MESHES AND CUBES: DISTRIBUTED SCATTERNET FORMATIONS FOR BLUETOOTH PERSONAL AREA NETWORKS

By

ANIRUDDHA SHRIRAM DAPTARDAR

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

MASTER OF SCIENCE

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

MAY 2004

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of ANIRUDDHA SHRIRAM DAPTARDAR find it satisfactory and recommend that it be accepted.

Chair

ACKNOWLEDGEMENT

First and foremost, I would like to thank that eternal power which exists in this world, we call GOD, which has helped me in a number of ways. I have faced many challenges during this eventful journey and I am grateful to GOD for helping me come out of all those safe and sound.

This thesis is dedicated to my family, *Aai* (Dr. Mrs. Vaidehi Daptardar), *Baba* (Mr. Shriram Daptardar) and my sweet younger sister, *chingi* (Ms. Sayali Daptardar). Without their love, support and motivation, I would not have reached this stage and this work would have been far from complete. *Aai* and *Baba* have always been more than a friend to me and have provided me with their valuable suggestions and shared their experience from time to time. They have shown me a path to walk on with their own footprints. They have given me the inspiration needed to face all the twists and turns of life with courage. The home-made food has always kept me healthy and enthusiastic. Sayali, with her sweet and friendly talk has always made me feel at home. She is the best sister.

I am very thankful to my advisor Prof. Murali Medidi, who has always played the role of a lighthouse for me. He has shown me the correct direction for research and saved me from veering in wrong direction. He has always been more than just an advisor, being a friend, philosopher and guide to me. It was fun working with him, learning from his past experiences and understanding how to improve myself in different ways. He has always motivated me to work hard and also showed me the importance of taking “breaks” while studying. His constant urge for perfection has helped me throughout this work. I would also like to thank Prof. Sirisha Medidi for serving on my thesis committee and for her support. Both, Prof. Murali and Prof. Sirisha have given me a homely feeling and parental support which has helped me stay so far away from my parents. I am very grateful to them.

A sincere thanks to Prof. Dyreson for serving on my committee, for sharing his technical expertise and for his valuable comments which helped me make this work more accurate. I would

also like to take this opportunity to thank Prof. Zhe Dang and Prof. Delgado-Frias for their support throughout my Masters program.

Thanks to all the members of Wireless Security Research (WiSeR) Group for their help and sharing their skills and their work with me. A million thanks to Ms. Balvinder Thind (Bindu) for being the friend I needed the most, not only supporting me but correcting me whenever needed. I am also thankful to her *Bhaiya*, *Bhabhi* and her cute little nephew Ranveer who gave me the warmth of a home and considered me a member of their family. The *Desi Janata* of Pullman has been fabulous in many ways; especially to mention that they made Pullman, an enjoyable place for me with all the celebrations of various Indian festivals, movies, dance parties through Indian Students Association(ISA) and not to forget, cricket matches.

Finally, I would like to thank all those who have helped me make this dream of mine come true.

MESHES AND CUBES: DISTRIBUTED SCATTERNET FORMATIONS FOR BLUETOOTH PERSONAL AREA NETWORKS

Abstract

Aniruddha Shriram Daptardar
Washington State University
May 2004

Chair: Muralidhar Medidi

Bluetooth Scatternet formation has received much attention recently, mainly because the structure of the scatternet has enormous impact on the overall network throughput. Along with the basic requirement of connectivity, a good scatternet is expected to provide many other properties. In this thesis, we propose two schemes for scatternet formation in Bluetooth personal area networks.

A careful study of the basic requirements and the properties of the Bluetooth technology showed that, the well known *Mesh* interconnection network is well-suited for scatternet formation. This thesis presents a decentralized algorithm for scatternet formation which guarantees a connected mesh of Bluetooth devices for those that are within transmission range of each other. Extending the same idea to 3-dimensions, we also provide another fully distributed algorithm to interconnect all the Bluetooth devices in a *cube* scatternet. Both the algorithms not only meet the minimum requirements of a Bluetooth scatternet but also provide other desirable properties such as higher connectivity, lower diameter, less node contention, multiple paths between any two nodes, in-built routing, easy inter-piconet scheduling and the ability to reconfigure for dynamic environments.

We implemented both the algorithms and used the Blueware simulator developed at MIT to test the algorithms. The simulation results indicate that we can create both the topologies in a time comparable to other scatternet formation algorithms. The results also show that both the scatternets

have much lower diameters and less node contention when compared with other proposed scatternets. Resulting meshes and cubes, from our simulations, make it very evident that both scatternets also can handle dynamic leave and join of Bluetooth devices very efficiently. We also tested our algorithms when some of the devices could be out-of-range of each other. The empirical evidence shows that the algorithms still generate connected scatternets with a high success rate.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

Dedication

CHAPTER ONE

INTRODUCTION

Bluetooth is a low cost, low power, short range radio technology, originally developed as a cable replacement technology to inter-connect devices such as cell phones, headsets, and portable computers. Bluetooth can be used to create a Personal Area Network (PAN), which is a kind of close range wireless network.

The Bluetooth specification is an open, global specification. It defines the complete system from the physical up to the application level. The protocol stack is usually implemented partly in hardware and partly as software running on a microprocessor, with different implementations partitioning the functionality between hardware and software in different ways.

1.1 Bluetooth Origins

Version 1.0 of the Bluetooth specification came out in 1999, but Bluetooth started five years earlier in 1994, when Ericsson Mobile Communications began a study to examine alternatives to cables that linked its mobile phones with accessories. The study looked at using radio links. Since, radio technology isn't directional and it does not need line of sight, it has obvious advantages over the infra-red links previously used between handsets and devices. There were many requirements for the study including handling both voice and data, so it could connect to both headsets and computing devices.

Out of this study was born the specification for Bluetooth wireless technology. The specification is named after Harald Blatand (Blatand is Danish for Bluetooth): Harald was a tenth century Danish Viking king who united and controlled Denmark and Norway. The name was adopted because Bluetooth wireless technology is expected to unify the telecommunications and computing industries.

1.2 The Bluetooth SIG

The Bluetooth Special Interest Group (SIG) is a group of companies working together to promote and define the Bluetooth specifications. The Bluetooth SIG was founded in February 1998 by the following group of core companies: Ericsson Mobile Communications AB, Intel Corporation, IBM Corporation, Toshiba Corporation and Nokia Mobile Phones.

In May 1998, the core promoters publicly announced the global SIG and invited other companies to join the SIG as Bluetooth adopters in return for a commitment to support the Bluetooth specification. The core promoters published the version 1.0 of the Bluetooth specification in July 1999, on the Bluetooth website http://www.bluetooth.com. In December 1999, the Bluetooth core promoters group enlarged with the addition of four major companies: Microsoft, Lucent, 3COM, Motorola.

Today over 3000 companies are members of the Bluetooth SIG and are working for the development of this open specification.

1.3 What Can You Do with Bluetooth ?

The Bluetooth specification enables you to connect a wide range of computing and telecommunications devices easily and simply, without cables. It will provide opportunities for rapid ad hoc connections, and make automatic connections between devices possible. Because Bluetooth wireless technology can be used for a variety of purposes, it also replaces multiple types of cable connections with a single radio link. It will allow you to concentrate about what you are working on, rather than how to connect everything to make the technology work.

The applications of Bluetooth are found to enhance the life in a myriad of ways. It is also possible to coordinate JAVA with Bluetooth since both have platform independent nature. Bluetooth can also provide the robust physical layer for *Jini* technology introduced by Sun Microsystems in 1997. Bluetooth can also provide connectivity with other technologies JetSend and HAVi.

Hence, Bluetooth has enormous capabilities which makes it a suitable candidate for general

purpose use and for short range ad hoc networking without cables. Bluetooth devices are cheap too. A normal Bluetooth chip costs mere US $5, providing a cheap and reliable solution.

1.4 Challenges faced by Bluetooth

The Bluetooth ad hoc networking brings with it new challenges. Bluetooth with its emphasis on low power and low cost put specific restrictions not present in other wireless networks. Bluetooth technology, though promising, is severely limited in terms of its data rate of about 1Mbps. The current range supported by Bluetooth devices is close to 10 meters which is very small as compared with many other wireless technologies. Bluetooth specification can scale upward to support wireless data transmission applications operating in 5 GHz range, supporting connections to devices upto 300 feet away. However, achievement of higher area coverage and higher data rates, is a long way to go from now.

The Bluetooth specifications are not yet complete and hence, there are many areas left open for research. Some of those areas include the topology for scatternet, inter-piconet scheduling, routing inside scatternets etc. Since, a number of devices are shared among piconets in order to construct the scatternet, their efficient scheduling is important for the network throughput. The scatternet structure also has enormous impact on the overall throughput of the network and hence, it should be designed carefully.

1.5 Motivation and Problem Statement

The Bluetooth specification allows an ad hoc network to be made up of piconets. Each piconet consists of a master and a number of slave devices. The piconets can be interconnected to form a *scatternet* through *gateways*. However, the Bluetooth specification does not define the structure of the scatternet. Scatternet formation recently became an area of intense research activity (see Chapter 3 for review of research work).

The scatternet formation problem is deciding role for each node i.e. assignment of roles to

Bluetooth devices. Depending upon this assignment, a node could be a master, slave or a bridge. This role assignment is responsible for the scatternet topology which has a significant impact on network performance. Routing, scheduling, throughput, and power consumption are all affected by this assignment of roles. Initially, devices have no knowledge of their surroundings or other Bluetooth devices. Each node operates independently and hence, a distributed scheme for scatternet formation has to be used as opposed to a centralized one. Devices are mobile, so topology changes can be frequent: thus the scatternet formation algorithm should be dynamic. It should allow for dynamic addition and deletion (leaving) of nodes. Further the scatternet should be formed within a reasonable time for any practical use.

In order to maximize the performance of the scatternet, the following goals are desirable: (1) the number of piconets in the scatternet should be minimized, (2) the degree of a node should be minimized, (3) the network diameter should be minimized, (4) node contention which essentially measures how many pair of nodes use this node as an intermediate relay node, should be minimized, (5) multiple paths between any two nodes should exist, (6) the time and message complexity should be minimized, and (7) the scatternet should be self-healing, that is, able to handle devices leaving and joining.

Creating a Bluetooth scatternet with these desirable properties is the focus of this thesis. In this thesis, we propose two distributed algorithms for scatternet formation in Bluetooth networks, *Meshes* and *Cubes*, which are well known interconnection networks in their own right. We believe that these two topologies are scalable to satisfy the requirements of personal area networks. Compared to previous research, our work is distinctive in the following ways. First, the total number of participating devices is limited only by radio interference concerns. Second, while most existing research ignores the routing and connectivity issues, our algorithms take care of those concerns implicitly. We also provide solutions for dynamic environments and show that our scatternet structures elegantly handle those cases. Both mesh and cube scatternets provide lesser diameters for the network which improves the overall throughput of the network. They also reduce the overall node

contention by providing multiple paths between any two nodes in the network.

1.6 Thesis Organization

The rest of the thesis is organized as follows. We provide background information about Bluetooth technology in Chapter 2. The research work on scatternet formation issues, done so far, is discussed in Chapter 3. The Mesh and Cube algorithms for scatternet formation are explained in Chapter 4. We point out various properties of the scatternets such as easier routing, multiple paths between any two nodes, low network diameter, low node contention. We also point out the ability of these networks to handle dynamic environments. The simulation results are presented in Chapter 5. We provide a comparison of the obtained results with those of the other scatternet formation schemes based on the identified performance metrics. We also compare the properties of Mesh and Cube scatternets with each other and provide explanation when one's use should be prioritized over the other. We summarize the work performed in the Chapter 6 providing directions for further research in future work section.

CHAPTER TWO

BLUETOOTH TECHNOLOGY

Bluetooth is a promising wireless technology that enables portable devices to form short-range wireless ad-hoc networks. It operates in the 2.4 GHz free Industrial, Scientific, Medical (ISM) frequency band. Along with Bluetooth, many other technologies such as 802.11, WLAN use this license free band and hence, it is subject to interference. In order to avoid interference from these other devices and technologies, Bluetooth makes use of the Frequency Hopping Spread Spectrum (FHSS) by dividing the ISM band into 79 channels. Every device is given a slot of 625 μ sec (1600 hops/sec) in which it can either transmit or receive data. Since, the frequency hopping sequences are unique for each device and are designed not to have any repetitive patterns over small periods of time, the Bluetooth technology is expected to be very robust.

2.1 Bluetooth Protocol Stack

Figure 2.1 shows the Bluetooth Protocol Stack. There are following layers present in the stack.

1. Bluetooth Radio :

   Bluetooth Radio is the lowest defined layer of the Bluetooth Specification. It defines the requirements of the Bluetooth Transceiver device operating in the 2.4 GHz ISM band. The requirements are defined for following two reasons :

   - Provide compatibility between the radios used in the system.
   - Define the quality of the system.

   The most common Bluetooth Radio has an output power of 0 dBm. These devices are called class 3 devices. There are two more classes; class 2 which has a maximum output power of 4 dBm and class 1 which has a maximum output power of 20 dBm. Channel spacing is 1 MHz and a guard range is used in the top and bottom of the frequency band. Thus, the ISM

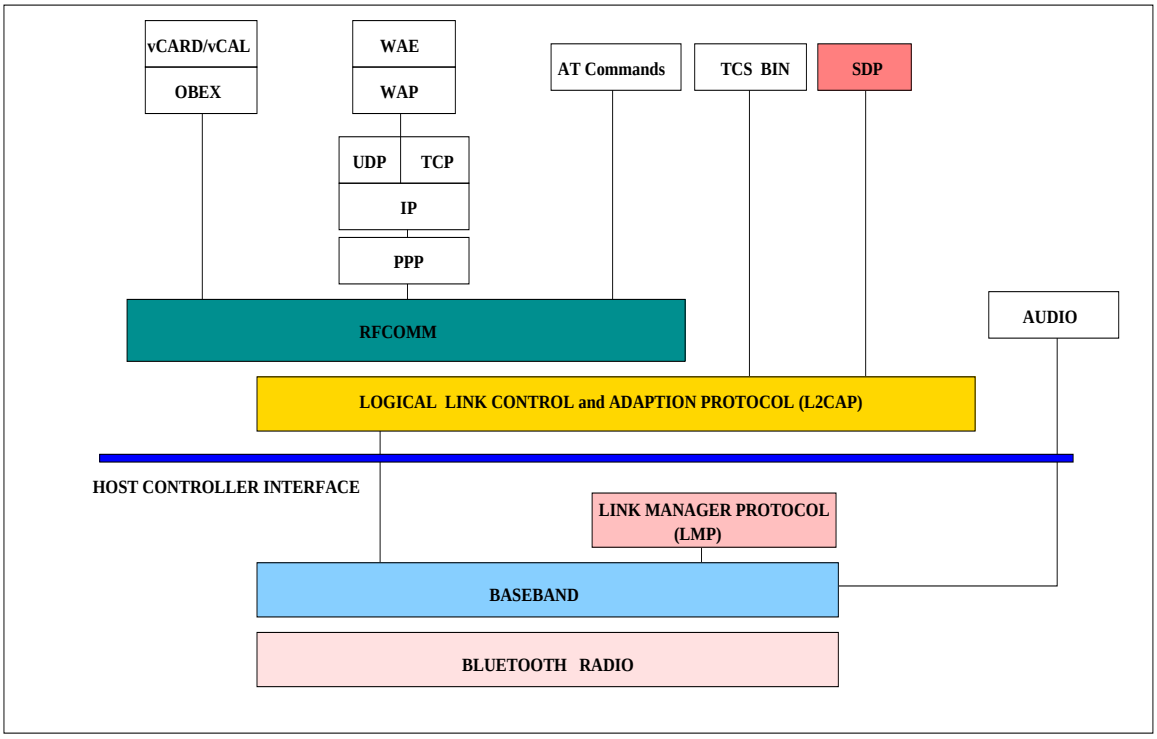


Figure 2.1: Bluetooth Protocol Stack

band is divided into 79 distinct channels. There is a 23 channel radio defined for countries with special radio frequency regulations. The modulation technique used is Gaussian Shift Keying (GFSK).

2. Bluetooth Baseband :

   Bluetooth Baseband (BB) is the physical layer of Bluetooth. It manages physical channels and links apart from other services like error correction, data whitening, hop selection and Bluetooth security. The Baseband layer lies on top of the Bluetooth Radio layer. The Baseband protocol is implemented as a Link Controller, which works with the Link Manager for carrying out link level routines like link connection and power control. The Baseband also manages the asynchronous and synchronous links, handles packets and does paging and inquiry to access and inquire Bluetooth devices in the area. The baseband transceiver applies

a time-division duplex (TDD) scheme.

Each Bluetooth device is assigned a 48 bit IEEE MAC address known as Bluetooth Device Address (BD_ADDR). This is used as a seed in some of the serial bit processing operations and in particular for the derivation of the access code. This MAC address is split into 3 parts BD_ADDR[47:32] Non-significant Address Part (NAP), BD_ADDR[31:24], Upper Address Part (UAP) and BD_ADDR[23:0] as Lower Address Part (LAP).

3. Link Manager Protocol :

   The Link Manager carries out the link setup, authentication, link configuration and other protocols. It discovers other remote LM's and communicates with them via the LMP. To perform its service provider role, the LM uses the services of the underlying Link Controller (LC). The LMP essentially consists of a number of Protocol Data Units (PDUs), which are sent from one device to another, determined by the address (AM_ADDR) in the packet header.

   The LMP also handles security and different low-power modes:

   - *Active*: A device participates in the piconet by listening (in the master-to-slave time slots) for packets containing its own AM_ADDR.
   - *Sniff*: A device acts similar to to an active device. Upon entering sniff mode, the master and slave decide a sniff interval; the time between two time slots where the slave will listen for packets.
   - *Hold*: A device that enters hold mode, does so for a specified *hold* time. During this time ACL packets are not supported but SCO packets can still be transmitted.
   - *Park*: when a slave enters park mode, it gives up its AM_ADDR but remains synchronized to the piconet. It is given a Parked Member Address (PM_ADDR) that the master

uses for un-parking the slave and an Access Request Address (AR_ADDR) that the slave can use to request to be unparked.

4. Host Controller Interface (HCI) :

   The HCI provides a command interface to the baseband controller and link manager and access to hardware status and control registers. Essentially this interface provides a uniform method of accessing the Bluetooth baseband capabilities. The HCI exists across 3 sections, the Host, Transport Layer and the Host Controller.

5. Logical Link Control and Adaption Protocol (L2CAP) :

   The L2CAP is layered over the baseband protocol and resides in the data link layer. L2CAP provides both connection-oriented and connectionless data services to upper layer protocols with protocol multiplexing capability, segmentation and reassembly and receive L2CAP data packets up to 64 KB in length.

6. RFCOMM protocol :

   The RFCOMM protocol provides emulation of serial ports over the L2CAP protocol. The protocol is based on the ETSI standard TS07.10. Only a subset of TS07.10 standard is used, and some adaptations of the protocol are specified in the Bluetooth RFCOMM specification. The RFCOMM provides transparent data stream and control channels over the L2CAP channels.

7. Service Discovery protocol (SDP) :

   The SDP provides a means for the applications to discover which services are available and

to determine the characteristics of those available services. A specific Service Discovery protocol is needed in the Bluetooth environment, as the set of devices that are available changes dynamically based on the RF proximity of the devices in motion, qualitatively different from the service discovery in traditional network based environments. The service discovery protocol defined in the Bluetooth specification is intended to address the unique characteristics of the Bluetooth environment.

SDP uses a server-client model where the server has a list of elements called Service Records. Each Service Record describes the characteristics of a service. There can only be one SDP server in a Bluetooth device; if a device has several services, one SDP server can act on behalf of all of them. Similarly, multiple applications may use a single SDP client to query servers of Service Records. If a device only acts as an SDP client it does not need to have SDP serve. A device may operate as both a server and a client at the same time.

2.2 Piconet Structure

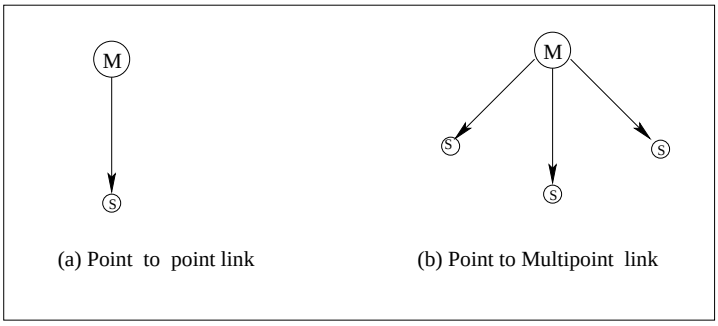


Figure 2.2: Bluetooth Link Types

As noted before, every device is assigned a unique address (BD_ADDR) from which the frequency hopping sequence for that device is generated. Bluetooth supports point-to-point as well as point-to-multipoint connection as shown in figure 2.2.

A group of devices sharing a communication channel is called as the piconet. Piconet is the basic building block for Bluetooth networks. Every piconet with at most *k* (Bluetooth 1.1 Specification, *k* is set to 7) active slaves is led by a Master of the piconet. The piconet might have more slaves which are not active or remain locked in the parked mode. These slaves are synchronized to the Master frequency. The Master node's clock decides the frequency hopping sequence for the piconet. There is no slave to slave communication and all the messages are exchanged between the slave and the master only. The role of a Master and a slave are only logical states; any device can potentially be a Master or a slave.

2.3 Bluetooth Device States

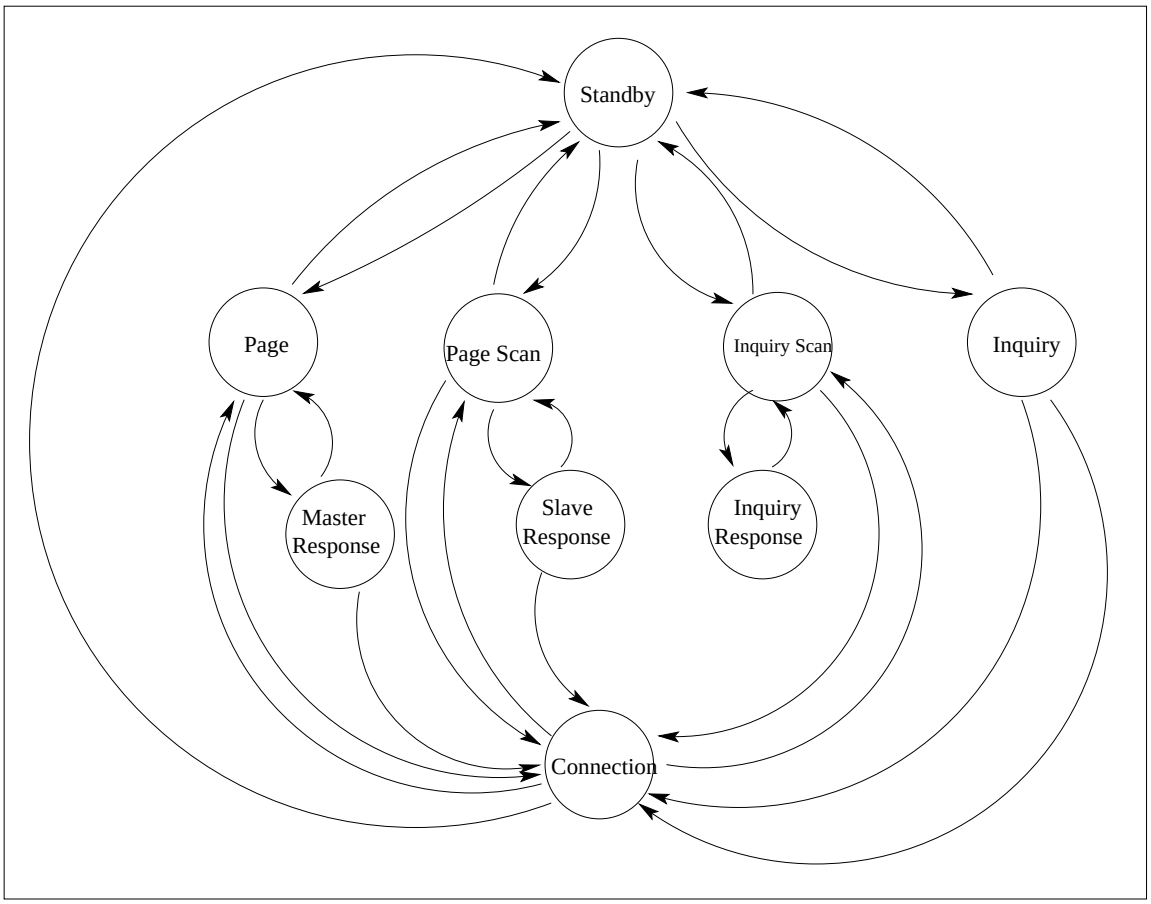


Figure 2.3: Bluetooth State Diagram

At any one time, a Bluetooth device is in one of a number of different states. The important states of Bluetooth devices are shown in Figure 2.3.

1. Standby: In the standby state, the device is inactive, no data is being transferred, and the radio is not switched on. Thus, the device is unable to detect any access codes. This state is used normally to enable low power operation.

2. Inquiry: Inquiry is the process whereby a device will attempt to discover Bluetooth enabled devices in its vicinity. During the inquiry procedure, the inquired upon devices will supply FHS packets to the inquirer. The FHS packets allow the inquirer to build a table of essential information required to make a connection, such as the address and clock.

3. Inquiry Scan: Inquiry Scan is the other half of the Inquiry procedure. Devices enter the inquiry scan state to make themselves available to inquiring devices. They listen for an extended time (this is necessary as they have no knowledge of the timing or frequency hop behaviour of any inquiring devices) for the inquiry packet. When they receive a valid inquiry message, they enter the *inquiry response* substate and respond with the FHS information needed. Both inquiry and inquiry scan states utilize a special hopping sequence (fast for inquiry and slow for inquiry scan) which is designed to reduce the amount of time before a frequency match occurs.

4. Page: To establish a connection, the device which is to become a master is instructed by the application to carry out the paging procedure. The master first enters the page state, where it will transmit paging messages directed at the intended slave device. The slave acknowledges the paging message and the master enters the *master response* substate and responds with FHS packet.

5. Page Scan: A device will typically enter page scan periodically to allow paging devices to establish a connection with it. Once a device in page scan state has successfully received a

paging packet, it will enter the *slave response* substate where it acknowledges the packet and awaits the FHS. On reception of FHS, it updates it own timing before entering the connection state.

6. Connection: On entry to the connection state, the slave switches to the master's clock and thus moves on to the master's frequency hop and timing sequence. Hence, both master and slave can communicate with each other on the established link between them.

