2009

A PROTOCOL SUITE FOR WIRELESS PERSONAL AREA NETWORKS

Karl E. Persson
University of Kentucky, karl@cs.uky.edu

ABSTRACT OF DISSERTATION

Karl E. Persson

The Graduate School
University of Kentucky
2009

A PROTOCOL SUITE FOR WIRELESS PERSONAL AREA NETWORKS

ABSTRACT OF DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By
Karl E. Persson
Lexington, Kentucky
Director: Dr. D. Manivannan, Associate Professor of Computer
Science
Lexington, Kentucky
2009

ABSTRACT OF DISSERTATION

A PROTOCOL SUITE FOR WIRELESS PERSONAL AREA NETWORKS

A Wireless Personal Area Network (WPAN) is an ad hoc network that consists of devices that surround an individual or an object. Bluetooth® technology is especially suitable for formation of WPANs due to the pervasiveness of devices with Bluetooth® chipsets, its operation in the unlicensed Industrial, Scientific, Medical (ISM) frequency band, and its interference resilience. Bluetooth® technology has great potential to become the de facto standard for communication between heterogeneous devices in WPANs.

The piconet, which is the basic Bluetooth® networking unit, utilizes a Master/Slave (MS) configuration that permits only a single master and up to seven active slave devices. This structure limitation prevents Bluetooth® devices from directly participating in larger Mobile Ad Hoc Networks (MANETs) and Wireless Personal Area Networks (WPANs). In order to build larger Bluetooth® topologies, called scatternets, individual piconets must be interconnected. Since each piconet has a unique frequency hopping sequence, piconet interconnections are done by allowing some nodes, called bridges, to participate in more than one piconet. These bridge nodes divide their time between piconets by switching between Frequency Hopping (FH) channels and synchronizing to the piconet's master.

In this dissertation we address scatternet formation, routing, and security to make Bluetooth® scatternet communication feasible. We define criteria for efficient scatternet topologies, describe characteristics of different scatternet topology models as well as compare and contrast their properties, classify existing scatternet formation approaches based on the aforementioned models, and propose a distributed scatternet formation algorithm that efficiently forms a scatternet topology and is resilient to node failures.

We propose a hybrid routing algorithm, using a bridge link agnostic approach, that provides on-demand discovery of destination devices by their address or by the services that devices provide to their peers, by extending the Service Discovery Protocol (SDP) to scatternets.

We also propose a link level security scheme that provides secure communication between adjacent piconet masters, within what we call an Extended Scatternet Neighborhood (ESN).

KEYWORDS: *Wireless Personal Area Network (WPAN), Bluetooth, Piconet, Scatternet Formation, Scatternet Routing*

Karl E. Persson

A PROTOCOL SUITE FOR WIRELESS PERSONAL AREA NETWORKS

By
Karl E. Persson

Director of Dissertation
Dr. D. Manivannan

Director of Graduate Studies
Dr. Andrew Klapper

RULES FOR THE USE OF DISSERTATIONS

Unpublished dissertations submitted for the Doctor's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgments.

Extensive copying or publication of the dissertation in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

A library that borrows this dissertation for use by its patrons is expected to secure the signature of each user.

Name	Date

DISSERTATION

Karl E. Persson

The Graduate School
University of Kentucky
2009

A PROTOCOL SUITE FOR WIRELESS PERSONAL AREA NETWORKS

DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By
Karl E. Persson
Lexington, Kentucky
Director: Dr. D. Manivannan, Associate Professor of Computer
Science
Lexington, Kentucky
2009

DEDICATION

This is dedicated to my parents, Hans Persson and Marianne Lindh-Persson, and to Katie for all their love and support. Without their encouragement and affection I would have never reached this milestone in my life.

ACKNOWLEDGMENTS

I would like to express my gratitute to my advisor Dr. D. Manivannan, who has supported me wholeheartedly throughout the entire process. His guidance and direction have been invaluable to me. I thank my committee members: Dr. Mukesh Singhal, Dr. Zongming Fei, and Dr. James E. Lumpp Jr., for their encouragement, support, and assistance.

In addition, I would like to thank: my former lab colleague Dr. Jianchang Yang, for many interesting discussions and his assistance with the formatting of my dissertation; Dr. Judy Goldsmith of the University of Kentucky Department of Computer Science, for her encouragement and advice that I pursue a Ph.D.; Mr. Joseph E. Smith III of Leesburg, Virginia and formerly of IBM Corporation, for his encouragement and advice that I pursue an advanced degree in the first place; Dr. Patricia Whitlow, Assistant Dean of the University of Kentucky Graduate School, for her inspiration and organization of a dissertation writing workshop; and Mrs. Kathleen Carter, for editing my dissertation and providing many helpful comments and suggestions.

I would also like to thank Dr. Jerzy W. Jaromczyk and Dr. Grzegorz W. Wasilkowski, both with the University of Kentucky Department of Computer Science, for their continuous support, encouragement, and positive impact during my entire academic career at the University.

Last I would like to thank all my friends, fellow students, and faculty in the Department of Computer Science and throughout the University of Kentucky for their collective inspiration and assistance throughout the time I have spent at the University.

Table of Contents

List of Tables

List of Figures

List of Files

1. Persson-Dissertation.pdf

Chapter 1

Introduction

Bluetooth® is a short range wireless networking technology suitable for both stationary and portable devices with low-power consumption. It was initially developed by LM Ericsson in Sweden, but is governed as an open specification by the Bluetooth Special Interest Group (SIG) and also adopted as the IEEE 802.15.1 standard for Wireless Personal Area Networks (WPANs). A WPAN is generally considered to be a small, short range ad hoc network, made up primarily of low power devices, that surround a person or an object, e.g. a desktop computer or a vehicle. Although Bluetooth® technology is suitable for WPANs, the basic Bluetooth® networking topology unit, a *piconet*, restricts membership to a single master and up to seven active slave devices with no direct interconnections between the slaves. Therefore the formation of Bluetooth® scatternets is essential for communication between larger groups of devices.

A scatternet topology is formed by interconnecting piconets. Since each piconet has a unique Frequency Hopping Sequence (FHS), piconet interconnections are accomplished by asking some selected nodes to participate in more than one piconet. These so called *bridge nodes* divide their time between piconets by switching between FH channels and synchronizing to each piconet's master. In general, a node can only be the master in one piconet but is allowed to participate in other piconets as a slave.

The Bluetooth® Special Interest Group (SIG) has published several versions of the Bluetooth specification® [11]. An overview of its evolution is provided in Chapter 2.2.5. The

specification defines the concept of a scatternet, but does not provide detail on how scatternets should be formed or how actual scatternet communication can take place.

1.1 Problems Addressed And Solved In This Dissertation

Bluetooth® technology has great potential to become the de facto standard for communication between heterogeneous devices in WPANs. However, the piconet size limit of a single master and up to seven active slaves prevents Bluetooth® devices from being part of larger Mobile Ad Hoc Networks (MANETs) or Wireless Personal Area Networks (WPANs). The Bluetooth® specification [11] has defined the concept of a scatternet for interconnecting multiple piconets using overlapping bridge nodes, but does not specify any protocols or procedures for the formation of scatternets, routing and data communication, or scatternet security.

In this dissertation we address scatternet formation, routing, and security to make communication in scatternets possible. With the adoptation of over one billion devices already and the Bluetooth® SIG on the verge of producing a specification that enables high-speed Bluetooth® communication, it is important to provide protocols for scatternet communication so that we can take full advantage of the increasing pervasiveness and capability of Bluetooth® enabled devices.

Scatternet formation is the precursor to any communication between devices in a scatternet. Numerous approaches have been proposed to solve the problem of scatternet formation. However, many of these approaches depend on centralized entities, consist of phase divided algorithms, lack fault-tolerance, and hence are of limited use. We discuss many of these approaches in detail, then establish criteria and topology models for defining efficient scatternet formation algorithms and classify existing solutions accordingly. We also propose the Bluetooth Distributed Scatternet Formation Protocol (BTDSP), which is a fault-tolerant distributed scatternet formation protocol. BTDSP is based on the observations that cen-

tralized leader elections, phase divisions, and use of Master/Slave (MS) bridges result in inefficient scatternet topologies.

For devices in a scatternet to be able to communicate, a routing protocol is necessary. We propose Hybrid Bluetooth Scatternet Routing (HBSR), which is a hybrid scatternet routing protocol above the L2CAP layer that utilizes a zone routing approach to proactively maintain an Extended Scatternet Neighborhood (ESN) between adjacent piconets and uses gossip-based reactive discovery of either individual devices (destination based) or devices that offer a service (service based) outside the ESN. We also take Inter Piconet Scheduling (IPS) into consideration to ensure that HBSR operates as efficiently as possible.

Security is another important area that is necessary for scatternets to be properly utilized. We discuss different security scenarios and present a link level algorithm for enabling security associations between adjacent ESN piconet masters in a scatternet.

1.2 Organization of the Dissertation

The rest of the dissertation is organized as follows. *Chapter 2* establishes the necessary background by defining major concepts in more detail. In *Chapter 3* we define criteria for scatternet formation, establish general scatternet topology models, and classify proposed solutions according to the aforementioned criteria and models. Thereafter, in *Chapter 4* we describe our Bluetooth Distributed Scatternet Formation Protocol (BTDSP) in detail and also compare and contrast it to other scatternet formation algorithms. In *Chapter 5* we introduce our Hybrid Bluetooth Scatternet Routing (HBSR) protocol and evaluate the performance of the algorithm. *Chapter 6* presents our approach for establishing link level security in a scatternet. Finally, in *Chapter 7* we conclude the dissertation and describe the direction of our future work.

Chapter 2

Preliminaries

In this section we define some of the concepts discussed in this dissertation.

2.1 Wireless Personal Area Networks (WPANs)

A wireless ad hoc network can generally be considered an autonomous group of communicating peers that use unreliable wireless links and does not rely on a central authority or infrastructure [34]. More specifically, a Personal Area Network (PAN) can be defined as a short range ad hoc network that inter-connects devices within close proximity; typically surrounding a single person or an object [55]. When wireless Infrared (IR) or Radio Frequency (RF) transmissions are used to inter-connect PAN members the network is called a Wireless Personal Area Network (WPAN) [23]. Further, the IEEE 802.15 Working Group (WG) for WPANs has developed a Wireless Personal Area Network (WPAN) standard based on version 1.1 of the Bluetooth specification [33]. The main purpose of a WPAN is to enable ubiquitous connectivity of heterogeneous wireless devices surrounding a person. Bluetooth® is especially suitable as the underlying WPAN technology because of its clustering properties and low-power operation.

2.2 Bluetooth Technology

In this section we cover Bluetooth® history, purpose, and technical details, as well as define networking concepts and nomenclature.

2.2.1 Overview

Bluetooth® is a networking technology aimed at providing short range wireless communication to low-powered devices. The name Bluetooth® was adopted from the 10th century Scandinavian Viking king Harald Bluetooth who successfully united Denmark and Norway. Bluetooth® was initially developed by LM Ericsson in Sweden, but has since 1998 been governed as an open specification by the Bluetooth® Special Interest Group (SIG) [30]. Given that Bluetooth was developed in Sweden, naming the technology after a Scandinavian Viking king could be seen as suitable without further explanation. However, Harald Bluetooth and the wireless technology named after him also share the characteristic as uniters of heterogeneous entities, which might have contributed to the naming choice. While Harald Bluetooth united Sweden and Denmark under Christianity in the 10th century, Bluetooth wireless technology unites, or inter-connects, heterogeneous electronic devices.

