nodes using Bluetooth 1.1 [1]). In phase two, piconets combine to form ACB-Trees, and these smaller ACB-Trees are combined with similarly sized ACB-Trees to form larger ACB-Trees. At the end of phase two, a forest of ACB-Trees exists. In phase three, this forest is merged into a single tree by loosening the height similarity restrictions.

4.3.1 Phase One: Piconet Merging

Purpose

The goal of phase one is to gather independent devices into piconets. Each node begins phase one as an independent master of its own piconet. Each device then performs the neighboring device discovery process described in the Bluetooth 1.1 specification. When devices discover each other, they attempt to merge their piconets together to create larger piconets.

Implementation

The algorithm starts with a collection of devices which are all in communication range of each other. Each device begins as a piconet of size 1. The merge process for these single node piconets is identical to the merge process for larger piconets.

To allow the masters of two piconets to discover each other, each master calls the SEEK procedure with the probability p or SCAN with the probability $1 - p$. During the SEEK procedure, the master enters the inquiry state, looking for other piconets with which to merge. In the SCAN procedure, the master enters the inquiry scan state, waiting to be found by another piconet master. Once two masters discover each other, they enter the corresponding PAGE and PAGE SCAN modes to form a connection.

Following connection establishment, the *MergePiconets(node u, node v)* procedure is called. Node u is the master node which called the SEEK procedure, while node v is the master node which called the SCAN procedure. MergePiconets calculates the total number of slaves which would exist in the new piconet if the proposed merger occurred. This summation can be described as the slaves in u's piconet plus the slaves in v's piconet, plus one more to account for v becoming a slave to u. If this summation is found to be 6 or less, then the merge takes place via v transferring all of its slaves to u. After the slaves have been transferred, v retires as a piconet master and assumes the role of slave to u. If the sum is greater than or equal to 6, then the masters disconnect and continue the discovery process looking for other piconets with which to merge. Slaves in

piconets do not serve any role throughout any of the three phases and can be thought of as being set to standby until scatternet formation is complete.

Termination

Piconets which acquire the maximum 6 slave nodes move to phase two immediately. However, the situation can arise where all the piconets are of such a size that no merges can take place. Additionally, even if such merging could take place given enough time, it is desirable to place an upper bound on phase one to ensure the overall speed of the algorithm. Because of this, a phase one timeout is specified and any piconet with less than 6 slaves which reaches this timeout in phase one will immediately transition to phase two.

4.3.2 Phase Two: Tree Building

Purpose

The goal of phase two is to create ACB-Trees from the individual piconets. Each piconet which enters into phase two consists of a master and up to 6 slaves. The empty slave slot is used to connect to other trees with the goal of merging two trees into a new larger tree.

Implementation

There are two main types of trees in phase two: *Lone Piconets* and *ACB-Trees*. A lone piconet is a single piconet, consisting of a master and its slaves, which has not yet merged with any other piconets. ACB-Trees are combinations of two or more piconets and have the basic ACB-Tree shape shown in Figure 4.1. These two trees operate differently as a result of their differing capabilities. Five types of nodes exist in phase two: Root, Handle, Lone Piconet, Internal, and Slave. As noted earlier, slave nodes do not perform any actions during scatternet formation. During phase two and three, Internal nodes do not perform any actions either.

Lone Piconets operate in a very similar fashion to phase one, randomly selecting to perform SEEK or SCAN based on probabilities p and $1 - p$, respectively. When a Lone Piconet connects to another tree, it calls the *MergeComponents(node u, node v)* procedure.

ACB-Trees, unlike Lone Piconets, consist of two or more piconets. Utilizing this property, we create a division of labor by introducing the concept of the Root node and the Handle node. The Root node always calls the SEEK procedure, while the Handle always calls the SCAN procedure. Effectively, the Root node of one tree searches for the Handle of another tree. Once the Root is connected to another tree, it calls the *MergeComponents(node u, node v)* procedure.

Procedure *MergeComponents(node u, node v)*

This procedure does the majority of the work required by phase two. This procedure is supplied with nodes u and v, where u is master of the u-v link. MergeComponent can be called by both Lone Piconets and by Root nodes, where u is either a Lone Piconet or a Root and v is either a HANDLE or a Lone Piconet. As a result of these varying combinations, there are four cases which exist in this procedure.

General Case: Node u is a Root, node v is a Handle.

This most general case handles the situation of two trees merging, which is implemented in the form of *recursive doubling*. As with cases 2 and 3, the initial step is to analyze the height of each tree. In this case, a merge is allowed if $abs(height_u - height_v) \leq d$, where d is a parameter to the algorithm specifying the allowed merge difference. Setting d to 0 results in a complete balanced binary tree, while setting it to an integer greater than zero will allow the tree to become more unbalanced (greater values of d allow greater imbalance). If d is set to infinity, then any merger of two trees will be allowed. The tradeoffs regarding the setting of d will be discussed in the performance evaluation, though it can generally be stated that tighter control over the balance will require more time to produce an ACB-Tree which spans all available nodes.

The merge process is illustrated in Figure 4.3.

1. Node u, the root node, discovers node v, a lone piconet.

2. Node u asks its Handle (node x) to enter PAGE SCAN.

3. Node v enters PAGE mode, seeking to connect with node x.

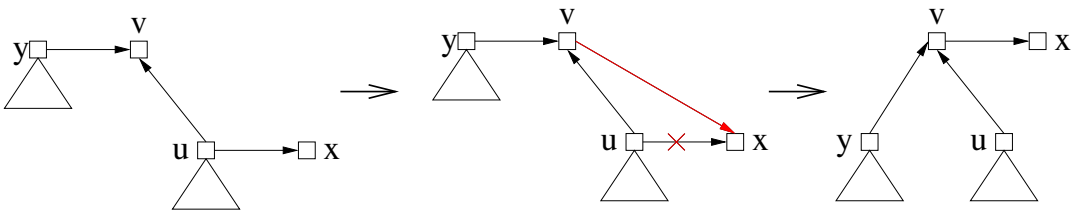


Figure 4.3: General Case of MergeComponents.

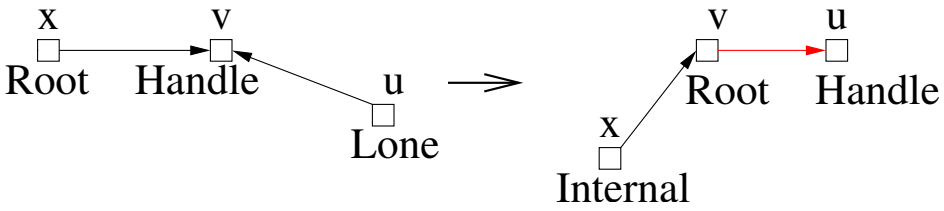


Figure 4.4: Case 2 of MergeComponents.

4. Once node v connects with node x, node u disconnects the u-x link.

5. Node v's Root and node u change their modes to become Internal nodes. Node v changes its role from Handle to Root.

6. Nodes v and x update their height records to indicate the new tree height.

Case 1: Node u is a Lone Piconet, node v is a Lone Piconet.

This is the most basic case, which occurs when both u and v are Lone Piconets. In this case, u changes its role to be a Root node and v changes to become the Handle. The result is an ACB-Tree with height 1, where there is a Root and Handle, but no Internal nodes.

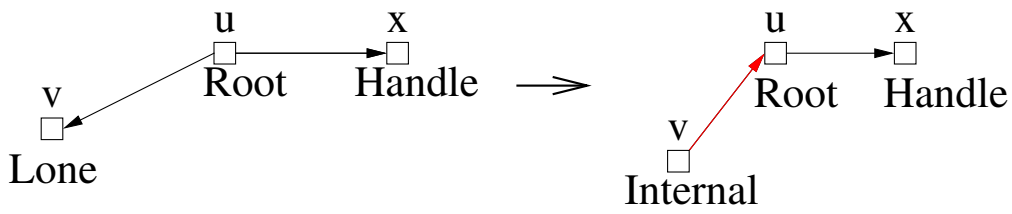


Figure 4.5: Case 3 of MergeComponents.

Case 2: Node u is a Lone Piconet, node v is a Handle.

In this case, a Lone Piconet has contacted a tree. The first step in this case is to check the height of v's tree. If the height of v's tree is equal to 1, then a merge is allowed to occur. The link between u and v is reversed, making u the slave to v. Next, v's Root changes to become an Internal node, while v changes to become the new Root. Finally, u changes to become the Handle, which creates an ACB-Tree of height 2. If the merge is not allowed to proceed, then the u-v link is broken and each of the nodes continues with their previous tasks.

Case 3: Node u is a Root, node v is a Lone Piconet.

In this case, a tree has connected to a Lone Piconet. In a similar fashion to case 2, the first step is to determine if u's tree is height 1 and if not then the merge is not allowed. While this case could be handled by the general merge case, an optimization can be made since the order of the piconet nodes in the tree does not matter. The general case requires an additional link to be established, which carries with it the overhead of the paging process. Instead, the u-v link can simply be reversed, with v changing its role to be an Internal node. The result is an ACB-Tree of height 2, which is equivalent to the tree which results from case 2. If the merge is not allowed to proceed, then the u-v link is broken and each of the nodes continues with their previous tasks.

The final result of this process is an ACB-Tree with a height of $max(height_u, height_v) + 1$. Should node v fail when paging node x, it will disconnect the u-v link. In this case, as with the case where merging was not allowed, each of the nodes continues with their previous tasks.

Termination

Due to the distributed nature of the algorithm, a situation can develop where no merges can occur due to height mismatches between trees. An example of such a situation would be a forest of trees with heights 7, 5, 3, and 1 when d is set to be 1. In this case, no amount of additional merge attempts could combine these trees based on the current value of d. As a result, we impose a timeout on phase two, with trees moving to phase 3 once this timeout has expired. Setting this value too low could result in an excess number of trees entering into phase 3, even though valid

merges exist, while setting it too high would result in an excess amount of time attempting to merge trees which cannot be merged.