2.4 Bluetooth Network Formation

Initially each device is a separate device being the master of its own piconet with no slaves. So, initially, each piconet consists of a single device. The Bluetooth specification requires the devices to discover the neighboring devices so as to form larger piconets consisting of more devices inside a piconet.

The neighbor discovery process in Bluetooth consists of two phases: **INQUIRY** and **PAGE**. In the INQUIRY phase, a master node attempts to discover its neighboring devices and gather necessary information to establish a connection with a subset of neighbors. If the INQUIRY phase is successful, it moves on to the PAGE phase, where the information gathered during the Inquiry phase is used to establish a bi-directional frequency hopping connection.

During the Inquiry phase, one of the two devices is in the INQUIRY state and the other is in the INQUIRY SCAN state. The device in the INQUIRY phase repeatedly alternates between transmitting short ID packets which consists of the Inquiry Access Codes (IAC) and listening to responses. In order to overcome the phase difference between devices in *inquiry* process, the Bluetooth specifications require the devices in INQUIRY to hop at a faster rate than devices in INQUIRY SCAN state. The Frequency Synchronization (FS) delay refers to the time it takes for the INQUIRY device to transmit at the frequency the INQUIRY SCAN device is currently listening on.

Upon receiving such an ID packet, the receiver backs off for an amount of time uniformly

distributed between [0, 1023] time slots. This avoids the contention when the two listeners listen to the same hopping frequency and respond simultaneously. After the random backoff time, it listens for the ID packets at the same frequency as before the backoff. If it receives the second ID packet, it responds to the other node with the FHS (Frequency Hop Synchronization) packet containing information about itself such as

- Address : This helps the master node to derive the DAC of the slave and also page hopping sequence that the node will use to page the master node.
- Clock : This is used by the master to derive the phase of the slave and hence it eliminates the Frequency Synchronization (FS) delay during the page phase.

This INQUIRY procedure leads to an asymmetric knowledge of two neighboring devices. The inquirer identity is not known at the device that received IAC packets. Only the inquirer upon receipt of the FHS is aware of the identity and also the clock of the other node. This enables the inquirer to estimate the frequency hopping sequence used by the other node and thus to invite it to join the piconet as its slave. This invitation is accomplished by means of the paging procedure.

The receiver after sending FHS packet, moves into the PAGE SCAN state. When the master node receives this information, it moves into the PAGE state. With the available information, the sender is able to send the DAC packet on the frequency the slave is listening to in the PAGE SCAN state. By definition, the device which goes into the PAGE mode is called the *master* and the node which goes into PAGE SCAN mode is called as the *slave*. The slave responds to master's DAC packet with another DAC packet. The master responds with an FHS packet which gives slave information about the master's frequency hopping sequence and its phase. Hence, it becomes the slave on that connection link and acknowledges the master with another DAC packet.

Consider the link formation between the two devices U and V. say, V moves into the PAGE mode and U into the PAGE SCAN mode and hence, V should take the role of a Master and U should take the role of a Slave. But it may happen that the device U, which is in PAGE SCAN, is

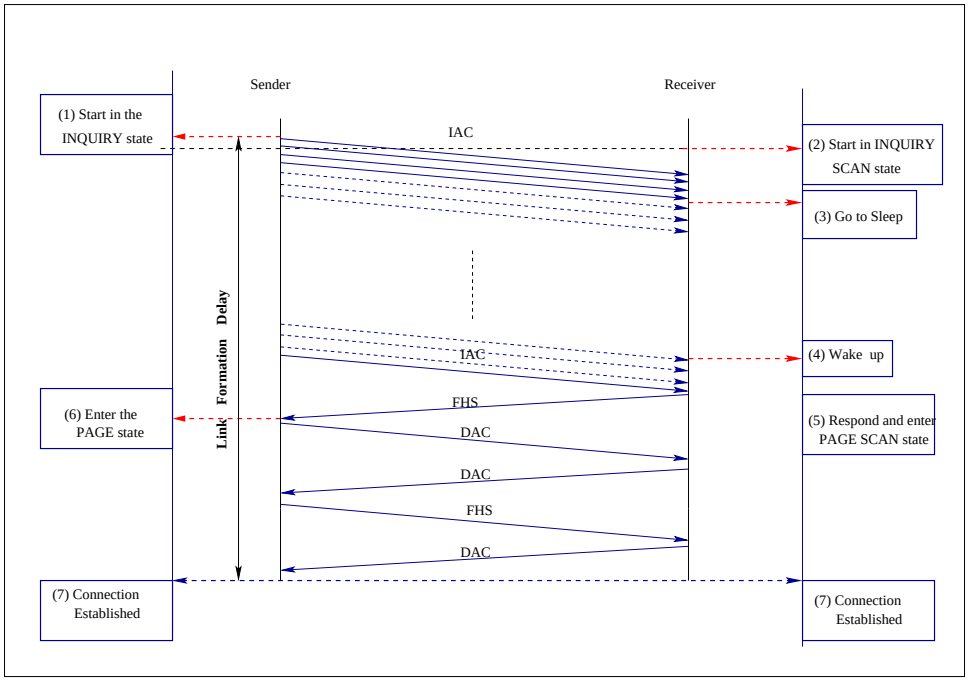


Figure 2.4: Bluetooth Link Formation

already a master of another piconet and that it could have host V as one of its slaves. In this case, once a piconet has been established between V and U, with V as master, the slave U can request a *switch* of roles. This situation is explicitly addressed by the Bluetooth specifications, and it is implemented via exchanging a specific Link Manager Protocol (LMP) packet that instructs the two devices to switch to the frequency hopping sequence of the new master.

2.5 Intra-Piconet Communication

As described earlier, the slave can only talk to the master of the piconet and hence the only data exchange possible is between the master and the slaves. For the intra-piconet communication, Time Division Duplex (TDD) scheme is employed. A slave can respond only in the time slot preceded by the time slot in which it was polled by the master. In the even numbered time slots, there is a master to slave transmission and the odd time slots are for the slaves to respond back to the master. The master determines which slave can send in the next slave to master time slot. The

decision is based on the type of the link and the round-robin scheduling used by the master.

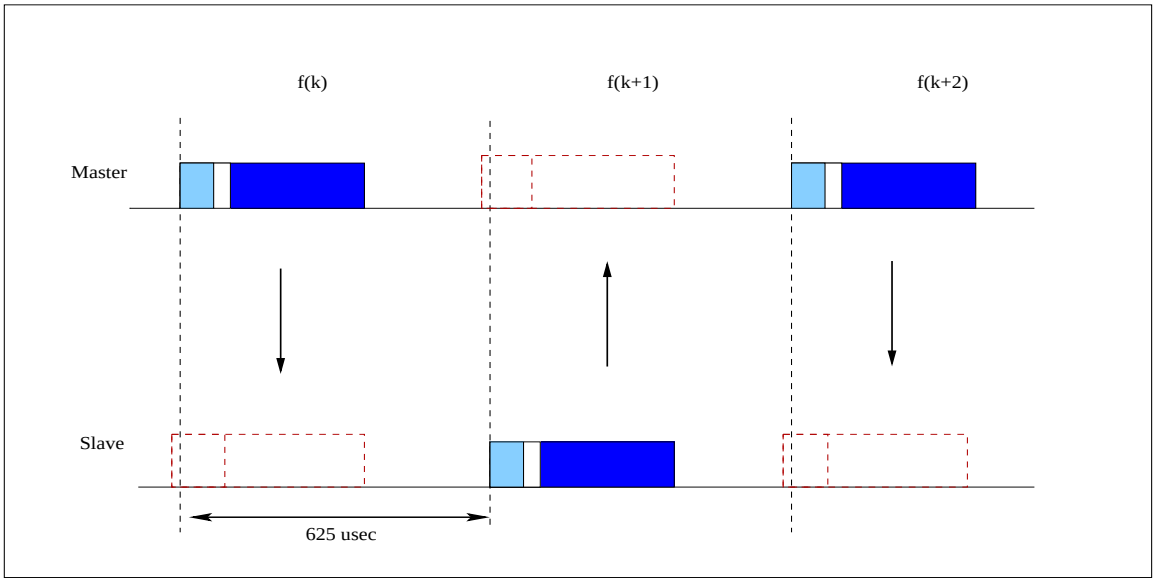


Figure 2.5: Single-slot Packet Communication

The RF hop frequency used for the packet remains fixed for the duration of the packet. Sometimes the master or slave may make use of the packet that lasts for more than one time slot i.e a multi-slot packet. For the single-slot of the packet, the RF hop frequency is derived from the current Bluetooth clock value. For the packets that range for more than one slot, the RF hop frequency is derived from the Bluetooth clock at the start of the first slot.

Between the master and the slave, there can be different types of links established out of which two link types have been defined.

1. Asynchronous Connection-Less (ACL): An ACL link exists between a Master and a Slave as soon as a connection has been established. A Master may have a number of ACL links to a number of different slaves, but only one link can exist between two particular devices. ACL link provides a packet-switched connection where data is exchanged sporadically as and when data is available from higher up the stack. The choice of which slave to transmit to and receive from is up to the Master on a slot basis and so both asynchronous and

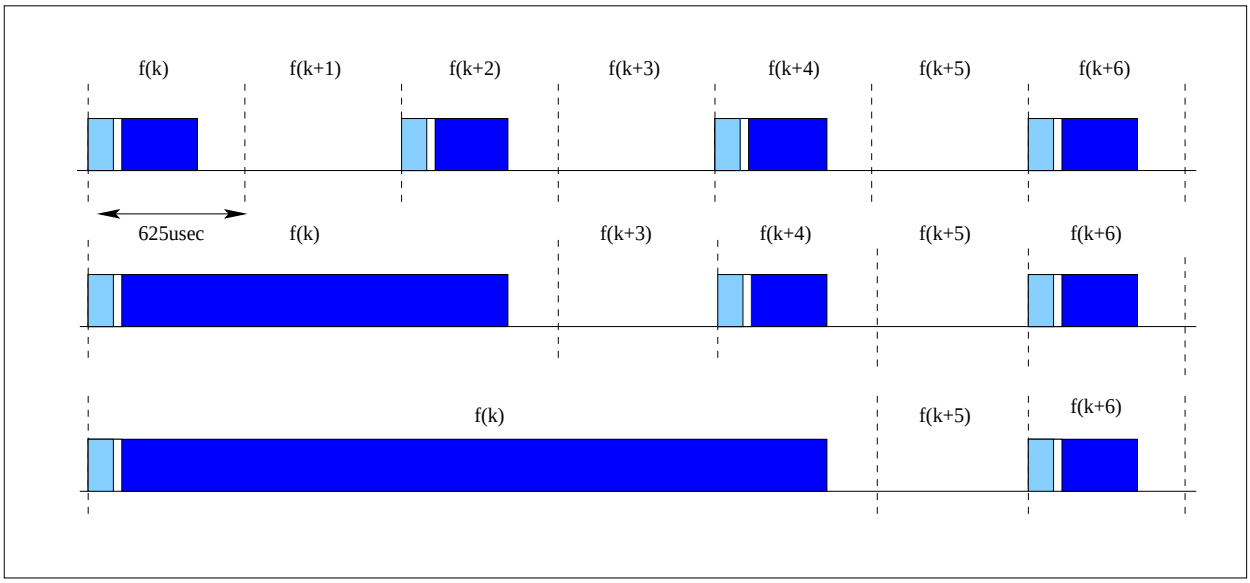


Figure 2.6: Use of Multi-slot packets

isochronous (time-bounded) services are possible. Most ACL packets facilitate error checking and retransmission to assure data integrity.

2. Synchronous Connection-Oriented (SCO): A SCO link is quite different and provides a symmetric link between Master and Slave with reserved channel bandwidth and regular periodic exchange of data in the form of reserved slots. Thus, the SCO link provides a circuit-switched connection where data is regularly exchanged and as such it is intended for use with time-bounded and regular data such as audio. A Master can support up to three SCO links to the same slave or different slaves. A slave can support up to three SCO links from the same Master. Due to the time bounded nature of SCO data, these packets are never retransmitted.

2.5.1 Bluetooth Packet Structure

Every packet consists of an access code, a header and a payload as illustrated in figure 2.5.1. The access code is used to detect the presence of a packet and to address the packet to a specific device. e.g. The slaves detect the presence of a packet by matching the access code against their stored

copy of the Master's access code. The header contains all the information associated with the packet and the link, such as the address of the slave for which the packet is intended. Finally, the payload contains the actual message information if this is a higher layer protocol message such as might be sent from L2CAP or LM, or the data if this is actually the data being passed down the stack.

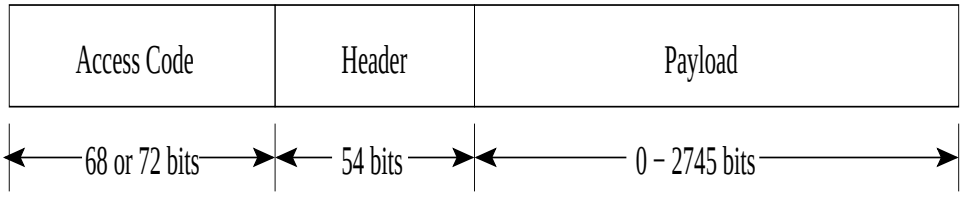


Figure 2.7: Bluetooth Packet Structure

Access Code :

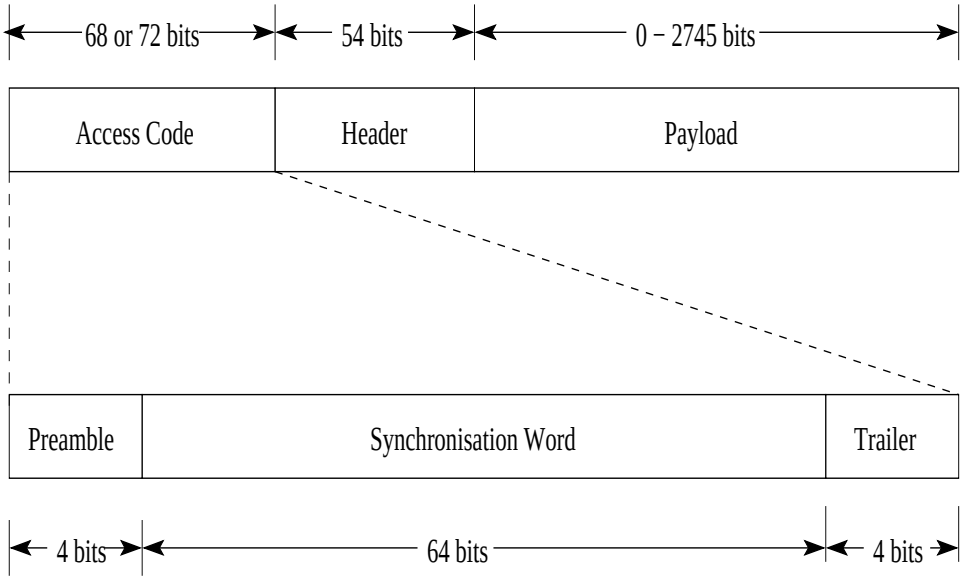


Figure 2.8: Bluetooth Access Code Structure

During a connection when a link is active, the access code (illustrated in figure 2.5.1) identifies the packet as being from or to a specific master. In other modes of operation, a special device address may be used e.g. in inquiry to produce the Inquiry Access Code (IAC).

The first part of access code is a 4-bit preamble which is used to detect the edges of the received data. The preamble is a fixed sequence of either 0101 or 1010, depending upon the value of the first bit of synchronization word so as to form a known 5-bit sequence. The synchronization word is formed from the 24-bit Lower Address Part (LAP) of the Bluetooth device address using an algorithm defined in the Bluetooth specifications. First, the algorithm appends a predefined 6-bit sequence to the LAP to improve autocorrelation properties of the sync word. This is then XOR'd with bits 34 to 63 of a full length, 64 bit PN sequence P[63:0]. The resulting 30 bit sequence is then encoded with a (64,30) BCH block code to yield a 34 bit parity word.

There are four different access codes defined which might be used at any one time as follows :

1. *Channel Access Code (CAC)*: The CAC is derived from the Master's LAP and is used by all devices in that piconet during the exchange of data over a live connection.

2. *Device Access Code (DAC)*: The DAC is derived from a specific device's LAP. It is used when paging a specific device and by the device in Page Scan while listening for paging messages.

3. *General Inquiry Access Code (GIAC)*: The GIAC is used by all devices during the inquiry procedures, as no prior knowledge of anyone's LAP will exist. The GIAC is fixed in the specification as 0x9E8B33.

4. *Dedicated Inquiry Access Code (DIAC)*: There is a range of DIACs reserved by the specification for carrying out inquiry procedures between a specific set of devices only i.e. printers or cellular handsets. These use LAPs in the range 0x9E8B00 - 0x9E8B3F. There have not been any definitions of specific DIACs for specific classes of devices as yet, and so the Generic Access Profile (GAP) specification indicates that only the Limited Inquiry Access Code (LIAC) should be used. This is the only currently defined DIAC and is based on the LAP 0x9E8B00.

Packet Header

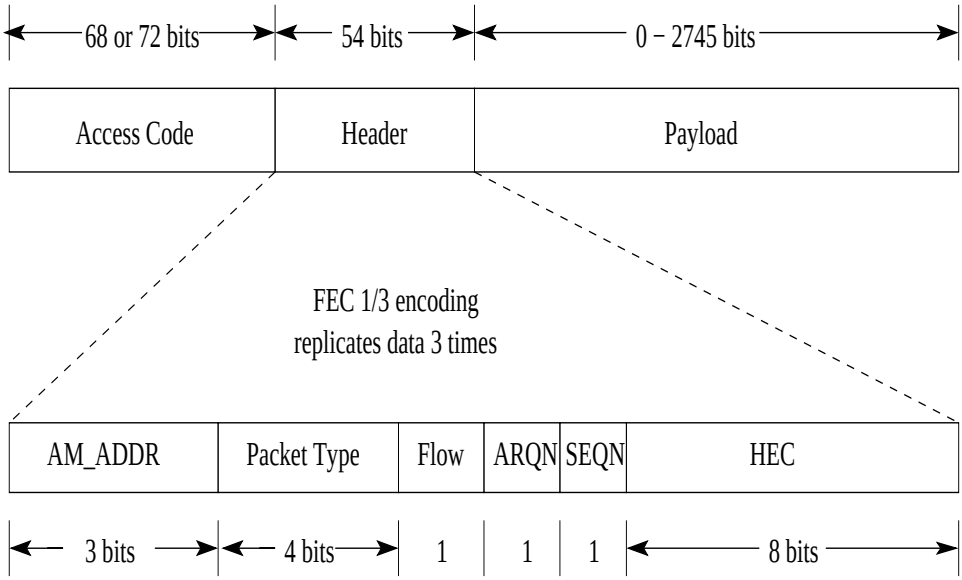


Figure 2.9: Packet Header Structure

The packet header shown in Figure 2.5.1, contains control information associated with the packet. In total, the header contains 18 bits of information, which are protected by a Forward Error Correction (FEC) code of 1/3. This encoding replicates the data three times so that each bit occupies 3 μs. The resulting header is 54μs in length.

During the paging process, the master will assign an Active Member ADDRess **(AM_ADDR)** to the slave. This will then form the connection handle used to address all communications to the slave and for the master to differentiate between responses from different slaves. The 3-bit field is sufficient for seven slaves. An AM_ADDR of zero corresponds to a broadcast packet from the Master to be received by all of its slaves. **Packet Type** defines which type of traffic is carried by this packet (SCO, ACL, NULL or POLL), the type of error correction used for the payload and how many slots the payload will last for. **Flow** flag is asserted by a device when it is unable to receive any more data due to its receive buffer not being emptied. The **ARQN** flag is asserted by a device to indicate that the previous reception was successful validation of the CRC (ACKnowledge

or ACK-condition). If, however, the ARQN was lost due to failure of the returned header, then the original sender of the data will assume a Negative-Acknowledgement (NAK) condition and retransmit the first packet again. Each time a new packet is sent, the **SEQN** flag is toggled. Header Error Correction **HEC** field is simply a CRC function performed on the header represented in octal notation by the generator polynomial 647. It is initialized with either the master or slave UAP or DCI, depending on the packet.

Payload

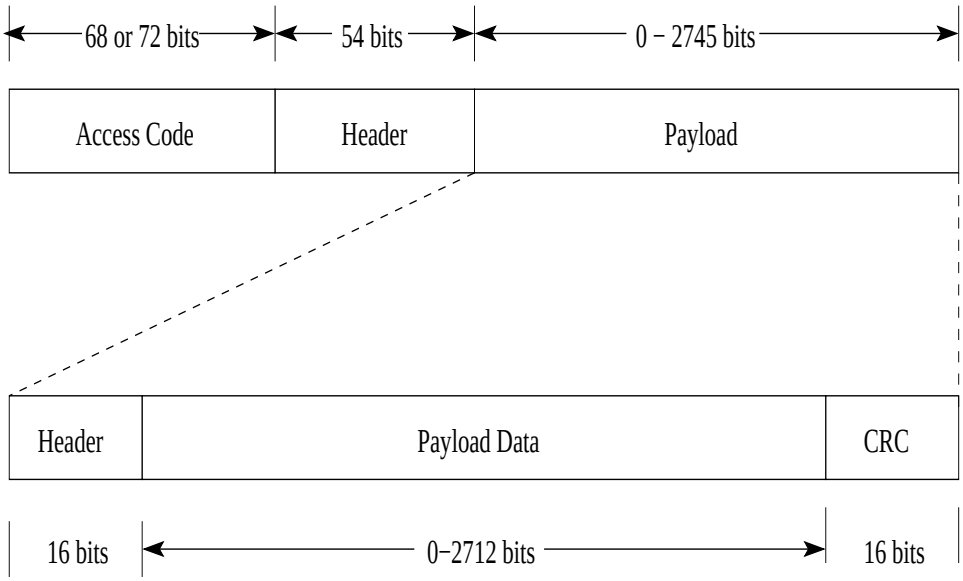


Figure 2.10: ACL Payload Structure

The ACL payload field of all ACL packets is split into three parts; the payload header, the payload data itself and the Cyclic Redundancy Check (CRC) field. The SCO payload field is 240 bits in length. The CRC field is absent, the flow, ARQ and SEQ fields are redundant since flow control and retransmissions do not apply to SCO links.

2.6 Networking

The Bluetooth can support more than two devices to be connected together and hence it allows for the networking of devices. A piconet can connect at the most only 8 devices at a time. But such piconets can be interconnected to form a *scatternet*. The ability to interconnect various piconets is basically because of the ability of any Bluetooth device to be a part of more than one piconet simultaneously. A Bluetooth device can act as a Master in one piconet and slave in the other constituting a Master-Slave (M/S) bridge. Similarly a device can be a slave in both the piconets that it is a part of which gives rise to the Slave-Slave (S/S) bridge. But never a device forms a Master-Master (M/M) bridge since that would mean a device to be in charge of more than one piconet. Bluetooth specifications allow a node to be a part of at most 7 piconets simultaneously. Hence, the device which is shared between two or more piconets, acts as a *gateway* between them. Piconets can thus be interconnected through gateways into a multi-hop ad-hoc network called a *scatternet*.

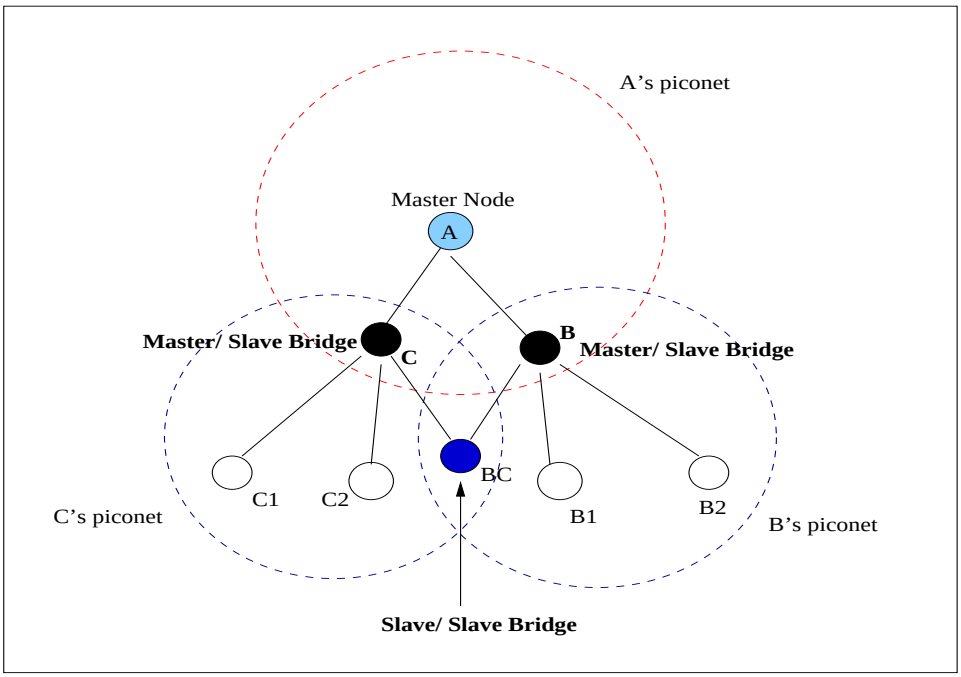


Figure 2.11: An Example Bluetooth Scatternet

Figure 2.6 shows a root device A with two children B and C which play slave role when in piconet of Master A. But B and C act as master nodes in their own piconets. Node B has two slaves B1 and B2 and node C has two slaves C1 and C2. Hence, devices B and C act as M/S bridges. The slave node BC is a part of piconets lead by nodes B and C and hence, BC acts as a gateway between the piconets of B and C. Node BC constitutes a S/S bridge.

Whenever, a device is a part of more than one piconet, it needs to schedule its time among those piconets. It can switch between the piconets as per the piconet scheduling algorithm in use. But whenever a device switches from one piconet to the other, it has to synchronize its clock with the clock of the master of that piconet.

2.7 Scheduling

Bluetooth scheduling can be of two types :

- Intra-piconet scheduling : Since there are more than one slave possible in a single piconet, the master has to find a way so that each slave gets a fair chance to communicate. Hence, for this purpose, the policy used is that of “polling”. The master decides which slave to poll and hence, the polled slave gets a chance to communicate with the master.

- Inter-piconet scheduling : The problem of scheduling becomes harder to manage when we consider more than one interconnected piconets, i.e., a scatternet. This scheduling mechanism would affect the throughput of the whole network and hence it is necessary to find a perfect inter-piconet scheduling mechanism. Since, the node which switches from one piconet to the other, has to synchronize with the master of the new piconet, for a certain time period, (called as the “switching time”), it remains inactive by entering the power save mode. Along with the switching time, the node also has to wait inside that piconet before it can start communicating. Hence, the total overhead for the piconet switch is the total time the node remains inactive. The scheduling algorithm should be designed so as to minimize

this overhead.