As the Bluetooth® specification [11] was developed, a number of basic usage scenarios, or profiles, were proposed to simplify adaptation and interoperability, and ultimately persuade device manufacturers to include Bluetooth® chipsets in their devices. These profiles can generally be classified into the following four categories [51]: generic profiles, including the Generic Access Profile (GAP) and Service Discovery Protocol (SDP) profiles: telephony profiles, including headset, cordless telephony, and intercom profiles; networking profiles, including fax, Dial-up Networking (DUN), PAN, and LAN access profiles; and object exchange profiles, including object push, file transfer, and automatic synchronization profiles.

The Generic Access Profile (GAP) [8] defines procedures for discovery of devices, link management and pairing of devices, as well as link level security aspects within a piconet [11]. GAP procedures are used extensively in the management of piconets and link level security. The use of security modes from the Generic Access Profile (GAP) is discussed in Section 6.3.

Another profile that fits into the generic profile classification is the Service Discovery Protocol (SDP) [11] profile, which allows devices to discover services that are offered by other devices. The Service Discovery Protocol (SDP) operates using a client/server model, without

knowledge of underlying Master/Slave (MS) relationships, to discover the capabilities of a specific piconet device or to determine which piconet devices support a requested application or service [50]. We discuss extending SDP functionality to scatternets in Section 5.4.

The PAN profile defines the inter-connection of Bluetooth® devices with heterogeneous networking devices using the Bluetooth Network Encapsulation Protocol (BNEP). This can be accomplished using either a Network Access Point (NAP), which functions as a gateway between the Bluetooth® piconet and the rest of the network, or in a Group Ad Hoc Network (GN) scenario where non-Bluetooth devices are able to form temporary links and exchange data with Bluetooth devices using BNEP for Ethernet encapsulation [9, 11].

2.2.2 Technical Details

Bluetooth® operates in the 2.4 GHz Industrial, Scientific, Medical (ISM) frequency band. ISM is unlicensed and available worldwide, making it suitable for pervasive technologies such as Bluetooth®. The Bluetooth® core system consists of the Bluetooth Controller, the Host Controller Interface (HCI), and a set of profiles outlined in the previous section. The basic layering and core architectures are illustrated in Figure 2.1[1].

The Bluetooth® radio is designed to minimize interference using a Frequency Hopping Spread Spectrum (FHSS) system across 79 hop carriers in the ISM band, with 1 MHz spacing, at a nominal rate of 1,600 hops per second. For devices to communicate with each other they must all follow the same pseudo-random Frequency Hopping Sequence (FHS) and be synchronized on the same Radio Frequency (RF) carrier at the same time. For the basic data rate of 1 Mbps the baseband signal is modulated using Gaussian Frequency Shift Keying (GFSK) and then transmitted on an RF carrier according to the FHS. The Enhanced Data Rate (EDR), which was introduced in version 2.0 of the specification [11], defines two new modulation techniques for transmission of the synchronization sequence, payload, and trailer sequence, while the access code and packet header are transmitted at

[1]Illustration derived from Bluetooth® Core v2.0 + EDR specification [11] page 21. Copyright 2004 © Bluetooth SIG [30].

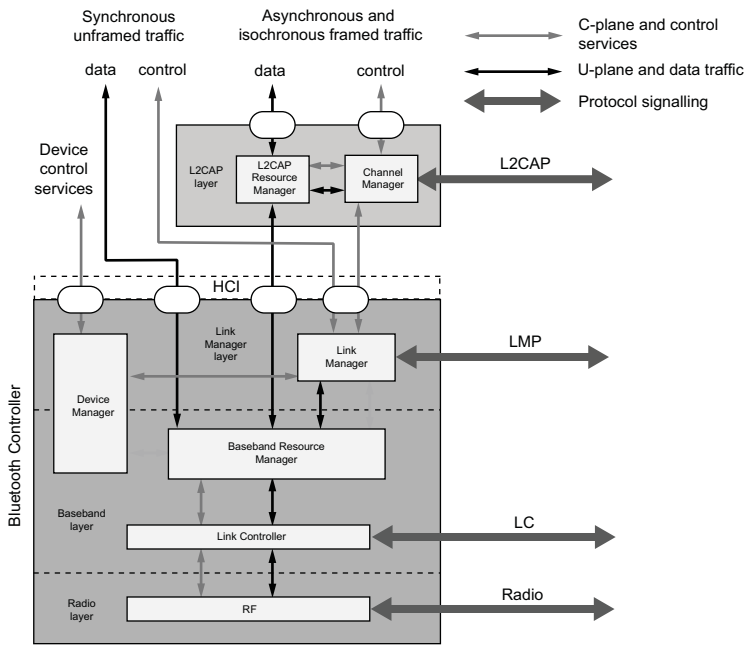


Figure 2.1: Bluetooth® Core Architecture

basic data rate for backwards compatibility. For 2 Mbps Enhanced Data Rate (EDR), $\pi/4$-rotated differential encoded Quarternary Phase Shift Keying ($\pi/4$-DQPSK) is used, while 3 Mbps EDR is achieved by using differential encoded 8-ary Phase Shift Keying (8DPSK) [11].

After each packet transmission on an RF carrier, following the FHS, devices hop to the next carrier in the sequence to remain synchronized to the Frequency Hopping (FH) channel. The FH channel can be thought of as a virtual channel as it consists of multiple RF carriers that are visited according to a FHS, and it is not a physical carrier itself. Further, the FH channel is divided into time slots. Each time slot is 625 μs and contains a single baseband transmission. A baseband transmission can last for one, three or five time slots. To maintain synchronization across different carriers and to align time slots, Bluetooth® uses loosely synchronized clocks, which means that slaves use an offset from the master's clock to stay synchronized. The master's *BD_ADDR* provides the identity of the piconet and its native clock determines the time slot boundaries. The *BD_ADDR* is a unique 48-bit MAC address that is assigned to every Bluetooth® device.

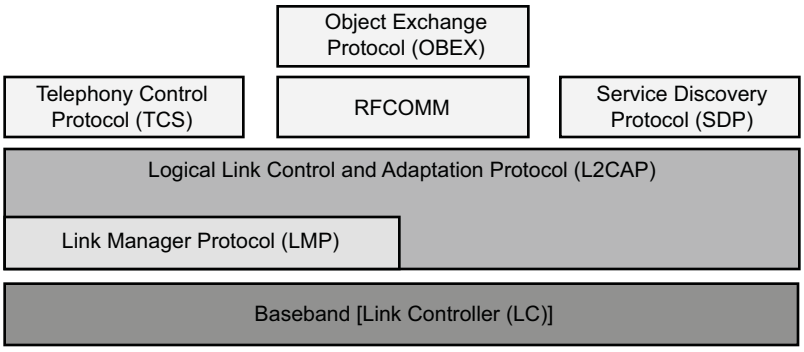


Figure 2.2: Bluetooth® Protocol Stack

The Bluetooth protocol stack is illustrated in Figure 2.2[2]. The Bluetooth® baseband protocol performs basic link management, frequency hop selection, piconet creation, timing and synchronization, error control, and basic link level security operations. The baseband protocol is also used for passing data between the radio and higher layers [50] in the Bluetooth Link Controller (LC) Finite State Machine (FSM). There are two types of physical links that can be formed between devices: Asynchronous Connectionless (ACL) and Synchronous Connection-Oriented (SCO). ACL links are used for non-time sensitive data communication and have error control functionality. SCO links are used for time-sensitive data communication such as voice or multimedia applications. For SCO links, there are two packet types available specifically for latency sensitive traffic. Depending on the required voice quality and type of Forward Error Correction (FEC) needed, either single slot Data/Voice Bluetooth packet type (DV) or High-quality voice Bluetooth packet type (HV) packets are available. For ACL links, data can be transmitted in one, three, or five slot Data - Medium Rate (DM) or Data - High Rate (DH) packets of different types depending on the type of error control needed, as well as in AUX1 packets for raw data.

Bluetooth® allows piconet-wide broadcasts as well as point-to-point logical links on Asynchronous Connectionless (ACL) and Synchronous Connection-Oriented (SCO) physical links to be established. These are contained within logical transports, which are used for Intra

[2]Illustration derived from Bluetooth® Core v2.0 + EDR specification [11] page 181. Copyright 2004 © Bluetooth SIG [30].

Piconet Scheduling (IRPS). Each active slave in a piconet is identified by a 3-bit logical transport address, or LT_ADDR. The intra piconet communication is exclusively controlled by the master, which periodically polls each slave to keep it synchronized. Slaves are only permitted to transmit during odd time slots directly following a master poll.

Above the baseband and Link Controller (LC) in the Bluetooth® protocol stack resides the Link Manager Protocol (LMP), which controls link formation and communicates with Link Managers (LMs) in peer devices. A number of LMP Packet Data Unit (PDU) commands have been defined to communicate link management functionality between LMs in peer devices to configure the links, manage the piconet, or implement link level security [50].

Above the LC and LM resides the Logical Link Control and Adaptation Protocol (L2CAP), which is used as a conduit for higher layer application protocols on connection-oriented ACL links [50]. L2CAP is not used for SCO real-time links as there are strict latency requirements for traffic across such links. L2CAP uses Connection-oriented (CO) or Connection-less (CL) channels, as well as a signalling channel, between devices to abstract details of the underlying Bluetooth® layers from application protocols for simplicity of implementation. L2CAP functionality can be divided into the following categories [50]: protocol multiplexing, packet segmentation and reassembly, Quality of Service (QoS), and group management. L2CAP protocol multiplexing is used to establish the virtual channels and hide lower layers from application protocols. Application protocol packet payloads are segmented into Bluetooth® ACL packets, and control is transfered down to the Link Controller (LC). Conversely, payloads are reassembled from Bluetooth® ACL packets and directed up to the corresponding application protocol. L2CAP also provides per application protocol Quality of Service (QoS) and device group management, since the piconets are not directly visible to higher layer protocols.

2.2.3 Piconets

The piconet is the fundamental networking unit for Bluetooth® Wireless Personal Area Networks (WPANs). A piconet consists of a single master device and up to seven active slave devices. The piconet master determines the Frequency Hopping Spread Spectrum (FHSS) that the slaves must follow in order to stay synchronized to the piconet channel. The Time Division Duplex (TDD) Frequency Hopping (FH) channel is divided into 625 μs time slots. It is driven by the master, which eliminates contention problems within a piconet. The master transmits during even time slots, while slaves are restricted to sending packets back to the master during the subsequent odd time slots. The master addresses each slave either individually by its LT_ADDR, or by a piconet-wide broadcast using LT_ADDR 000.

Devices can be connected into piconets, the basic Bluetooth® networking unit, directly, or they can use a device discovery mechanism to locate peers. By entering the inquiry state in the Link Controller (LC) Finite State Machine (FSM), a device transmits special inquiry packets on a pre-determined subset of frequencies. If other devices simultaneously engage in inquiry scanning and receive the inquiry, they respond and can subsequently be directly paged and incorporated into the piconet. This is called device discovery and is the primary method for locating devices to form a piconet.