4.3.3 Phase Three: Final Merging

Purpose

In this final phase, the goal is to combine the forest of trees which remain after phase two has completed. While balance is a desirable trait to bound the diameter, it is more important to obtain a network in which all devices are connected.

Implementation

The process for phase three is essentially identical to that of phase two except for two main differences. First, the allowed merge difference condition is relaxed. This allows trees of differing heights to merge into a single tree, at the cost of allowing the tree to become unbalanced. Using the example noted from phase 2, this modification allows the trees of heights 7, 5, and 3 to merge.

However, the condition can still exist where only a Lone Piconet and a tree exist. This condition cannot be resolved by simply modifying the allowed merge difference. Because of this, a special procedure called *SwallowLonePiconet(node u, node v)* is used. The procedure can be called by either a Root node after connecting to a Lone Piconet or by a Lone Piconet after connecting to a Handle.

Procedure *SwallowLonePiconet(node u, node v)*

The idea between SwalloLonePiconet is that when d is set to a non-zero integer, 'holes' will be left in the tree. These holes can be defined as spaces in the tree where a piconet could be added without changing the overall height of the tree and any tree which has allowed a merge between trees of differing height will have at least one hole. The holes can easily be identified by any Internal node which has a height not equal to 1 has one child or less. In the case where there are no holes in the tree, the Lone Piconet can be added as an Internal node below a leaf node. The overall height of the tree will be increased by one in this case.

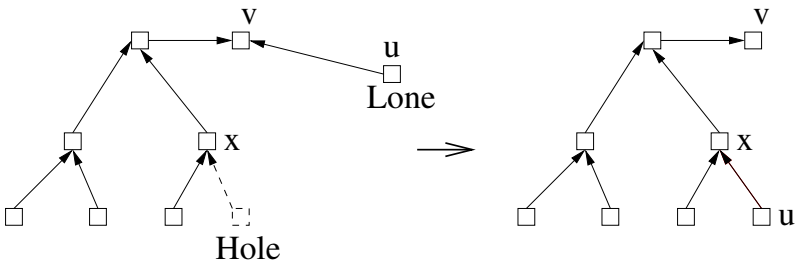


Figure 4.6: Tree swallowing a Lone Piconet.

The Root node begins by running a search of its left and right sub-trees to find the hole which is highest in the tree. Though any hole could be used, filling the highest hole will improve the balance of the tree overall. If no hole is found, then an arbitrary leaf node is chosen to accept the Lone Piconet.

A special case occurs when the Root node of the tree contacts the Lone Piconet and the highest hole is adjacent to the Root node. In this case, the Root node reverses the link between it and the Lone Piconet, such that the Lone Piconet becomes the master of the connection. After the link has been reversed, the Lone Piconet changes its mode to become an Internal node.

In all other cases, the process of incorporating the Lone Piconet into the tree occurs as follows.

1. The parent of the hole, node x in Figure 4.6, is put into PAGE SCAN mode.

2. The Lone Piconet goes into PAGE mode attempting to connect to node x.

3. After the link has been established, the Lone Piconet changes its mode to become an Internal node.

4. If the Lone Piconet was swallowed by a leaf node, then the height of the tree is recalculated.

Termination

Phase three terminates when all nodes have been brought together into a single ACB-Tree.

4.4 Energy Efficiency

It has been shown [38] that under a uniform traffic model, nodes which are closer to the Root node of a tree will have a higher load overall. As a result, it is natural to construct the tree such that the most energy capable nodes serve as bridges and are located higher in the tree. These bridge nodes carry a higher load due to the handling of routing of nodes, deal with the overhead of switching between multiple piconets and handling their own traffic. The least energy capable nodes should be assigned the role of slave nodes, where they are only required to handle their own traffic.

The basic idea behind our implementation is the use of a knock-out tournament. In short, during phases one and two merges are conducted such that the most energy capable nodes 'win' the role of leading the resulting tree. Each device stores an energy capability value, which is used to reflect available battery power and the type of device. The following is a description of the modifications to the standard algorithm to implement energy efficient ACB-Tree construction.

4.4.1 Phase One: Piconet Building

The standard MergePiconets procedure is used, with one modification. Each time a merge occurs, u and v compare their available energy values. If v is found to be more energy capable than u, then v is chosen to become the new piconet's master. Otherwise, u is set as the master. The remainder of the MergePiconets procedure remains unchanged. As a result of this, every piconet ends phase one with the master as the most energy capable node in its piconet.

4.4.2 Phase Two: Tree Building

As with phase one, the same basic procedure is used for phase two, with the addition of the energy efficiency checking. The same cases exist, but some additional sub-cases are required due to the possible ordering of nodes. At the end of phase two, there will be a forest of ACB-Trees, where each tree has the most energy capable node as the Handle and less capable nodes will be located further from the Root node.

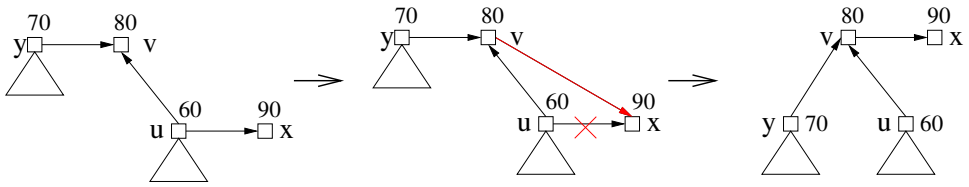


Figure 4.7: Energy Efficiency - General case of MergeComponents. Existing link is usable.

General Case: Node u is a Root, node v is a Handle.

The general merge case is illustrated in Figures 4.7 and 4.8. The energy levels of node u's Handle (node x) and node v are compared. The node with the higher energy capability is selected to be the Handle, while the other node is selected as the Root. If x has a greater or equal energy capability than v, the merge process occurs identically to the general case which was described earlier. In the case where v has a greater energy capability than x, the existing u-v link must be disconnected. Before this occurs, v asks its Root (node y) to go into PAGE looking for node x, while node u asks node x to go into PAGE SCAN.

If y is successful in paging node x, then the merge process described in the non-energy efficient general case is utilized. Should paging fail, each tree goes back to their respective tasks of searching for other trees.

The result of either sub-case is the same: an ACB-Tree which combines the two smaller trees. The Handle of the new ACB-Tree is the most energy capable of the Handles of the two smaller trees, while the Root node is the least energy capable.

Case 1: Node u is a Lone Piconet, node v is a Lone Piconet.

The energy levels of node u and node v are compared. The node with the higher energy capability is selected to be the Handle, while the other node is selected as the Root. If u ends up as the Handle, then the u-v link is reversed to ensure that all links in the tree point upward.

Case 2: Node u is a Lone Piconet, node v is a Handle.

This case can be split into three sub-cases, characterized by the relative energy capability of each device. Note that node y is considered to be v's Root node.

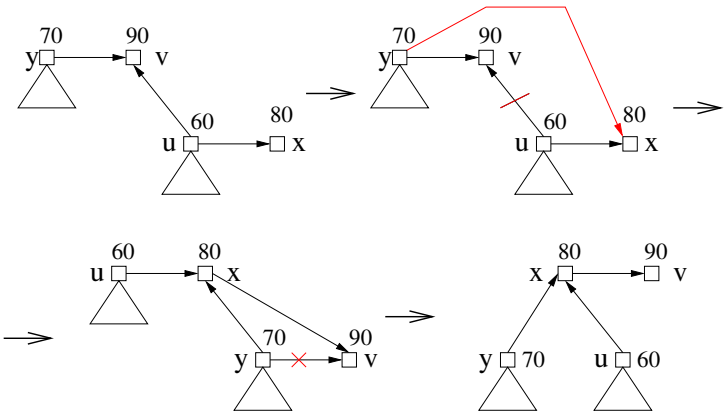


Figure 4.8: Energy Efficiency - General case of MergeComponents. Existing link is not usable.

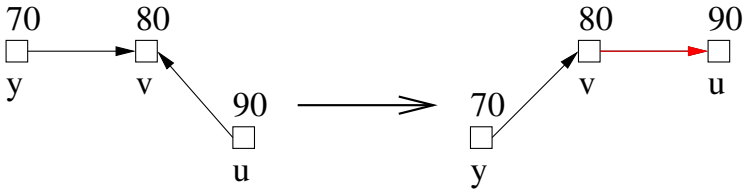


Figure 4.9: Energy Efficiency - Case 2a of MergeComponents.

Case 2a: Energy Capability of $u \geq v \geq y$

In this case, shown in Figure 4.9, the u-v link can simply be reversed. Node u is assigned as the Handle, node v is assigned the role of Root and node y retires to the role of Internal node.

Case 2b: Energy Capability of $y \geq v > u$

Figure 4.10 illustrates case 2b. After the final ACB-Tree has been constructed, y must be the master of u which must be the master of v. However, no y-u connection exists. Node v will ask node y to enter PAGE looking for node u and ask node u to go into PAGE SCAN. Once the connection is made, the y-v link is broken. Finally, node u changes from a lone piconet to be the Root and node y changes to become an Internal node.

Case 2c: Energy Capability of $v > y > u$

In the final ACB-Tree, node u should be the master of node y, which in turn should be the master of v. As with case 2b, no link between y and u exists in case 2c (Figure 4.11). Node v will ask node u to enter PAGE looking for node y and ask node y to go into PAGE SCAN. Once the

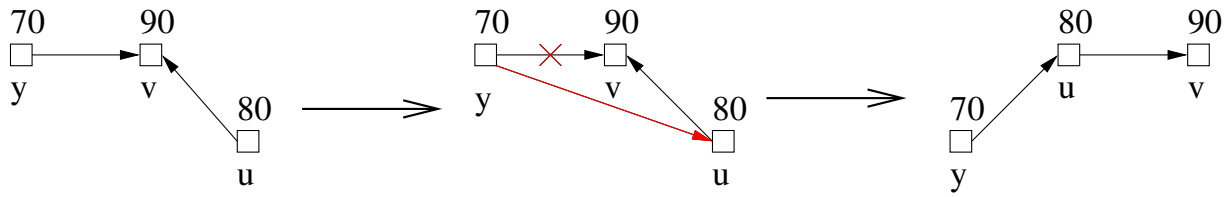


Figure 4.10: Energy Efficiency - Case 2b of MergeComponents.