2.8 Current problems

The current open problems from the perspective of networking of Bluetooth devices are the Scatternet Formation and the Inter-Piconet Scheduling.

- Scatternet Formation :

  The Bluetooth specification enables the formation of a larger network from many nodes but it does not specify an exact method for scatternet formation. There are many aspects which must be taken care to form a scatternet in order to maximize the throughput. One of the most important factors that affect the performance is the network topology. Since, every node can potentially act as a both master and a slave, we need to choose nodes as masters or as slaves. The network topology is responsible for the ability to handle traffic, the network diameter, routing, etc., which are major factors affecting efficiency. Several scatternet formation techniques have been proposed. There are various topologies that have been suggested including the *loop* scatternet, *tree* scatternet, etc. Due to the various trade-offs encountered, a scatternet formation algorithm can satisfy only a few of these desirable properties.

- Inter-Piconet Scheduling :

  The *scatternet* is possible to build due to ability of the various Bluetooth devices to participate in more than one piconet at a time. As explained earlier, these devices which are shared between two or more piconets have to switch between the piconets to allow communication. But, since switching from one piconet to the other, asks for a lot of overhead in terms of switching and synchronizing time required, it is desirable to keep the piconet switches to as minimum as possible. But as the switching between various piconets decreases, the routing

takes more time. And the routing delay may increase. Hence, we face with trade-off situation. Moreover, the inter-piconet scheduling would in some respect also depend upon the scatternet topology.

CHAPTER THREE

RELATED WORK

The problem of scatternet formation for a Bluetooth network has been studied from many angles. The proposed algorithms can be roughly divided into two broad classes : those that assume all the devices are within transmission range of each other and aim to provide an optimal topology with respect to certain performance metrics [21], [20], [15], [26], [28],[6], [30], [13]; and those that aim to form a connected network when all the devices are not necessarily one hop away from one another [19], [24], [18], [29]. When all of the devices are within the transmission range of each other, a scatternet is expected to provide connectivity plus additional capabilities. However, for out of range devices, connectivity is the single most important property.

3.1 Scatternets for In-Range Devices

The initial attempts include a scatternet formation algorithm given by Salonidis *et al* [21] called *Bluetooth Topology Construction Algorithm* (BTCP). The algorithm consists of three phases. The first phase is called as *Coordinator Election* which selects one leader out of the given set of nodes. For the leader election process, each node maintains a variable called *votes* initialized to 1. When two nodes get in touch with each other, they exchange their values of *votes*. The node with higher value of *votes* is declared the winner. Ties are broken using the ID of the nodes. The winner node increments its *votes* variable by the *votes* of the loser node. The loser node then hands over all the FHS packets it had won along with its own FHS payload packet. The FHS payload packet essentially contains the address and clock information of the node. This way, after *(n-1)* confrontations, the chosen leader has the FHS payload packets of all the other devices in the network. Hence, the leader node gets a global view of the network. In the second phase, the leader assigns a role to each device that it is supposed to play in the final scatternet. The final phase includes the actual connections between the nodes. Due to the complete connectivity of the

scatternet, every piconet leader is one hop away from every other piconet leader and as a result, the scatternet formed, supports a maximum of 36 nodes and is not scalable to a higher number of nodes.

Ramachandran et al [20] redefine the problem of Scatternet formation as a problem of *clustering* and propose a new randomized clustering algorithm which can be directly applied to Bluetooth networks. They view the independent piconets as separate clusters. The aim of the algorithm is formation of a single cluster i.e. a scatternet. The algorithm consists of two phases. The first phase determines the role to be played by each node. The number of masters are chosen to be close to ideal number of upper bound of $\lceil \frac{N}{S+1} \rceil$ where N is number of devices and S is the maximum cluster size. However, they do not address the problem of determining bridges between piconets. Hence, the resulting scatternet is not a connected one violating the basic property of a scatternet that of connectivity.

Law and Siu [15] propose a single phase fully decentralized scatternet formation algorithm that minimizes the number of piconets and the degree of a node in the network. The devices are partitioned into components during algorithm execution. Each component is led by the master that chooses either to call SEEK or ask an unshared slave (or itself) to call SCAN at the beginning of every round. A master calling SEEK attempts to connect to nodes performing SCAN in other components to form bigger components. The resulting scatternet is a tree which has minimal number of bridge nodes, reducing the synchronization delay and effectively the overall end-to-end formation time.

Tan, Miu, Guttag and Balkrishnan [26] suggest a tree structure suitable for dynamic environments. TSF is a scatternet which at any point of time is a forest consisting of tree components. Each node transitions between two states: FORM and COMM. Devices in the FORM state attempt to establish connections with other neighboring devices that are part of other components. Devices in the COMM state are involved in the data communications with other nodes in the same tree component. A root device may only connect to another root device and non-root devices can

acquire single devices only. This enforces and maintains the tree structure of the scatternet. The tree scatternet suggested, is a self healing scatternet, accommodating any node leave and join and reorganizing to retain the tree structure.

Baatz *et al* [6] have proposed a scatternet formation algorithm based on 1-factors. They model the scatternet as a directed graph with nodes as the Bluetooth devices and graph links as the master-slave links. Different scatternet topologies can be obtained by different combinations of the 1-factors of the network graph. 1-factors are chosen with the aim of perfect 1-factorization. Once the 1-factors are chosen, the directions to the edges are assigned in a consistent manner. The distribution of master and slave roles is done equally among the Bluetooth devices since the traffic pattern in the network is unknown. However, no implementation was reported and no simulation studies were performed.

Chong and Chaing [9] proposed Bluering, a Bluetooth scatternet with ring structure. All nodes are interconnected together to form a ring. Each node belongs to two piconets, acts as a Master-slave relay, and has exactly two links in total. Initially, all nodes are independent lines of length 1. These lines are interconnected together to form a single line of devices. Finally, The line is then closed by connecting the piconets forming the end-points of the line. This way we get a ring scatternet. The *BlueRing* has benefits in terms of reliability, ease of packet routing and scheduling. The BlueRing scatternet provides two disjoint paths between any two devices in the network. Hence, it increases the reliability of the scatternet, however, the scatternet suffers from partitioning of the network due to a single node join.

Wang *et al* [28] proposed a scatternet formation scheme called *Bluenet*. Bluenet is composed of three phases. In phase one, each node collects information about its neighbors within the radio range, and then randomly invites up to 7 neighbors to become his slaves. This keeps the number of slaves in every piconet limited to 7. At the end of phase 1, a few piconets would be formed along with a few unconnected separate nodes. In phase two, these separate Bluetooth nodes begin to page all their neighbors, attempting to become a part of a piconet. A node which becomes a part of more

than one piconet becomes a bridge node between those two piconets. The third phase is where the piconets get connected to form a single scatternet. At this point the master of each piconet instructs their slaves to form outgoing links. In spite of its simplicity, Bluenet cannot always guarantee a connected network be established.

Zhang, Hou and Sha [30] propose formation of *loop scatternets* for Bluetooth networks. The authors formalize the notion of maximum node contention. The algorithm consists of two phases. The first phase groups the individual devices into piconets. After the piconets are formed, they are connected together to form a ring scatternet using specified procedure. At the end of phase 1, each piconet has a maximum of *k* - 1 slaves and hence can accept one more slave. The second phase called as an *optimization phase* where a slave from each piconet is explicitly selected and shared with another piconet which is $\sqrt{P}$ hops away, to reduce the network diameter and maximum node contention. Simulation results provide comparisons with tree scatternet proposed by Law *et al* [15].

Kawamoto *et al* [13] propose formation of an on-demand scatternet suitable for dynamic environments. The algorithm consists of two phases. First phase creates a *Control Scatternet* to support dynamic topology changes and route determination. The design goals include minimizing the number of piconets. The formation of control scatternet is divided into three periods during which (1)the nodes discover their neighbors with whom they exchange status information (2)the nodes congregate into piconets by electing a master among the nodes within the radio range of each others, and (3)the piconets are connected to form the *Control Scatternet*. The second phase creates a separate *on-demand* Scatternet. The scatternet is called as *on-demand* in the sense that a route between any two nodes is established whenever either one of those nodes, wants to initiate a data communication with another node. This scatternet is torn down once the data communication is complete. They claim to achieve high aggregate throughput at the expense of slightly higher connection delays. Hence, the scatternet creation process is done on-demand and routes between any two are created only when they want to start data communication between each other.

Zhen *et al* [31] present a two-phase scatternet formation algorithm: in the first phase, all the devices are self-organized into "blue-star islands" that consist of piconets interconnected by a joint slave node. In the second phase, the "blue-star islands" are bridged by means of a routing trigger to form a fully connected network. The routing trigger is implemented by means of Route Request(RREQ) messages, that have been used in on-demand routing protocols. However, for higher number of nodes, the network is expected to get flooded with the route request messages increasing the message complexity of the network.

Sun *et al* [25] provide an algorithm to embed b-trees into a scatternet which enables self-routing in a network. The algorithm is distributed and asynchronous and requires only a fix-sized message header and no routing table. Various operations for adding a node or a different bluetree which joins the network are provided. The protocol also provides ways in which the bluetree can be balanced.

Recent research shows efforts of using evolutionary approaches for the scatternet formation. Sreenivas and Ali [23] provide an *evolutionary algorithm* consisting of two phases. The initial phase is that of role determination and the second phase is that of connection establishment. The role for each node in the scatternet is decided based upon a random genetic algorithm. A genetic algorithm selects random groups of node: these groups constitute the initial population. Each group corresponds to a combination of masters, slaves and bridge nodes. The fitness of each group is evaluated based on the number of piconets(or master nodes) in the network. The algorithm is applied repeatedly until a group has number of piconets equal or within a limit of 2 greater than the universal lower bound of $\lceil \frac{n-1}{k} \rceil$. This group is chosen to be the best solution and the algorithm terminates. In the second phase, each master pages its slaves and connection establishment takes place. Since, there is no regular structure to the scatternet formed, it is difficult to predict a definite routing scheme for such a scatternet. A regular structure has an inbuilt routing and scheduling schemes which reduce the overall communication delay. Moreover, the time required to build the scatternet is very large as the number of nodes increase.

Another approach by Pamuk and Karasan [17] includes creation of SF-DeviL, a distributed scatternet formation algorithm which considers energy efficiency as one of the metrics for scatternet formation. Each device is assigned a device grade and the master and slave roles are assigned to those devices which are best fits for the same. They choose a device having a high battery capacity as a master or a bridge node hoping to get a more stable and power efficient scatternet. At the end, a resulting spanning tree has devices with the largest device grade as root and the smallest device grade devices as leaves. The simulation results however indicate that the scatternet formation delay is very high for the proposed scheme.

Barriere *et al* [7] propose another distributed scatternet formation algorithm called projective scatternet formation. The construction of the so called *projective scatternets* is dynamic in the sense that nodes can join and leave the network at their convenience. For fixed constant degree of nodes, the resulting diameter is polylogarithmic in the size of the network and the connectivity of the masters is high. They also provide a routing protocol for the suggested topology which is based on simple labeling of the devices participating in the scatternet. The algorithm reacts to a node leave or join request with $O(log^2 n log^2 log n)$ time in local computation where n is number of nodes and the message complexity is found to be $O(log^4 n log^4 log n)$ bits.

Song *et al* propose formation of dBBlue scatternet. The backbone of the scatternet is well known *de Bruijn graph*. The algorithm is claimed to have achieved a diameter of O(log *n*) and the congestion at every node is suggested to be O(log *n*/*n*). The scatternet is updated dynamically using a *token-based* updating scheme in order to handle node leaving and joining; the cost of updating is claimed to be better than [7]. For the purpose of easier routing, the authors propose a MAC address assignment for piconets. However, no implementation details or simulation results are provided.

3.2 Scatternets for Out-of-Range Devices

Algorithms proposed in this category [19], [24], [18], [29] attempt to form a connected network when all the devices are not necessarily one hop away from each other.

Petrioli, Basagni, Chlamtac [19] propose a mesh-like star topology for Bluetooth scatternets. The algorithm consists of three phases with an initial device discovery phase followed by the second phase for formation of piconets called as *Bluestars*. For the device discovery phase, the devices are required to gather symmetric knowledge of their neighbors which is done by establishing temporary piconets between them. Based on a locally and dynamically computed weight (a number that expresses how suitable a node is for becoming a master), each node decides whether it is going to be a master or a slave. These devices with their appropriate roles, connect with each other to create *Bluestars*. In the final phase, all such bluestars are connected by selection of a gateway between a pair of piconets to create a *BlueConstellation*. However, there is no bound kept on the number of slaves present in a piconet. It is desirable to keep the number of slaves in a piconet confined to k, reducing the additional overhead of parking and unparking.

Stojmenovic [24] provides a *dominating set based* protocol for scatternet formation, The protocol consists of three phases. In phase 1, all the neighbors within the transmission range are discovered and sparse connected graphs are built based on *unit graph*. In phase 2, Yao graph is constructed to keep the number of slaves in a piconet to a bounded number (k). In the final phase, each node is assigned either a master or slave role based on some key comparisons. However, these techniques require each node to be aware of its geographic location which is possible only through certain additional hardware support such as GPS receivers. This increases the cost of the Bluetooth devices.

Petrioli and Basagni [18] propose a degree-constrained scatternet formation algorithm called as *Bluemesh*. The algorithm consists of two phases. The initial phase is that of topology discovery followed by second phase of scatternet formation. The first phase requires each node to discover both its one-hop and two-hop neighbors. The two hop neighbor discovery is suggested using the temporary setup of piconet between *every pair* of neighboring nodes. Due to asymmetric nature of the Bluetooth discovery process, discovery of one-hop neighbors itself is a time consuming process. Hence, setting up the temporary piconets and exchange of neighbor information between

every pair of neighboring nodes is expected to take more time as number of nodes increase significantly.

Similarly, Zaruba et al [29] propose two protocols for scatternet formation. In the first protocol, a designated node called *blueroot* builds a spanning tree rooted at itself. A parent-child relation in the tree represents a master-slave relation in the resulting scatternet. The second protocol(BGB) is composed of two phases. In phase one, several devices are selected to build a "bluetree" rooted at each of them. In phase two, each subtree is mapped to a node in a virtual graph and the first protocol is executed on a virtual graph. Bluetrees limit the number of slaves to *k* and also claim to have *Blueroutes* which are fairly close to the minimum shortest path available.

Dong and Wu [10] present Three Bluetree Formations for constructing efficient scatternets. Their algorithm uses the neighbor's neighbor set and/or neighbor's location to construct the Bluetree [29] to efficiently balance two conflicting goals between the number of piconets and the average shortest path. They provide three modifications to the Bluetree algorithm provided [29]. In each of the modifications, the first phase of the algorithm remains the same. Modification 1 requires the knowledge of neighbors and neighbor's neighbors. Modification 2 uses the knowledge of relative distance between the two nodes which can be determined using the relative signal strength. Modification 3 needs the actual geometric positions of the neighbors found using GPS. A simulation study is provided proves that these modification help reduce the number of piconets as well as the average shortest path ratio in comparison with BGB.

Ghosh *et al* [11] provide BTSpin as a single phase distributed scatternet formation protocol. A spin is defined as an occurrence of an INQUIRY mode followed by an INQUIRY SCAN mode. The master in each piconet schedules each of its slaves for a spin in a round robin fashion. After one round, the master goes for a spin. The algorithm provides rules for connecting separate components into a single component using a "mesh-based approach"; however, their algorithm creates structure similar to a "tree". The algorithm also provides for dynamic node leave and join which is taken care of using a *backup gateway*.

Miklos *et al* [16] present performance aspects for Bluetooth scatternet formation. They identify several scatternet parameters that can be used as performance metrics. Some of the important scatternet requirements are enlisted below :

1. Scatternet Formation Delay: The scatternet formation delay measures the total time required to form a complete scatternet. This delay should be kept as low as possible. It is desired to connect the devices together as soon as possible so as to start the communication between devices.

2. Number of Piconets: Bluetooth specification defines 79 different channels. Due to the limited number of channels, as the number of piconets increase, the probability of collisions also increases, adding to the end-to-end packet delay and decreasing throughput. It also adds to the scheduling complexity since many devices may transmit simultaneously through some common piconets. Hence, in order to to increase the throughput by reducing the collisions, the number of piconets in a particular network should be maintained to minimum.

3. Degree of a Node: The degree of a node specifies the number of piconets it belongs to. As the degree increases, the device becomes shared between more and more piconets. This adds to the piconet switching and synchronization delay, resulting in communication bottlenecks. So, the degree of a node should be minimized.

4. Network Diameter: The network diameter is the maximum number of hops between any pair of devices in the network. It essentially measures the largest distance between any two nodes in the network. Larger the distance, more time it takes for packets to reach from one node to another. This results in higher communication delays. Since, low diameter leads to end-to-end delays, we need to minimize the network diameter.

5. Maximum Node Contention: Node contention measures how many node pairs use a particular node as a relay node in their communication. If a node has high contention, it becomes

a performance bottleneck. Along with the node being overloaded, it consumes large amount of battery power from such a bridge node. Hence, maximum node contention should be minimized.

6. Multiple Paths: Having multiple paths between any pair of nodes, increases the connectivity of the network. It also reduces the possibility of a node on a particular path having high contention. It avoids congestion in the network due to which overall throughput of the network increases. Moreover, leaving of a single node does not partition whole network. Providing multiple paths asks for higher connectivity between nodes. The connectivity can be improved through the use of higher number of bridge nodes which also increases the number of piconets in the network. Hence, we can see that the connectivity and the number of piconets are mutually exclusive properties.

7. Handling Dynamic Environments: The nodes in the scatternet can be mobile. Hence, they may join or leave the scatternet at any time. The scatternet should handle such dynamic movement of nodes with the least cost for updating the scatternet. Leaving of a node should be implemented without causing the scatternet to be partitioned. Special efforts are needed to handle and overcome network partitioning. That puts an overhead of reorganizing the scatternet. Similarly the scatternet should allow for new nodes joining in without having to reformulate the scatternet.

In addition to these fundamental properties expected out of the scatternets, there are several other secondary properties that have been identified. Scatternets need to be evaluated based on these metrics. However, we face a number of trade-off situations while satisfying these properties, since some of the properties contradict each other.

Our approach falls in the first category since we assume all the devices are in transmission range of each other. We observe that most existing scatternet formation algorithms in the first category are trees. It has been proved [30] that tree scatternets have at least one bottleneck node

under the uniform traffic model. Along with the bottlenecks, it also suffers from disadvantages such as potentially higher network diameter, higher contention, easy partitioning of the network, single path between any two nodes. Hence, it is not a suitable structure for Bluetooth scatternet formation purposes. We propose mesh and cube structures for scatternet formation purposes which not only avoid any bottlenecks in the networks but also provide lesser diameters, high connectivity, lesser node contentions and multiple paths between any pair of nodes.

CHAPTER FOUR

MESH AND CUBE SCATTERNET FORMATION

4.1 2-Dimensional Mesh Scatternet Formation

We propose mesh as a scatternet topology for Bluetooth devices, because of its many desirable properties such as symmetry, connectivity, low fixed degree etc. Mesh structure is a very well known interconnection network. A mesh's redundant message paths provide an inherent fail-safe reliability that applications demand. A typical embedded requirement is achieving maximum energy efficiency for battery-powered applications. Mesh architecture helps to meet this challenge by reducing total transmitter output power for a given network span thereby lowering the overall power drain. Mesh also provides loop-free, state-less, memory and processor efficient routing algorithms which make it suitable for a limited computation platform. All these properties are highly desirable for a Bluetooth network. For any two points in a (non-trivial) mesh, there are always more than one path available. Hence, it does not suffer from single connectivity. Moreover, the contention at any node is lesser when compared with any other network topology. It also prevents the network naturally from being congested. Mesh structure is highly scalable and does not have a limit on the number of nodes in the network. Hence, we find mesh structure highly suitable for Bluetooth scatternet and so, aim to build a mesh structured Bluetooth scatternet.

Our algorithm consists of three conceptual phases. The first phase is dedicated to grouping the nodes into independent piconets. Here, all the independent devices are connected into piconets each with a maximum of k - 2 slaves. In phase 2, all such piconets are connected together into larger and larger meshes. By the end of the second phase, meshes of piconets are formed, with each piconet as a node in the mesh. The second phase does not guarantee generation of a single mesh, in which case, we may need a third phase to merge all the smaller meshes into a single mesh in order to achieve connectivity and retain the mesh structure.

4.1.1 Phase 1: Piconet Formation

Purpose:

The purpose of phase 1 is to connect the independent set of devices into piconets. All the devices start execution of phase 1 and through *device discovery* process specified in Bluetooth specifications 1.1, search for their neighbors. Initially, every device is a component of size 1. We aim at creating bigger components by creating master-slave links between devices.

Implementation:

Initially, we assume a set of *N* isolated devices that are within transmission range of each other. At this point of time, each device is a leader of the piconet consisting of only itself. It remains in phase 1 until either it acquires *k* - 2 slaves or when it decides to move on to the next phase due to the time constraint we have imposed. This local constraint is based on the time spent in phase 1 and the slaves acquired in that time by that node.

```
Procedure MakePico(leader u, leader v)
{
    if (Slave(u) + Slave(v) < k − 2 )
    {
        Make all slaves of v (including v) slaves of leader u
        u remains as the leader and v retires.
        if Slave(u) = k − 2 then u moves to phase 2.
    }
    else
    {
        u an v remain as independent leaders of their own piconets.
        Both u and v restart the process of finding neighbors.
    }
}
```

Figure 4.1: Procedure MakePico

In phase 1, a leader calls procedure SEEK with probability *p* and calls the procedure SCAN with probability 1 - *p*. When a leader executes procedure SEEK, it goes in the INQUIRY state and tries to acquire one or more slaves. On the other hand, when a leader executes procedure SCAN,

it enters the INQUIRY SCAN state and waits to be contacted by another leader. A successful execution path includes two leaders in opposite modes contacting each other after which they enter PAGE (from INQUIRY) and PAGE SCAN (from INQUIRY SCAN) states and create a connection between them by calling the procedure MakePico.

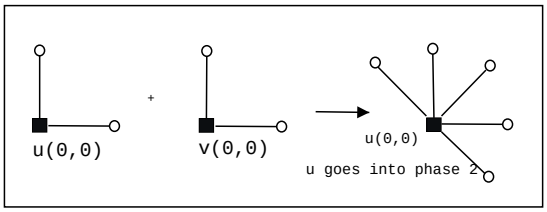


Figure 4.2: Procedure MakePico Example

Procedure MakePico(node *u*, node *v*) is called by two leaders who successfully contact each other in Phase 1 and are in opposite modes of INQUIRY and INQUIRY SCAN. The leaders of two independent piconets decide to merge into a single piconet in phase 1, only if the total number of slaves of *u* and *v* together including *v*, is less than *k* - 2. This restriction leaves space for 2 more slaves in every piconet which are acquired in the following phases. If this condition is satisfied, *u* and *v* merge into a single piconet led by *u* as shown in Figure 4.2. node *v* retires from its duty as a leader and becomes a slave in the piconet led by node *u*. However, if the condition is not satisfied, then they decide not to merge in Phase 1 and remain as two separate piconets. Both the leaders in this case, restart the process of finding neighbors.

Termination of Phase 1:

A leader is forced to move to the next phase whenever it has (*k* - 2) slaves in its piconet. But, this condition solely is not enough to move all nodes into phase 2 since it would take an inordinate amount of time for all the nodes to acquire (*k* - 2) slaves. It is expected that if a node spends more time in Phase 1, it has a better chance to acquire more slaves. But this is also expected to delay the whole process of scatternet formation. Hence, it is worthwhile for a leader with fewer (0 to *k* - 4)

slaves to spend more time in phase 1 trying to acquire more and more slaves, whereas leaders with more (k - 3 and k - 2) slaves spend relatively lesser time in phase 1. Hence, the decision to move from phase 1 to phase 2 depends upon the time spent in phase 1 and the number of slaves acquired in that time.

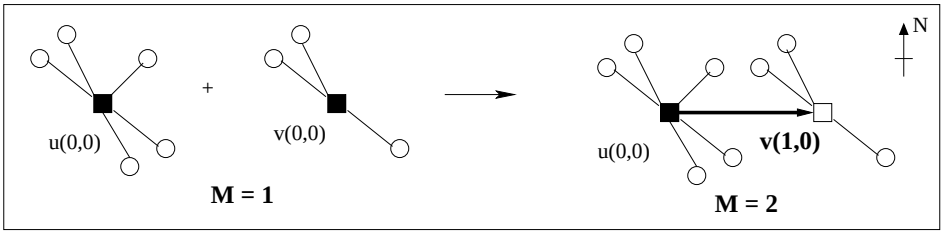


Figure 4.3: Procedure MergeEvenMesh - M = 1

4.1.2 Phase 2: Mesh Growth

Purpose:

The main aim of this phase is to combine independent piconets together to form bigger component meshes. Every piconet leader at the end of phase 1 will have at most (k - 2) slaves under it. Using the available additional space for two slaves, we can interconnect two piconets into a single component. By connecting such smaller sized components together into meshes, we can obtain a single mesh consisting of all the nodes in the network. This phase focuses on growing the scatternet and reducing the number of independent components in the network.

Implementation:

Similar to Phase 1, a leader will randomly choose to be in INQUIRY or INQUIRY SCAN. If successful, procedure CONNECT is called. Each leader is aware of the number of components under it given by M. Initially every leader has just one component, i.e., its own piconet. As a leader spends more time in phase 2, it is expected to gain control of more components, hence increasing the value of M. The decision to merge and the way in which merging can happen depends upon the value of M. For ease of mesh formulation, we assign the first bridge node acquired by a leader

as its EAST (x-direction) slave and the second bridge node as NORTH (y-direction) slave. It also keeps track of all these device addresses in the EAST and NORTH directions.