2.2.4 Scatternets

Although multiple piconets can coexist without significant interference from each other, it is desirable to be able to interconnect them to form larger WPANs with more than eight nodes. This is where the concept of a scatternet comes into play. The Bluetooth® specification defines scatternet as an interconnection of piconets, but states that it is not within the scope of the core protocols [11]. Conceptually, any time a piconet member participates in more than one piconet a scatternet exists. However, in order to utilize the benefit of additional connectivity the bridge nodes must be able to relay data packets between piconets. This requires both Inter Piconet Scheduling (IPS) and a scatternet routing algorithm. This is

discussed in more detail in Chapter 5.

2.2.5 Evolution of the Bluetooth Specification

For the purpose of this dissertation we assume that all algorithms and assumptions are based on version 1.2 of the Bluetooth® specification [11]. For clarity, we outline the evolution of the specification and the changes between versions.

- *Version 1.0* - Initial version released by the Bluetooth® Special Interest Group (SIG) in 1998. This version has been deprecated.
- *Version 1.0B* - Included many errata items from the Bluetooth® Special Interest Group (SIG) website [30]. This version has been deprecated.
- *Version 1.1* - Included many errata items from the Bluetooth® Special Interest Group (SIG) website [30] and fixed compatibility problems in initial versions based on feedback from *unplugfests* held by the Bluetooth SIG [30]. Received Signal Strength Indicator (RSSI) was also added. This was the first truly interoperable version. This version has been deprecated.
- *Version 1.2* - Many new features were added in this version including faster connection and discovery, Adaptive Frequency-Hopping Spread Spectrum (AFH), Extended Synchronous Connection-Oriented (eSCO) links, enhanced error detection and flow control, enhanced synchronization capability, and enhanced flow specification.
- *Version 2.0 + EDR* - Enhanced Data Rate (EDR) for 2.0 and 3.0 Mbps transmission speeds added.
- *Version 2.1 + EDR* - New features added including Encryption Pause and Resume, Extended Inquiry Response, Secure Simple Pairing, Sniff Subrating, and Security Mode 4.

Chapter 3

Bluetooth Scatternets: Criteria, Models and Classification

3.1 Introduction

In this chapter we describe the fundamental challenges related to scatternet formation, define general scatternet topology models, establish different topology criteria, and classify existing solutions for scatternet formation.

A scatternet topology is formed by interconnecting piconets. Since each piconet has a unique frequency hopping sequence, piconet interconnections are done by allowing some nodes to participate in more than one piconet. These *bridge nodes* divide their time between piconets by switching between Frequency Hopping (FH) channels and synchronizing to the piconet's master. In general a node can be the master only in one piconet but is allowed to participate in other piconets as a slave.

In the Bluetooth® specification [11] the functionality and membership properties of piconets are described in detail while scatternet formation is only mentioned briefly. Numerous approaches for scatternet formation have been proposed since the specification was first published. The main idea behind scatternet formation is to interconnect adjacent piconets. We can interconnect piconets by scheduling disjoint slots during which the interconnecting bridge nodes can communicate with the master of each connected piconet.

The rest of this chapter is organized as follows. In Section 3.2 we cover the fundamen-

tal concepts behind Bluetooth® network topologies. We briefly overview how Bluetooth® piconets are formed in Section 3.2.1. In Section 3.2.2 we discuss different formation metrics and constraints for scatternet formation. Thereafter, in Section 3.2.3 we define scatternet models and describe their differing characteristics. We cover link formation in Section 3.2.4, since Bluetooth® devices must discover each other and form links before they can communicate. Section 3.2.5 describes some proposed solutions for Intra Piconet Scheduling (IRPS). Section 3.2.6 covers Inter Piconet Scheduling (IPS) and its implications for better scatternet performance. Section 3.2.7 discusses the general problem of routing in scatternets. In Section 3.3 we review the state-of-the-art approaches with respect to scatternet formation and classify the approaches according to the models defined in Section 3.2.3. We begin with single-hop protocols in Section 3.3.1, in which all devices must be within transmission range of each other to form a scatternet. Thereafter, we cover multi-hop protocols in Section 3.3.2. These approaches offer more flexibility since not all devices are required to be within radio proximity of each other. Section 3.3.3 describes optimized solutions that provide theoretical results on how scatternet topologies could be constructed efficiently. Finally, in Section 3.4 we offer some summarizing remarks.

3.2 Bluetooth Topology Fundamentals

3.2.1 Piconets

Bluetooth® devices communicate with each other in a Master/Slave (MS) configuration. That means that devices do not communicate as equal peers on a collision prone medium, as in IEEE 802.11 [35]. Figure 3.1(a) shows a two node piconet with one master and one slave device. The FH channel is divided into time slots. Each time slot is 625 μs and contains a baseband transmission. A baseband transmission can last for one, three or five time slots. After each transmission, the piconet devices hop to another one of the 79 FH carriers. To maintain synchronization Bluetooth® uses loosely synchronized clocks, which means that slaves use an offset from the master's clock to stay synchronized. The master's BD_ADDR

provides the identity of the piconet and its native clock determines the time slot boundaries.

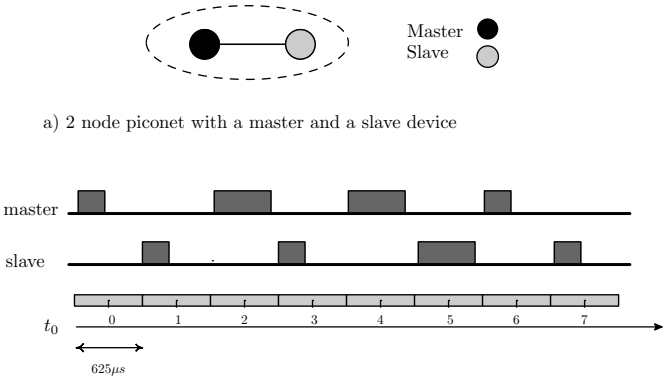


Figure 3.1: Basic description of a Bluetooth® Piconet

Communication pattern in a piconet is controlled by the master. The master polls each slave in a round-robin fashion during even numbered time slots. The slaves are allowed to transmit only during odd time slots subsequent to a poll from the master. Figure 3.1(b) illustrates an example of communication on the TDD slotted channel between the master and the slave. The figure shows the communication pattern where the master transmits during even time slots and the slave responds during odd time slots. Based on the specification, in larger piconets the master addresses each slave using a round-robin Intra Piconet Scheduling (IRPS) algorithm. Section 3.2.5 covers other proposed IRPS algorithms.

Multiple piconets can coexist without significant interference from each other since all slaves maintain synchronization to a uniquely identified FH channel. In [77] Zurbes concludes that inter-piconet interference is only significant in dense scenarios of 50-100 overlapping piconets. Although piconets can coexist, it is also desirable to be able to interconnect them. As previously stated, the eight device upper bound on active piconet capacity and the use of FH channels necessitate piconet switching for piconet interconnections. To interconnect two or more piconets at least one device must participate in more than a single piconet. By allowing certain devices to function as bridges and switch between participating piconets, these *bridge nodes* can relay packets between piconets. The bridge node must also keep track of all its connected piconet masters' identities and clock offsets, since clocks are merely loosely

synchronized. Upon switching from a piconet, the bridge node changes its operational mode from *ACTIVE* to either *HOLD* or *SNIFF*. Consequently the bridge node cannot respond to master polls while it is absent from the piconet. In *HOLD* mode, the device maintains an LT_ADDR but is not active for a fixed time interval. In *SNIFF* mode the device also maintains an LT_ADDR and schedules periodic sniff intervals during which is it available. The main difference is that *SNIFF* is scheduled periodically while *HOLD* is a one-time operation that does not begin until $6*T_{poll}$ slots from the time of the request [50]. Upon switching back the bridge must re-synchronize to the master to follow the FHS again. Miklos *et al.* [31] conclude that piconet switching poses a significant overhead and has a major impact on system performance. It is therefore important for overall scatternet performance not only that the scatternet topology is carefully constructed, but also that piconet switching is scheduled as efficiently as possible. Section 3.2.6 describes some of the proposed approaches for Inter Piconet Scheduling (IPS).

3.2.2 Scatternet Formation Metrics and Constraints

In the Bluetooth® specification [11] scatternets are entirely conceptual. There are no guidelines for forming scatternets. Two scatternet topologies that are formed by separate approaches can appear radically different and retain dissimilar characteristics. The resulting scatternet topology is therefore based on how the piconets are inter-connected. The underlying purpose for formation of a specific scatternet and any constraints placed on how devices form connections may have a significant impact on the resulting topology.

Some criteria used for scatternet formation are listed below:

- Complete scatternet connectivity
- Maximized aggregate bandwidth
- Minimized average routing path length

- Maximized average node availability
- Minimized bridge switching overhead
- Communication group clustering
- Self-healing
- Multi-hop node participation
- On-demand scatternet formation

Complete scatternet connectivity means that every node in the scatternet is able to reach every other node. In other words, there are no partitions in the scatternet. By choosing to *maximize aggregate bandwidth* the user wishes to increase goodput and reduce overhead from formation, scheduling, or routing. To *minimize average routing path length* means that the scatternet topology is as compact as possible with few routing hops between nodes. *Maximized average node availability* refers to ensuring that a dense and connected topology with multiple routes between nodes is available. *Minimizing bridge switching overhead* is a commonly used criterion and refers to controlling the number of piconets in which a bridge node participates. Scatternets that are optimized for *communication group clustering* often have special purposes which necessitates that certain devices are always clustered together. *Self-healing* refers to the ability of the algorithm to re-incorporate nodes that either move, lose power, or otherwise are disconnected from the scatternet. *Multi-hop node participation* means that the scatternet is able to accommodate devices that might or might not be within direct radio vicinity of each other. *On-demand scatternet formation* refers to the ability to form temporary scatternets only when they are needed.

One of the most important decisions to make before designing a new or choosing an existing algorithm for scatternet formation is to determine the desired constraints placed on bridge nodes. A node is allowed to participate only as a master in a single piconet, but can join any number of other piconets as a slave on a time share basis. This type of

bridge is called a Master/Slave (MS) bridge. The other more restrictive type of bridge is the Slave/Slave (SS) bridge. An SS bridge is allowed to operate only as a slave node in the piconets in which it participates, but not as a master in any of them. In [48] Misic *et al.* find that the mean access delay in scatternets with SS bridges is lower than when MS bridges are used. The main reason is that intra-piconet communication ceases while the MS bridge is participating in another piconet, for which it is not a master. In contrast, disallowing MS bridges eliminates some tree, ring, and mesh topologies from consideration that can, in some cases, produce shorter average path lengths, prevent routing loops, and provide simpler scatternet routing.

Another formation metric that directly affects scatternet performance is the *bridge degree*, defined as the number of piconets in which a node is allowed to participate. In [31] and [61] the authors conclude that there is a direct correlation between bridge switching overhead and scatternet performance. Scatternet performance can be increased by placing constraints on the maximum bridge degree [61, 67, 16, 5]. Restricting bridge overlap by *allowing only one bridge between any two piconets* is another possible constraint [67].

A performance metric that is difficult to determine a priori is the traffic pattern [19]. Traffic-flows between nodes have significant impact on throughput and overall performance [14]. More specifically, groups of devices that frequently communicate with each other make up Communicating Groups (CGs) and can be clustered together for better performance [44]. Generally this metric has only theoretical significance for designing scatternets that improve performance, since it requires knowledge of traffic patterns and node relationships. This is described in more detail in Section 3.3.3.

3.2.3 Scatternet Models

Based on the metrics and constraints discussed in the previous section, a number of general scatternet models can be derived. In this section we categorize different types of scatternet topologies and compare their characteristics.