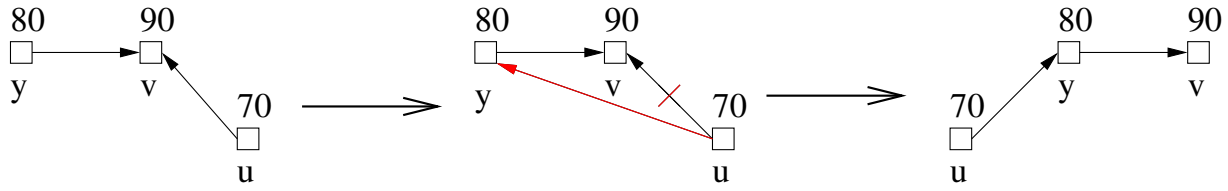


Figure 4.11: Energy Efficiency - Case 2c of MergeComponents.

connection is made, the u-v link is broken. Finally, node u changes from a Lone Piconet to be an Internal node. Nodes v and y maintain their respective roles.

Case 3: Node u is a Root, node v is a Lone Piconet.

Similarly to case 2, this case can be split into three sub-cases, characterized by the relative energy capability of each device. Due to the similarities between cases 2 and 3, the specifics of case 3 are skipped.

4.4.3 Phase Three: Final Merging

The most important goal of phase three is to obtain a connected scatternet. As was the case in phase three of the standard algorithm, the height restrictions are relaxed and lone piconets are swallowed by ACB-Trees. The energy efficient *MergeComponents* procedure used in phase two is used in phase three, while the *SwallowLonePiconet* remains unchanged from the standard algorithm. This provides a good balance between obtaining a connected scatternet in a short period of time while using a best effort technique to unify all remaining trees.

At the end of phase three, we will have a single ACB-Tree with the property that, in general, the most energy capable nodes will serve as bridge nodes closer to the Root and Handle. Those nodes which are less energy capable will be located closer to the leaf nodes or serve as slaves.

4.5 Dynamic Environments

The ATSF algorithm is designed for the situation where all nodes are within communication range of each other. This assumption is useful for analyzing static networks and determining key properties of the algorithm but neglects the likely occurence of nodes leaving the network or attempting to join once the topology has been established. An algorithm must address the case of dynamic node joins and leaves in order to be truly general purpose. To consider the case of dynamic environments, we analyze the situation where an already formed scatternet is subjected to either new nodes attempting to enter or existing nodes attempting to leave.

To handle node joins and leaves in the network, a new fourth phase is created. As soon as a single ACB-Tree is created in phase three, the tree transitions directly into phase four. Any nodes which arrive after the tree has been established will be placed into phase four. The goal of phase four is to maintain or establish connectivity with all available nodes.

4.5.1 Dynamic Node Arrival

In the ATSF algorithm described above, phase one creates piconet with a master and up to 6 slaves. Though the best case scenario would be to have every piconet full, which yields the lowest piconet count possible, it is likely that many piconets will have less than the maximum 6 slaves allowed. It is also likely, given proper selection of the phase one timeout, that a majority of the piconets will have at least one slave. Though this would seem to be inefficient at first glance, it actually provides an opportunity to cope with dynamic node arrival in a graceful manner.

In a binary tree, roughly half of the nodes will be leaf nodes. This is an important property to consider when combined with knowledge that some piconets will have less than the maximum number of slaves. As a result, on average half of the nodes with unfilled slave slots will end up as leaf nodes. The leaf nodes are ideal locations to accept new node arrivals, because they have the lowest communication burden among the bridge nodes in the network.

In phase four, each of the leaf nodes will select an unused slave (one that is not serving as a

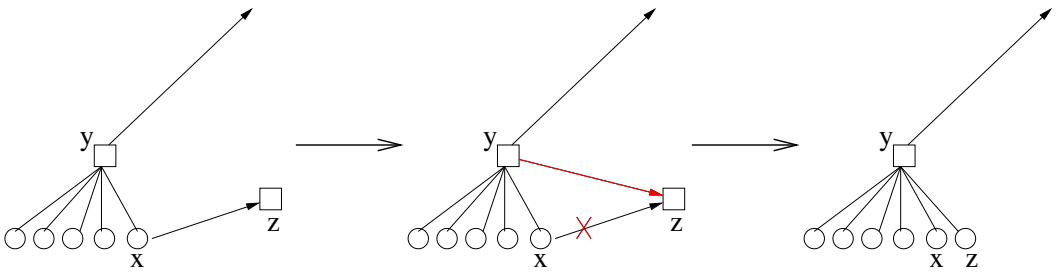


Figure 4.12: Dynamic Node Arrival - Case 2.

bridge) to search for individual devices seeking to enter the scatternet. The devices which are not yet part of the scatternet will call the SCAN function while the slaves in the scatternet will call SEEK. The slave chosen will only spend 50% of its time in SEEK, allocating the remainder of its time for its own communications. SCAN takes a longer amount of time than seek, so the device seeking to enter the scatternet will be burdened with the task. The device will spend all of its time in SCAN mode, because its sole purpose is to become a member of the scatternet. In the event that there are no unshared slaves available, the master of the piconet will spend 50% of its time in SEEK mode looking for devices desiring entry into the scatternet.

Case 1: Master finds lone device.

When the master node finds and connects to a lone device, then the device changes its mode to slave. The slave then is selected as the device which performs SCAN looking for other devices needing to be incorporated into the scatternet.

Case 2: Slave finds lone device, unused slots available.

The slave (node x in Figure 4.12) will ask its master (node y) to go into PAGE looking for the lone device, and tells the lone device (node z) to go into PAGE SCAN. Once the connection between the master and lone device is established, the lone device changes to become a slave. The x-z link is then broken, and node x continues its task of searching for lone devices.

There are also two special cases which exist:

Special Case 1: Internal node which only has one child.

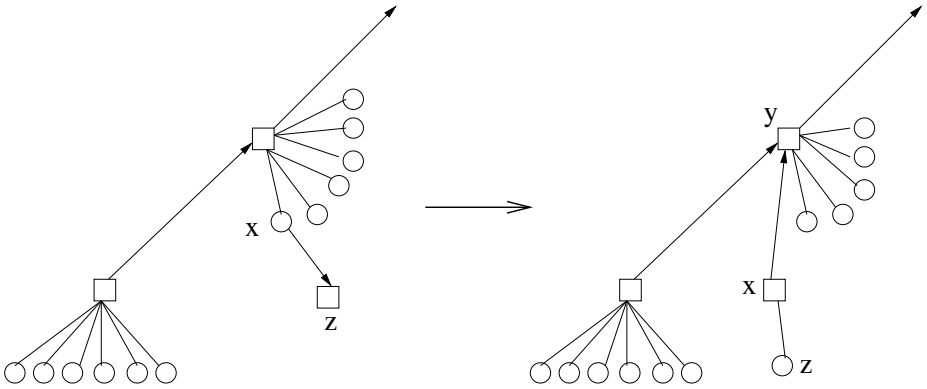


Figure 4.13: Dynamic Node Arrival - Special Case 1.

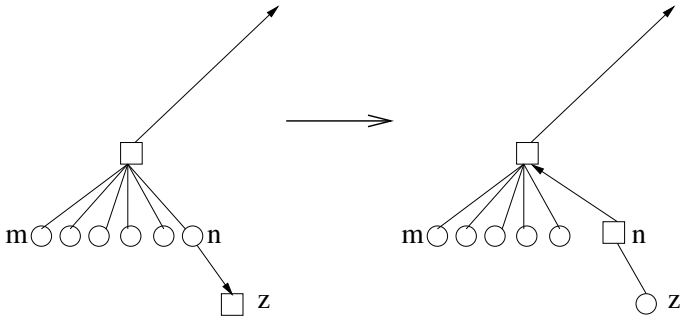


Figure 4.14: Dynamic Node Arrival - Special Case 2

The purpose of including this case is to improve balance in the tree by allowing Internal nodes with only one child to acquire a second child node. If an unused slave slot is available, the situation is identical to case 2. If all unused slots have been filled, then the slave (node x in Figure 4.13) which found the lone device (node z) will convert from being an unshared slave to become a Leaf node. It will then reverse the link to its master, ensuring that all bridge links consistently point towards the Root of the tree. Node z will change to be a slave to x.

Special Case 2: Leaf node with no unused slave slots.

Two unshared slaves (m and n in Figure 4.14) are selected to look for lone devices. When either node connects to a lone device, it will convert to an Leaf node then reverse the link to its master.

4.5.2 Dynamic Node Departure

When handling node departures, the goal is to maintain the ACB-Tree structure as best as possible with as little communications disruption as possible. The Bluetooth specification dictates that when a node leaves a network, it must select a replacement for itself. The trivial case is where an unshared slave is leaving the network. In this situation the node can simply leave the network without selecting a replacement because it is responsible only for its own communications. However, three more complex cases exist.

Case 1: Leaf node with no slaves.

This is the most trivial case for a non-slave node. The node has no slaves, so it is only responsible for its own communications. The node can leave the network without selection of a replacement.

Case 2: Internal, Root or Handle with spare slaves.

The node (x in Figure 4.15) has unshared slaves, which can be used to fill the void left by the departing bridge node. An unshared slave (node y) is selected as a master by the departing node and then all connections to and from node x are re-established with node y.