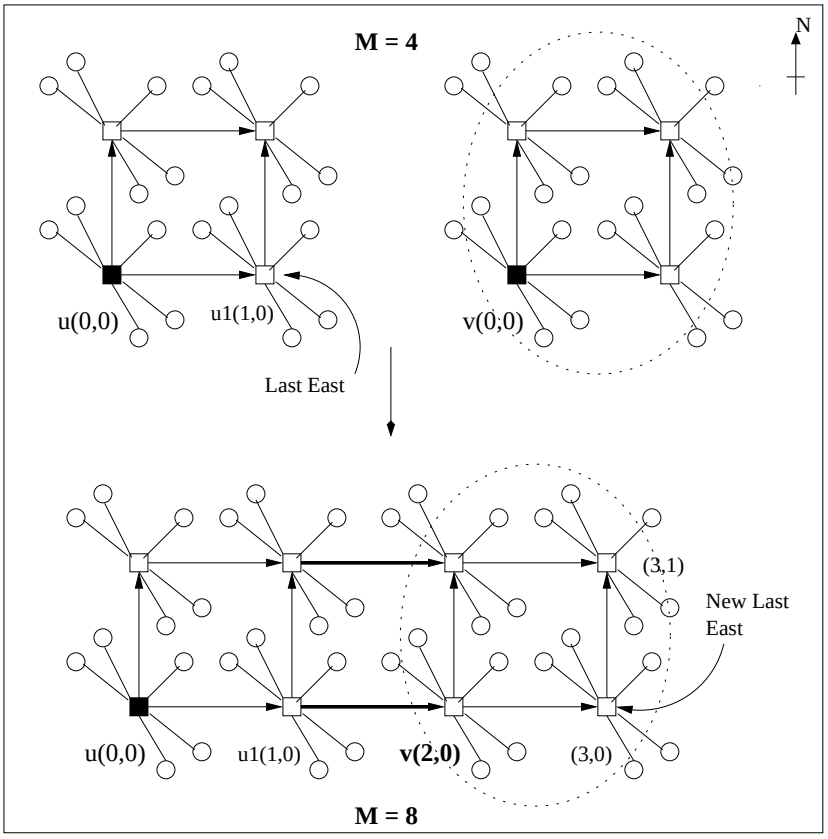


Figure 4.4: Procedure MergeEvenMesh - M $>$ 1

When leaders of two independent piconets contact each other, they first compare their values of *M*. When both leaders have equal values of M, then if M is an EVEN power of 2 (i.e. M = 1, 4, 16 etc.), procedure MergeEvenMesh(*u*,*v*) is called, else if it is an ODD power (i.e. M = 2, 8, 32 etc.), procedure MergeOddMesh(*u*, *v*) is called. If both leaders have unequal values of M, then the procedure MergeUnequal(*u*, *v*) is called.

As explained, we essentially follow the *recursive doubling* approach for our mesh creation. We expect the same sized meshes to get together and form a bigger mesh of double the original size. The restriction to have $M(u) = M(v)$ makes it easier to control the growth of the Mesh. This approach is necessary in order to ensure formation of the expected compact mesh topology.

Without the said restrictions, the mesh growth would be uncontrolled.

Procedure MergeEvenMesh(u,v)

Procedure MergeEvenMesh which imparts connections in the EAST-WEST direction, is called by the leaders which have *M* as an even power of 2 (i.e. 1, 4, 16 etc.). Figure 4.3 shows a simple case wherein both the leaders have a single component. First, an attempt is made to coalesce them into a single piconet with a maximum of (*k* - 2) slaves, similar to a successful connection in phase 1. However, if the two piconets can not be merged into a single piconet, then a scatternet with two piconets is formed as shown. Leader *u*, makes leader *v* a slave in the EAST direction. Node *u* also updates the value of M for the new scatternet. We get a mesh of size 2 when the connections are done.

Figure 4.4 shows a more general case of MergeEvenMesh when two meshes of size more than one combine together. Here, the leader *u* has to communicate with its LAST EAST node, *u1*, asking him to make leader *v*, as its EAST slave providing him with the address and clock information of leader *v*. It also provides node *u1* with the list of addresses of all nodes in its NORTH direction. The information packet from node *u* is passed hop by hop to node *u1* through intermediate nodes. In order to obtain the necessary information, leader *u* goes ahead with a temporary connection with leader *v* and necessary data is exchanged. This temporary connection is then teared down. With the necessary information received from node *u*, node *u1* pages node *v* and makes it as an EAST slave. It also forwards the list of NORTH nodes of node *v* to its own NORTH neighbors so that they can get connected with appropriate nodes by dequeuing a node from the list and then passing the list on to its NORTH neighbor. The mesh structure is maintained with all the nodes in the NORTH direction of *u1*, getting connected with the corresponding nodes of *v*. The coordinates of all the nodes in *v*'s mesh are modified accordingly. Node *u* also needs to update the value of *M* accommodating the components led by *v*. Node *u* has to update its LAST EAST node address to the LAST EAST node address of node *v*, which is also obtained through the temporary connection.

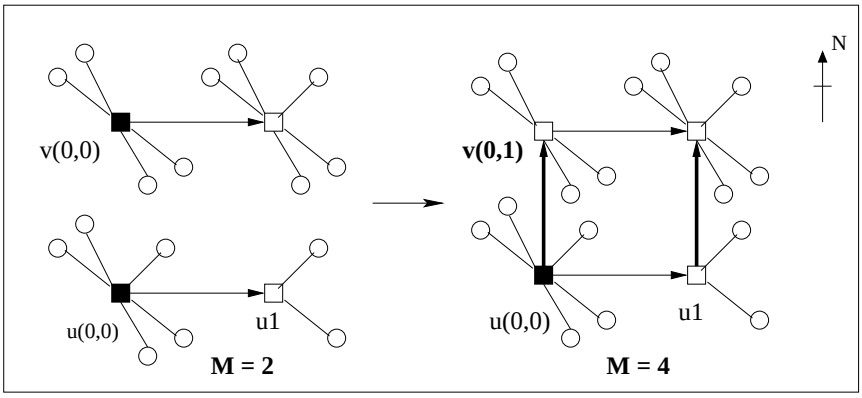


Figure 4.5: Procedure MergeOddMesh - M = 2

*Procedure MergeOddMesh(*u,v*)*

This procedure (illustrated in Figures 4.5 and 4.6) is similar to the Procedure MergeEvenMesh except that the connection of the components happens in the NORTH-SOUTH direction as opposed to the EAST-WEST direction in procedure MergeEvenMesh. This procedure is called by the leaders which have *M* as an odd power of 2 (e.g. 2, 8, 32 etc.).

A simple case would be the connection of two scatternets which have only two piconets under them. Figure 4.5 shows this special case when *M* = 2 whereas a more general case of connections is presented in figure 4.6. In the first case, node *u* connects to node *v* and assigns it as NORTH slave. However, we need the EAST piconet leader of node *u* to also form a link with the EAST slave of node *v*. This is possible since node *u* and node *v* can communicate with each other through the established link and can exchange the addresses of their EAST neighbors. This information can be passed by leader node *u* to its EAST slave and then this EAST slave can connect with the EAST slave of node *v*. This way a mesh of 4 nodes is formed by connecting 2 smaller meshes each of size 2.

Figure 4.6, illustrates a more general case. This case is when two independent meshes of size more than 2, connect with each other. In this case, the leader *u*, has to communicate with its LAST

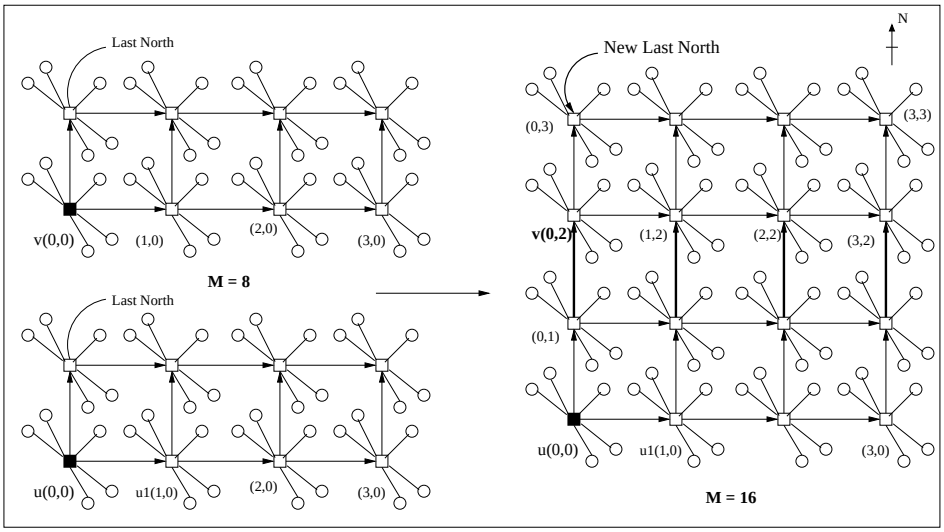


Figure 4.6: Procedure MergeOddMesh - M $>$ 2

NORTH node, *u2* asking him to make node *v* its NORTH slave. There is a temporary connection link established between leader node *u* and node *v*. Necessary information such as the list of addresses and clocks of all of node *v*'s EAST neighbors, LAST NORTH neighbor of v is exchanged through it. In order to maintain the mesh structure, all the nodes in the EAST direction of *u2*, have to connect with corresponding nodes of *v*. The list of EAST neighbors of node *v* is passed on to node *u2* which in turn passes it on to its EAST neighbors. Each neighbor dequeues a node from the list, *pages* that node using the clock and address information. It then forwards the list to its EAST neighbor. The coordinates of all the nodes in node *v*'s scatternet are modified accordingly. Node *u* doubles the value of *M* accommodating the new components. It also changes it LAST NORTH to LAST NORTH of node *v*.

*Procedure MergeUnequal(*u,v*)*

This procedure is called when the leaders *u* and *v* contact each other, but either have unequal number of components (i.e. value of *M*) under them or both the leaders belong to different phases of the algorithm. The successful merging of two such components is possible only if either one

of the leaders have a single component under them i.e. $M = 1$. In order to combine into a single piconet, we still need to make sure that the number of non-bridge slaves of *u* and *v* together is less than (k - 2). If the condition can be satisfied, *u* and *v* merge into a single component by moving all slaves of *v*, including *v* into piconet led by *u*. Node *u* remains as the leader and *v* retires from its duty of leader. This is similar to a successful connection in phase 1. The failure case is one wherein the two leaders do not merge and remain separate components. Both *u* and *v* in this case start looking for other components to get connected to.

Fine-tuning Phase 2:

At the end of phase 2, we would expect to have a single mesh consisting of all the devices. However, this is not always possible e.g. cases where there are two leaders left with values of *M* being 8, 4 resp. Due to the condition that both leaders need equal values of *M* to get connected, they can never form connection in Phase 2. Hence, we need more control over the mesh connections that can happen in phase 2. This control can only be achieved from the local information available to the nodes due to the unavailability of global information. During the life of a leader node in phase 2, it may get connected with many other leader nodes with different values of *M* and hence disconnect with them. However, in order to exchange their values of M, they need to form a connection link and again disconnect. This connection establishment and disconnection is a costly operation. It accounts for multiple duplicate node discovery as same unequal sized mesh leaders may discover each other again and again resulting in more disconnections. Hence, in order to get a single mesh at the end of phase 2 faster, we add following logic to phase 2 algorithm.

1. In phase 2, when two leaders discover that the values of *M* are different and hence, they can not continue to connect with each other, each leader keeps a record of the address of other node and number of disconnections happened with that leader node.

2. After a certain number of disconnections between same pair of leaders, the leader with a bigger value of *M* decides to take control of the other leader by storing its address and

clock information. The leader of the bigger mesh also adds smaller mesh leader's list of disconnected leaders into its own list. The leader with smaller value of *M* then retires from searching for neighbors.

3. Every piconet leader stores a list of such disconnected leaders and when a new leader node is added into the list, a check is made to see if there are any leaders with a value of *M* same as the node newly added. If there is a leader with duplicate value of M, duplicates are removed by forcing a connection between these leaders forming a bigger mesh of double value of M. The list is again updated for any new duplicates. This process continues till we are left with no duplicates in the list.

The proper value of number of disconnections that can happen before a leader takes charge of the other leader in phase 2, depends upon the way the nodes discover each other. During the experiments performed, we see that if two leaders discover each other twice, then there is a high tendency of the same leaders to discover each other again and again. Hence, we set the minimum number of disconnections as 2 i.e. after 2 disconnections, leader with higher value of *M* would ask the other leader to retire and then it would store this leader's address and clock information.

This way, at the end of phase 2, if there is a single mesh that is created with only one leader node left in the network, we do not need execution of phase 3 at all and the algorithm terminates. We however, may run into a situation where along with the existence of a single mesh there may be a few leaders which can not get connected due to different value of *M* which are stored in this bigger mesh. Since, we have not yet achieved a single scatternet, we need to interconnect these independent meshes together retaining the mesh structure of the scatternet.

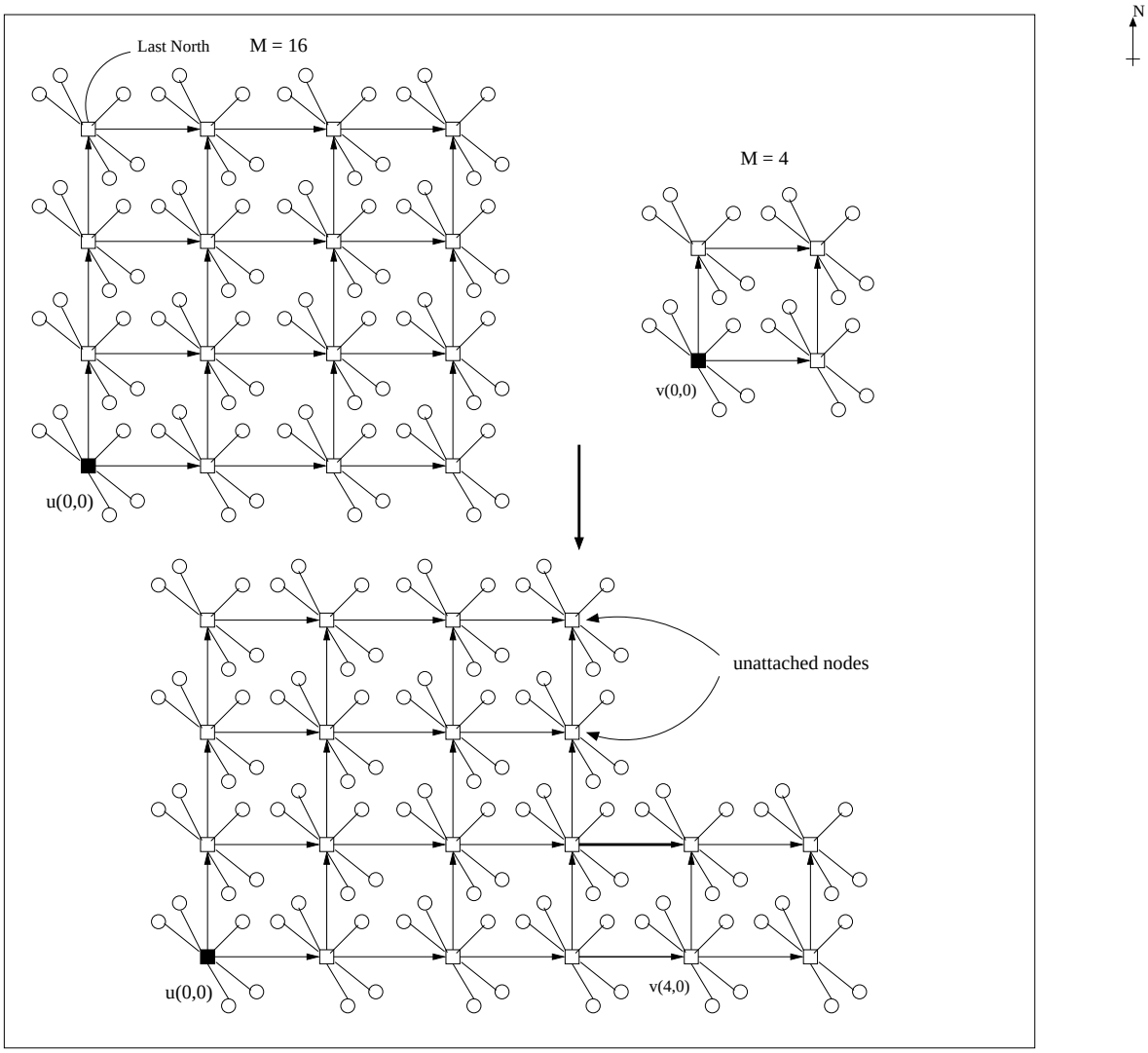


Figure 4.7: Phase 3 - Initial EAST - WEST directed connections of two meshes with M = 16 and M = 4

4.1.3 Phase 3: Reduction to a Single Mesh

Purpose:

By the end of the second phase, most of the nodes get involved to form a bigger mesh. But it might happen that there is more than one leader left and all the leaders represent meshes of different sizes and hence could not get merged in the second phase. Our job in this phase is to merge all the existing meshes into a single component while maintaining the mesh structure and keeping the

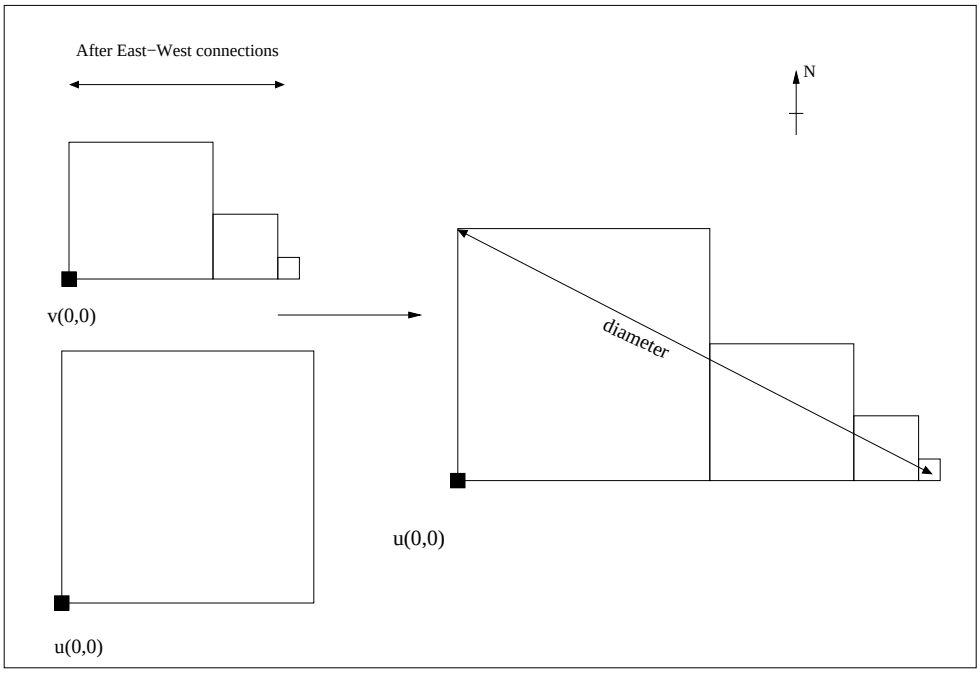


Figure 4.8: Phase 3 - Single Mesh Generation - Case 1

diameter to minimum. The diameter can be kept to minimum by controlling the way these meshes connect together. By ensuring connections in both directions, we try to avoid any run-offs in any one direction.

Implementation:

As shown in figure 4.7, we connect all the leader nodes present in the disconnected leader list, in the EAST direction allowing all the possible connections that can happen. Hence a few nodes in the bigger meshes may not have nodes to get connected to due to the unequal size of meshes. e.g. a mesh of size 16 when gets connected to another mesh of size 4 in EAST direction, a few nodes on EAST of mesh of size 16 will remain unattached as shown in figure 4.7. After all such EAST connections are formed, we are left with one perfect mesh and another scatternet of EAST wise connected smaller meshes. Now there are two different ways in which may combine these two meshes. Both the cases are shown in figures 4.8 and 4.9. Now, in order to reduce the diameter, we connect these two scatternets in the NORTH-SOUTH direction as in the second case. This reduces

the diameter since the combination of smaller meshes fits on top of the bigger mesh so as to create a compact mesh. This way, at the end of phase 3, we get a single mesh scatternet.

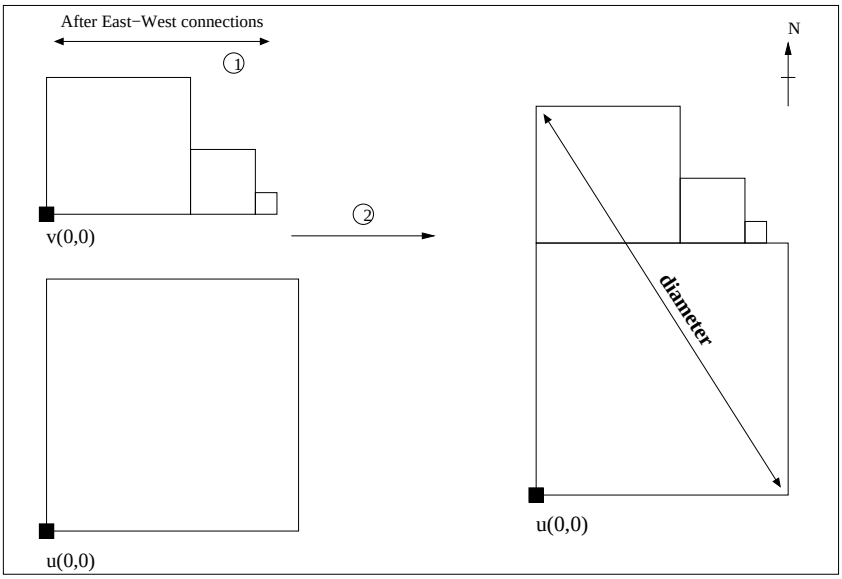


Figure 4.9: Phase 3 - Single Mesh Generation - Case 2

4.2 3-Dimensional Mesh (Cube) Scatternet Formation

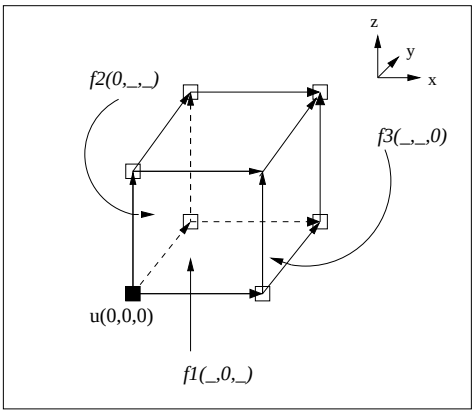


Figure 4.10: Cube Scatternet Formation

The 3-dimensional mesh scatternet, also called *cube* scatternet can be generated in a similar way accommodating for the third dimension in every connection or merging that happens. The

algorithm to form Cube also consists of 3 similar phases. The first phase is dedicated to piconet formation with a slight change from phase 1 of the mesh scatternet formation algorithm. We organize these piconets into cubes in Phase 2 by forcing an order in the way connections happen. Phase 2 of the algorithm is where smaller cubes get together to form bigger cubes. We face a similar situation when phase 2 may not be sufficient to guarantee formation of a single cube which makes use of phase 3 necessary. We now look at each phase briefly understanding the differences in the way cubes are formed from mesh scatternet.

4.2.1 Phase 1: Piconet formation

Purpose:

The purpose of Phase 1 of cube scatternet formation is very much similar to the purpose of phase 1 of mesh scatternet formation. We aim at grouping the given set of independent devices into piconets by creating master-slave links between devices. We also keep in mind that using these piconets, we need to build the cubes in further phases and hence, we keep the maximum number of slaves in a piconet to a certain fixed value.

Implementation:

We assume a set of *N* devices that are within the transmission range of each other. Each device is a leader of piconet consisting itself. In order to form cube scatternet, a leader of the piconet has to form a maximum of 3 slave links with the leaders of other piconets and hence, while forming piconets in phase 1, we need to keep 3 slots open in every piconet for these links that may happen in further phases. Hence, we make a restriction on the number of slaves a piconet leader can acquire as *k* - 3. This will allow every piconet to capture 3 more slaves later.

The execution of the phase is similar to the phase 1 of 2-dimensional mesh scatternet formation algorithm and hence we omit the execution details here.

Termination of Phase 1:

A piconet leader is forced to move into the phase 2, when it acquires its quota of *k* - 3 slaves. However, as discussed in mesh algorithm, this condition solely is not enough to move all the nodes into phase 2. Hence, we base our decision to move nodes into phase 2, on the number of slaves acquired by the leader and the time spent by the node in phase 1. A node which acquires higher number of slaves, close to *k* - 3 slaves, moves to phase 2 quicker than a node which has relatively less number of children.

4.2.2 Phase 2: Cube Growth

Purpose:

The purpose of this phase is to interconnect the piconets formed at the end of phase 1, into cubes. These cubes can again be interconnected to form bigger cubes. Phase 2 essentially builds up the cube scatternet by allowing smaller sized components to merge with each other to form bigger components. There are 3 slave slots open in each of the piconets which allows us to create bridges from one piconet to another interconnecting them in 3 different directions forming a cube.

Implementation

A piconet leader in phase 2 has a maximum of *k* - 3 slaves under it with a space for minimum three more slaves. In order to form a cube scatternet, we need to form connections between components in (x, y, z) directions as opposed to (x, y) in mesh. Every leader randomly chooses between INQUIRY and INQUIRY SCAN states. Every leader maintains the number of components under it given by *M*. The initial value of *M* is set to 1 indicating that there is only one piconet under this leader. The leader of the piconet, now has to do more book keeping as compared to a leader in mesh scatternet algorithm. Now, a leader has to keep information about the 3 planes that it is a part of as shown in the figure 4.10. Whenever two leaders discover each other, they exchange their values of M and depending upon these values, merging can happen. The bridge node slaves are acquired in x(EAST), y(NORTH) and z directions.

The two smaller cubes can merge to form a bigger cube only if their values of *M* are equal. If the values of *M* do not match, procedure mergeUnequal(*u*,*v*) is executed. When two cubes have equal values of *M*, procedure MergeXCube(*u*,*v*) or MergeYCube(*u*,*v*) or MergeZCube(*u*,*v*) is called. The following formula is used for making a choice of procedure that should be called.

$$(log_2 M)\%3 = \begin{cases} 0, & \text{execute Procedure MergeXCube(u,v)} \\ 1, & \text{execute Procedure MergeYCube(u,v)} \\ 2, & \text{execute Procedure MergeZCube(u,v)} \end{cases} \tag{4.1}$$

This ensures that the size of the cube increases in all the 3 directions one by one by connecting in x, y and then z-direction and so on.