Scatternet formation approaches can be classified into the following general topology models:

- Single Piconet Model (SPM)
- Slave/Slave Mesh (SSM)
- Master/Slave Mesh (MSM)
- Tree Hierarchy (TH)
- Master/Slave Ring (MSR)
- Slave/Slave Ring (SSR)

The simplest type of scatternet is the Single Piconet Model (SPM), illustrated in Figure 3.2(a) [44]. SPM is the only scatternet type that is natively supported in the Bluetooth® specification. In an SPM, as in a traditional piconet, a single master and up to seven active slaves communicate. The rest, if any, of the participating slaves, up to 255 of them, are placed in *PARK* mode and can be substituted in when the master needs to communicate with them. Although this model is simple and natively supported in the specification, it is very inefficient and requires active slaves to be placed in *PARK* mode to accommodate a parked slave.

Two variations of mesh topologies are shown in Figures 3.2(b) and (c). In Figure 3.2(c), a Master/Slave Mesh (MSM) that interconnects piconets using MS or SS bridges is shown. MSM is the least restrictive scatternet model, since it allows any type of piconet interconnection to be formed. Degree constrained MSMs place an upper-bound on the number of piconets in which a bridge node participates. For example, the Bluetooth Topology Construction Protocol (BTCP) protocol [67] forms a 2-MSM degree constrained scatternet, while our Bluetooth Distributed Scatternet Formation Protocol (BTDSP) (described in Section 4.3) forms a 2-SSM degree constrained scatternet. In both cases the number 2 corresponds to

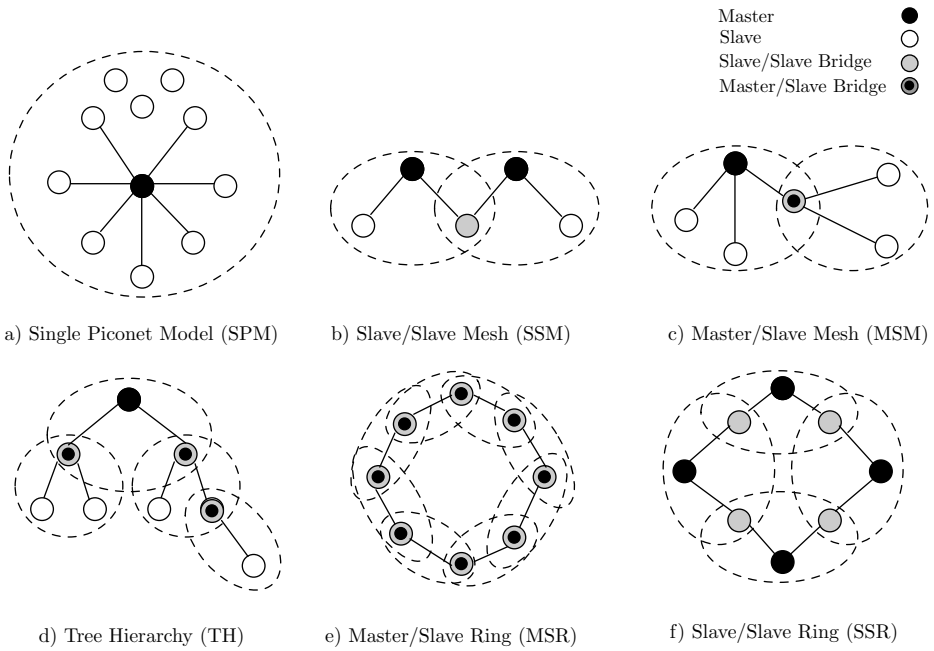


Figure 3.2: Illustration of Scatternet Topology Models

the upper-bound on the number of piconets in which a bridge node participates. Slave/Slave Mesh (SSM) scatternets are illustrated in Figure 3.2(b) and are very similar to MSMs, except that they allow only SS bridges. This limitation generally increases scatternet performance at the cost of additional protocol complexity.

Figure 3.2(d) shows a Tree Hierarchy (TH) topology. TH scatternet topologies differ from the preceding models in that they have a single root node and descendant tree nodes. The Tree Scatternet Formation Protocol (TSF) protocol by Tan *et al.* is an example of a TH scatternet [27]. TSF is discussed in detail in Section 3.3.1. The main advantages of TH topologies over the preceding models are that they have logarithmic average path length and simplify scatternet routing. On the other hand, the root node is a bottleneck point and node disconnections close to the root node can partition the scatternet and substantially reduce node availability.

Master/Slave Ring (MSR) and Slave/Slave Ring (SSR) models, in Figures 3.2(e) and (f), are ring structured scatternets. The main reason to form a ring topology is to alleviate the bottleneck problems seen in TH topologies [17] while maintaining simple scatternet routing

[68]. However, ring models suffer from partitioning problems and significantly longer average path lengths and mean access delays.

Depending on the purpose of the scatternet and the metrics that are emphasized, any of the aforementioned models could be appropriate. For instance, in applications where routing and access delay are of great importance, Tree Hierarchy (TH) solutions might be the most suitable. In other scenarios where connectivity and availability are the predominant metrics, the mesh models might work better.

3.2.4 Link Formation

After determining the purpose of the scatternet, the participating nodes must be discovered before a topology can be formed. This is called device discovery. Devices cannot simply overhear other nodes within radio proximity, as done in IEEE 802.11 [35], due to the FH channel scheme employed in Bluetooth. Instead, Bluetooth® devices must enter *INQUIRY* or *INQUIRY_SCAN* states to transmit inquiry packets or listen for inquiries respectively. Two trains of 16 predetermined frequency hop carriers are used for device discovery. Since there is no way for inquiry-scanning devices to know exactly at which FH channel an inquiring device is transmitting, the inquiring device broadcasts Inquiry Access Code (IAC) packets and changes hop carriers at twice the rate of the inquiry scanning device. This allows devices to discover each other independent of clock synchronization, since there is no a priori clock synchronization in Bluetooth® and each device merely has a native clock. If the inquiry scanning device receives an IAC packet it backs off for a Random Backoff (RB) delay period and then responds with a Frequency Hopping Sequence (FHS) packet, which contains its own *BD_ADDR* and clock value. Thereafter, paging procedures can commence and the previously inquiring device can page the respondent using a Device Access Code (DAC) packet. If the paging procedure is successfully completed, a link is established with the initially inquiring device as the master. If the roles need to be reversed, a Master/Slave (MS) switch can also be performed. After the link has been established, the slave stays synchronized to the

piconet using the FHS and clock value it received from the master during paging.

As described in the Bluetooth® specification [11], device discovery is not a spontaneous process. For device discovery to begin roles must be preassigned and devices must enter either *INQUIRY* or *INQUIRY_SCAN* states. However, for scatternet formation the link establishment process should happen gratuitously. Devices that wish to participate in the scatternet should automatically engage in device discovery and form a link, either as a master or as a slave. Gratuitous link establishment can be accomplished either symmetrically or asymmetrically.

Salonidis *et al.* propose a symmetric link protocol in [67] and [66]. In their approach devices alternate independently between *INQUIRY* and *INQUIRY_SCAN* states with randomized state residence intervals. Two devices successfully discover each other if they remain in complementary states for longer than a lower-bound interval.

Ching *et al.* propose an asymmetric approach to link establishment in [16]. Their randomized approach probabilistically determines inquiry and inquiry scan roles. Devices then repeat this procedure until a link is established.

Ramachandran *et al.* evaluate both a deterministic and a randomized algorithm that are variations of the two preceding solutions respectively [2]. Their results suggest that the randomized approach produce significantly lower link establishment delay than the deterministic (or symmetric) algorithm.

3.2.5 Intra Piconet Scheduling (IRPS)

The Bluetooth® specification describes a round-robin Intra Piconet Scheduling (IRPS) algorithm, in which the master polls each slave in a cyclic manner without considering queue lengths or other traffic dependent metrics [11]. This naive polling scheme is generally called Pure Round Robin (PRR). PRR provides fairness between the slaves, but wastes unnecessary time slots and does not provide any Quality of Service (QoS) bounds.

Kalia *et al.* [43] propose more efficient polling schemes. Their polling algorithms utilize

the queue state of the masters and slaves to determine the polling order. Their Priority Polling (PP) scheme prioritize links where both the master and the slave have non-empty queues. To prevent link starvation and provide fairness among the links, they also propose the K-Fairness Scheduling Policy (KFP). KFP performs round-robin scheduling among links on which one or both queues are non-empty. When one of the queues is empty, the algorithm sacrifices the current link polling to a link where both queues are full. An absolute fairness bound of K slots is guaranteed by not sacrificing additional link polls when the difference between the maximum and minimum provided service exceeded K slots. Both algorithms significantly increases slot utilization over the PRR scheme. However, both PP and KFP assume that the master has complete knowledge of the queue lengths at the slaves and is therefore not practically feasible.

Capone *et al.* [3] point out that the algorithms in [43] are idealistic, since the master has only partial knowledge of the queue lengths. Due to the master-driven polling structure, slaves cannot inform the master of their queue lengths until after being polled by the master. The authors emphasize that the PRR scheme provides fairness but is not exhaustive. Instead, they propose the Limited and Weighted Round Robin (LWRR) intra-piconet scheduling scheme. In LWRR slaves are polled in a weighted round-robin manner. The weights are adaptively decreased when an empty queue was encountered and reset to their original values when the queue is non-empty again. The authors do suggest, based on their simulation results, that the Exhaustive Round-Robin (ERR) IRPS algorithm outperforms both LWRR and other polling schemes, except under extremely high load where using LWRR results in less retransmissions.

Golmie *et al.* present the Bluetooth Interference Aware Scheduling (BIAS) algorithm, which reduces the impact of interference in the channel [53]. The idea is to detect excessive interference on frequency carriers and then avoid packet transmissions. Thereafter the bandwidth leftover by the constrained (by interference) devices is reallocated fairly among the rest using a credit system. The algorithm provides short term fairness between unconstrained

(interference-free) devices and also maintains their guaranteed service rates.

A precursor to efficient scatternet communication is the existence of an Inter Piconet Scheduling (IPS) algorithm. This is described further in the next section.

3.2.6 Inter Piconet Scheduling (IPS)

Inter Piconet Scheduling (IPS) is a prerequisite to scatternet formation. Efficient IPS minimizes wasted time slots when bridge nodes that are participating in other piconets are polled. Therefore, it is important that time slots are scheduled efficiently. Miklos *et al.* suggest in their performance analysis that decreasing the bridging overhead and number of links is fundamental for good performance in a scatternet [31]. It is therefore necessary to introduce an IPS algorithm.

Johansson *et al.* propose an IPS approach based on Rendezvous Points (RPs) in [55]. A Rendezvous Point (RP) is a scheduled time slot at which the master will poll the bridge node. Thus, the bridge node should be present in the piconet at the RP. Johansson *et al.* present an RP-based algorithm in [56]. They emphasize that for IPS algorithms to be efficient there should be some coordination with an Intra Piconet Scheduling (IRPS) algorithm. The authors also assume that master devices are *never* selected as bridges. This assumption is well supported by the experimental analysis by Misic *et al.* [48], who conclude that the mean access delay is lower when Slave/Slave (SS) bridges are used. If Master/Slave (MS) bridges are used slaves in a piconet cannot communicate at all when their master is away, since all communication is master initiated. The Rendezvous Point (RP) IPS algorithm by Johansson *et al.* in [56] is called Maximum Distance Rendezvous Point (MDRP). The basic idea is to maximize the distance between RPs within a periodic super frame. Based on the analysis by Miklos *et al.* in [31], maximizing the time between the polling of bridge nodes increases performance for static traffic models. However, the MDRP algorithm is not adaptive and therefore does not react to bursty traffic. It also requires complex coordination of the Rendezvous Points (RPs).