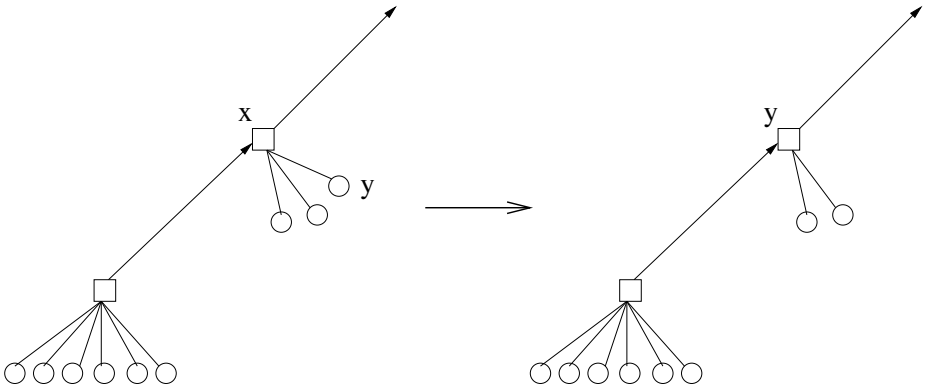


Figure 4.15: Dynamic Node Departure - Case 2.

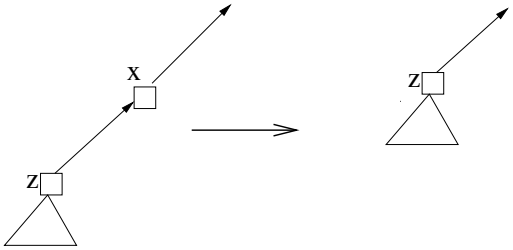


Figure 4.16: Dynamic Node Departure - Case 3. Only one child available.

Case 3: Internal, Root or Handle without spare slaves.

In the case where there the departing node has no unshared slaves, the most straight forward method is to select a child node as a replacement. In the case where there is only one child as in Figure 4.16, that child (node y) is pulled up as a replacement.

Case 4: Internal, Root or Handle without spare slaves.

The node (x in Figure 4.17) has no unshared slaves. As a result, it will be forced to borrow

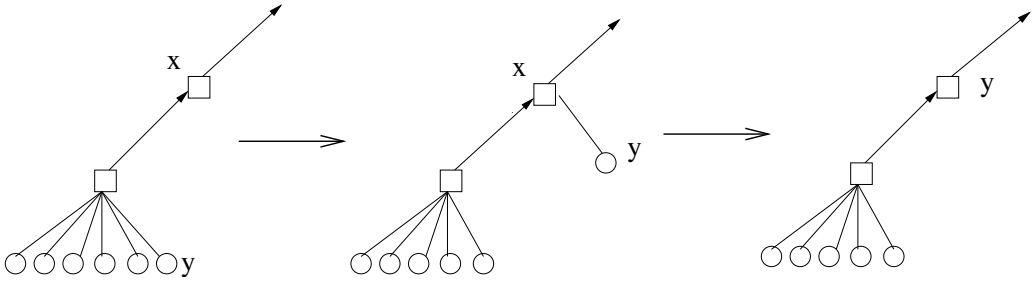


Figure 4.17: Dynamic Node Departure - Case 4. Unshared slaves available.

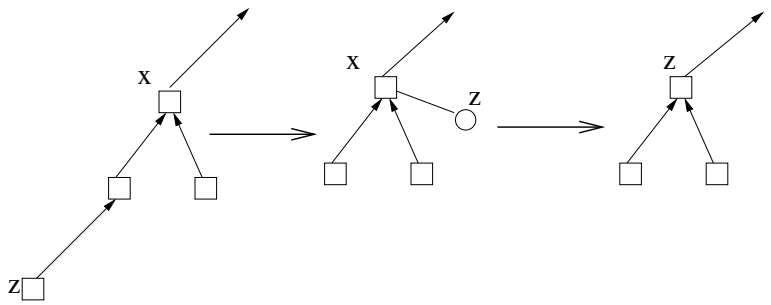


Figure 4.18: Dynamic Node Departure - Case 4. No unshared slaves available.

a slave from another piconet in the tree. x will ask its children (if any) and parent (if one exists) for an unshared slave. This request is recursive, in that if the child has no unshared slaves, it will ask each of its children for an unshared slave. As soon as an unshared slave is located (node y), its information will be sent to node x. The node x will go into PAGE, while node y will go into PAGE SCAN. Once the connection is established, node y will break its link to its old piconet. At this point, node x now has an unshared slave, node y, and the standard process of case 2 can be used.

Should the situation occur where there are no unshared slaves in the tree, then a Leaf node will be selected, even though it is a master. This will not negatively affect the tree because the master has no slaves. As such, it is only responsible for its own communications. The master (node z in Figure 4.18) will return its own information to node x, then enter PAGE SCAN. Node x will enter PAGE mode and then make the connection to the former leaf. Once connected, the leaf will disconnect from its former parent. Node z is now an unshared slave, and case 2 can be used. If z is a leaf and a child if x, then the x-z link can be reversed and case 2 can be used.

CHAPTER FIVE

PERFORMANCE EVALUATION

The ACB-Tree scatternet formation algorithm (ATSF) was implemented using Blueware 1.0, a Bluetooth network simulator built as an extension to NS2. This simulator closely follows the Bluetooth 1.1 specification and has been used to test a number of other algorithms. Three algorithms were used to evaluate performance characteristics, including TSF, BlueMesh and BlueCube. These algorithms were chosen to provide a breadth of formation shapes (a tree scheme, a mesh scheme and a cube scheme), and each had a readily available implementation on Blueware for a fair and direct comparison.

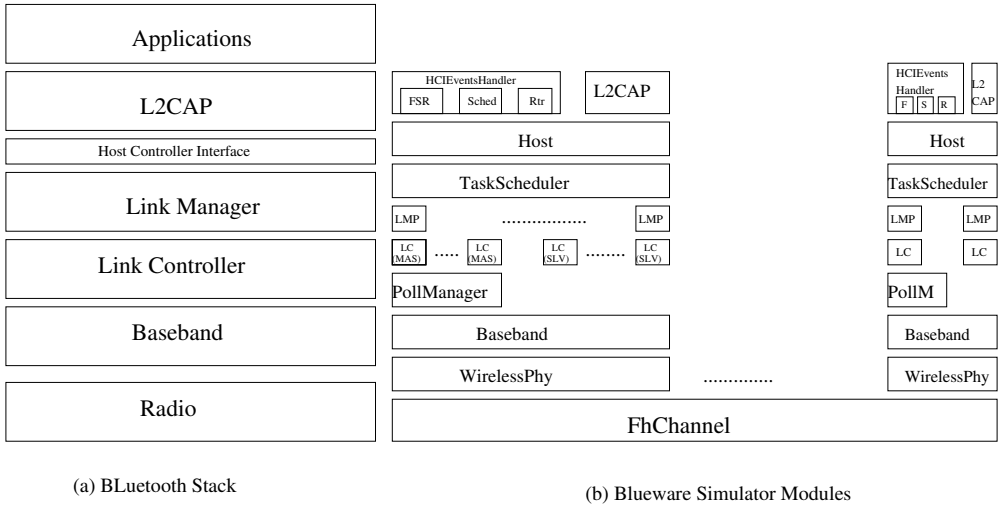
5.1 Blueware Simulator

As noted previously, the Blueware 1.0 [4] module adds Bluetooth simulation capabilities to the NS2 discrete time network simulator. NS2 provides a thorough infrastructure for simulation of wireless and wired environments, multiple stream types including TCP and UDP, as well as routing and multicast protocols for a variety of networks. Parameters for simulations are easily manipulated via a scripting interface, and the open source nature of the simulator has created a community which provides a vast array of add on modules, including Blueware.

The Blueware module closely follows the Bluetooth 1.1 specification, implementing most aspects of the Bluetooth protocol stack. An illustration of the Blueware protocol stack versus the standard Bluetooth protocol stack can be seen in Figure 5.1. [34] provides full description of the Blueware implementation.

5.2 Blueware Limitations

Though the Blueware simulator provides a reasonable enough approximation of the Bluetooth specification to allow analysis of scatternet formation algorithm performance, there are a number



(a) BLuetooth Stack

(b) Blueware Simulator Modules

Figure 5.1: Bluetooth protocol stack and the corresponding Blueware stack.

of caveats which exist that can have an effect on the simulation results. Among other aspects, some key aspects are the lack of SCO link support, an insufficient interference model and the inability to place nodes in SNIFF or PARK modes. Fixing or improving these aspects would make Blueware follow the specification more closely, but these limitations do not significantly impact the result.

5.3 Simulation Parameters

The simulation parameters were chosen to closely match those used by other formation algorithms to ensure a fair comparison. During testing, each experiment was run with node counts varying between 20 and 200 nodes. Each data point is the average of ten runs with the given number of nodes, each run using a different random seed. Due to all nodes being set as Class 3, the simulated area was set to be 7.07 x 7.07 meters, ensuring that all nodes would be within range of each other. Unless otherwise noted, node arrival was en masse, with all nodes beginning the simulation at time 0 seconds. The default phase two timeout was set at 10.0 seconds, with an allowed merge difference of 1. The results of the simulations were analyzed to determine the formation time and the number of piconets in the resulting scatternet. The piconet count and scatternet formation delay of the ACB-Tree algorithm was compared with all three other schemes. The height of the

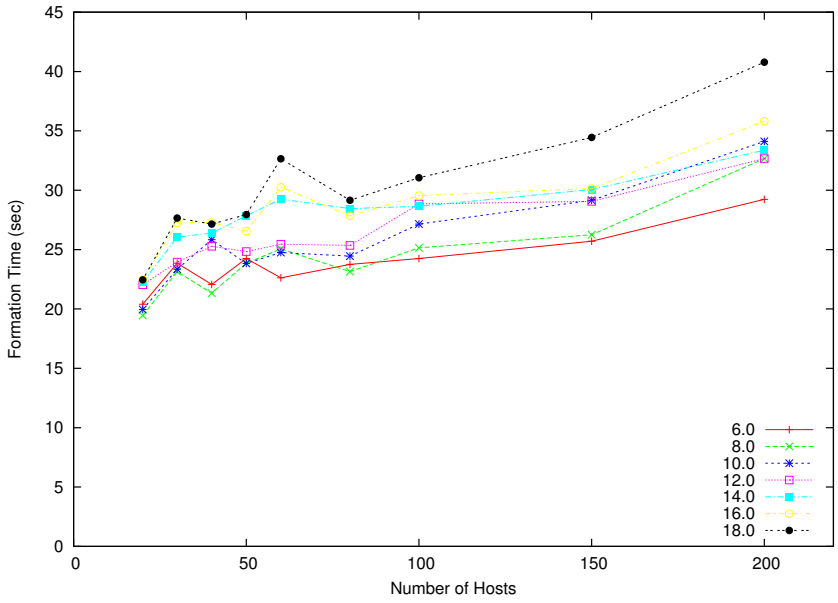


Figure 5.2: Average Formation Time vs. Number of Hosts.

final ACB-Tree scatternet was compared to the height of TSF, the only other tree based algorithm.