Procedure MergeXCube(u,v)

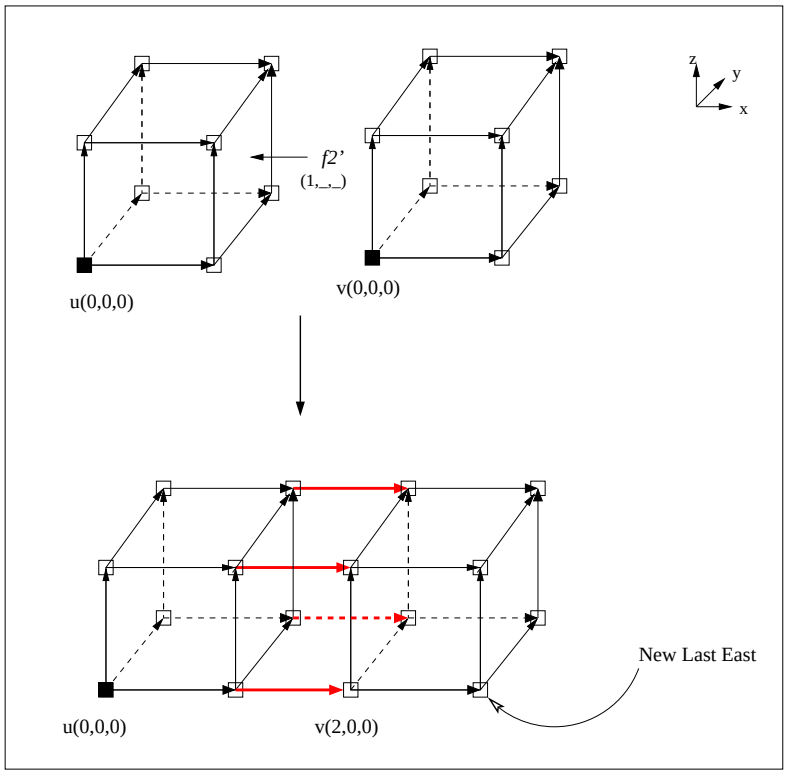


Figure 4.11: Cube Scatternet MergeXCube(u,v)

Procedure MergeXCube(*u*,*v*) connects the two cubes in the x-direction as shown in figure 4.11. Now, the faces *f1* and *f3* of the leader *u* are changed by appending them with corresponding faces *f1* and *f3* of cube led by node *v*. Node *u* and node *v* form a temporary connection link between each other through which information about the nodes constituting faces *f1*, *f2* and *f3* of node *v* is passed on to node *u*. Node *u* appends the faces *f1*, *f3* of node *v* to its own faces *f1* and *f3* respectively and then the information about node *v*'s face *f2* is passed on to LAST EAST node *u1* which is also common node between faces *f1* and *f2* of node *u*. Node *u1* then pages node *v* and makes him its EAST slave. It then passes the information about face *f2* to its neighbors in y-direction, columnwise. Each node in each column dequeues a node from the list and then forwards the list to its z-direction neighbor. When all the nodes of face *f2'* form EAST connections with all the corresponding nodes of face *f2* of node *v*, we get a bigger cube of double the original size. Node *u* then doubles the value of *M* and modifies its LAST EAST node to LAST EAST node of node *v*. The coordinates of all the nodes in cube led by *v* are modified accordingly.

Procedure MergeYCube(u,v)

This procedure (illustrated in figure 4.12) is similar to the procedure MergeXCube, except that the cube growth is achieved in y-direction as opposed to x-direction. The temporary connection between nodes *u* and *v* is used for exchanging information about the faces *f1*, *f2* and *f3*. Node *u* appends faces *f2* and *f3* with the corresponding faces *f2* and *f3* of cube of node *v*. Then the information about face *f1* is passed on to LAST NORTH node *u1* hop by hop. Using this information, face *f3'* of node *u* gets connected to the face *f1* of node *v*. The coordinates of all the nodes in the cube led by *v* are modified accordingly. Node *u* doubles its value of *M* and also modifies the LAST NORTH node to reflect addition of cube led by node *v* in y-direction.

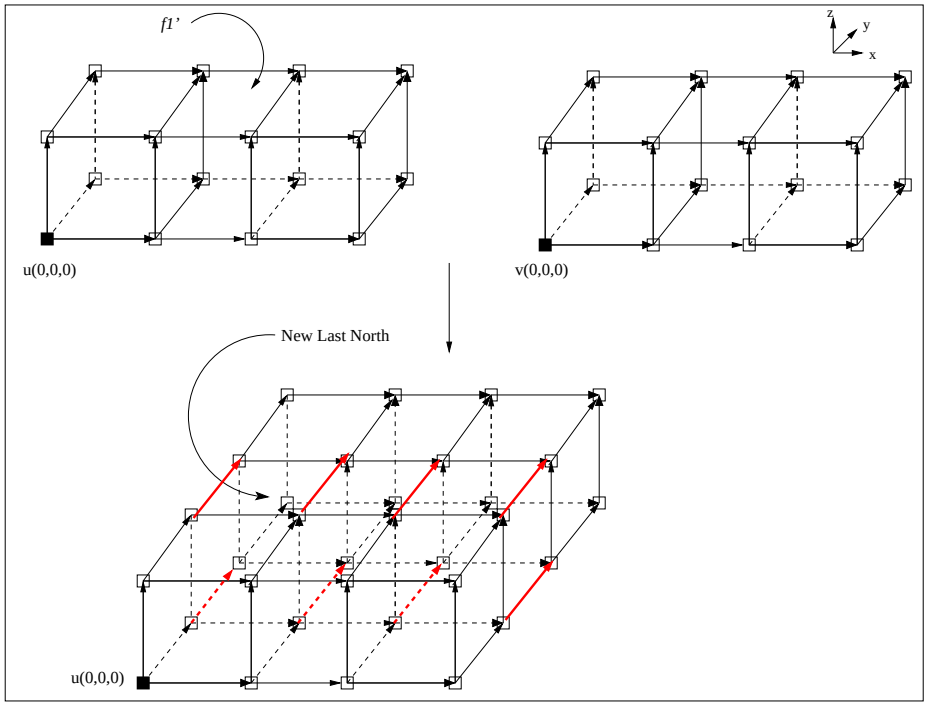


Figure 4.12: Cube Scatternet MergeYCube(u,v)

Procedure MergeZCube(u,v)

This procedure connects the two cubes in the z-direction as shown in figure 4.13. Here, faces *f1* and *f2* of node *u* are appended with the corresponding faces of the cube led node *v*. Each node constituting face *f3'* of node *u* gets connected with the corresponding node of face *f3* of node *v*. The modifications to the coordinates of nodes are done as in the previous cases. Node *u* changes its LAST Z node to LAST Z node of node *v*.

*Procedure MergeUnequal(*u,v*)*

This procedure is called either when two cubes of unequal size contact each other or when the two leaders belong to two different phases of the algorithm. We can combine the two cubes into one if and only if one of cubes has $M = 1$ and the total number of non-bridge slaves together of node *u* and *v* is maximum of $k - 3$. If these conditions are satisfied, then a successful merging of two piconets into one, takes place which is similar to a successful connection in phase 1. However, if

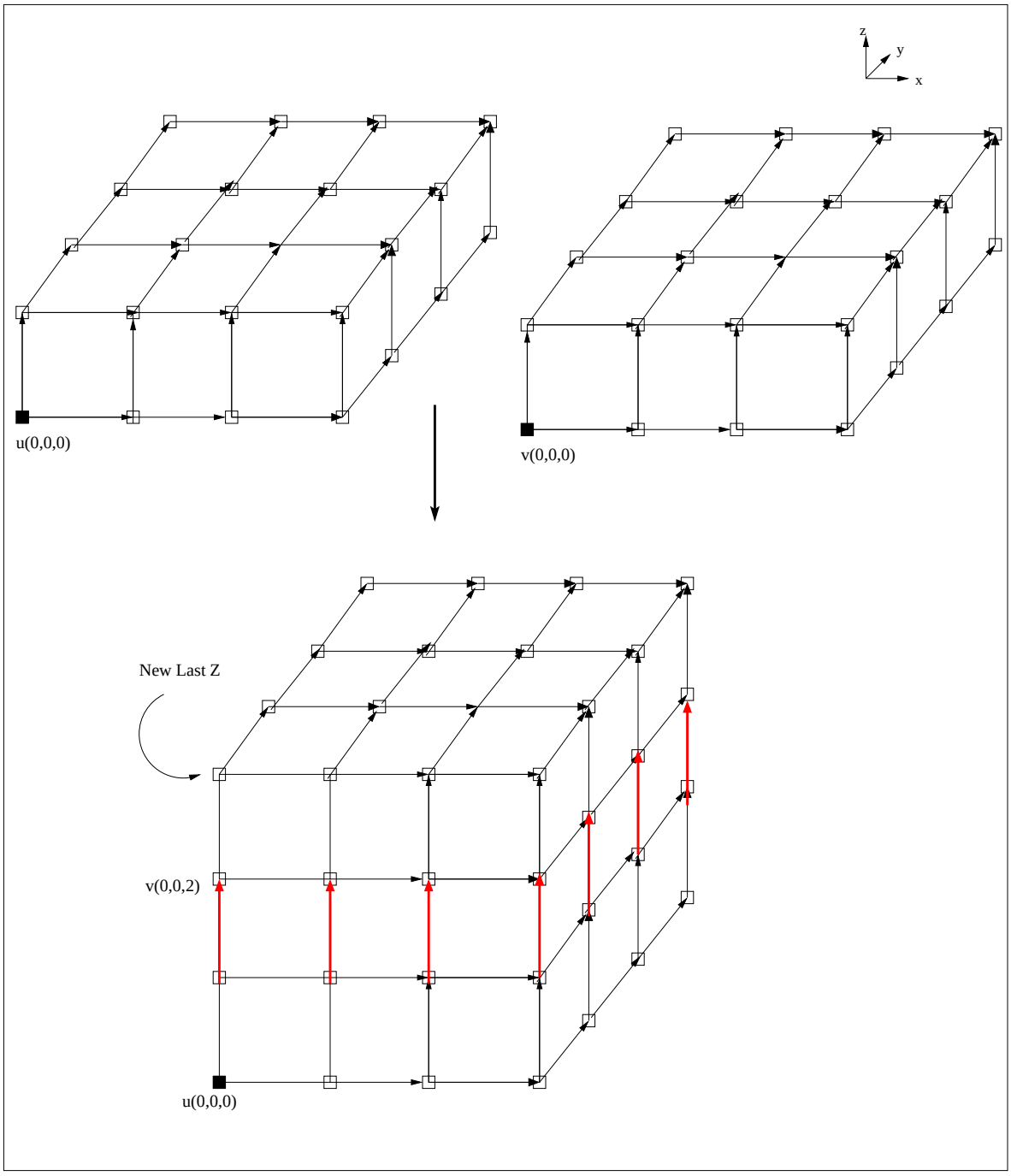


Figure 4.13: Cube Scatternet MergeZCube(u,v)

both the conditions are not satisfied, the two cubes remain separate entities and keep searching for neighbors.

Fine-tuning Phase 2:

As in the case of Mesh scatternet, we can not guarantee the formation of a single cube scatternet by the end of phase 2 with only this information. We would observe that there are many unsuccessful connections that happen due to the discovery of neighbors which have different size. In order to control these disconnections, we add exactly similar logic to phase 2 of cube scatternet algorithm as in mesh scatternet. Hence, a leader of a cube would take charge of another smaller cube if it discovered and disconnected with the leader of the cube twice in phase 2. The leader of smaller cube is added in the list which is updated to remove any duplicates.

This would ensure us the existence of a single big cube and several smaller cubes with different sizes if any. If there is only one leader remaining in the network, we do not need phase 3 in which case, scatternet formation algorithm terminates. In other case, we execute phase 3 with similar goals of achieving a single cube with smallest diameter possible.

4.2.3 Phase 3: Single Cube scatternet formation

Purpose:

The phase 2 does not guarantee us that there will be a single connected scatternet in the network. Hence, purpose of phase 3 is to wrap up the scatternet formation algorithm by interconnecting the remaining leaders, maintaining the cube structure of the scatternet and keeping the diameter as low as possible.

Implementation:

Phase 3 is called when we are left with more than one leaders after phase 2. Here, along with the basic goal of connectivity, we also keep the goal of diameter reduction in mind and do the connections in phase 3. We connect all the smaller cubes in the direction where the total length of the a face of the formed cube in that direction is smaller or equal to the corresponding face of leader of biggest cube. For this purpose, we need to add the length of the faces of each of the smaller cubes in each directions and compare it with the corresponding faces of the bigger mesh.

This can be found out since leader of the bigger mesh knows the values of *M* for all the smaller cubes. Hence, we are left with only two leaders of unequal size cubes in the network. Then a final connection is made in the z-direction so that the diameter of the network is minimized. This phase guarantees us formation of a single cube quickly.

4.3 Handling Dynamic Environments:

We assume all the Bluetooth devices to be stationary during the simulations and hence do not have to worry about the mobility of nodes. However, in real world Bluetooth networks, the nodes are mobile and may leave and join the network. In this section, we explain how both scatternet schemes handle these cases of dynamic node leave and joins without incurring much cost.

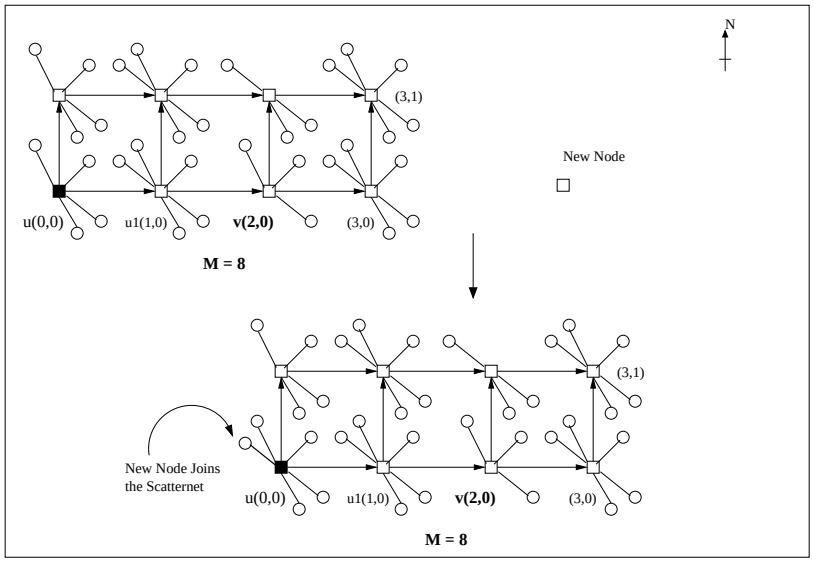


Figure 4.14: Node Joining in Mesh scatternet

4.3.1 Dynamic Node Joins

The mesh (2D and 3D) scatternet algorithms described above, consist of the first phase of piconet formation which puts a bound of maximum of *k* - 2 slaves in each piconet for mesh and a maximum of *k* - 3 slaves for cube, so as to allow for bridge nodes to be acquired in further phases. However, not all piconets in the final mesh scatternet would be complete and consist of *k* slaves. This is

because of the constraint that we put on the time spent by a node in phase 1. We do not leave piconets in phase 1 till they acquire maximum slaves possible. We put a restriction of a certain limit on the time spent in phase 1 and hence, we expect a few piconets to be made up of less than maximum slaves when they move to phase 2. This leaves space for more than 2 slaves in case of mesh and more than 3 slaves in case of cube, to be acquired in phase 2. However, as explained in the algorithm, leader of the piconet acquires a maximum of 2 slaves in phase 2 for mesh and a maximum of 3 slaves in the case of cube. This leaves some slots open where new nodes can be accommodated. Moreover, not all the leaders make use of all of the bridge node slots for mesh connection. Only the internal nodes of a mesh and a few other nodes, utilize both their slave connections for mesh generation. The nodes which form the corners and edges of the mesh may use 1 or none of those two slots. Hence, after the final scatternet is formed, we can see that the mesh scatternet (2D as well as 3D), can still accommodate many more slaves without affecting the mesh structure. One such example is shown in figure 4.14. As we can see till the capacity of the mesh to acquire more slaves, is not reached, we do not have to reorganize the scatternet. In order to handle dynamic node joins, we suggest the following:

1. When a new node wants to join the fully formed scatternet, it contacts the leader of the mesh (2D or 3D) scatternet requesting him to absorb this new node into the scatternet.

2. The leader of the scatternet then checks its own piconet, to see if it has any space for accommodating this new node. If it has less than k slaves in its own piconet, then it forms a slave link with this new node asking him to join the piconet as a slave. The node gets added to the mesh scatternet and still mesh structure remains unaltered.

3. If the leader node finds that its own piconet has no space for new node joins, it asks its immediate EAST neighbor for the same. If its immediate EAST neighbor can accommodate this new node, it does so. In case of failure, it asks its immediate EAST neighbor and so on.

4. If all the EAST neighbors of the leader fail to absorb the new node, leader node asks its NORTH node for the same. NORTH node repeats the similar steps to check if it has any space for accommodating new node or else tries the same with its EAST and NORTH neighbors.

5. This way each piconet leader in the scatternet checks if it can add the node as a slave in its piconet.

Our scatternet can accommodate a very high number of node joins keeping the original mesh structure (2D as well as 3D) intact. This belief is corroborated by our simulation results presented in chapter 5, section 5.2.4.

4.3.2 Dynamic node Leave

Our main aim while handling node leaving the scatternet, is to maintain the mesh structure as much as possible and still allow for a node leaving the scatternet. We face following conditions:

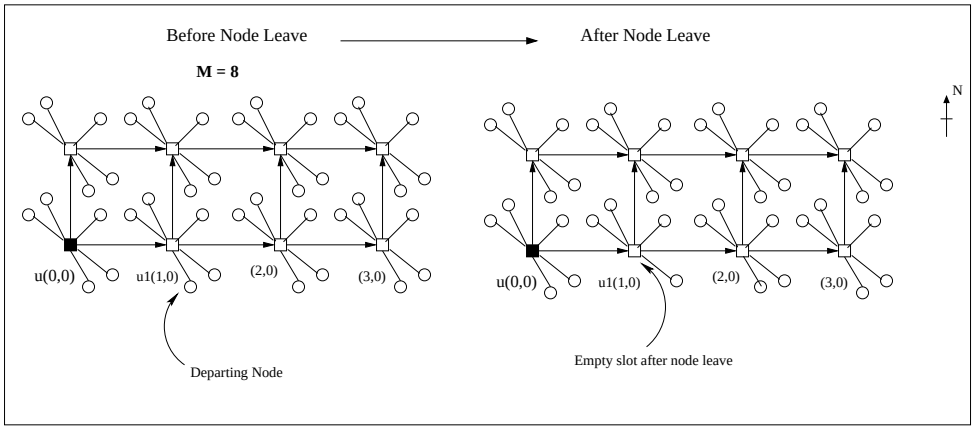


Figure 4.15: Dynamic node leave - Simple case

1. When a non-bridge node leaves the scatternet: When a node which does not play the role of a bridge between any two piconet wants to leave the scatternet, we do not have to handle the case, in any special way. The leaving of a node only affects the piconet it is a part of,

by reducing the number of slaves by 1. However, this does not affect the connectivity of that piconet or the whole scatternet at all. Hence, we do not have to worry about any of the non-bridge nodes leaving the scatternet. This simple case is illustrated in figure 4.15.

2. When a bridge node leaves the scatternet: This is a more complicated case that the first case. All bridge node perform the duty of a master in their own piconets. A bridge node essentially provides the connectivity to the mesh scatternet and hence, not handling these cases would leave holes in the meshes. This is not desirable. Hence, we decide on handling this case in following way:

   - Case 1: When the departing node has a minimum of one non-shared slave, it assigns one of its non-shared slave as the new master in its own piconet and asks that node to take its place in the scatternet. This is as per the Bluetooth specifications 1.0b which states that when a master node leaves its piconet, it has to assign one of its slave as temporary master of the piconet before leaving. This case is shown in figure 4.16. Here, the slave *u1* which is assigned the new master of the piconet, takes up the duties of the departing node *u* replacing previous connections with new ones. Hence, the EAST slave *e1* of node *u*, now becomes the EAST slave of *u1* and similarly for NORTH slave *n1*. The leaders *v1* and *v2* replace their old connections with node *u* with new connections with node *u1*.
   - Case 2: When the departing node does not have any non-shared slaves, it can not assign its duties to any of its slaves and hence have to rely on some other technique for handling this case. There are two sub-cases that may be considered:
     - when the departing node has no EAST and NORTH neighbors: This is the case when the departing node has no connections outgoing to other piconets and hence has no EAST and NORTH neighbors. When this situation occurs, the leaving of this node implies very insignificant harm to the connectivity of the scatternet

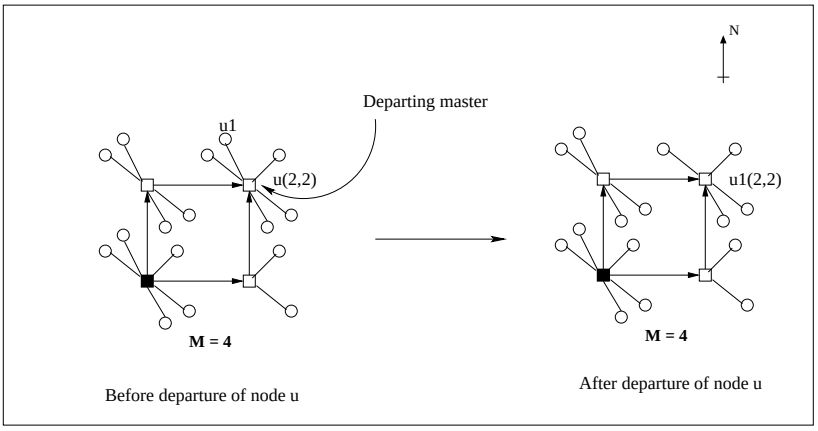


Figure 4.16: Dynamic node leave Case 1

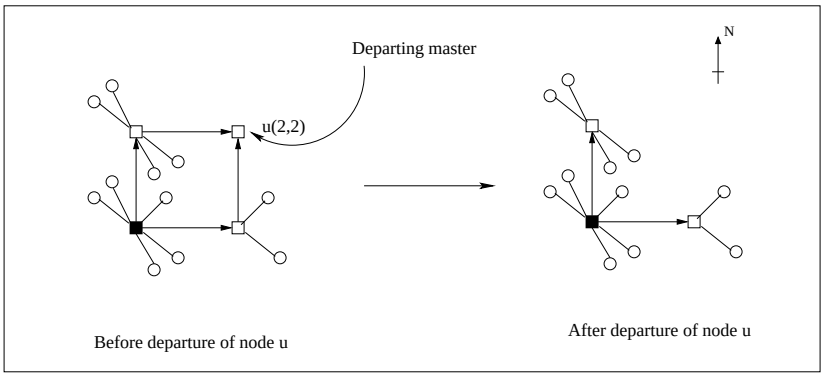


Figure 4.17: Dynamic node leave Case 2(a)

as shown in figure 4.17. As shown, node *u* has no outgoing EAST or NORTH connections and hence, it can leave the scatternet without any harm to the structure of the scatternet. Node *v1* and *v2*, for whom node *u* was a slave, tear down those slave connections when node *u* leaves.

– when the departing node has either EAST or NORTH or both neighbors: In this case, departure of the node which has minimum one outgoing connection to other piconets, would create holes in the mesh. In order to solve this problem, we suggest that the departing node in such cases, asks one of its immediate EAST or NORTH neighbors for any non-shared slave they might have. In case when either of immediate EAST or NORTH neighbors (priority given to EAST neighbor) has

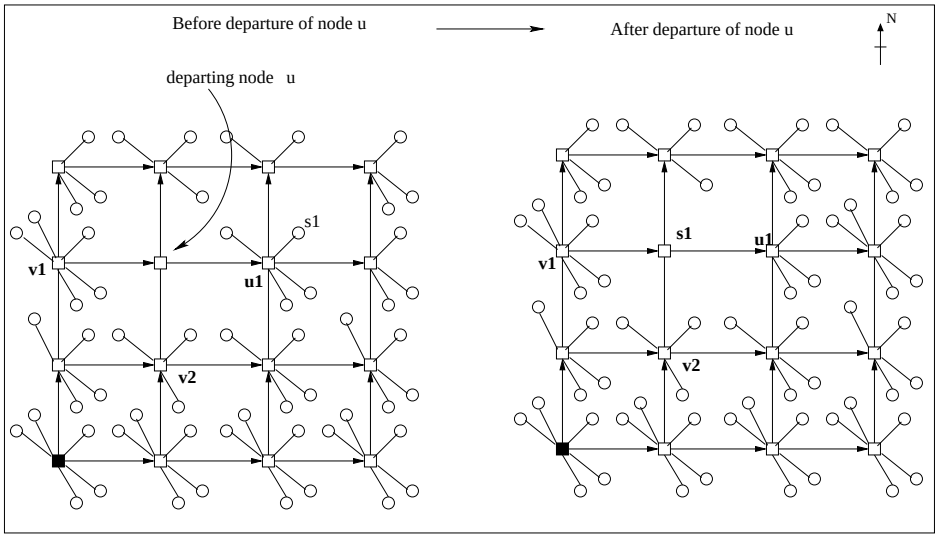


Figure 4.18: Dynamic node leave Case 2(b)

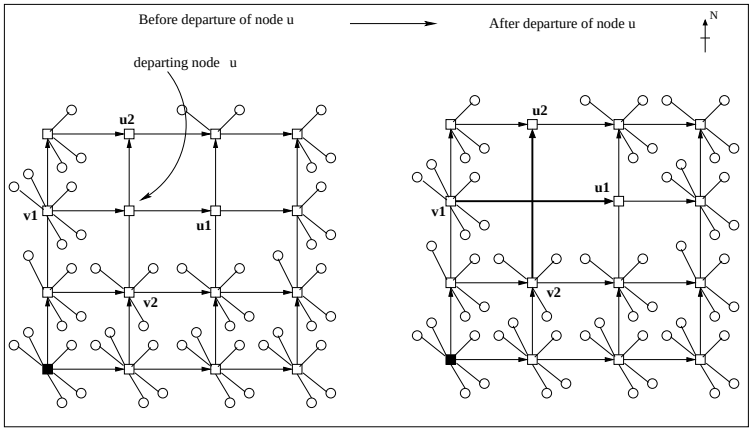


Figure 4.19: Dynamic node leave Case 2(c)

a non-shared slave, this slave is asked to take up the role of the leaving node in the scatternet. This case is shown in figure 4.18. Node *u* which has no non-shared slave, asks its EAST neighbor *u1* to lend a non-shared slave *s1* and asks this node *s1* to perform its duties. Hence, nodes *v1* and *v2* replace their slave connections to node *u* with similar connections to *s1*. We face the worst case scenario when the departing node's immediate EAST and NORTH neighbors also lack the non-shared slaves. This case can be handled as shown in figure 4.19. Hence, node*u*

and its immediate EAST neighbor *u1* and immediate NORTH neighbor *u2*, do not have a non-shared slave which can replace the node *u*. So, node *v1* which has *u* as its EAST slave, now makes node *u1* as its EAST slave and node *v2* which has *u* as its NORTH slave, now makes node *u2* as its NORTH slave. This retains the mesh structure maintaining similar connectivity we had in the previous mesh before modification. Hence, we do not suffer from any mesh structure reorganizations for any node leaves that happen.