Racz *et al.* propose the Pseudo-Random Coordinated Scatternet Scheduling (PCSS) IPS algorithm in [4]. In their approach, they use a pseudo-random sequence of checkpoints generated from the master clock and *BD_ADDR* of the bridge slave, instead of the RPs used in [56]. In the event of checkpoint collisions (when a bridge has multiple checkpoints scheduled in different piconets or when traffic patterns change), the checkpoint intensity can be increased or decreased adaptively.

Johansson *et al.* propose a scheduling algorithm which they called *JUMP* mode [54]. *JUMP* mode is a propose new operational mode. Hence, it requires modification to the current specification. The *JUMP* mode approach is very similar to the approaches proposed in [56] and [4]. Bridge nodes signal their presence on a link at pseudo-random rendezvous windows. Both Slave/Slave (SS) and Master/Slave (MS) bridge nodes are allowed and are called *jumping slaves* and *jumping masters* respectively. By signaling its presence on a link, the jumping node notifies the link peer that it will be present during the entire RP window. To reduce wasted polling of jumping slaves, Johansson *et al.* allow jumping slaves to establish a long term signaling scheme. A jumping master can also signal to its slaves that it will be absent during an RP window, so that the slaves will not have to listen for polls during the master's absence.

Zhang *et al.* propose another approach to mitigate the problem of masters polling absent bridge nodes, called the *bridge conflict problem*, in [75]. Their Flexible Scatternet-wide Scheduling (FSS) scheme polls slaves in a weighted round-robin fashion. The polling weight is represented by a tuple (P,R) where P indicates the scheduled cycle period of the node and R indicates the maximum polls in a scheduled cycle. The polling weight can be adaptively adjusted by the master, based on estimated traffic on the link. Each bridge node has an associated switch-table that indicates when the bridge will be present in a piconet. This bridge table is replicated at the connected master, but modified by each bridge node independently (based on its queue length). The algorithm provides an adaptive scheduling scheme. However, it requires a multi-phased single-hop scatternet formation protocol, such

as BTCP [67], since the switch tables are initially created by a coordinator.

Har-Shai *et al.* [39] present an approach applicable to smaller scatternets that do not require complex coordination. Their Load Adaptive Algorithm (LAA) is a completely dynamic approach, in which the next rendezvous point os scheduled when the bridge switches piconets. This approach avoids establishing inflexible periodic schedules. The Slave/Slave (SS) bridge node is also assumed to be connected to exactly *two* piconets. LAA determines when a bridge node should switch piconets based on several factors. The algorithm uses queue length to determine how soon it should switch piconets. An upper bounded time commitment interval is used to notify the master of how long the bridge will be absent to prevent wasted polls. This algorithm is adaptive and requires minimal coordination.

Wang *et al.* [70] propose the Dichotomized Rendezvous Point (DRP) approach, which attempts to coordinate RPs throughout the scatternet. The authors assume that only Slave/Slave (SS) bridges are used and that the bridge degree is exactly 2, or in other words that a 2-SSM degree constrained scatternet topology is used. Their two-phased algorithm assigns suboptimal RP during the first phase and thereafter utilizes traffic flow information during the second phase to dynamically adjust the Rendezvous Point (RP) schedule. As an enhancement to the DRP algorithm they also propose Dichotomized Rendezvous Point Broadcasting (DRPB), which adds a third phase that attempts to seek common RP reference points across the scatternet.

3.2.7 Scatternet Routing

In most topologies a scatternet routing protocol is necessary to efficiently accommodate data traffic. There are numerous different approaches to form a scatternet. This is due to the fact that scatternets are not part of the Bluetooth® specification. Therefore, the resulting topologies can have very different characteristics and consequently a general scatternet routing protocol that works across all possible topologies is highly unlikely. Further, although general wireless ad hoc network routing protocols such as Ad hoc On-demand Distance

Vector (AODV) [13] and Dynamic Source Routing (DSR) [21] have the potential to work in Bluetooth® scatternets, they would require extensive modification. Depending on the scatternet formation protocol that is used, different modifications are necessary. In some strictly structured topologies, such as tree (TH) or ring (MSR and SSR), routing is trivial. For instance, in the protocol by Sun *et al.* routing reduces to traversing a binary search tree with packets forwarded unidirectionally around the ring [68]. However, most scatternet mesh topologies require an explicit routing protocol. We cover scatternet routing fundamentals in more detail and present our hybrid scatternet routing algorithm in Chapter 5.

3.3 Topologies Resulting from Scatternet Formation

Scatternet formation can be accomplished in several different ways. We distinguish between *single-hop*, *multi-hop*, and *optimized* solutions. Single-hop solutions require all devices to be within radio transmission range of each other, while multi-hop solutions allow devices to join the scatternet if they are within the proximity of at least one participating device. Optimal solutions for scatternet formation are often not of practical use, but offer theoretical insight into how scatternet topologies can be efficiently constructed.

Multi-hop solutions are distributed by nature. Within the category of single-hop solutions we also distinguish between *coordinated* and *distributed* approaches. For coordinated solutions some entity has complete knowledge of the network and assigns roles and connections to all participating devices. This can be done by electing a coordinator using a leader election process [29].

3.3.1 Single-hop Topologies

The scatternet formation approaches described in this section are classified to produce single-hop topologies. This means that they rely on the assumption that all devices involved in the formation are within radio transmission range of each other. The Bluetooth® specification specifies three power control classes and allows transmit power control between 0 dBm and 20

dBm [11]. Although power control schemes have been proposed, e.g [62], we only consider approaches that use static transmission power. We further classify single-hop topologies based on whether the formation uses a coordinator or it is completely distributed.

Coordinated Solutions

Coordinated single-hop topologies are formed by an centralized, and in many cases omnipotent, device that has been elected to coordinate the formation.

Salonidis *et al.* present one of the first protocols for scatternet formation [66, 67]. The idea behind their Bluetooth Topology Construction Protocol (BTCP) attemps to solve the problem of asymmetric link formation. The authors emphasize that spontaneous link formation requires devices to automatically engage in device discovery. Whereas device discovery in the specification is designed to manually select an inquiry state, Salonidis *et al.* suggest that devices should voluntarily enter either the *INQUIRY* or *INQUIRY_SCAN* state. In their symmetric link formation protocol, devices alternate between *INQUIRY* and *INQUIRY_SCAN* after a random state residence time. If two devices meet in complementary states for longer than the required formation delay a link is formed. Although the authors claim that the protocol is distributed, since devices spontaneously engage in scatternet formation, it does require a centralized leader with global knowledge and can therefore not be considered a distributed solution. This formation protocol creates a 2-MSM connected scatternet using single bridge links between exactly two piconets. Further, BTCP is divided into three phases. During the first phase a coordinator is chosen using a leader election process. Thereafter, the coordinator determines the master and bridge roles based on a formula that imposes a 36 node upper bound on the scatternet. Finally the links are formed. BTCP requires *en masse* node arrival and does not take mobility and node failures into consideration.

Ramachandran *et al.* present both a deterministic and a randomized algorithm to form ad hoc clusters in [2]. Both approaches involve a leader election of a super-master, which subsequently forms the actual topology in a centralized manner. The deterministic approach

is similar to the symmetric link formation in BTCP, in which nodes alternate between *INQUIRY* and *INQUIRY_SCAN* states. Nodes that discovered each other form a virtual inquiry response tree with the root node as a master. The master then forms a cluster, or a piconet, from the devices in the virtual response tree. Thereafter, a super-master is elected among the masters and the clusters are interconnected. In the randomized approach, devices determine their inquiry role probabilistically using several rounds of Bernoulli trials. This determines the master and slave designates and ultimately the makeup of the clusters. The MSM scatternet topology is then formed by the super-master in the same manner as in the deterministic approach. Both algorithms require *en masse* node arrival, but do not specify how bridges are used to interconnectthe clusters.

Zaruba *et al.* present Bluetrees in [28]. The authors present two variations of their algorithm: Blueroot Grown Bluetrees and Distributed Bluetrees. The Distributed Bluetrees approach is a multi-hop solution and is described further in Section 3.3.2. A Blueroot Grown Bluetree creates a TH topology and is formed from an arbitrarily selected coordinator node called the *blueroot*. A rooted spanning tree is built from the blueroot using a neighborhood topology graph. The root node is a master and every one-hop neighbor is a slave. The children are then assigned an additional role as a master in another piconet and the tree is recursively formed. Moreover, each internal tree node is a Master/Slave (MS) bridge node. The authors limit the number of slaves to five to prevent exceeding the seven active slave limitation in piconets and to introduce excessive overhead. Blueroot Grown Bluetrees require radio vicinity for all nodes while its distributed counterpart does not.

Sun *et al.* present a self-routing Bluetree TH scatternet topology in [45]. Their approach is directly based on Blueroot Grown Bluetrees and binary search trees. After the tree is formed routing is trivial. The node insertion position in the Bluetree is determined by the root based on the *BD_ADDR* of the new node. *QUERY* messages are passed up the tree to inform internal nodes and the root of the current range of *BD_ADDR*'s of each node's children. Therefore, when the root node receives a *JOIN* request from a new node it can

easily determine in which of its children's ranges the node should be placed. Once the root finds the correct insertion position, the new node is inserted as a leaf node. If the newly inserted node is the root of a subtree, some of its children might violate the Bluetree range constraints. This is solved by the root(s) propagating allowable range messages down the tree. Once a node detects a child that is outside the allowable range, that child is disconnected and subsequently rejoined by the root at the correct position. The protocol supports incremental node arrival and is self-healing, even when multiple nodes fail. It also makes scatternet routing trivial with the tradeoff of continuous tree maintenance. As with other TH topologies, nodes close to the root are bottlenecks and failure of bottleneck nodes results in scatternet partitioning. It is also possible that unbalanced trees are formed, which eradicates the benefit of the logarithmic average path length.

Lin *et al.* present the Bluerings protocol in [68]. Their approach forms a 2-SSR topology, in which only Slave/Slave (SS) bridges are used. In contrast to the Bluerings approach in [17] (described in Section 3.3.1), this protocol allows complete piconets to join the ring structure. The protocol relies on a leader election process to form the piconets and assigns bridge connections in accordance with the ring topology constraints. Packet routing is done unidirectionally, except during maintenance operations when the ring is broken. In those instances the packets can be routed in the reverse direction. Exactly two bridge nodes are present in each piconet: one upstream and one downstream. Further, the bridges are connected between exactly two piconets to ensure the ring structure. The protocol is self-healing and reconstructs the ring upon node failures using Dedicated Inquiry Access Codes (DIACs) to reconnect missing bridge links. The protocol simplifies routing with the tradeoff of longer average path lengths.

Distributed Solutions

In contrast to the previous section, distributed single-hop approaches do *not* depend on a single device to form a scatternet.

Law *et al.* present a randomized protocol that formed a Slave/Slave Mesh (SSM) in [16].