5.4 Simulation Results

Daptardar and Medidi [13] showed that extending the phase one timeout had limited influence on either the overall formation delay or the final piconet count. Based on these results, the phase one timeout for all experiments was set at 3.5 seconds. In contrast, the phase two timeout can have a significant effect on the resulting scatternet.

We varied the phase two timeout between 6.0 seconds and 18.0 seconds, inclusive, in two second increments. Figure 5.2 shows the results of these tests in regards to the overall scatternet formation delay. As evidenced by the simulation results, the timeout used for phase two has little effect on the overall delay. The final height of the scatternet, shown in Figure 5.3, is nearly identical for all of the timeout values tested.

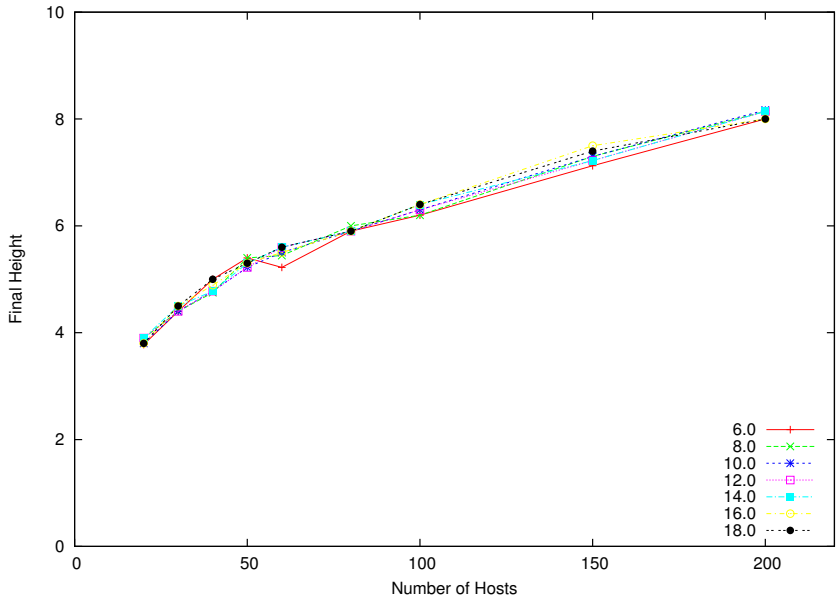


Figure 5.3: Height vs. Number of Hosts.

Phase 3 Height Change

An important aspect of selecting the proper phase two timeout is determining the tradeoffs between formation time and height. As was seen in Figure 5.3, the final height of the scatternet is consistent regardless of the phase two timeout selection. Therefore, a better method for determining the correct timeout is to analyze the height change which occurs in phase three. As can be seen in Figure 5.4, increasing the phase two timeout will decrease the number of merges which occur in phase three. The main benefit of minimizing the number of phase three merges is that phase two only allows merges of trees which have a similar height. This in turn will result in better balance in the final tree.

Scatternet Formation Delay

Figure 5.5 shows the average formation time for each of the four scatternet formation algorithms with regards to the number of hosts in the scatternet. As predicted, TSF performed the best in the formation time comparison, yielding scatternet formation in roughly 12 seconds on average. This is due to the design of TSF, in that it only focuses on forming a loop-free connected scatternet

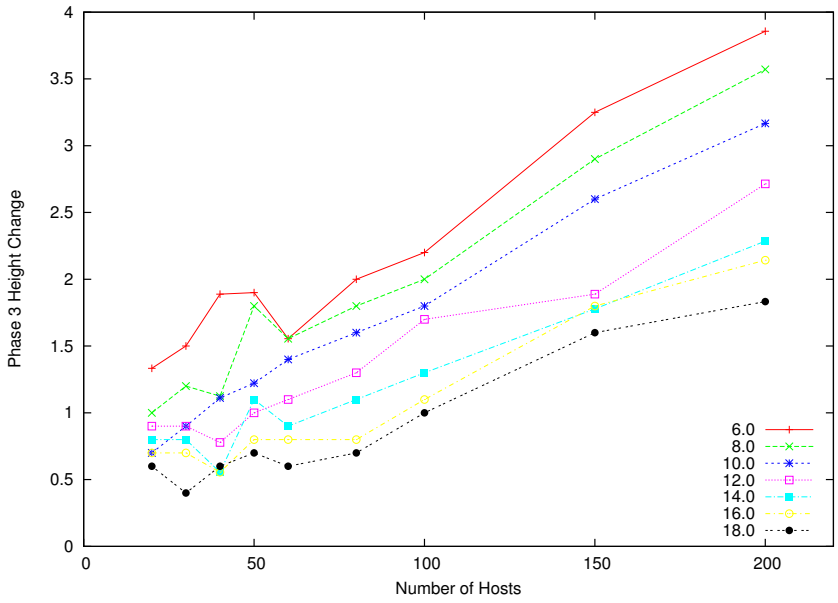


Figure 5.4: Phase 3 Height Change vs. Number of Hosts.

and allows for joins at any point in the tree. This also allows TSF to combine multiple smaller trees simultaneously, while the other three algorithms require merges only in pairs. Because of TSF's sole focus on obtaining a connected scatternet quickly, there is no attempt to control over scatternet characteristics such as height, diameter or the number of piconets. The merge process used in TSF also provides no natural method for obtaining an energy efficient scatternet.

Among the remaining three algorithms, ATSF significantly performed better than both BlueMesh and BlueCube. The difference is due to the numerous connections which must be made when forming mesh or cube structures, while ATSF only requires one additional connection beyond the one used for discovery. ATSF also allows for more flexibility during merges by allowing a merge to occur if two components which are close enough in height based on the allowed merge difference. This results in fewer disconnections, reducing the number of times the device discovery process must occur. The similar overall trend of these three algorithms reflects the fact that device discovery dominates the time required to form a scatternet.

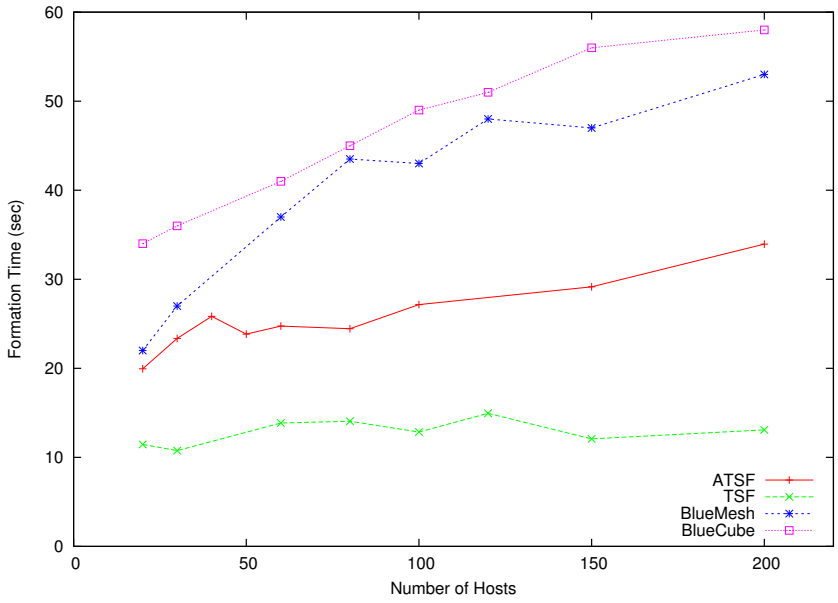


Figure 5.5: Formation Time vs. Number of Hosts.

Piconet Count

As can be seen in Figure 5.6, ATSF produced the fewest piconets of all four algorithms. BlueMesh and BlueCube each have higher piconet counts due to the extra links needed for mesh and cube structures and yielded respectively higher piconet counts. TSF, which does not attempt to minimize the number of piconets, produced the highest piconet count.

Scatternet Height

In terms of height, ATSF performed better than TSF, the only other tree based scheme. While ATSF scatternets averaged a height which was 2-3 levels less than the scatternets produced by TSF, the biggest difference lies in the ranges of heights produced. As can be seen by Figure 5.7, ATSF scatternet heights varied by only 1 on the high and low side of the average. In contrast, TSF heights varied wildly, with ranges of 7 levels around the average being observed.

Overall, the experimental performance results show that ATSF performed well under the conditions under which it was designed. ATSF produced connected scatternets quickly and with tight control over the height of the resulting tree. ATSF also produced significantly fewer piconets than

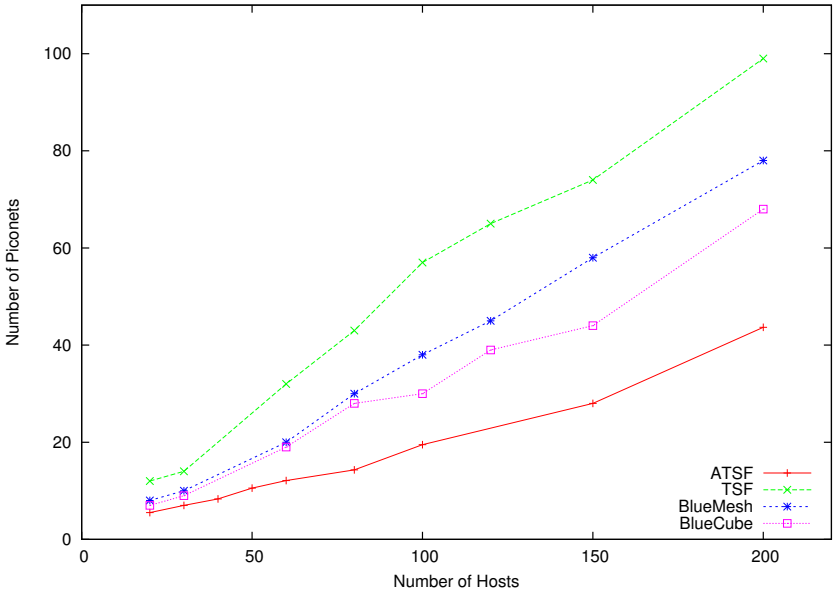


Figure 5.6: Piconet Count vs. Number of Nodes.