The cases can be handled in exactly similar way by extending the solution to 3-dimensions for cube scatternet. Hence, the proposed scatternet formation algorithms handle the cases of dynamic node leave and join very naturally. The routing in a static node mesh without any node leave and join, is a trivial issue. When the devices are assigned coordinates, there are a number of routing strategies available for this purpose. However, node leave and join creates imperfect meshes. The routing inside these imperfect meshes will need special handling. A trivial routing algorithm can no longer be applicable to this topology. A routing algorithm for these imperfect meshes is subject to further research.

CHAPTER FIVE

PERFORMANCE ANALYSIS

In this chapter, we present results obtained for a series of experiments performed and analyze the performance aspects of the *mesh* and *cube* scatternet formation schemes on the basis of identified performance metrics. In the next section, we discuss these metrics based on which the schemes are evaluated. In the simulation environment section, the tools used for simulations are explained and the numerical results obtained from experiments are presented. We have compared our simulation results with a number of other scatternet formation schemes proposed, highlighting the salient properties of mesh and cube scatternets.

5.1 Performance Metrics

The scatternet formation schemes are evaluated based on the following performance metrics :

1. *Scatternet Formation Delay*: This metric measures the amount of time required to form a single scatternet. As discussed in the previous chapters, Bluetooth device discovery process is an asymmetric process and hence, devices require certain time in order to discover and then form a link with their neighboring devices. This metric refers to the total time required to form a single scatternet by connecting all such Bluetooth devices together. The scatternet formation delay is expected to increase with increase in the number of devices.

2. *Number of Piconets*: Though a scatternet with a large number of piconets allows for better spatial reuse, it incurs more end-to-end delays. As discussed in chapter 2, the number of piconets should be kept as low as possible in order to reduce the number of collisions. However, there is a trade-off between the connectivity of the network and the number of piconets. As the network becomes more and more connected, more and more piconets have to share their components between each other, resulting in increase in number of piconets.

3. *Network Diameter*: The Network diameter is defined as the maximum number of hops between any pair of devices. The network diameter essentially measures the maximum distance packets have to travel in a network for communication between any two devices. The longer the distance, larger will be the delay in communication and hence, throughput of the network suffers. A scatternet topology should provide minimum diameter in order to incur smaller communication delays. A network that provides higher connectivity is expected to provide smaller network diameters. This fact is evident from the simulation results obtained.

4. *Maximum Node Contention*: The contention at node *i* is defined as

$$contention(i) = \sum_{1 \leq j,k \leq N} f(i,j,k) \tag{5.1}$$

where

$$f(i,j,k) = \begin{cases} 1, & \text{if the shortest path between node pair j,k traverses i} \\ 0, & \text{otherwise} \end{cases} \tag{5.2}$$

Node contention thus, measures how many node pairs use node *i* as a relay node in their communication. If a node is common node between several devices, then it may become a performance bottleneck increasing the contention in the network. The node contention leads to congestion in network resulting in dropping of packets and loss in throughput. The congestion in the network must be kept to minimum.

5. *Average Roles played by a node*: The average role is the metric which measures how many different roles are played by an arbitrary node in the given scatternet. The ideal value would be 1 since that would mean, a node is a part of only one piconet and is performing only one function either as a slave or a master. However, in this case, the scatternet would not be connected since there would be no bridge nodes available. A bridge node always has to

perform more than one role. Hence, the goal of the scatternet should be keeping the average roles played by a node as low as possible. It is also an indication of the number of bridge nodes used in the network. We would like to state that the properties *lesser average roles played by a network* and *higher connectivity of the network* can not both be satisfied simultaneously. Connectivity of the network is a highly desirable property since that increases the robustness of the network and does not partition easily obviating the need to reorganize. However, higher connectivity can only be achieved using higher number of bridge nodes, increasing the average roles played by a node. Hence, we face a trade-off situation.

6. *Degree Deviation*: Degree deviation measures the balance of the network and is calculated as follows:

$$\alpha = \sqrt{\frac{N * (\sum x^2) - (\sum x)^2}{N * (N-1)}}) \tag{5.3}$$

where

N : Number of nodes in the network and

x : degree of each individual node

A very high degree deviation is seen in case of the "star" networks where a few central nodes have a very high degree and most traffic flows go through them. Whereas the degree of other nodes is very less. On the other hand, small deviation refers to a topology where on each node approximately has the same number of traffic flows going through, which allows for more even distribution of traffic in the network. Miklos *et al* [16] prove that the performance of the network is better with lower degree deviation, however this effect is less significant for small networks.

7. *Number of nodes that can join without reorganization*: In the real world scenario, nodes may leave and new nodes may join an existing scatternet. A scatternet is said to be robust when the network can handle such node leaves and node joins without having to re-construct the

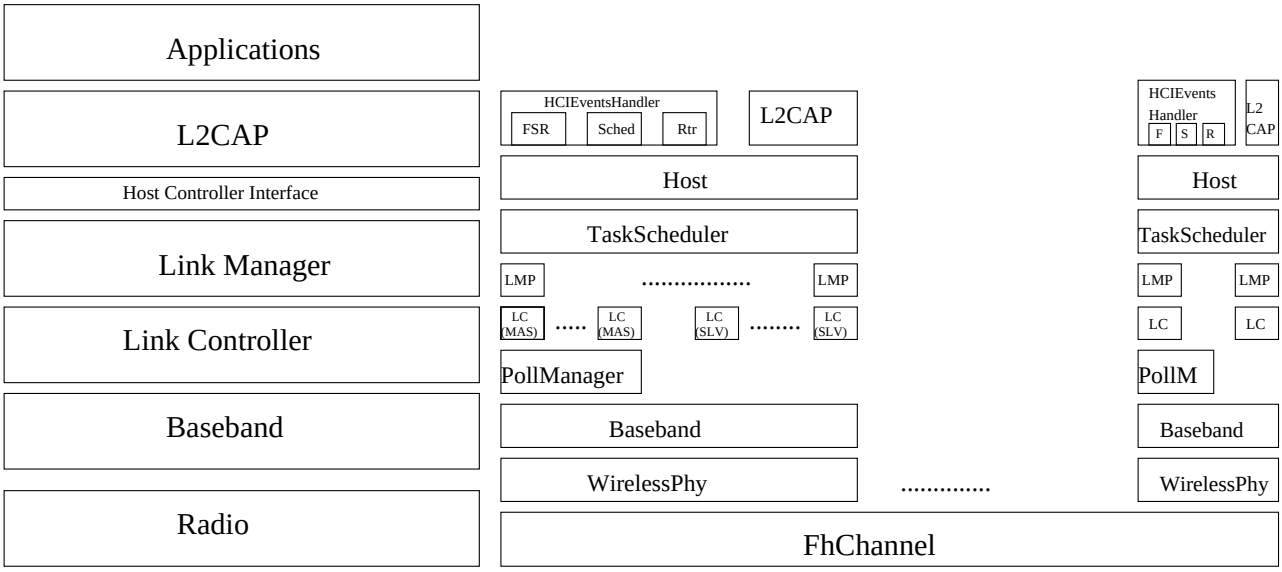
scattenet. Through simulations, we take a look at the capacity of the proposed schemes to handle joining of such new nodes to the scatternet formed without having to reorganize.

8. *Average number of links at a node*: This average measures the total number of incoming and outgoing links from the node. Higher number of links implies that the network can carry and evenly distribute the increasing amount of traffic. However, this also means increasing the number of bridge nodes which represents increasing overhead due to scheduling complexity.

5.2 Simulation Environment

For our simulations, we make use of Blueware 1.0 [2]: Bluetooth Simulator for *ns* developed at MIT which closely follows Bluetooth specifications 1.0. It is available for public use and can be downloaded from http://nms.lcs.mit.edu/projects/blueware. The next section describes Blueware simulator. We then explain the various simulation parameters set in the simulation model.

5.2.1 Blueware Simulator



(a) BLuetooth Stack

(b) Blueware Simulator Modules

Figure 5.1: Comparison between Bluetooth Stack and Blueware modules

Blueware 1.0 is built on top of well known network simulator *ns-2*[1]. *NS-2* is a discrete event simulator targeted at networking research. It provides substantial support for simulation of TCP, routing and multicast protocols for various networks. It provides a set of interfaces for configuring a simulation and for choosing the type of event scheduler used to drive the simulation. *Ns-2* implements an extended network stack that makes simulations of wireless networks possible. Blueware is built on top of this wireless extension provided to *ns-2*.

Blueware simulator implements most aspects of the Bluetooth protocol stack according to the Bluetooth Specifications 1.0b. Figure 5.1(a) shows the the Bluetooth protocol stack and Figure 5.1(b) depicts various modules of the Blueware simulator. Each device in the simulator is equipped with several modules through which it discovers and communicates with other devices. The WirelessPhy module provides functionality of the Bluetooth radio. The FhChannel module simulates the frequency-hopped wireless medium. The Baseband module of the simulator implements two main functions:

1. to discover neighboring nodes using Inquiry and Page operations and

2. to generate pseudo-random frequency hopping sequence.

It implements the Inquiry, Inquiry Scan, Page, Hold, and Role-Change operations as specified in the Bluetooth Baseband specifications. Poll Manager manages the master-slave links within a piconet and decides which active slave to poll. The Link Manager Protocol (LMP) module implements detailed LMP protocols and is responsible for setting up and configuring a new link, switching master and slave roles on a particular link, putting a link into low power modes such as Hold, Sniff and Park and tearing down an existing connection. The Link Controller (LC) module implements the link control operations such as the Automatic Repeat Request (ARQ) scheme. As shown in Figure 5.1(b), every master-slave link has unique instances of the LC and LMP modules which are responsible for carrying out all the Bluetooth primitive link establishment, control and

communication operations. TaskScheduler implements the scheduling schemes for various Bluetooth operations related to communication and neighbor discovery. The Host module provides the Host Controller Interface (HCI) and interacts with various scatternet protocols. Each scatternert formation scheme can be implemented by extending the HCIEventsHandler module. Finally The Logical Link Control and Adaption Protocol (L2CAP) module provides segmentation and reassembly of application data units. Blueware introduces notion of a *session* at the Baseband module. Use of *sessions* allows the baseband module to serialize the Baseband operations and avoid undesirable situations of an event being triggered interrupting an ongoing operation. Blueware also allows host to setup an un-interruptible session.

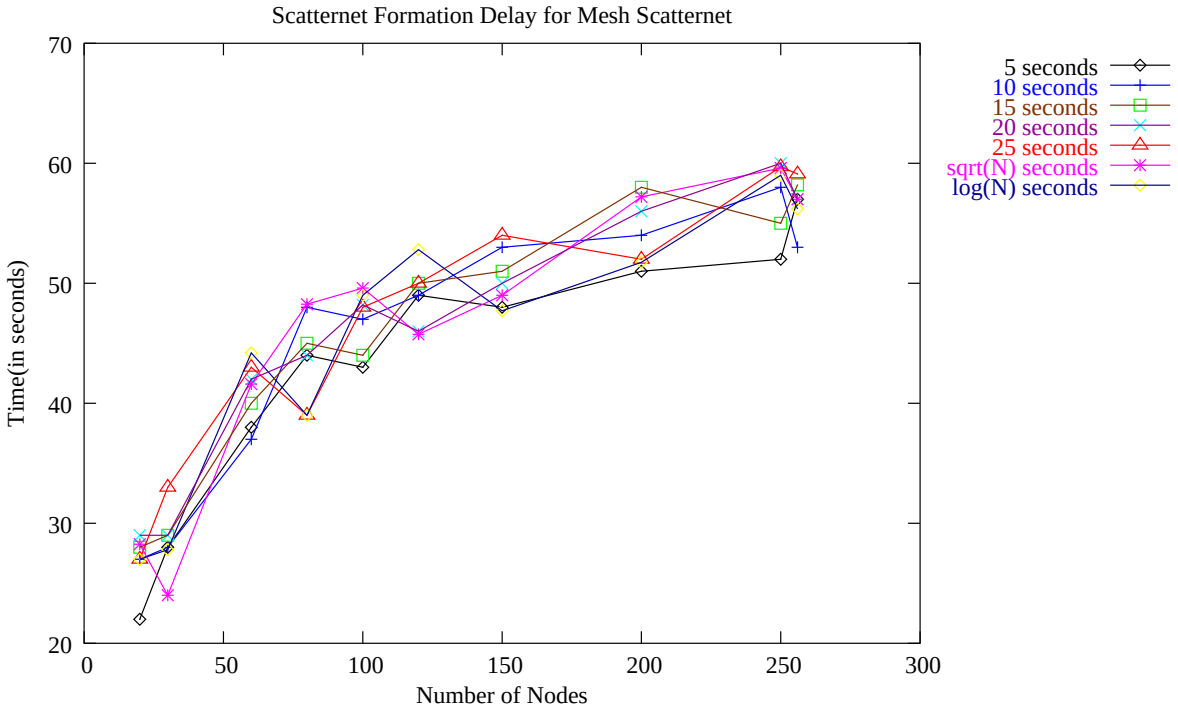


Figure 5.2: Mesh Scatternet - Formation Time

However, there are a few limitations to the Blueware simulator. The simulator does not implement a reasonable interference model. However, when evaluating topology formation schemes, a lack of interference model does not significantly impact the results when the node density is not too high. Blueware currently does not support synchronous connections (SCO). Further, it does

not support SNIFF and PARK modes for a Bluetooth link, yet. Availability of SCO links and low power modes such as PARK would help Blueware closely follow the Bluetooth specifications and provide even more realistic implementation of Bluetooth specifications. However, these shortcomings do not invalidate or deviate the results obtained from their real world implementations by a significant amount.

5.2.2 Simulation Parameters

We used Blueware 1.0 to create an ad-hoc network of Bluetooth devices, varying the number of nodes N as 20, 30, 60, 100, 150, 200, 250, 256. Each data point reported is an average of results obtained through 10 different simulation runs with different seeds. The power class 3 Bluetooth (BT) nodes (i.e. nodes with a maximum transmission radius of 10 meters) are assumed to be randomly distributed in the area of 7.07 meters * 7.07 meters and are stationary during simulations. We assume that all nodes start executing the scatternet formation scheme at the same time, though it is not a requirement of our algorithm. We also tested the effect of different maximum time spent in phase 1 on various performance measures.

We first provide some of challenges faced during Blueware implementation followed by the simulation results obtained for Mesh topology and then move on to similar experiments for the Cube topology. We then evaluate both with respect to other scatternet formation schemes based on the performance measures discussed above. We then provide the purpose for this research by analyzing the advantages of using the proposed scatternet formation schemes. We also compare the mesh and cube schemes and provide scenarios when each one should be preferred over the other.

5.2.3 Blueware Implementation

In this section, we discuss some of the challenges faced during the Blueware implementation of our algorithms for mesh creation.

As explained in chapter 2, Bluetooth device discovery is a time consuming process. It does

take time for two or more independent components i.e. piconets or scatternets to discover each other. In order to expedite this process of device discovery, in each component, we assign more than one device to search for neighboring components. We introduce the notion of a *coordinator* node for each component which is chosen by the leader of the component. The job assigned to the coordinator is to discover neighboring devices and inform about them to its own leader. The leader can then form a connection link with the component leader discovered by the coordinator. There is a certain time penalty that is added in choosing a node as a *coordinator*, however this penalty is compensated the fact that the device discovery happens faster. Along with the time penalty, we also face some scheduling problems when the leader of the component and the coordinator both discover two different components. Hence, the leader receives two connection requests simultaneously, one for the component it has discovered and the other one for the component discovered by its *coordinator*. At this time, we have to give preference to one of the connection requests and discard another. A coordinator continuously performs INQUIRY SCAN as opposed to a random choice of INQUIRY or INQUIRY SCAN by the leader node. Hence, it may happen that a leader node and the coordinator of the same mesh, may discover itself and try to form a connection; however, we need to discard such connections.

Blueware puts a slave link in a piconet into the HOLD mode as soon as a connection is formed. However, when forming a mesh, we need to use the existing links to make further connections and there is communication involved on the links formed through the bridge nodes. Hence, the process of communication is delayed till the link becomes active. More than one communication event may be pending till this time, which may be dependent on each other, and may fail to happen when the link becomes active. This is one of the biggest implementation problems faced as far as Blueware simulator goes.

When there are two independent mesh scatternets present in the network, the leader of each mesh is required to discover the other and form a temporary connection link with it through which enough information can be exchanged so as to determine the way in which merging can happen.

Once the exchange of information takes place, we no more need this connection link between two leaders if the mesh structure does not require it. For this connection link to establish, we need to leave one slave slot open in each master node / piconet. If we allow the piconets to acquire 5 dedicated slaves in phase 1, then with the addition of 2 bridge nodes, the piconet would be complete and so, in order to form another temporary slave link with the leader of other mesh, we would need to PARK one of the dedicated slaves. This is necessary only for the purpose of scatternet formation. The particular slave can be unparked immediately after the scatternet formation is over. Blueware does not allow for *parking* of slaves and hence, we had to dedicate a slave slot for this final connection reducing the maximum number of slaves in every piconet in phase 1 to 4 as compared to a possible maximum of 5 slaves. By *parking* one of the acquired slaves in phase 1 temporarily, a master can essentially complete its piconet with 7 slaves, however, due to simulator limitations, our results show a maximum of 6 slaves (including bridge nodes) in every piconet. Hence, the number of piconets tend to be higher than the expected lower bound of mesh.

A mesh connection needs more than one connection to happen when connecting two components. These connections include devices which have retired after becoming a part of the bigger mesh. Hence, in order to form connections, we need to provide these devices with the addresses of the device to connect to and their clock information. This way, we can ask a device to page another device with the specified address. However, a connection is not guaranteed to be formed. This connection might be needed in order for formation of few more connections and hence, another problem that we faced is, the need to serialize the way in which the connections take place. Failure of a connection could ruin the formation and growth of mesh. However, serialization of events is a very difficult task in distributed computing such as the one we are employing.

Our experiments show that the final connection in phase 3 of the algorithm takes longer time than expected. Essentially, it is an operation of *directed paging* and a connection that is expected to be established quickly. However, we see that this connection takes a long time. The leaders of each of two meshes connect and disconnect with their own chosen coordinators many times during

this span, but, the two meshes do not get in touch with each other for a long time.

5.2.4 Simulation Results

Mesh Scatternet :

Scatternet Formation Delay

We study the time required to form a single mesh scatternet with varying time bound on phase 1 by all the devices. We vary the maximum time for phase 1 from 5, 10, 15, 20, 25 seconds and also for log(N) and sqrt(N). Figure 5.2 shows the results for the experiments performed. We can see that on an average, the time required for scatternet formation increases logarithmically irrespective of the maximum time spent in phase 1. However, the results also point out that the total time taken for scatternet formation is in a large part the time spent by the devices in phase 2 and hence, varying the maximum time for phase 1 does not show a great effect on overall time taken.

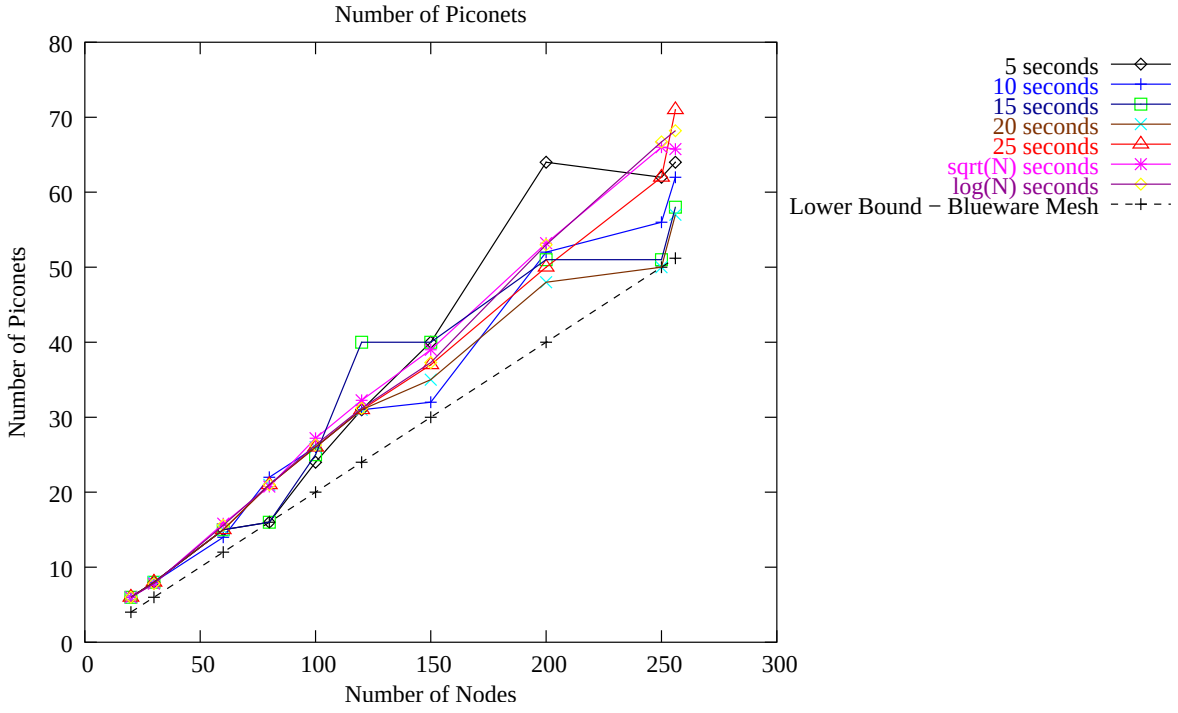


Figure 5.3: Mesh scatternet - Number of Piconets

Number of Piconets

Figure 5.3 shows the number of piconets with varying number of nodes in the generated mesh. We also measured the effect of the maximum time spent in phase 1 on the number of piconets. We find that when the nodes spent lesser time in phase 1, they tend to form a higher number of piconets for a few of the experiments as expected. However, this result is not uniform. The reason lies in the fact that the reduction in the number of piconets can be done even in phases 2 and 3 and is not limited to only phase 1. Hence, the number of piconets do not solely depend on the maximum time spent in phase 1.

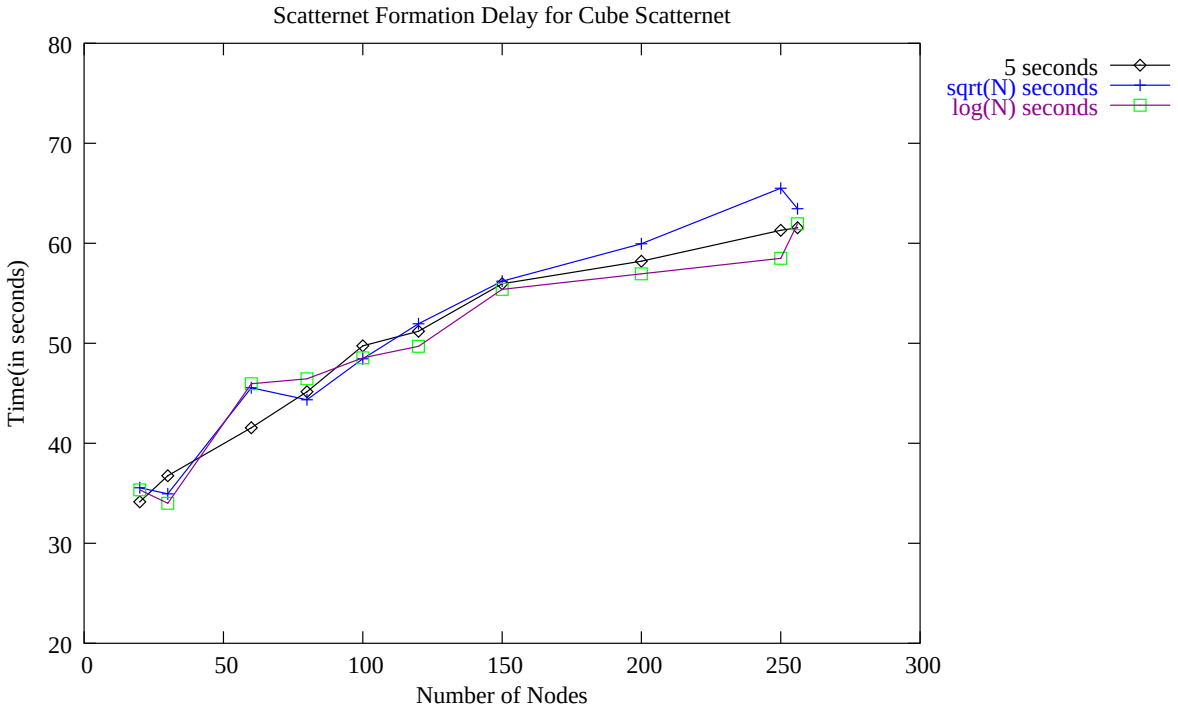


Figure 5.4: Cube scatternet formation delay

Further, our experiments show that in phase 1, most of the piconets acquire a minimum of 2 slaves quickly. After that, even if these 3 node piconets, contact each other and get connected, they can not be merged into a single piconet due to our forced limitation of a maximum of 4 slaves in a piconet for phase 1 which allows for mesh growth in phase 2 keeping 2 slave connection open.

Hence, irrespective of the time spent in phase 1, there is no more slave acquisition that takes place, leaving the number of piconets same after 5 sec till 25sec of time spent in phase 1. Hence, we see that the changing the bound on time spent in phase 1, does not affect the number of piconets and also the total time taken to form a scatternet a great deal.

We expect the number of piconets to be even lower when the simulator limitations are overcome. However, the simulation results closely match the expected lower bound for meshes formed with Blueware simulator with its limitation as shown in Figure 5.3.