Devices are partitioned into components, which consist of a single device or a piconet in which the master is the component leader. Similar to the randomized link formation approach in [2], Law *et al.* also use asymmetric link formation. Devices probabilistically determine whether to enter the *INQUIRY* or *INQUIRY_SCAN* state. Each leader of a component, disjoint device or master attempt either to add additional slaves to its piconet or, if it currently has no slaves, to join another piconet. Each leader of a component tries to find other leaders and relinquish leadership. Finally, only one component leader is still active to connect additional slaves. To optimize the scatternet topology, the protocol allows merging of piconets and migration of slaves between them. The protocol incrementally forms a 2-SSM with bridge nodes participating in exactly two piconets. It is optimized to minimize time and message complexity while allowing incremental joins. However it does not handle device failures.

Chun-Choong *et al.* propose Bluerings to form ring structured scatternets in [17]. The Bluerings approach produces a 2-MSR topology in which all bridge nodes are MS bridges and each bridge is connected to exactly two piconets. For a complete Bluering to be formed, *en masse* node arrival and radio vicinity is assumed. The algorithm is categorized as a distributed solution since each node is assumed to be engaged in formation independently, although simultaneously. There are two variations of the algorithm. In the first, the head of the semi-connected ring performs *INQUIRY* while the tail performs *INQUIRY_SCAN*. Disjoint nodes probabilistically choose between the two inquiry states. The head of the semi-connected ring has to have the largest identifier of the connected nodes to prevent loops from forming, before all nodes are included. Loops are avoided by allowing inquiry-scanning nodes to connect only to nodes with smaller identifiers. In the second variation, loops are prevented by disallowing the tail from engaging in inquiry and inquiry scanning all together. Further, only disjoint nodes and the head are allowed to probabilistically select an inquiry state. Both variations terminate the topology formation after a timeout period during which no additional nodes are connected. A ring topology has the advantage of every node having two paths to any other node (as long as the ring is maintained) and constant

path length. As another advantage the path is determined trivially. The disadvantages of this specific approach include excessive packet latency and much longer average path length (N/2) than, for instance, in TH topologies. In addition, neither of the two algorithms provide any methods for maintaining the ring topology, which takes care of incremental arrivals and node failures.

Tan *et al.* present a Tree Hierarchy (TH) scatternet formation protocol in [27] and [26]. The Tree Scatternet Formation Protocol (TSF) forms a distributed TH scatternet, which allows incremental arrivals and handles node failures. As in [16] TSF partitions devices into components. Each component in TSF is either a single free node or a subtree that seek to join another tree in the forest. Similar to BTCP, TSF use symmetric state transitions to establish links. After a links is formed, the master becomes the root and the slave becomes a leaf node. TSF restrict free nodes to connect only to other non-root nodes and other free nodes, while root nodes can connect only to other root nodes. When two root nodes are connected, one becomes the master while the other becomes a slave. These restrictions prevent loops from forming. Self-healing of the tree is native to the protocol. Internal nodes that lose connectivity to their parent become roots and attempt to connect to another root node, while roots that lost all their child nodes becomes free nodes. TSF isolates communication between root nodes using the Limited Inquiry Access Code (LIAC), which is native to the specification. TSF is self-healing, distributed, produces a connected scatternet, and simplifies scatternet routing. However, due to the nature of TH topologies, the root node is a bottleneck in the network and node failures close to the root partitions large portions of the scatternet. The authors note that TSF is not guaranteed to heal network partitions when all devices are not within radio vicinity [27]. The protocol is therefore classified as a single-hop topology. Based on its decentralized nature and the interconnection of rooted subtrees it could, however, function in a multi-hop scenario. Although in a such case it would not necessarily maintain its self-healing and connected properties. A multi-hop optimization of TSF called SHAPER [24] is described further in Section 3.3.2.

3.3.2 Multi-hop Topologies

In this section we cover protocols that do not require all devices to be within radio vicinity of each other.

In Section 3.3.1 we describe the coordinated single-hop Bluetrees approach by Zaruba *et al.* [28]. The authors also propose a multi-hop version called Distributed Bluetrees that works analogous to Blueroot Grown Bluetrees, except that multiple *init nodes* are used. The node with the highest ID in the local neighborhood is selected, using an election process, as an *init node*. The second phase of the protocol merges the subtrees into a single TH scatternet topology. Whereas Blueroot Grown Bluetrees requires radio vicinity for all nodes, Distributed Bluetrees does not. Distributed Bluetrees functions in a distributed manner to form a multi-hop scatternet, but cannot always guarantee connectivity of all subtrees. However, it does simplify scatternet routing since the formed topology has a tree structure.

Wang *et al.* present Bluenets in [74]. In their approach the resulting scatternet topology is formed as a 2-SSM. They emphasize that average path length in Bluenets are shorter and a Bluenet can sustain higher traffic flows than in Bluetrees. The authors also explain both these observations by stating that, in the heavily connected mesh topology (2-SSM), paths are not required to go along the congested and non-optimal path through the root node. The trade-off is that routing in the Bluenet scatternet is much more complex than in Bluetrees. Analogous to Bluetrees, Bluenets are not able to guarantee scatternet-wide connectivity.

Basagni *et al.* propose a multi-hop solution in [5]. In their three-phased approach, devices first engage in discovery using a symmetric link formation protocol. In addition to the traditional symmetric link formation behavior found in [67], devices also exchange a weight parameter. After the discovery and information exchange, devices have knowledge of the local neighborhood graph. Based on the exchanged weights, devices in the local neighborhood with the highest weight, called *init* devices, become masters. These devices then initiate the formation phase and start to form Bluestars, which essentially are just piconets. Disjoint devices that receive a page from a device with a larger weight join as slaves. After the *init*

devices have successfully paged all its neighbors with smaller weight, the second phase is done. Devices that have only neighbors with larger weight become masters and page other devices with smaller weight. In the final phase, a BlueConstellation scatternet is formed. The BlueStar masters determine the *init* masters by comparing weights with neighboring masters that are either two or three hops away. The *init* masters then instruct their slaves to page specific neighbors to form gateways to neighboring BlueStars. Thereafter, they engage in a Master/Slave (MS) switch and the paging slave become a bridge node between the two BlueStar piconets. However, only neighboring masters that are two hops away can receive the page from the gateway slave. In the event that the neighboring master is three hops away, a gateway piconet is created to join the two BlueStars. The protocol guarantees scatternet connectivity in a multi-hop environment. However, it provides neither self-healing nor does it allow incremental node arrival, because the protocol is divided into multiple phases.

In [24] Cuomo *et al.* present the SHAPER algorithm. It is based on TSF [27], but ensures connectivity and self-healing in a multi-hop scenario. The main difference between TSF and SHAPER is that while TSF uses a set of coordinators (that were roots of subtrees), SHAPER allows both root and non-root nodes (of partitioned subtrees) to form links and initiate tree reconfiguration. There is a trade-off from allowing all nodes to form links to other subtrees. Namely the fact that they all have to periodically engage in either inquiry or inquiry scanning. In TSF this is limited to coordinator nodes, which prioritize scatternet communication over costly formation for non-root nodes. However, as the authors of SHAPER also state, this potentially prevents healing of the TH topology when coordinators are not within radio transmission range of each other. SHAPER avoids this and ensures connectivity. Initial tree formation is conducted in the same fashion as in TSF. The important differences lay in the way SHAPER handles link establishments between nodes in different tree partitions. There are several cases. In the first case, in which both connected nodes are roots, the root of the larger tree becomes the new root and the topology is reconfigured accordingly. If the slave on the link has the larger tree, a Master/Slave (MS) switch has to be performed first. This

case is similar to TSF. In the second case, either the slave device is a root and the master a non-root or the slave device is a free node. Either way the tree is simply attached and the parameters are updated. The third case occurs when the master on the link is either a root or a free node and the slave is already part of a tree. This is analogous to the first case, except that tree sizes does not have to be compared. A Master/Slave (MS) switch has to be performed first and the topology is then reconfigured. In the last case, both connected nodes are non-root nodes. The smaller of the two trees is reconfigured and the parent-child relationships are inverted so that all nodes in the reconfigured tree are descendants of the connected node. To prevent the simultaneous initiation of multiple reconfigurations, SHAPER implements a locking mechanism that forces non-root nodes to obtain permission from the root before reconfiguring the tree. The authors claim that SHAPER have a lower average formation delay than TSF. As previously mentioned, the trade-off is that non-root nodes have less time to engage in communication, which reduces communication efficiency.

A radically different approach from the solutions described so far is proposed by Liu *et al.* in [73]. The authors empasize that approaches that attempt to ensure scatternet connectivity have to be periodically maintained regardless of whether there is any traffic across the links. Instead they propose a solution that establishes a scatternet along multi-hop routing paths on-demand. Their scatternet topology is built as part of the route discovery and includes only devices along the route from the source to the destination. For intermediate nodes along the scatternet route, master and slave roles also have to be assigned. Ideally every other device on the route will be a master and the rest will be slaves. All bridge nodes will then be Slave/Slave (SS) bridges. However, due to dynamic route changes and uneven route lengths they also allow Master/Slave (MS) bridges when necessary. Similar to general ad hoc on-demand routing protocols, a route request packet is flooded from the source node [13, 21]. Once the destination receives a route request packet it sends a route reply packet along the reverse path back to the source. The route flooding is somewhat more difficult in Bluetooth® than in traditional Direct Sequence Spread Spectrum (DSSS) ad hoc

networks, since a link has to be explicitly established before information can be exchanged between nodes. The authors note that establishing point-to-point links along all potential paths to broadcast the route request imposes an excessive amount of overhead. Instead they incorporate the route request packets in the pre-existing inquiry broadcast mechanism. Two types of Extended ID (EID) packets, containing the additional route request information, are needed since enough information cannot not be accommodated in a single EID packet. This is accomplished by sending the type 2 packet during the response time slot (after the inquiry broadcast), since the type 1 EID packet does not require instant acknowledgement. In fact, the Random Backoff (RB) delay mechanism from the Bluetooth® specification discourages immediate replies to thwart collisions. After the destination receives the first route request, the scatternet is formed backward along the reverse route based on the transmission of the route reply packet. Thereafter data packet routing is accomplished by next-hop entries at each intermediate node, similar to [13]. To reduce the path latency due to bridge switching overhead the authors propose to align the time slots along a route for efficient path traversal. However, when multiple routes co-exist, this alignment becomes increasingly difficult.

3.3.3 Optimized Topologies

Scatternet formation approaches based on theoretical foundations offer important insight into how optimized topologies can be constructed. This section covers some of the proposed theoretical approaches.