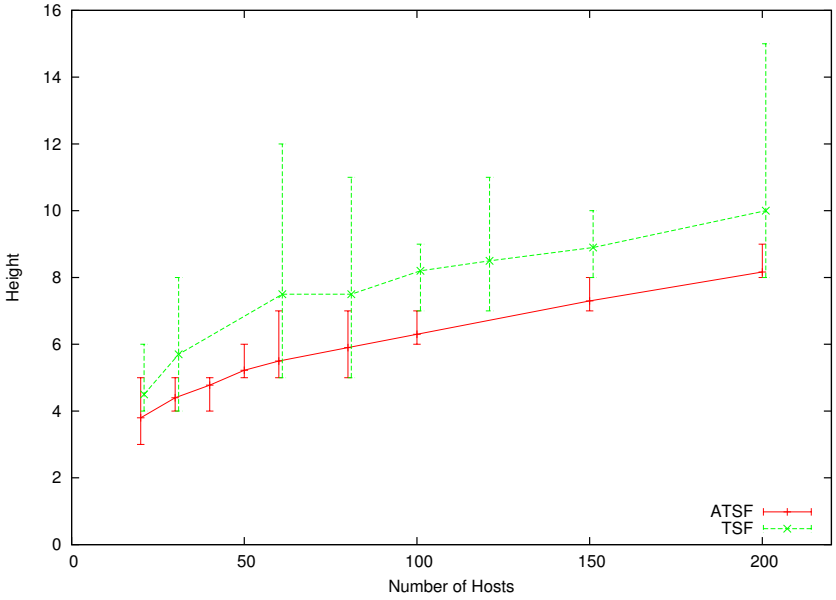


Figure 5.7: Height vs. Number of Nodes.

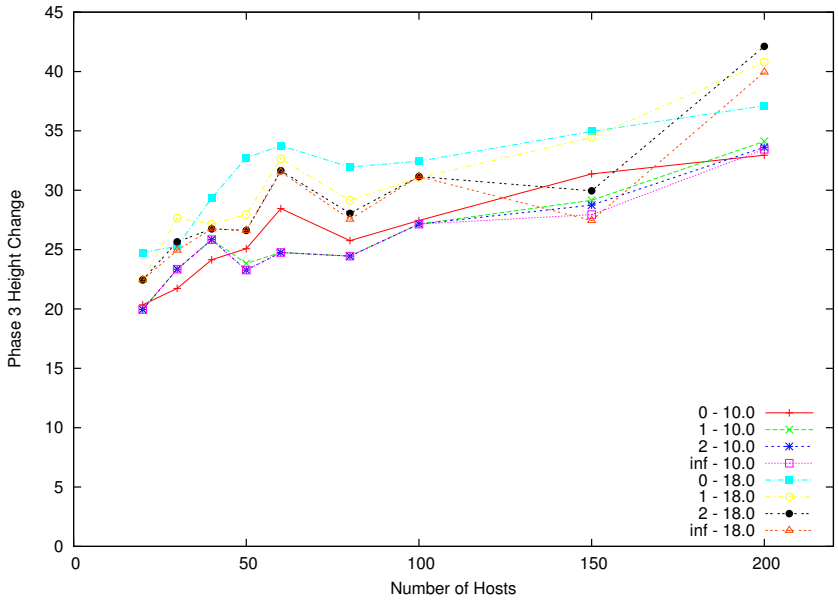


Figure 5.8: Formation Time vs. Number of Nodes.

other algorithms, especially TSF, which will yield better scatternet performance. Finally, ATSF provided consistent results with little variation in all three of the metrics.

Allowed Merge Difference

We also conducted simulations with the allowed merge difference set to 0, 1, 2 and infinity. The goal was to determine the effects of using tighter or looser merge constraints. The results of this can be seen in Figures 5.8, 5.9, and 5.10.

Figure 5.9 closely matches the results found when varying the phase two timeout but keeping the merge difference fixed at 1 (Figure 5.3). Other than a slight reduction in final tree height when setting the allowed merge difference to 0, the results indicate that modifying the allowed merge difference does not create a significant effect on the final height of the scatternet.

In terms of the phase three height change (Figure 5.10), we can see the two groupings, defined by the common phase two timeout, produce similar changes in height during phase three. These results coincide with those found in Figure 5.4.

In essence, these results indicate that the penalties are minimal when seeking a tree which is

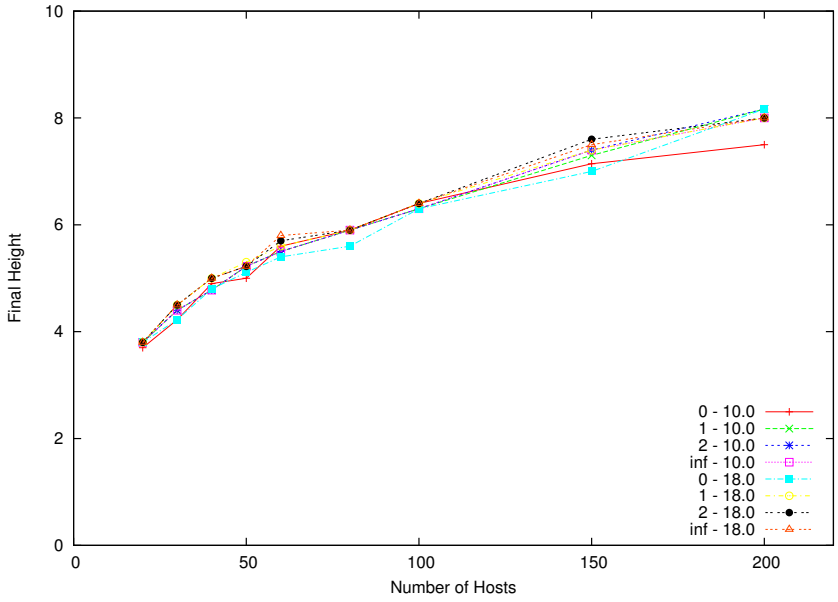


Figure 5.9: Height vs. Number of Nodes.

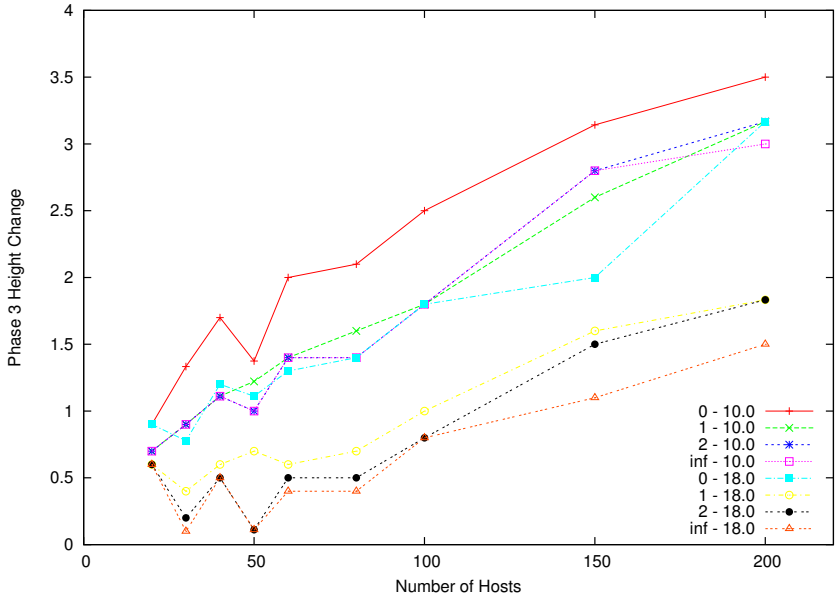


Figure 5.10: Phase 3 Height Change vs. Number of Nodes.

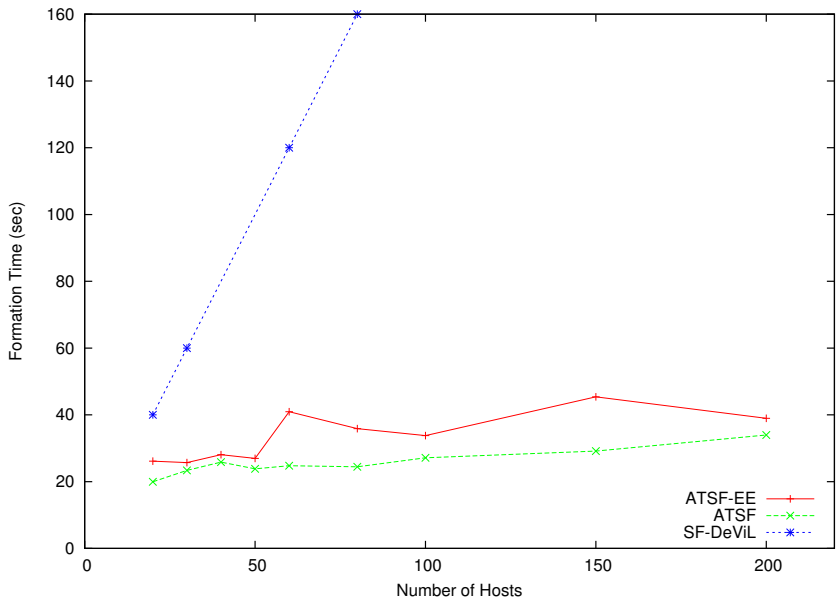


Figure 5.11: Formation Time vs. Number of Nodes. ATSF-EE vs. ATSF, SF-DeviL.

more balanced. Therefore, the allowed merge difference should be set at either 0 or 1, which produces a tree which is balanced or almost balanced, since there are no benefits to allowing larger differences.