Cube Scatternet :

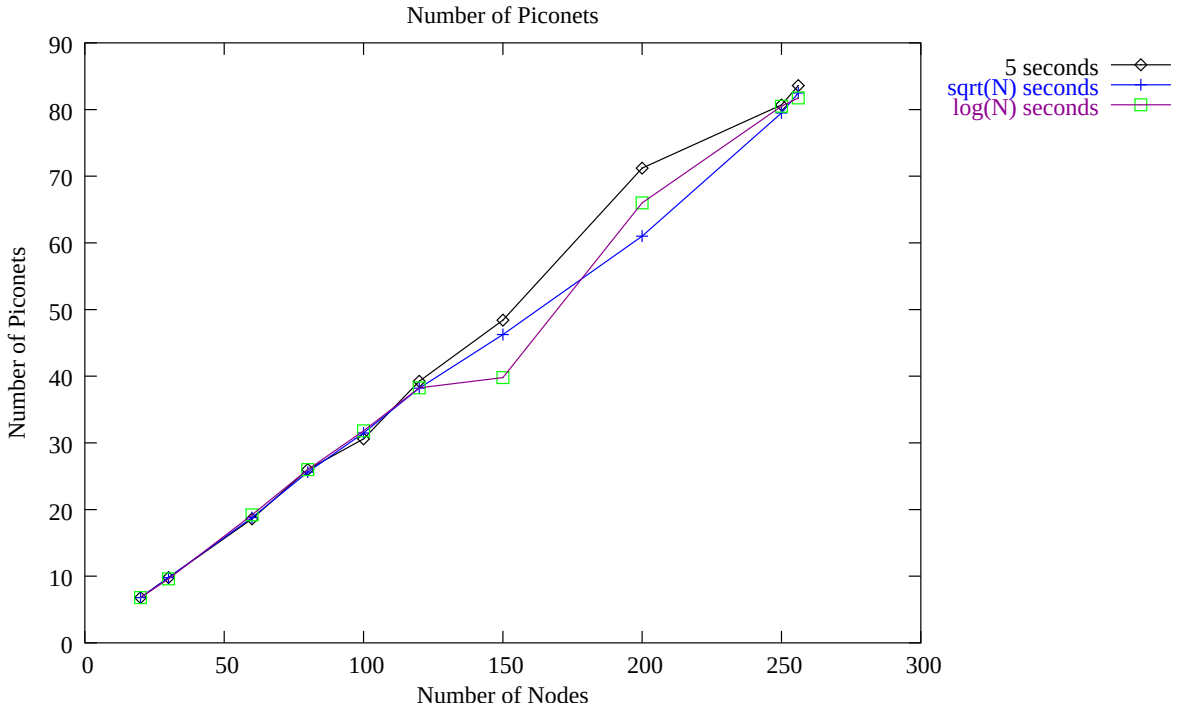


Figure 5.5: Cube Scatternet - Number of Piconets formed with varying bound on time spent in Phase 1

Scatternet Formation Delay

As shown in Figure 5.4, we study the effect of various maximum times for phase 1 and the number of nodes on the total time taken for scatternet formation. For the sake of clarity, we omit

the results for other maximum time bounds for phase 1 and provide results for the maximum time spent in phase 1 as 5sec, sqrt(N) and log(N). However, the results are similar to those obtained for mesh scatternet. We can see that the overall scatternet formation delay for cube scatternet increases, again logarithmically. However, similar to mesh scatternet, we see that the bound on the time spent in phase 1, has little effect on total time taken to form the cube scatternet. This can be explained since in phase 2, we make many more connections than in phase 1. Also that we have a lot more restrictions on the way connections can happen and the number of devices doing device discovery also reduces in phase 2. Hence, phase 2 of the algorithm would take more time than phase 1. Experimental results support this expectation.

Number of piconets

We plot the number of piconets, with increasing number of nodes as shown in Figure 5.5. We can see that the number of piconets increase linearly as the number of nodes increase which is as expected. We also observe that there is a very little effect of the bound on time spent in phase 1 on the number of piconets. This is similar to the results obtained for the mesh scatternet formation scheme and can be explained on similar grounds. Due to simulator limitations, we have to restrict the number of non-bridge slaves in a piconet to 3. Overcoming the limitations will help us improve on the number of piconets further.

5.2.5 Comparison of Mesh and Cube Scatternets

We can see that the time spent in phase 1, has no impact on the scatternet formation time and the generated number of piconets. Empirical evidence suggests that the time spent in phase 1, has no impact on the quality of the scatternet formed. So, for comparison purposes, we set the maximum time spent in phase 1 to the minimum of the selected values i.e., 5 sec and provide results for the same.

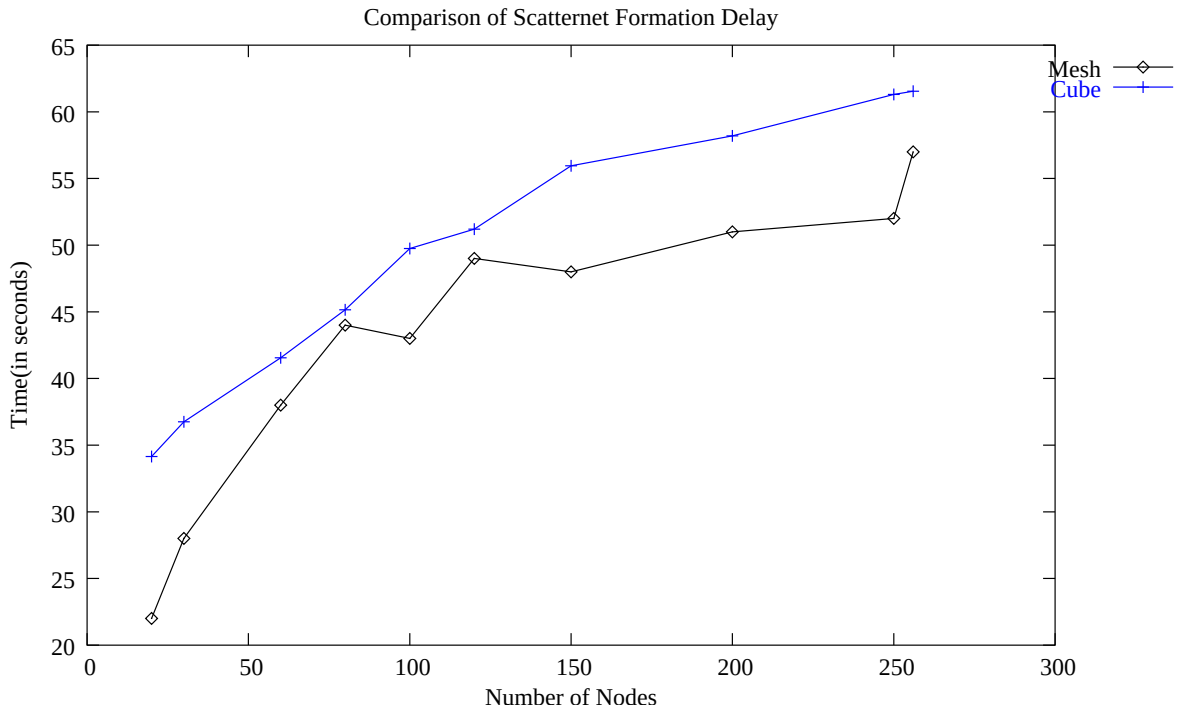


Figure 5.6: Scatternet formation Delay Mesh v/s Cube

Figure 5.6 shows the comparison of the scatternet formation delay for mesh and cube scatternets with varying number of nodes. We can see that for all the nodes, the time taken by a cube scatternet is a little higher but comparable to the mesh scatternet. This ran counter to our expectation that the cube needs more formation time than a mesh. Experimental results show that the time taken for cube is not significantly higher than meshes. One of the reasons for this is the time taken for *device discovery* dominates the total time for scatternet formation. Once a device discovers another device and finds that it can form a connection link with this neighbor, the connection establishment happens quickly. Paging a discovered node does not cost as much time penalty as much a new connection establishment since new connection establishment also adds the delay for discovery of another node. A leader continues to search other devices when its mesh members are forming connection links. Hence, the connection establishment happens in parallel with the device discovery process. So, cubes may need more connections than a mesh for same number of nodes, but it needs similar time for device discovery as long as the number of devices remain the same.

Hence, the closeness in formation time.

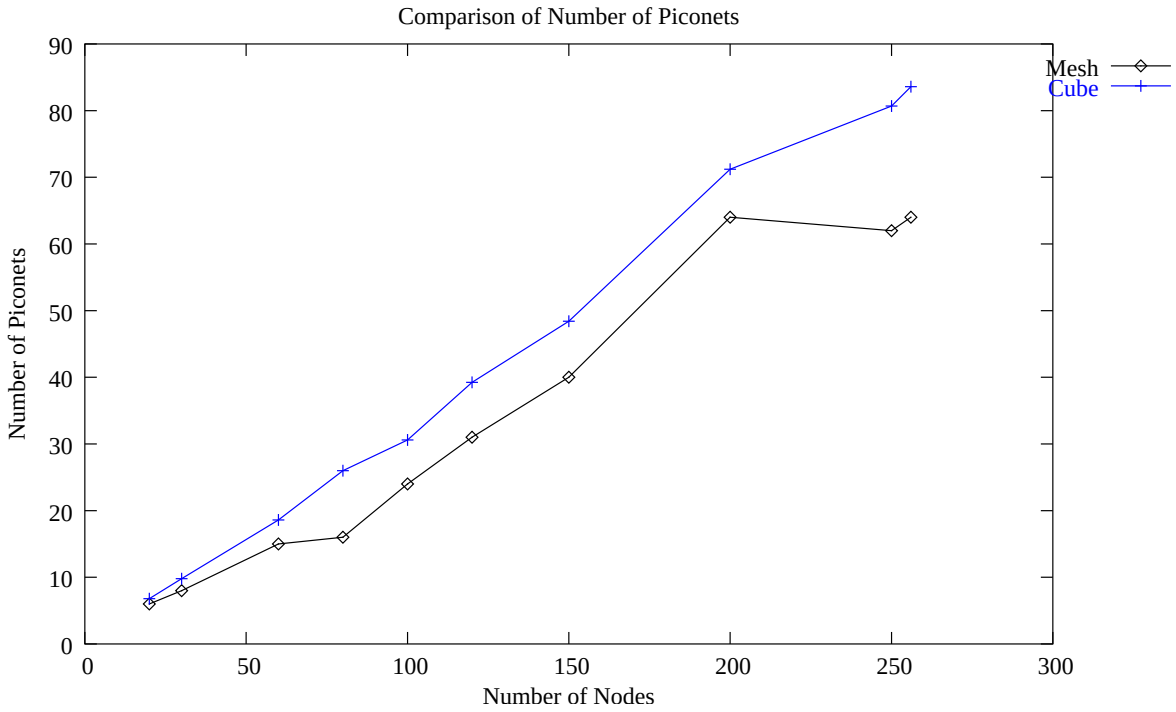


Figure 5.7: Number of Piconets - Mesh v/s Cube

Figure 5.7 compares the number of piconets for both mesh scatternet and the cube scatternet. We can see that as expected the number of piconets in a cube are always more than that in mesh scatternet. When moving from mesh scatternet to cubes, we expect to pay the penalty of more number of piconets, because of the higher connectivity.

Figure 5.8 shows the piconet density for mesh and cube scatternets with varying number of nodes. As the number of piconets increase when we move from mesh scatternet to a cube scatternet, we see a reduction in piconet density which is expected. We also observe that the difference in piconet densities for mesh and cubes remains constant irrespective of the number of devices in the network.

Figure 5.9 shows the average roles played by a node with varying number of nodes for mesh and cube. We observe that a node on an average plays close to 2 roles in mesh network whereas in the case of a cube, this number goes up a little bit, however not significantly. This can be

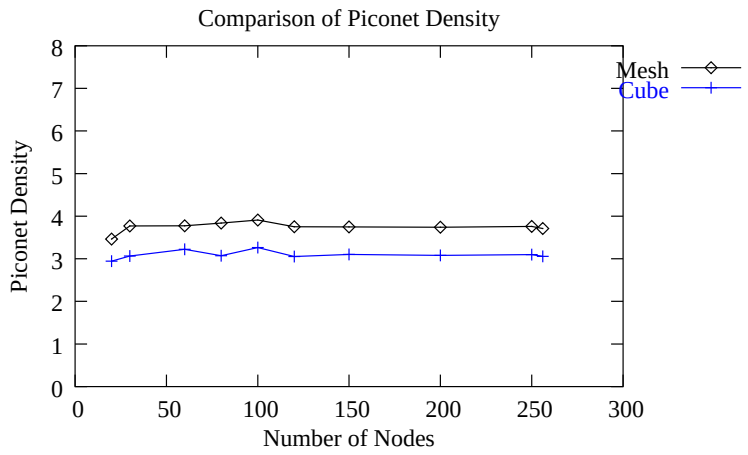


Figure 5.8: Comparison of Piconet Density for Mesh and Cube scatternets

explained as follows. Our scatternet formation algorithms currently require Master-Slave (M/S) bridges. Hence, almost every master of the piconet, is also a slave in other piconet(s). An arbitrary node has to play role of a master in its own piconet and slave in other (two for mesh and three for cube) piconets. However, the dominating number of non-bridge slaves (approximately 2 in every piconet) play the only role of a slave in one piconet. Hence, the average number of roles played by a node reduces significantly. We see that this average increases slightly as we move from 2 dimensional meshes to 3 dimensional cubes and that this difference also seems to remain same irrespective of the number of nodes.

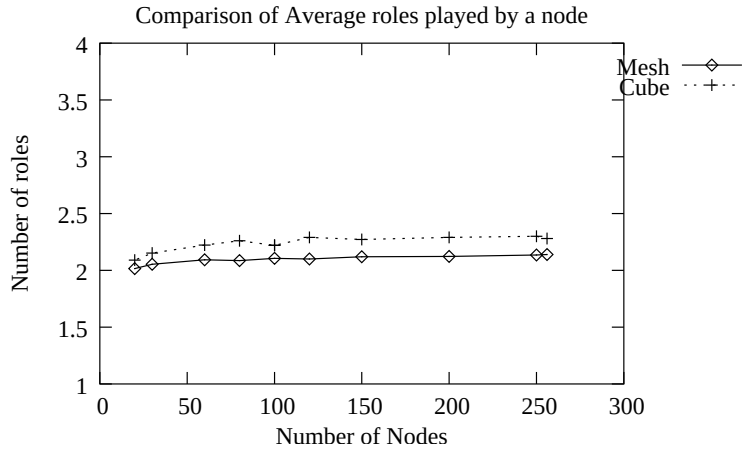


Figure 5.9: Comparison of Average Roles for Mesh and Cube scatternets

This fact can be explained more with the result shown in Figure 5.10. Figure 5.10 compares the percentage degree deviations for both mesh and cube networks. The degree of a node in a mesh network varies from 1 till 3 whereas for a cube network, this degree can be anywhere in the range of 1 till 4. Hence, we can see a higher degree deviation for cube networks as compared to mesh networks which is as per our expectations. This means that degree of nodes vary less in case of mesh scatternet in comparison with cube scatternets.

Figure 5.11 shows the average number of links at a node for mesh and cube. As expected the cube scatternet has higher links on an average than a mesh network. We observe that for smaller number of nodes, the average number of links remain similar for both mesh and cubes but, this gap widens as the number of nodes increase. The average number of links at a node for both mesh and cube remain close to 2 and 2.5 respectively, because of a very high number of non-bridge node slaves which are not shared and hence have just 1 link at them. This brings down the overall average number of links by a significant amount.

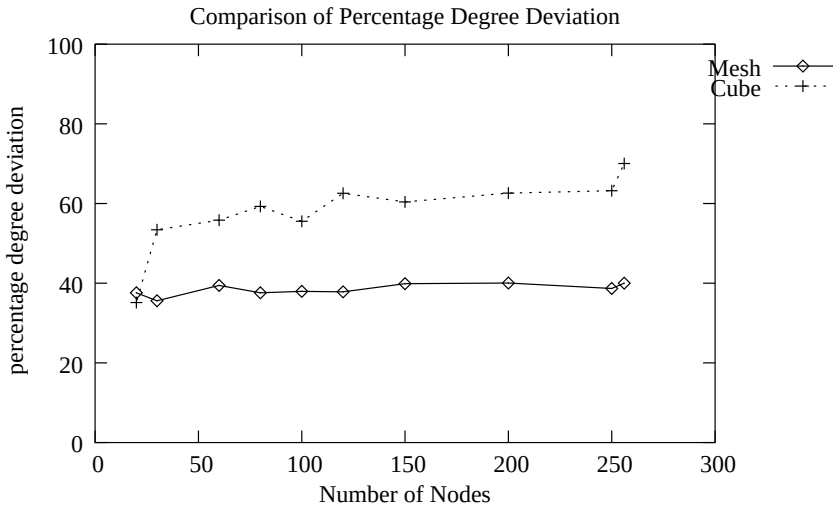


Figure 5.10: Comparison of Degree Deviations for Mesh and Cube scatternets

Figure 5.12 compares the number of nodes that can join the scatternet without the need for reorganization. As explained in Figure 5.8, mesh has piconet density close to 4 and hence, every

piconet can accommodate 4 more nodes. Whereas, in the case of cube networks, the piconet density is close of 3 which means every piconet is capable of essentially allowing 5 more nodes on an average. Hence, we expect the cube to accommodate more such joining nodes as compared to the mesh structure. Empirical evidence supports this expectation. As we move from mesh to cube scatternets, the number of piconets also increase and hence, we see that the network can accommodate for higher number of node joins. When the number of devices is small, both mesh and cube can accommodate comparable number of slave joins. However, for higher number of devices, the cube can accommodate many more slave joins.

Comparison with other scatternet formation algorithms :

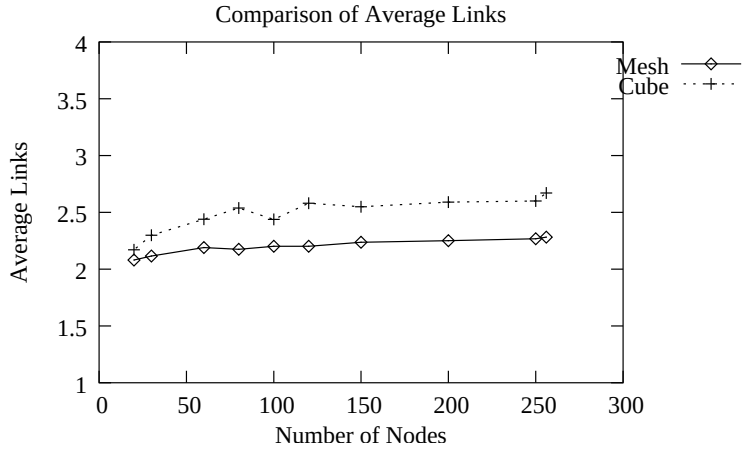


Figure 5.11: Comparison of Average Number of Links for Mesh and Cube scatternets

Figure 5.13 shows the comparison of scatternet formation delays for mesh and cube scatternets with TSF and SF-Devil scatternet formation schemes. These are the only two scatternets which provide the formation times for their scatternets. As expected the scatternet formation delay in case of the mesh and cube scatternets is higher as compared to TSF. Our algorithm performs much better and generates a scatternet in a much shorter time span when compared with SF-Devil. The mesh and cube scatternets are expected to take more time naturally, since they are much more regular

and robust structures, with a higher number of connections, when compared with tree . Along with the fact that the number of connections is higher, we also put some special restrictions on connections in order to maintain the mesh and cube structures and control its growth. We need to make sure that the meshes of equal size connect with each other and form a bigger mesh, whereas no such restrictions are necessary in case of TSF scatternet. Hence, TSF is expected to have a few unsuccessful connections between leaders of individual meshes, before a final successful connection. These reasons cause the time required for scatternet formation to increase. We also would like to note that due to Blueware limitations, we are left with more piconets than the expected lower bound, hence increasing the time for connections accordingly. Overcoming the limitation will help us reduce the scatternet formation delay even further.

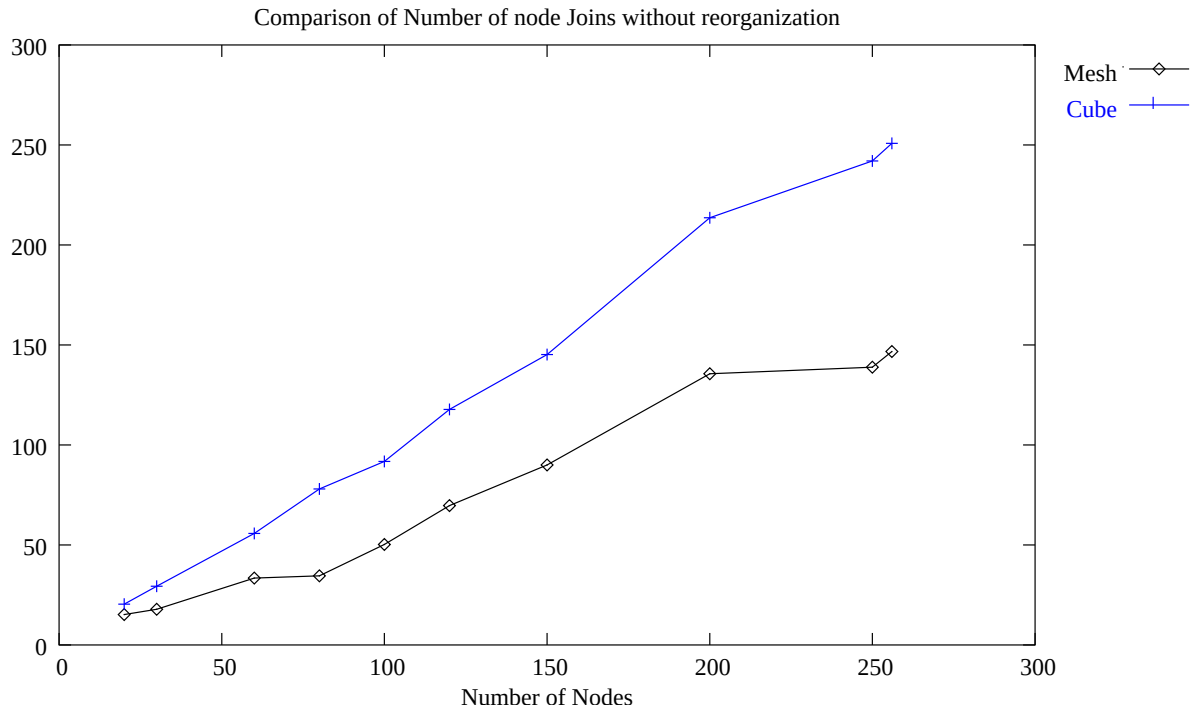


Figure 5.12: Comparison of Number of Node Joins possible without reorganization

Figure 5.14 compares the number of piconets present in the final scatternet for mesh and cube scatternets with several other schemes. We find that irrespective of the number of nodes, both mesh and cube produce a smaller number of piconets when compared with TSF. TSF allows for

minimum connectivity between nodes and does not put any other restrictions of a certain number of slaves in a piconet and hence we see that the number of piconets is very high. There is always a trade-off between the connectivity and the number of piconets. Due to higher connectivity achieved in mesh and even higher in cube, we have fewer slaves in each piconet and hence the number of piconets is slightly higher as compared to the evolutionary scatternet formation scheme which aims at reducing the number of piconets to a global minimum of $\lceil \frac{n-1}{7} \rceil$ and continue to perform iterations of the algorithm till the number of piconets is within certain limit of this number. Hence, they generate a scatternet with very few piconets. However, there is no specific and regular structure imposed on the generated scatternet. We also observe that our mesh scatternet has much fewer piconets when compared with Bluemesh. This is because, Bluemesh algorithm assures connectivity by choosing the gateways such that there is an inter-piconet route between all masters which are 3-hops apart. In order to do that during the gateway selection phase, a common slave is chosen between two piconets. However, when there are no common slaves in neighborhood of two piconets, intermediate gateways are selected which create extra piconets and this process continues till the piconets are connected through a common slave in order to provide connectivity. This increases the number of piconets present in Bluemesh.

Figure 5.15 compares the network diameter for various scatternet schemes in comparison with mesh and cube scatternet formation schemes. The diameter for TSF is obtained by doubling the height of the tree, the metric readily available for comparison. We can see that both mesh and cube have much smaller diameter as compared to TSF. The loop scatternet provides smaller diameter than mesh scatternet however, cube scatternet formation scheme provides least diameter of all. This is because of cube provides for the highest connectivity among all the scatternets presented. For smaller number of nodes, the difference in diameter of the network for all the scatternet topologies is very small; however, as the number of nodes increase more than 50, we see that cube scatternet shows a drastic reduction in the diameter of the network.