Yun *et al.* present an approach that forms an MSM topology in [36]. Their Bluestars approach models the discovery neighborhood as an inquiry graph I. From the inquiry graph I, $2^{|I|}$ topology subsets are available and one is selected. A combination of asymmetric and symmetric link formation is also used. Symmetric link formation is used for *en masse* arrival, while asymmetric link formation is used for incremental arrivals. After determining a topology graph T, neighboring nodes are grouped into Bluestars (piconets) and links are established between each neighboring Bluestar. The Bluestars are further pruned into

Table 3.1: Protocol Comparison Chart

Protocol Authors	Coordinated	Distributed	Multi-hop	Incremental Arrival	Topology Model	Link Formation	Self-healing	Ensures Connectivity	Comments
Salonidis et al. [67],[66]	Yes	No	No	No	2-MSM	Symmetric	No	Yes	*Max 36 devices*
Ramachandran et al. [2]	Yes	No	No	No	MSM	Asymmetric/ Symmetric	No	Yes	*Bridge connections are not defined*
Zaruba et al. [28]	Yes	No	No	No	TH	Asymmetric	No	Yes	*Blueroot Grown Bluetrees version*
Sun et al. [45]	Yes	No	No	Yes	TH	Asymmetric	Yes	Yes	*Self-routing*
Lin et al. [68]	Yes	No	No	Yes	2-SSR	Asymmetric	Yes	Yes	*Stateless routing*
Law et al. [16]	No	Yes	No	Yes	2-SSM	Asymmetric	No	Yes	*Optimized for time and message complexity*
Chun-Choong et al. [17]	No	Yes	No	No	2-MSR	Asymmetric	No	Yes	*Stateless routing*
Tan et al. [27],[26]	No	Yes	Yes, if root nodes are within vicinity	Yes	TH	Symmetric	Yes	No	*Can function in multi-hop environment*
Zaruba et al. [28]	No	Yes	Yes,if init nodes are within vicinty	No	TH	Asymmetric	No	No	*Distributed Bluetrees version*
Wang et al. [74]	No	Yes	Yes	No	2-SSM	Asymmetric	No	No	
Basagni et al. [5],[61]	No	Yes	Yes	No	Degree constrained SSM	Symmetric	No	Yes	*Ensures connectivity in mult-hop environment*
Cuomo et al. [24]	No	Yes	Yes	Yes	TH	Symmetric	Yes	Yes	*Multi-hop version of TSF*
Liu et al. [73]	No	Yes	Yes	Yes	MSM	Asymmetric	Yes	No	*Scatternet is formed along on-demand route*

Bluestars* by removing redundant links. This solution is similar to [5], but requires a costly determination of the optimal topology subset before the scatternet can be formed.

Cuomo *et al.* define a methodology for scatternet formation based on graph theory [19]. They describe the scatternet as a bipartite graph and model it using an adjacency matrix. After evaluating all possible matrices with respect to the chosen optimization metrics, an optimized topology is determined. This SSM topology can then be formed using a centralized strategy. The evaluation metrics used are classified as either traffic dependent or traffic independent. The traffic dependent metrics include residual scatternet capacity and average node load, while the traffic independent metrics include maximized overall capacity and average path capacity.

Chiasserini *et al.* present an optimized approach for scatternet formation that attempts to minimize the traffic load [14]. The authors assume that traffic patterns and routes are known *a priori* and formalize the topology formation as a min-max problem. First, they locate the bottleneck node in the network and then determine the topology to minimize the traffic load at that node. Although the approach is not applicable in practice, the authors note that it does provide insight into how topologies that fulfill the capacity constraints in Bluetooth® scatternets should be formed. They also discuss a distributed approach that incrementally forms a scatternet and utilizes Dedicated Inquiry Access Codes (DIACs) to classify nodes, based on individual characteristics such as load, battery power, and computational capacity.

Baatz *et al.* present an approach to optimize scatternet capacity in [64]. They note that due to the FH structure and communications scheduling employed in Bluetooth, only a subset of nodes can communicate on disjoint links at any time. Therefore, they model the scatternet as a directed graph and determined matchings between nodes so that no two links (edges) communicate with the same node (vertex) at the same time. Thereafter, they define a set of these matchings as a *scatternet schedule*. Based on the schedule, a feasible rate vector is defined that conforms to the scatternet capacity constraint. A max-min fair rate

vector that equally distributes the scheduling across all the maximal matchings is obtained to maximize capacity, while ensuring that all links are served in a fair way. A scatternet topology is then determined by partitioning the links into disjoint sets so that (near) perfect matchings, called (near) 1-factors, are obtained. The resulting topology will differ depending on how these 1-factor matchings are chosen. The authors also state that the actual topology formation can be done using a centralized single-hop approach such as BTCP [67].

Scatternet formation based on dominating sets is of interest due to the similarity between the Connected Dominating Set (CDS) and the subsets of masters and bridges in scatternets. Wan *et al.* present a distributed approximation algorithm to construct a Connected Dominating Set (CDS) in ad hoc networks [57]. Their algorithm consists of two phases, during which a Minimum Independent Set (MIS) and a dominating tree are formed respectively. The MIS consist of a subset of nodes that are separated by exactly two hops. The dominating tree T^* is constructed during the second phase and its internal nodes form a CDS. The algorithm is not directly applicable to scatternets, but present a constant factor approximation to the NP-hard Minimum Connected Dominating Set (CDS) problem.

Stojmenovic present an algorithm for scatternet formation that is based on the concept of dominating sets in [65]. This approach is based on the CDS algorithm from [72], in which a CDS is constructed by including all nodes and then removing locally redundant nodes. Stojmenovic eliminates redundant neighbors by applying a Yao subgraph construct on the Relative Neighborhood Graph (RNG) or the Gabriel Graph (GG) to produce the dominating set. Thereafter, device roles are determined so that the scatternet can be connected.

Zussman *et al.* study capacity assignment in scatternets in [78]. They develop some algorithms to minimize the average delay in a scatternet. They observe that scatternets using only Slave/Slave (SS) bridges are necessarily bipartite and state the problem of optimal capacity assignment for both bipartite and non-bipartite graphs. By assuming an existing topology and traffic flow information, they develop an optimized algorithm that finds the optimal capacity vector for the capacity assignment problem. They also develop some

heuristic algorithms that find approximate solutions to the same problem. They conclude that bipartite scatternet topologies (using only SS bridges) provide the best performance.

3.4 Summary

In this chapter we discussed different criteria for scatternet formation and presented topology models that conform to varying constraints. We reviewed the state-of-the-art approaches with respect to scatternet formation and categorized these approaches into single-hop, multi-hop, and optimized solutions. We further classified the resulting topologies by the general topology models. Single-hop solutions required that all devices were within radio vicinity of each other, whereas multi-hop protocols did not impose such a constraint. We also subdivided the single-hop solutions into coordinated and distributed approaches. Coordinated algorithms required a leader to control the formation, while distributed solutions did not. The reviewed scatternet formation approaches were compared and contrasted in Table 3.1. As previously mentioned, depending on the criteria for the scatternet and the constraints imposed any of the solutions from Table 3.1 could be suitable. We also reviewed some optimized solutions that for the most part were not directly usable in practice. For that reason we chose not to include the solutions from Section 3.3.3 in Table 3.1.

Chapter 4

A Fault-Tolerant Distributed Formation Protocol for Bluetooth Scatternets

4.1 Introduction

The Bluetooth® specification [11] and the IEEE 802.15.1 [33] standard define properties of a scatternet but do not explicitly provide a protocol for scatternet formation. A scatternet can be formed by inter-connecting piconets in a way that does not violate the existing constraints imposed on the participating piconets. It must be formed by explicitly inter-linking disjoint piconets. Due to the unique structure of a scatternet, traditional wireless ad hoc topology formation protocols cannot be directly applied. In this chapter we present our Bluetooth Distributed Scatternet Formation Protocol (BTDSP).

Our scatternet formation approach is designed to operate in a multi-hop environment, which means that nodes are not required to be within transmission range of each and every other node. The approach does not rely on a single coordinator and the algorithm is not subdivided into multiple phases. A scatternet is formed in a distributed fashion by allowing each node that is not already part of the scatternet to make a probabilistic decision whether it should attempt to form links as a master or as a slave. Since each piconet can only contain a single master and as many as seven active slaves, we aim to optimize the proportions of designated master and slave roles.

For fault-tolerance it is also extremely important that incrementally arriving nodes and previously disconnected nodes are quickly either incorporated into an existing piconet or are allowed to form their own piconet. Our approach handles this requirement by using a local threshold value at each node. For disjoint nodes that are not a part of any piconet, the threshold is set low so that slave roles are chosen by them more often than master roles. Once a node becomes a slave, it relinquishes its pursuit to form new links (unless specifically directed by the master to form a bridge inter-connection). For piconet masters, the threshold is increased proportional to the number of connected slaves. Instead of pursuing inclusion into another piconet, the master makes a probabilistic decision between attempting to incorporate an additional slave or assigning an existing slave as a new bridge designate. The concept of bridge designates will be described in detail later in this chapter. This strategy effectively makes the master of a larger piconet able to incorporate additional slaves and increase its own piconet size, with higher probability.

We assume that only pure slaves (i.e., slaves that are connected to only a single master) are considered as potential bridge designates. This assumption further ensures the formation of a flat scatternet topology that is free from the bottlenecks found in tree-based algorithms (e.g., [28], [27]). We make this assumption since a Master/Slave (MS) bridge node wastes time slots when it is away from a piconet (in which it is a master) and participates as a slave in a different piconet. Kalia *et al.* suggest in [44] that the use of Master/Slave (MS) bridges negatively affect piconet performance since all intra piconet communication must be put on hold while the master is participating in another piconet. Misic *et al.* present similar results in [48] and conclude that inter-piconet delay is also significantly increased when Master/Slave (MS) bridges are used.

In our protocol nodes are allowed to arrive incrementally and join the existing scatternet. The protocol is also self-healing, meaning that nodes that have been disconnected can easily be re-incorporated into the scatternet. Depending on whether a link was broken due to wireless interference or node failure, links are either re-established or the node is incorporated

elsewhere in the scatternet. We explain this further in detail in Section 4.3.

The rest of this chapter is organized as follows. We summarize previous approaches for device discovery and scatternet formation in Section 4.2 before presenting the algorithm in Section 4.3. Section 4.3.1 established some preliminaries. In Section 4.3.2 we discuss the issues involved in device discovery and scatternet formation. We outline the basic idea behind our algorithm in Section 4.3.3. In Section 4.3.4 the complete scatternet formation algorithm is presented. In Section 4.4 we present the performance evaluation and discuss our results. The chapter is concluded in Section 4.5.

4.2 Related Work

Scatternet formation can be accomplished in several different ways. Depending on what constraints are placed on the algorithm, the resulting topology can have very different characteristics. Scatternet formation approaches are classified as single-hop or multi-hop solutions [60]. Single-hop solutions require that all nodes are within transmission range of each other, while multi-hop solutions do not impose such a restriction.

Single-hop solutions, such as [2, 67, 16, 17], require that all devices are within proximity of each other. The solutions proposed in [2] and [67] also require an election process that first locates a coordinator, which assigns roles to devices and forms the topology.

On the contrary, multi-hop solutions are distributed by nature. They do not require all devices to be within proximity of each other and form a variety of different topology models. For example, in [28] and [27] scatternets are formed as rooted trees while in [74] and [5] mesh and star topologies are formed respectively. In [73] the authors take a different approach and form scatternets on-demand for the duration of a route.

We briefly summarize previous solutions that influenced our algorithmic design decisions. A comprehensive review and classification of these scatternet formation approaches can be found in Section 3.3.

The Bluetooth Topology Construction Protocol (BTCP) protocol by Salonidis *et al.* is a

single-hop solution that utilizes asymmetric link formation and a leader election mechanism across multiple phases to form a 2-MSM scatternet topology [67, 66]. The protocol also imposes a 36 node upper bound, requires *en masse* node arrival, uses inefficient Master/Slave (MS) bridges, and does not take into consideration mobility and node failures.

The deterministic and randomized clustering algorithms by Ramachandran *et al.* use a leader election of a super-master and form the MSM scatternet topology in a centralized manner [2]. Both algorithms require *en masse* node arrival and do not specify how bridge nodes interconnect the clusters into a scatternet.