Energy Efficiency

Energy efficiency was implemented as described in Section 4.4. The energy efficient modification to the ATSF algorithm (ATSF-EE) was evaluated against the only other scatternet formation algorithm which addresses energy efficiency, SF-Devil [21]. Experimental results for ATSF-EE were compared with those presented in [21] for SF-DeviL as well as results from the standard ATSF results. It should be noted that the simulation environment used in [21] was not available for testing. Therefore, the results presented for SF-DeviL are derived from the graphs shown in [21].

Formation Time

Figure 5.11 shows the results for ATSF-EE, ATSF and SF-DeviL. As the number of hosts increases the formation time increases exponentially for SF-DeviL. Specifically, a network of only 60 nodes takes two minutes to form. In contrast, ATSF-EE takes less than 40 seconds for the same

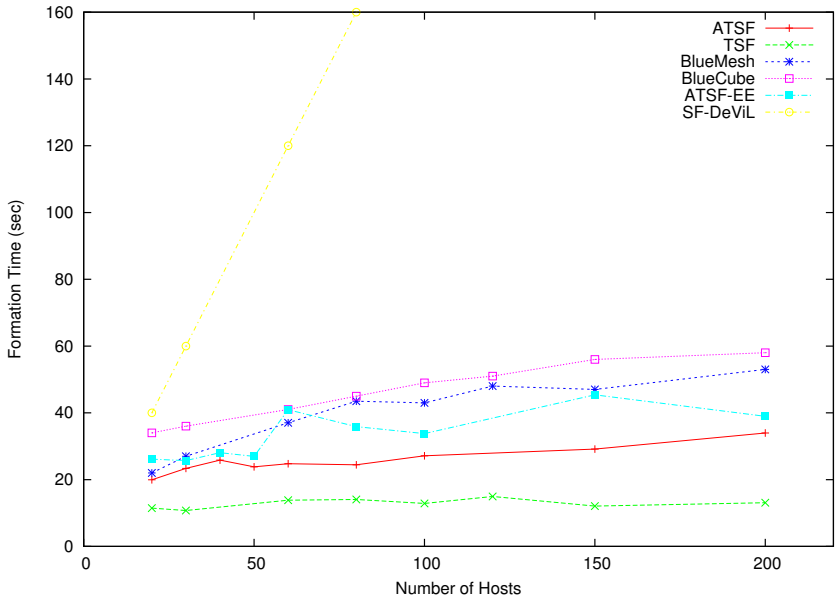


Figure 5.12: Formation Time vs. Number of Nodes. All tested algorithms.

60 nodes.

When compared to the standard ATSF algorithm, ATSF-EE takes less than 20 additional seconds to form a connected scatternet. On average, the cost for forming an energy efficient scatternet is roughly 5 extra seconds.

These results correlate with previous expectations for ATSF-EE. During the merge process, roughly half of the time the existing links can be used to create a new energy efficient component. In the situation where those links are not usable, only one additional link must be established. This is in sharp contrast to SF-DeviL, which has no bound on the number of links which must be rearranged.

Figure 5.12 shows ATSF-EE in comparison to all other tested algorithms. This illustrates that obtaining energy efficiency using ATSF-EE has a low cost when compared to ATSF and other algorithms.

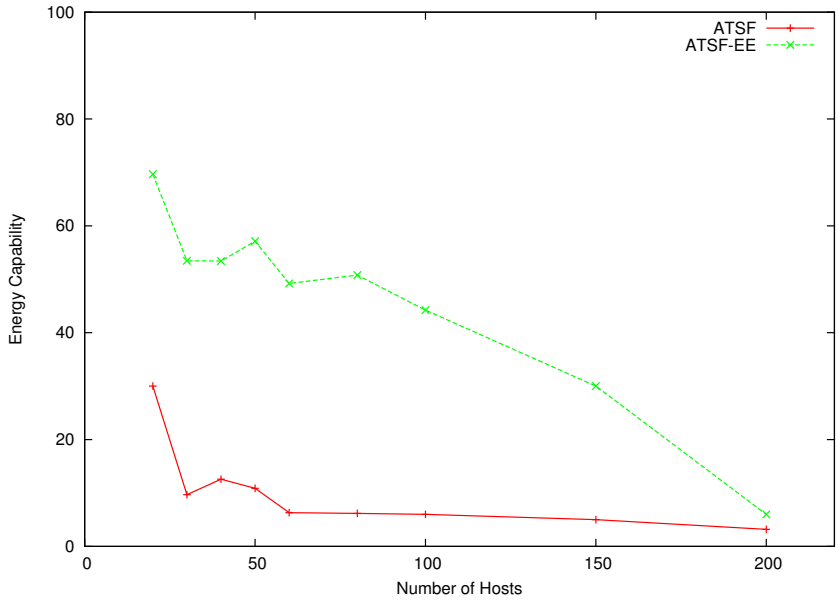


Figure 5.13: Lowest energy capability among piconet masters.

Lowest Master

Figure 5.13 shows the lowest energy capability among nodes which are piconet masters. Because the energy capabilities were uniformly distributed, a high value indicates that the piconet masters are all fairly capable in comparison to the unshared slaves in the scatternet. As can be seen in the graph, the standard ATSF algorithm has masters which are consistently lower in energy capability than the masters in ATSF-EE. This is expected since ATSF is essentially choosing the piconet master nodes randomly from a given piconet, while ATSF-EE is choosing the most capable node from the piconet.

The convergence which occurs as the number of nodes increase is expected. This occurs because, as the number of hosts increase, the likelihood of establishing a piconet where all the nodes are not very capable is increased.

Battery Level Violations

Figure 5.14 shows the number of battery level violations which exist in the final scatternet. We define a battery level violation as the situation where a node has a higher energy capability than its

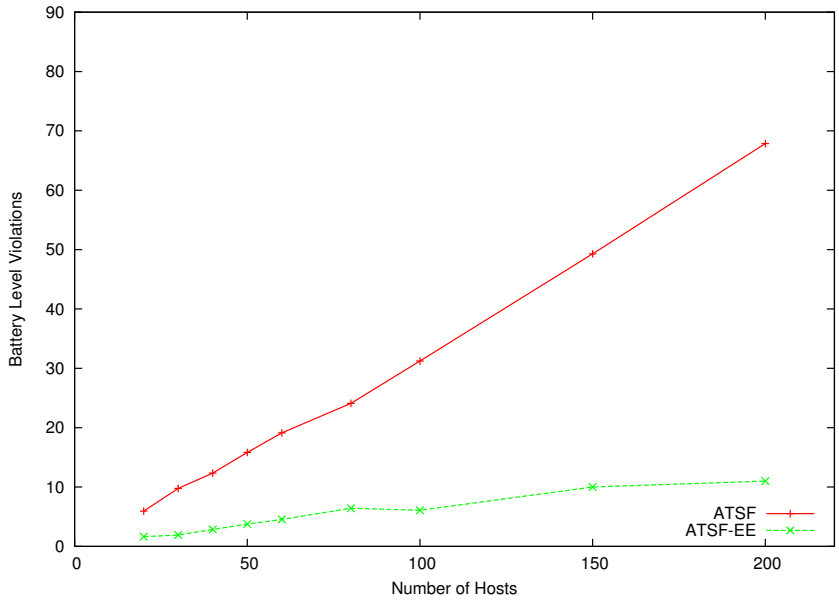


Figure 5.14: Number of Battery Level Violations vs. Number of Nodes.

parent, or when an unshared slave has a higher energy capability than its piconet master.

The number of violations which occur in ATSF increases linearly with respect to the number of hosts in the scatternet. Because the connections are made without regard to energy capability and the energy levels are uniformly distributed, it is reasonable that roughly half of the time the more energy capable node would "win" out over the less energy capable node. This trend was observed in the experimental data.

ATSF-EE yielded a violation trend which was nearly flat, with only 11 violations occurring with 200 nodes in the scatternet. In comparison, ATSF had 70 violations for the same number of nodes. One cause for these violations is when a lone piconet is swallowed and placed into a hole where the parent has a lower energy capability than the swallowed piconet. The other situation which can create an battery level violation is illustrated in Figure 5.15, which is characterized by the Handle (node v) of a tree has a lower energy capability than the both the Root (node u) and Handle (node x) of the other tree. In both of these situations, one battery level violation will result.

While having zero violations would be ideal, a low number of violations could be considered

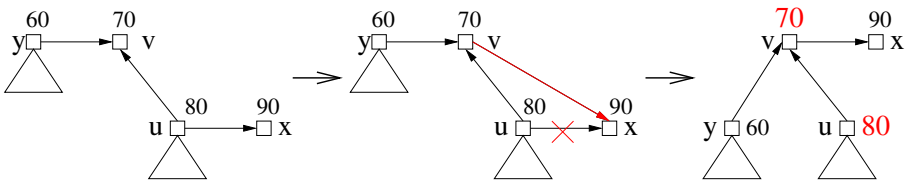


Figure 5.15: Battery level violation in the general merge case.

acceptable if the cost for fixing them was too high in terms of the overall scatternet formation delay. At 200 nodes, the scatternet had a 5.5% battery violation rate (the number of violations divided by the total number of nodes). These violations do not destroy the overall energy efficient nature of the tree, they simply indicate that a more energy efficient structure could be produced.

Overall, the results for ATSF-EE show that it is feasible to obtain a scatternet with energy efficient properties in a short period of time. The addition of the energy efficient consideration does not invalidate any of the standard properties of the ATSF algorithm. ATSF-EE is able to produce tree scatternets with bounded diameter, a fixed number of roles per device and fast formation time while adding the additional benefit of energy efficiency.