Zhang, Hou and Sha [30] prove that the maximum node contention for the TSF is at least

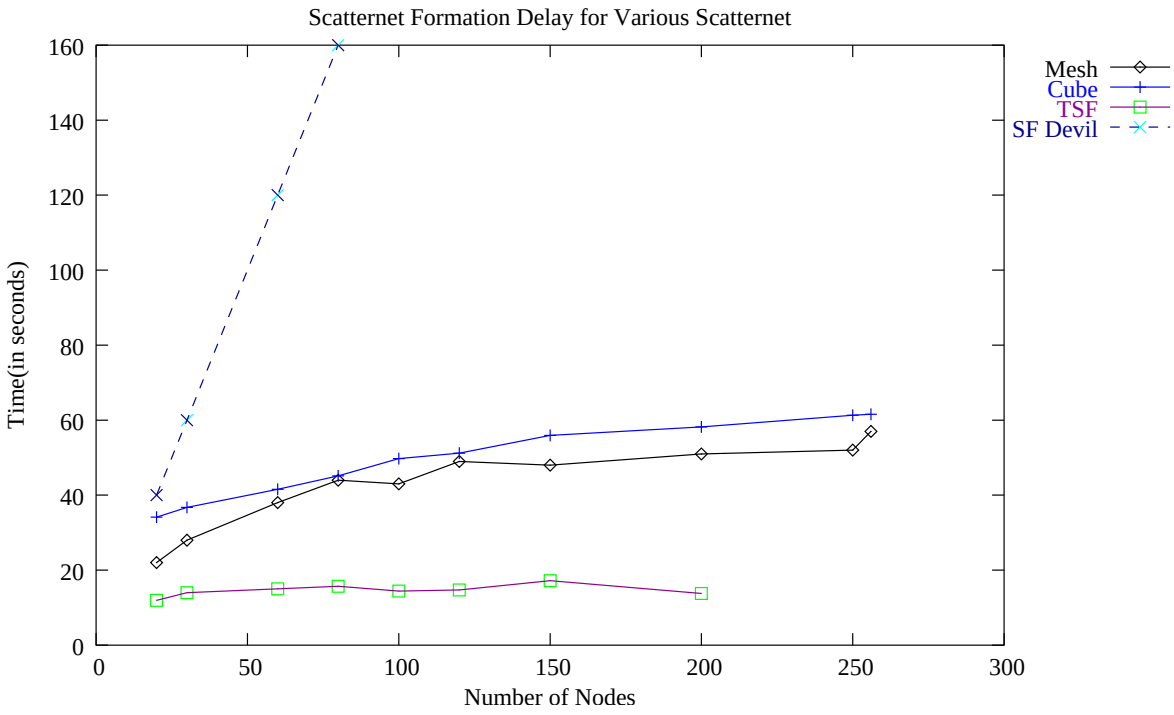


Figure 5.13: Comparison of Scatternet Formation Delays for various scatternet schemes

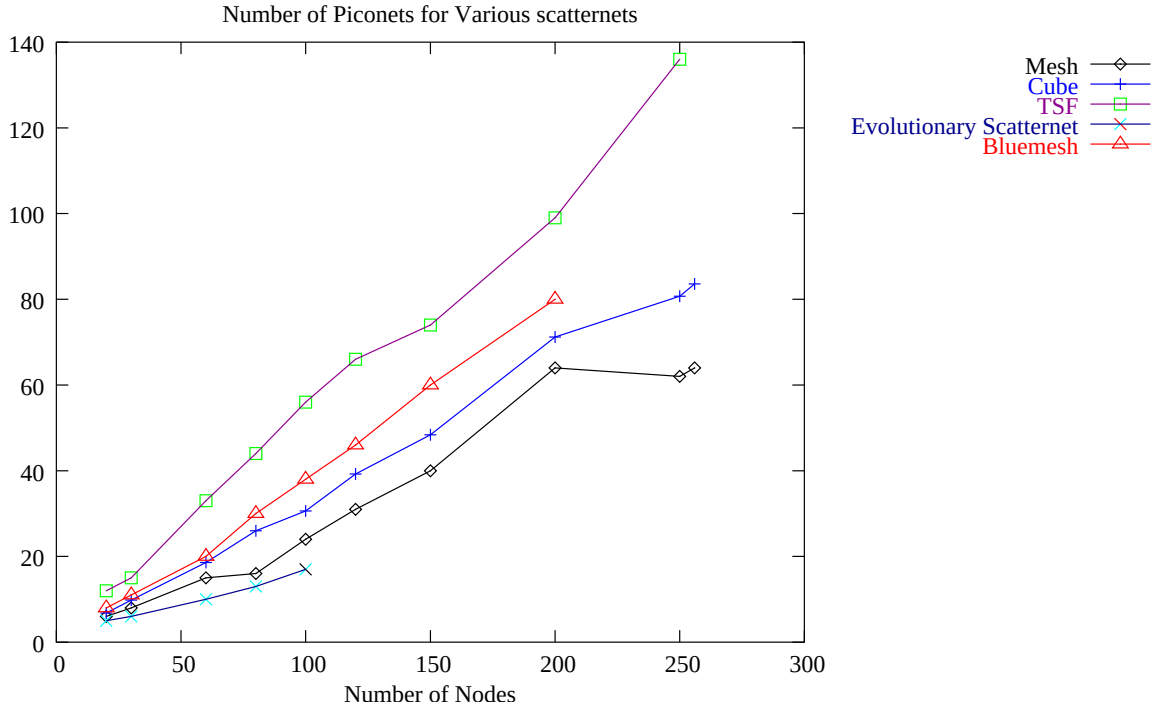


Figure 5.14: Comparison of Number of Piconets for various scatternet schemes

$N^2/4 + N/2 - 1$. As discussed above, we know that tree structure suffers from many bottlenecks which increase the overall node contention. The maximum node contention in the case of the loop scatternet is $\frac{(k-1)^2 P(\sqrt{(P)}+2)}{4}$ where *P* is the number of piconets. This is better as compared to TSF. However, the maximum node contention in the case of a mesh scatternet is $(k-1)^2\sqrt{(P)}$ which is significantly less when compared with loop scatternet and definitely better when compared to TSF. The 3-dimensional meshes reduce the maximum node contention even further to $(k-1)^2\sqrt[3]{P}$. Hence, we can see that both mesh and cube have significantly lower node contention and hence are better suited to handle higher amount of traffic with increasing number of nodes.

5.2.6 Out of Range Devices

For our mesh and cube scatternet algorithm, we assume all the devices to be in range of each other and hence the area in which the devices are randomly distributed is considered to be less than 10 meters in diameter since BT power 3 nodes have a transmission range of maximum 7.07 meters. However, we experimented with increasing the size of this area to 10 and 14 meters allowing some of Bluetooth devices to be out of range of each other. We then run the same algorithm for these devices. Figure 5.17 shows the comparison of how many times, we successfully get a connected mesh scatternet as opposed to TSF.

We can see that mesh and cube scatternets always provide a better chance of connectivity as compared to TSF though the connectivity of the scatternet is not guaranteed always. This is because the mesh and cube algorithms are built with a basic assumption of all in-range devices. In order to guarantee 100 percent connectivity of the scatternet, we need to modify the algorithms which is subject to further research.

5.3 Analysis of Experimental Results

After looking at the experimental results and analyzing them, we can draw following conclusions:

- Mesh and Cube scatternet formation schemes need more time to generate final scatternet as

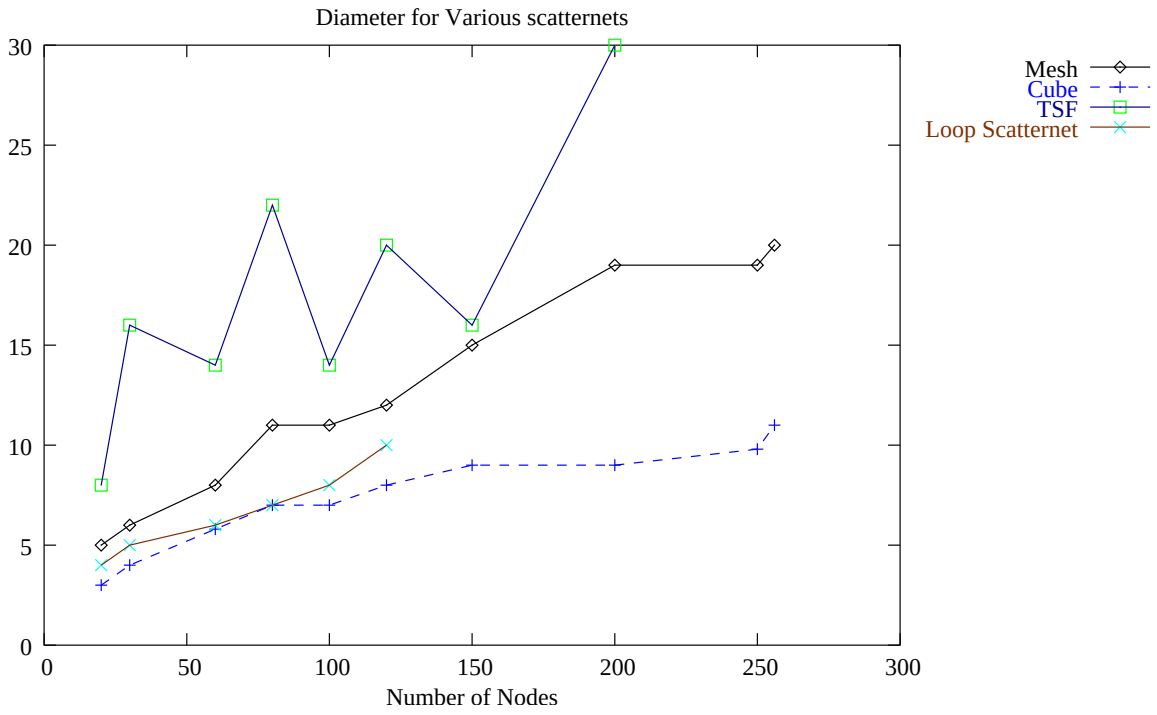


Figure 5.15: Comparison of Diameters for various scatternet schemes

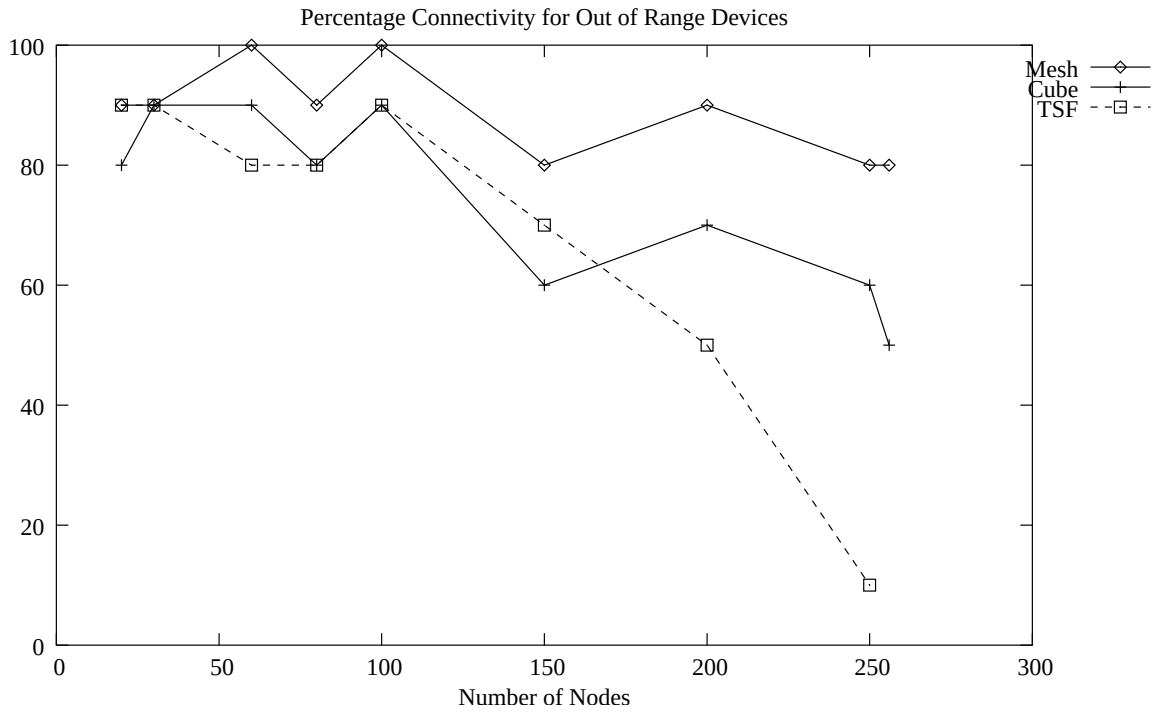


Figure 5.16: Results for out of range devices, Range = 10 meters

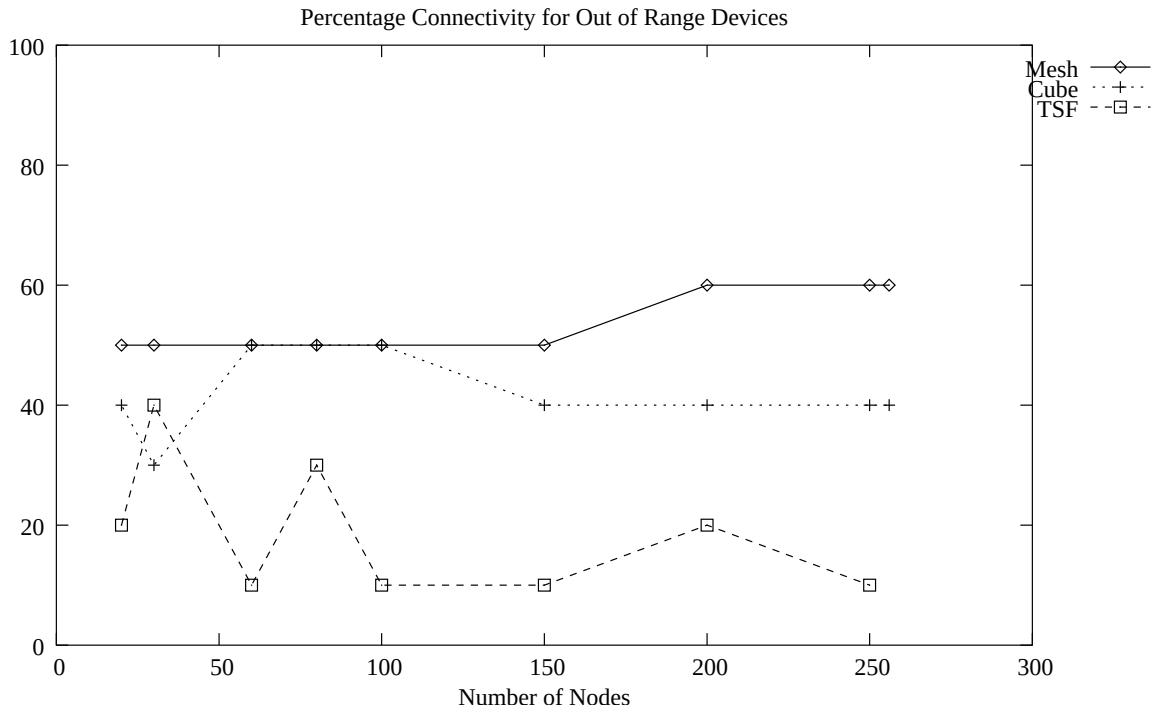


Figure 5.17: Results for Out of Range Devices, Range = 14 meters

compared with a few other scatternet formation schemes which provide lower connectivity. However, that is the price paid to get a highly connected and regular structure with several desirable properties. The scatternet formation delay is still comparable to other scatternet formation schemes and is acceptable. We believe that the time required for scatternet formation for mesh as well as cube can be improved further once we overcome the limitations posed by Blueware simulator.

- The diameter provided by mesh and cube structures are smaller compared to other scatternet topologies; in fact cube provides the least diameter of all the scatternets so far. This is a highly desirable property since it reduces the maximum number of hops between two devices in the network which reduces the communication delay in the network. The reduction in delay results in improvement in throughput.

- The connectivity provided by robust structures like mesh and cube is very high which reduces the maximum node contention in the network drastically. This has two-folded use as with connectivity comes resistance to network partitioning and also provides multiple paths between any two nodes reducing network congestion. We believe that meshes and cubes can handle a very high amount of traffic and with a properly chosen routing scheme can balance the traffic flows in the network. This also helps to improve on the overall throughput of the network.

- We see that the current scatternet formation scheme for mesh and cube makes use of M/S bridges. The scheme can be easily modified to implement Slave/Slave (S/S) bridges. We can reduce the average number of roles for a node by replacing the M/S bridges by Slave-Slave (S/S) bridge between two piconets. In this case, a master node would not be shared between two piconets and hence would be performing the role of a master only. A slave node would have to perform more than one role.

- Both mesh and cube can accommodate a very high number of node joins. As shown in Chapter 4, both can handle node leaves elegantly. Hence, the structures are well suited for dynamic environments with highly mobile nodes.

From the simulations, we have shown that complex and regular structures like mesh and cube can be successfully generated for Bluetooth networks. Both, meshes and cubes provide us with a numerous other highly desirable properties: they have low network diameter and node contention compared to other scatternet schemes. They also provide multiple paths between any nodes and prevent single point of failures. The scatternets can also handle nodes leaving and joining the network without significant reorganization. For all these desirable properties, we pay the price of slightly higher scatternet formation time.

The meshes get formed quicker compared to cubes. They also provide fewer piconets in comparison to cubes. However, the cubes provide lower diameter and lower node contention. This

advantage provided by cubes becomes significant for higher number of Bluetooth devices. When the number of devices in the network are less, the meshes should be preferred over cubes. However, as the number of devices increase, cubes should be preferred over meshes. Similarly, we also see that when a few devices could be out of range of others, meshes have a higher chance of forming a single scatternet. Hence, metrics such as number of devices and area in which these devices are present, should be taken into account when choosing the scatternet topology.

CHAPTER SIX

CONCLUSIONS AND FUTURE WORK

6.1 Conclusion

In this thesis, we studied the problem of scatternet formation for Bluetooth Personal Area Networks. Essentially, the scatternet formation is the problem of assignment of roles to Bluetooth devices. A Bluetooth device can act as a Master, Slave or a Gateway between two piconets. Hence, deciding which device performs which role so as to form a single connected network of these devices is the problem of scatternet formation. There are two cases that can be considered; when all devices are within the transmission range of each other and when all devices are not within the transmission range of each other. We focused our research on the first case, i.e. when all Bluetooth devices are within the transmission range of each other. A good scatternet formation algorithm, for this case, is expected to provide additional desirable properties such as smaller scatternet formation delay, fewer piconets, low diameter, minimum node contention, multiple paths between any two nodes, higher connectivity, and ability to handle dynamic node leave and join besides the minimum requirement of forming a connected scatternet. However, some of the properties are contradictory to each other and pose a number of trade-off situations when building a scatternet out of Bluetooth devices.

Based on these facts, we proposed two completely distributed scatternet formation schemes, *Mesh* and *Cube* scatternet algorithms, in this thesis. The algorithms consist of three conceptual phases which use a bottom-up approach to generate a single scatternet from a number of devices. In the first phase, we group individual devices together into piconets. Phase 2 creates bridges between these piconets with certain restrictions in order to control growth of mesh and third phase is used in order to assure the formation of a single mesh in the network. In order to allow for piconets to interconnect in a mesh structure in further phases, the number of slaves in any piconet

are restricted to k - 2. Any master which has acquired k - 2 slaves, moves on to phase 2. However, since this condition cannot be assured at all the masters to phase 2, we also put a maximum time bound on the time spent by each master in phase 1. We expect that the longer the time spent in phase 1, lesser the number of piconets and vice versa. Similarly, we expect that the total scatternet formation delay would reduce if we spend lesser time in phase 1 and vice versa. Hence, we have experimented with different time-out periods for phase 1 to observe their impact on the total quality of the scatternet. In phase 2, we connect equal sized meshes to connect to each other and grow them into a bigger meshes allowing connections in both x and y directions. In phase 2, we also keep track of the disconnections that happen. Phase 3 is used as a wrap-up phase and connects all the meshes in the scatternet into a single mesh with the aim of minimizing the diameter. Extending the same idea to three dimensions, we can form a *cube* scatternet. While forming the cube scatternet, we have to an additional dimension while allowing connections between piconets. Hence, we restrict the number of devices in every piconet in phase 1 to k - 3. Similarly in phase 2, the piconets are connected in all three x, y and z directions. Phase 3 generates a single cube scatternet with minimum diameter.

Both algorithms guarantee formation of a single connected scatternet when the devices are within transmission range of each other. We find the both mesh and cube have a very symmetric structure which provide many desirable properties such as in-built routing, multiple paths between any pair of nodes. Since these scatternets provide more paths than any other suggested scatternets so far, we find that the maximum node contention is reduced to minimum when compared to other topologies suggested so far. The scatternets provide low diameters and also high connectivity. Hence, the scatternets do not suffer from single point of failure; in fact the cases of node leave and node joins can be handled very elegantly.

Simulation studies were performed to test the suitability of the proposed scatternet formation algorithms. We used the Blueware simulator, developed at MIT which models Bluetooth Specifications 1.1 closely. We measured the time required for scatternet formation varying the maximum

time spent in phase 1 along with the number of piconets in the generated scatternet. We find when the maximum time spent in phase 1 is varied, it has a very small impact on the total scatternet formation delay. We find that this result is due to the dominating device discovery time for Bluetooth devices. Both mesh and cube scatternets require similar amount of time for formation irrespective of the maximum time spent in phase 1. We also find that the number of piconets generated at the end, do not solely depend on the time in phase 1, mainly because of the reduction in number of piconets continuing in phase 2 and phase 3.

We compared our algorithms with various other scatternet formation techniques based on the identified metrics such as the scatternet formation delay, number of piconets, network diameter, maximum node contention, average number of paths between any two nodes in the network etc. We find that TSF scheme requires lesser scatternet formation time when compared with our algorithms. This is as expected: since meshes and cubes are regular structures with higher connectivity and we put a number of restrictions on the way two piconets / scatternets connect. SF-DeviL is the only other scatternet formation technique (of all the other scatternet formation schemes) which reported the scatternet formation time. We note that the algorithms require much lesser formation time than SF-DeviL.

Comparing the number of piconets generated for mesh and cube with others, it is clear that both mesh and cube algorithms generate higher number of piconets than the lower bound. This is due to the restriction of a maximum of $k - 2$ slaves for meshes and $k - 3$ for cubes in a piconet in phase 1. However, we find that both the algorithms always generate lesser number of piconets as compared to TSF and Bluemesh for any number of nodes. Our simulation results show that mesh and cube scatternets provide lower diameters as compared to TSF; in fact Cube provides the least diameter of all the scatternet structures suggested so far. We find that due to higher number of piconets, both the scatternets can accommodate large number of dynamic node joins without the need for any reorganization. The node leave can also be handled very elegantly. The price paid to get such desirable properties is a little higher scatternet formation time. However, we find this price worth

paying for, due to other immense benefits towards Bluetooth scatternet performance. As a result, we provide the desired properties of keeping the network diameter and network contention close to minimum and also provide for in-built routing and scheduling.

Mesh scatternet provides smaller number of piconets when compared with the cube scatternet. However, cube provides smaller network diameter and also reduces the node contention. For smaller number of devices, this difference is not significant and hence, we find that the price of higher number of piconets for cube, is not worth paying for, in the case of smaller number of devices (less that 100). However, as the number of devices increase to more than 100, the cube provides significantly lesser node contention and network diameters compared to a mesh and hence, we suggest the use of cube scatternet when the number of devices in the network is high.

This research work proves that despite the time consuming and asymmetric nature of device discovery process in Bluetooth networks, it is possible to create regular and robust structure like *mesh* and *cube*. The generated scatternets due to their regularity and symmetry, not only allow for quicker in-built routing but also allow for easier scheduling. Though there is some penalty paid in terms of the time required for scatternet formation, the penalty is not much and is worth paying for. In today's state of art, the problems of scatternet formation, scheduling and routing are seen as three separate problems and are approached at in different ways. However, this work makes it evident that with the creation of regular structures, these problems need no longer be considered as separate, in fact should be considered as three inter-related problems which can be solved with proper choice of topology for scatternet. More and more efforts should be put in, to form symmetric and regular structures, which will help Bluetooth technology to overcome its challenges.

6.2 Future Work

There are several avenues for future research directions. For the scatternet formation purposes, we currently make use of the Master/Slave (M/S) bridges between any two piconets. Every bridge performs the role of a master in its own piconet and a slave in two other piconets. A M/S bridge adds certain synchronization delay due to role switching for bridges. Hence, it is desirable to replace to replace these M/S bridges with S/S bridges. This will facilitate better intr-piconet scheduling and reduce the communication delay further improving the overall throughput. We expect a few simple modifications to the current algorithm would allow us convert M/S bridges into S/S bridges.

The routing on a perfect mesh is a trivial issue. However, when after several nodes leave the scatternet, incomplete meshes are formed. The routing in these meshes has to be handled in a different way and the previous trivial routing algorithms do not apply anymore. We need to find out such a routing strategy. Similarly, an intra-piconet as well as inter-piconet scheduling scheme is also needed. We plan to work on these aspects of both mesh and cube scatternets.

Considering the reduction in diameter and increase in connectivity achieved when moving from 2-dimensional meshes to 3-dimensional cubes, we are interested in moving to higher dimensions. However, that is possible only with the presence of M/S bridges due to the restriction of a maximum of 7 slaves in each piconet. We expect that as we move to higher dimensions, we will have higher scatternet formation delays and higher number of piconets, a price paid in exchange for still lower diameters and lesser node contentions.

We also think that one more regular structure hexagon, could be a suitable topology for scatternet. A hexagon has many desirable properties such as easier routing, multiple paths. It would be interesting to compare performance of a hexagonal scatternet with mesh and cubes.

BIBLIOGRAPHY

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

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

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

[4] BLUETOOTH 1.1 Connect Without Cables, Jennifer Bray and Charles F. Sturman, Second Edition, Upper Saddle River, New Jersey, Prentice Hall, 2002.

[5] Bluetooth Demystified, Nathan J. Muller, New York, McGraw Hill, 2001.

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

[7] 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, January 2003.

[8] C. Chiasserini, M. Marsan, E. Baralis and P.Garza, *Towards Feasible Topology Formation Algorithms for Bluetooth-based WPANs*, 36th Hawaii International Conference on System Science, Hawaii, January 2003.

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

[10] Y. Dong and J. Wu, *Three Bluetree Formations for Constructing Efficient Scatternets in Bluetooth*, Proceedings of the 7th Joint Conference on Information Sciences, September 2003, pp. 385-8.

[11] J. Ghosh, V. Kumar,X. Wang, C. Qiao, *BTSpin - Single Phase Distributed Bluetooth Scatternet Formation*, Technical Report, University of Buffalo, New York.

[12] M. Kalia, S. Garg and R. Shorey, *Scatternet Structure and Inter-Piconet Communication in the Bluetooth System*, IEEE National Conference on Communications 2000, New Delhi, 2000.

[13] 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, New orleans, Lousiana, March 2003.

[14] M. Kazantzidis, *Locally Optimal Bluetooth Scatternet Formation*, Technical Report 010033, UCLA Computer Science WAM Lab.

[15] C. Law and K. Siu, *A Bluetooth Scatternet Formation Algorithm*, IEEE Symposium on Ad Hoc Wireless Networks 2001, San Antonio, Texas, November 2001.

[16] 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), August 2000, pp. 147-8.

[17] 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.

[18] C. Petrioli and S. Basagni, *Degree-Constrained Multihop Scatternet Formation for Bluetooth Networks*, IEEE Transactions on Computers, Special issue on Wireless Internet, September 2002.

[19] C. Petrioli, S. Basagni and I. Chlamtac, *Configuring Bluestars : Multihop Scatternet Formation for Bluetooth Networks* IEEE Transactions on Computers, Special issue on Wireless Internet, 2002.

[20] 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, Boston, MA, August 2000, pp. 54-63.

[21] T. Salonidis, P. Bhagwat, L. Tassiulas and R. LaMaire, *Distributed Topology Construction of Bluetooth Personal Area Networks*, Proceedings of the IEEE INFOCOM 2001, Anchorage, Alaska, April 2001.

[22] W. Song, X. Li, Y. Wang and W. Wang, *dBBlue : Low Diameter and Self-routing Bluetooth Scatternet*, Mobile and ad-Hoc Networking and Computing(MobiHoc) 2003, pp. 22-31.

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

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

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

[26] G. Tan, A. Miu, J. Guttag and H. Balkrishnan, *An Efficient Scatternet Formation Algorithm for Dynamic Environments*, MIT Technical report, MIT-LCS-TR-826, MIT Technical Report, October 2001.

[27] G. Tan, *Blueware : Bluetooth Simulator for ns*, MIT Laboratory of Computer Science.

[28] Z. Wang, R. Thomas, Z. Haas, *Bluenet - Scatternet Formation to Enable Bluetooth-Based Ad Hoc Networks,* Proceedings of IEEE ICC 2001, Helsinki, Finland, June 2001.

[29] G. Zaruba, S. Basagni and I. Chlamtac, *Bluetrees- Scatternet Formation to Enable Bluetooth-based Ad Hoc Networks,* IEEE International Conference on Communications,June 2001,volume 1,pp 273-7,.

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

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