CHAPTER SIX

CONCLUSIONS

Summary

This thesis presented a novel scatternet formation scheme for Bluetooth networks. Various metrics were identified via existing approaches, with a reasonable subset selected as the focus for the proposed algorithm. Scatternet formation delay, piconet count, piconet membership count and network diameter were selected as important traits for scatternet formation algorithms. Additionally, energy efficiency and dynamic node arrival/departure were also identified as important traits for a generally acceptable formation algorithm.

The Almost-Complete-Binary Tree algorithm was designed to create energy efficient bounded diameter scatternets in a distributed environment. A recursive doubling technique is used to combine two components into a single larger component. The algorithm is robust in spite of mobility in the form of dynamic node arrival and departure. ACB-Trees also create trees with energy efficient characteristics, such that the energy capability of nodes increases towards the root of the tree. The piconet membership for nodes is fixed at one for all slaves and leaf nodes, two for the handle, and three for all internal bridge nodes. The tree structure provides a natural routing structure, while the limitations on piconet membership and minimal piconet count simplify the link scheduling which must occur.

Experiments showed that the ACB-Tree scatternet formation algorithm met these goals, creating balanced tree structures in a short period of time. The resulting scatternet had fewer piconets than any of the other algorithms tested. In comparison to TSF, the other tree based algorithm, ATSF produced trees with a smaller diameter and maintained a significantly tighter bound on the variation in diameter. ATSF yielded the lowest scatternet formation delay compared to BlueMesh and BlueCube, which were the only algorithms tested which sought to control the scatternet diameter. Experiments also showed that variation of the allowed merge difference and/or the phase two

timeout had no effect on the final scatternet height and little effect on the formation delay.

Future Work

In regards to further research, the first direction should be toward the development of link scheduling and routing algorithms. Implementation of these two remaining aspects of the scatternet formation problem would allow end-to-end delay to be analyzed. While existing generic methods could be utilized, a specialized algorithm taking advantage of the ACB-Tree structure could produce better results.

Test results indicated that, depending on the selection of the phase two timeout, a significant height change can occur in phase three. Possible areas for improvement could include the use of an orphan list similar to the one used in [13] or iterative relaxation of the merge difference constraint. Allowing lone piconet swallows, currently only permitted in phase three, to occur in phase two could further reduce the final height and improve formation delay.

Finally, the algorithm should eventually be adapted to handle the condition where some devices are out-of-range. This is important both in terms of areas which are physically expansive and when the scatternet contains devices with a mixture of Class 1, 2 and 3 radios.

BIBLIOGRAPHY

[1] Bluetooth Special Interest Group, *http://www.bluetooth.com*

[2] NS-2 Network Simulator, *http://www.isi.edu/nsnam/ns*

[3] BlueHoc Simulator, *http://bluehoc.sourceforge.net*

[4] Blueware Simulator, *http://nms.lcs.mit.edu/software/blueware*

[5] GloMoSim, *http://pcl.cs.ucla.edu/projects/glomosim*

[6] SimJava, *http://www.dcs.ed.ac.uk/home/hase/simjava*

[7] University of Cincinnati Bluetooth Simulator, *http://www.ececs.uc.edu/∼cdmc/ucbt*

[8] A. Aggarwal, M. Kapoor, L. Ramachandran, and A. Sarkar, *Clustering Algorithms for Wireless Ad Hoc Networks*, Proceedings of the 4^{th} International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, pp. 54-63, Boston, MA, August 2000.

[9] S. Baatz, C. Bieschke, M. Frank, P. Martini, C. Scholz and C. Kühl, *Building Efficient Bluetooth Scatternet Topologies from 1-factors*, Proceedings of the IASTED International Conference on Wireless and Optical Communications, WOC 2002, Banff, Alberta, Canada, 2002.

[10] L. Barriere, P. Fraigniaud, L.Narayanan and J. Opatrny, *Dynamic Construction of Bluetooth Scatternets of Fixed Degree and Low Diameter*, Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2003.

[11] H. Chen et al. *Topology-Controllable Scatternet Formation Method and Its Implementation*, International Workshop on Wireless Ad-Hoc Networks, Oulu, Finland, 2004.

[12] F. C. Chong, C. K. Chaing, *Bluerings - Bluetooth Scatternets with the ring structure*, Proceedings of the IASTED Wireless and Optical Communications WOC, 2002.

[13] A. S. Daptardar, *Meshes and Cubes: Distributed Scatternet Formations for Bluetooth Personal Area Networks*, M. S. Thesis, Washington State University, 2004.

[14] Y. Dong and J. Wu, *Three Bluetree Formations for Constructing Efficient Scatternets in Bluetooth*, Proceedings of the 7^{th} Joint Conference on Information Sciences, pp. 385-388, September 2003.

[15] J. Ghosh, V. Kumar, X. Wang, C. Qiao, *BTSpin - Single Phase Distributed Bluetooth Scatternet Formation*, Technical Report 2004-06, New York, 2004.

[16] Y. Kawamoto, V. W. S. Wong and V. C. M. Leung, *A Two Phase Scatternet Formation Protocol for Bluetooth Wireless Personal Area Networks*, IEEE Wireless Communications and Networking Conference, 2003.

[17] M. Kapoor, M. Sanadidi, M Gerla, *An Analysis of Bluetooth Scatternet Topologies*, IEEE International Conference on Communications, vol. 26, no. 1, May 2003, pp. 266-270, 2003.

[18] C. Law and K. Siu, *A Bluetooth Scatternet Formation Algorithm*, In Proceedings of the IEEE Symposium on Ad Hoc Wireless Networks 2001, San Antonio, Texas, USA, November 2001.

[19] M. Medidi and A. Daptardar, *A Distributed Algorithm for Mesh Scatternet Formation in Bluetooth Networks*, Proceedings of the International Conference on Wireless Networks (ICWN), pp. 295-301, 2004.

[20] O. Miklos, A. Rácz, Z. Turányi, A. Valkó and P.Johanson, *Performance Aspects of Bluetooth Scatternet Formation*, First Annual Workshop on Mobile and ad-Hoc Networking and Computing(MobiHoc), 2000.

[21] C. Pamuk and E. Karasan, *SF-devil : Distributed Bluetooth Scatternet Formation Algorithm based on Device and Link Characteristics*, Proceedings of the Eighth IEEE International Symposium on Computers and Communication, 2003.

[22] K. Persson and D. Manivannan. *Distributed Self-Healing Bluetooth Scatternet Formation*, Proceedings of the International Conference on Wireless Networks, pp. 325-334, 2004.

[23] C. Petrioli, S. Basagni and I. Chlamtac, *Configuring Bluestars: Multihop Scatternet Formation for Bluetooth Networks* IEEE Transactions on Computers, Volume: 52 , Issue: 6, pp. 779-790, June 2003.

[24] C. Petrioli and S. Basagni, *BlueMesh: Degree-Constrained Multihop Scatternet Formation for Bluetooth Networks*, Mobile Networks and Applications, vol. 9, no. 1, Feb. 2004.

[25] B. Prabhu and A. Chockalingam, *A Routing Protocol and Energy Efficient Techniques in Bluetooth Scatternets*, IEEE International Conference in Communications (ICC) 2002, New York, 2002.

[26] L. Ramachandran, M. Kapoor, A. Sarkar and A. Aggarwal, *Clustering Algorithms for Wireless Ad-Hoc Networks*, Proceedings of the 4^{th} International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, 2000.

[27] T. Salonidis, P. Bhagwat, L. Tassiulas and R. LaMaire, *Distributed Topology Construction of Bluetooth Personal Area Networks*, Proceedings of the IEEE INFOCOM, IEEE INFOCOM, Anchorage, AK, pp. 1577-1586, Apr. 2001.

[28] W. Song, X. Li, Y. Wang and W. Wang, *dBBlue : Low Diameter and Self-routing Bluetooth Scatternet*, MobiHoc, 2003.

[29] H. Sreenivas and H. Ali, *An Evolutionary Bluetooth Scatternet Formation Protocol*, Proceedings of the 37^{th} Hawaii International Conference on System Sciences, 2004.

[30] I. Stojmenovic, *Dominating Set Based Bluetooth Scatternet Formation with Localized Maintenance*, International Parallel and Distributed Processing Symposium, 2002.

[31] M. Sun, C. Chang and T. Lai, *A self-routing topology for Bluetooth Scatternets*, Parallel Architectures, Algorithms and Networks, 2002.

[32] S. Sunkavalli and B. Ramamurthy. *MTSF: A Fast Mesh Scatternet Formation Algorithm for Bluetooth Networks*, Globecom 2004, 2004.

[33] G. Tan and J. Guttag, *An Efficient Scatternet Formation Algorithm for Dynamic Environments*, IASTED International Conference on Communications and Computer Networks, Cambridge, MA, 2001.

[34] G. Tan, *Blueware: Bluetooth Simulator for ns*, MIT Technical Report, MIT-LCS-TR-866, October 2002.

[35] Q. Wang and P. Agrawal, *A Dichotomized Rendezvous Algorithm for Mesh Bluetooth Scatternets*, Ad Hoc & Sensor Wireless Networks, Vol. 1, pp. 65-88, 2004.

[36] Z. Wang, R. Thomas, Z. Haas, *Bluenet - Scatternet Formation to Enable Bluetooth-Based Ad Hoc Networks*, Proceedings of IEEE International Conference on Communications, 2001.

[37] G. Zaruba, S. Basagni and I. Chlamtac, *Bluetrees- Scatternet Formation to Enable Bluetooth-based Ad Hoc Networks*, IEEE International Conference on Communications, 2001.

[38] H. Zhang, C. Hou and L. Sha, *Design and Analysis of a Bluetooth Loop Scatternet Formation Algorithm*, IEEE International Conference on Communications, 2003.

[39] B. Zhen, J. Park and Y. Kim, *Scatternet Formation of Bluetooth Ad hoc Networks*, Proceedings of the 36^{th} Hawaii International Conference on System Sciences, 2003.