Using probabilistic asymmetric link formation, the single-hop algorithm by Law *et al.* forms a mesh topology by merging disjoint components [16]. Component leaders try to find other leaders to whom they can relinquish leadership until a single component leader remains active to connect additional slaves. Although the algorithm does not require centralized coordination and allows incremental joins, it does not handle device failure and scatternet healing.

The Bluetrees protocol by Zaruba *et al.* is presented in two variations: Blueroot Grown Bluetrees and Distributed Bluetrees [28]. The Blueroot Grown Bluetrees algorithm form a single-hop TH topology from an arbitrarily selected coordinator node called the *blueroot*. The distributed version merely adds multiple blueroot nodes and later merges the partitions. Blueroot Grown Bluetrees requires that every node is within radio vicinity of every other node, while its distributed counterpart does not. The main drawback of the approach is that the hierarchical topology makes higher level nodes bottlenecks and susceptible to partitioning.

The BlueRing protocol by Lin *et al.* forms a 2-SSR scatternet topology by interconnecting piconets into a ring [17]. It relies on a leader election process to form the piconets and assigns bridge connections in accordance with the ring topology constraints. Packet routing is simple and done unidirectionally with the trade-off of longer average path lengths. However, it is still based on a leader election process and suffers from expensive ring maintenance operations.

The distributed Bluerings approach by Chun-Choong *et al.* produces a 2-MSR topology, in which all bridge nodes are Master/Slave (MS) bridges and each bridge connects to exactly two piconets [17]. Their algorithm requires *en masse* node arrival and radio vicinity; however, it does not depend on a leader election process. The ring topology has the advantage of trivial routing, two paths to any node, and constant path length. However, the approach produces excessive packet latency and much longer average path length (N/2) than in Tree Hierarchy (TH) topologies (log N). Further, Bluerings neither provides ring maintenance for incremental arrivals nor handles node failures.

The distributed Tree Scatternet Formation Protocol (TSF) approach by Tan *et al.* forms a TH scatternet topology while allowing incremental arrivals and handling node failures [27, 26]. TSF partitions nodes into components that are either a single free node or a subtree that seeks to join another tree in the forest. Although TSF is distributed, self-healing, and produces a connected scatternet, it does not guarantee to heal network partitions when all devices are not within radio vicinity of each other. Furthermore, it uses inefficient Master/Slave (MS) bridges and the tree topology introduces bottlenecks in the scatternet.

The Bluenets approach by Wang *et al.*, an extension of Bluetrees, attempts to minimize inter-piconet connections and forms an MSM scatternet topology [74]. In contrast to Bluetrees, they emphasize a shorter average path length and the fact that a Bluenet can sustain higher traffic flows since, in their heavily connected Bluenet mesh topology, paths are not required to go along the congested and non-optimal path through the root node. Analogous to Bluetrees, the Bluenet is not able to guarantee scatternet-wide connectivity. It also suffers from the use of inefficient Master/Slave (MS) bridges and a phase-divided algorithm.

In the three-phased multi-hop solution by Basagni *et al.*, BlueStar piconets are connected into a BlueConstellation scatternet [5]. Using symmetric link formation with a weight parameter to form a local neighborhood graph, Bluestars (piconets) are formed by devices in the local neighborhood with the highest weight, called *init* devices. The BlueConstellation scatternet is formed when *init* masters instruct their slaves to page specific neighbors to

form gateways to neighboring BlueStars. The protocol guarantees scatternet connectivity in a multi-hop environment. However, it neither provides self-healing nor allows incremental node arrivals because the protocol is divided into multiple phases.

The SHAPER algorithm by Cuomo *et al.* is directly based on TSF with the modification that it ensures connectivity and self-healing in a multi-hop scenario. The main difference between TSF and SHAPER is that TSF uses a set of coordinators, roots of subtrees, to form links and initiate tree reconfiguration, while SHAPER allows both root and non-root nodes of partitioned subtrees to perform these operations. The authors claim that SHAPER has a lower average formation delay than TSF. The trade-off is that non-root nodes have less time to engage in communication, which reduces efficiency.

The radically different approach by Liu *et al.* builds a scatternet on-demand along multi-hop routing paths as part of route discovery. The approach has the advantage that it does not require a network-wide scatternet. However, it requires complex IPS coordination and significant modification to the Bluetooth® specification to function.

The following three key points summarize the main disadvantages of the previously described approaches:

- Reliance on a leader/coordinator to initiate and control the formation is not scalable and requires that all nodes are within transmission range of each other.
- Approaches that are phase divided do not allow incremental node arrival and require complex coordination.
- The use of Master/Slave (MS) bridges negatively impacts performance and also incurs bottlenecks and severe partitioning problems for hierarchical (tree-based) solutions.

In the next section we describe our distributed and self-healing scatternet formation approach. It is designed to remedy some of the drawbacks found in previous approaches.

4.3 A Fault-Tolerant Distributed Scatternet Formation Algorithm

4.3.1 Preliminaries

Before we get into the details of our scatternet formation algorithm, we first establish some preliminaries based on the observations from the previous section. It is important to take into consideration the manner in which devices are distributed to piconets, as well as how many piconets each bridge node must switch between. The *bridge degree*, or piconet connectivity degree, reflects the number of piconets in which a bridge node participates. Limiting the bridge degree reduces routing latency since inter-piconet switching is very expensive and a high bridge degree results in less frequent piconet visits from the bridge node [31]. In addition to reducing bridge degree, our algorithm assigns bridge roles to only pure slaves and forms a 2-SSM flat mesh topology. Pure slaves are piconet members that participate in only *one* piconet as a slave, i.e., they do not participate in multiple piconets as bridge nodes. This restriction prevents scatternet performance degradation, found in Tree Hierarchy (TH) topologies, since the use of Master/Slave (MS) bridges significantly increases inter-piconet latency [44, 48]. Performance degrades when using MS bridges because all intra-piconet communication is put on hold while an MS bridge is away participating in another piconet. Next we describe how device discovery is performed.

4.3.2 Device Discovery

To form a scatternet neighboring devices must first discover each other. This is done by initiating inquiry procedures. Devices discover other nodes by forming a brief peering in the complementary *INQUIRY* and *INQUIRY_SCAN* states. When a device enters the *INQUIRY* state, it starts transmitting *INQUIRY* packets across a predetermined set of inquiry frequencies. Peers that simultaneously reside in the complementary *INQUIRY_SCAN* state can thereafter respond to the inquiry. Consequently, an inquiring node must briefly connect to each one of the inquiry scanning nodes in order to discover and subsequently incorporate

them into a piconet.

Traditionally, as outlined in the Bluetooth® specification [11], device discovery is accomplished by explicitly assigning a role to each device: either master or slave. However, in a scatternet topology this should be done in an ad hoc manner without the need for specific role assignments. In both [27] and [67] the authors propose self-configuring schemes based on symmetric device discovery. Devices alternate between *INQUIRY* and *INQUIRY SCAN* states until pairs of devices meet in complementary states and form a connection. Although these methods produce a scatternet, piconet master and slaves roles are assigned randomly. These approaches frequently produce piconets of uneven size.

In a piconet, slaves are clustered around the master and share the bandwidth of the piconet. It is therefore important to distribute nodes evenly in piconets to maximize throughput. Uneven node distributions result in some bandwidth saturated piconets while other piconets have a lot of available capacity. An abundance of small-sized piconets also increases the scatternet diameter, leading to an added number of bridge traversals along a route. As previously mentioned, inter-piconet switching is an expensive operation and hence is not a desirable scatternet characteristic. Thus, in our approach we take piconet density into special consideration and attempt to incorporate willing scatternet participants into fewer piconets, but without relying on merging procedures or other artificial mechanisms to increase piconet density.

As we emphasized in the previous section, nodes with master roles cannot be efficiently used as bridge nodes since they control the intra-piconet communication. We therefore designed our algorithm to form a flat topology that does not suffer from the drawback of using Master/Slave (MS) bridge nodes. Our algorithm is also capable of operating in a multi-hop environment, i.e., it does not require all devices to be within radio transmission range of each other.

4.3.3 Basic Idea And Motivation

In this section we describe the basic idea and the motivation behind our Bluetooth Distributed Scatternet Formation Protocol (BTDSP). It is desirable that an ad hoc topology formation algorithm does not to rely on any centralized processes or a specific set of nodes. Our scatternet formation algorithm is therefore completely distributed in nature. Further, we do not want the resulting topology to have any specific congestion points or bottlenecks. Tree Hierarchy (TH) topologies that are formed using MS bridges exhibit this undesirable behavior [60]. Additionally, to prevent excessive inter-piconet switching we limit the number of piconets in which a bridge can participate to exactly two. This decreases the bridge node's piconet visit intervals with the trade-off of a sparser mesh topology. Thus, our algorithm forms a 2-SSM flat mesh topology.

We initially assume that all Bluetooth® devices are disjoint. However, the algorithm is capable of accommodating existing piconets and late arriving devices without modification. Before a scatternet can be created, individual piconets must first be formed and populated. This is accomplished by extending the device discovery mechanism provided in the Bluetooth® specification [11]. The first step is to determine the master and slave roles of participating devices. Our probabilistic approach assigns a small number of devices the master role and the rest slave roles. As more and more slaves join a piconet, a local probabilistic threshold value at each master is increased. As the threshold is increased, the probability that the master device scans for additional slaves increases. On the other hand, disjoint devices with a low threshold are more likely to join one of the existing piconets than trying to form their own new piconet. Thereby, our algorithm favors compactness and alleviates formation of large number of small piconets, which could lead to longer routing paths.

As the piconet reaches full capacity, the master stops attempting to incorporate more slaves into its piconet. However, the master continues to attempt to increase scatternet connectivity by forming bridge connections. To form a bridge connection the master randomly picks a pure slave, if one exists, as a new bridge designate. This procedure is also performed

by existing piconet masters, with less than full piconet capacity, that are not probabilistically designated by our algorithm to incorporate more slaves into their piconets. A bridge designate is defined as a connected slave, in an existing piconet, that responds to inquiries (from another master) in order to form a bridge connection between the two piconets. Note that a bridge designate is merely a piconet node that has been chosen by the master. Whether it will become a new bridge node or not depends on the aptitude of surrounding masters to establish a new connection.

The Bluetooth® master-driven slotted Time Division Duplex (TDD) allocation scheme restricts slave devices to sending packets only after being addressed by the master. Therefore, bridge designate slaves have unused slots during which they can scan for inquiries. Each slave is informed of the next slot during which it will be polled by the master through the IRPS and IPS schemes. In between these times the bridge designate can engage in inquiry scanning and respond to inquiries from neighboring masters. If a new bridge connection is formed, the bridge node informs each piconet master of the identity of the other master. As previously mentioned, we restrict bridge nodes to participate in exactly *two* piconets, since inter-piconet switching is an expensive operation. Each master also updates its *Bridge_Table* to indicate the *LT_ADDR* of the bridge and the *BD_ADDR* of the connected piconet's master. The *Bridge_Table* is also used for scatternet routing, which is addressed in Chapter 5.

4.3.4 Algorithm

Our Bluetooth Distributed Scatternet Formation Protocol (BTDSP) is based on periodic execution of the initialization procedure BT-Init (illustrated in Figure 4.1) by every participating device. Within this procedure only masters and nodes that are not part of a piconet are able to choose whether to initiate a device discovery, as a master or slave, or scan for bridge designates. This restriction is enforced by checking that a device does not have an *LT_ADDR*. Every active slave device is assigned a 3-bit *LT_ADDR* by its master, so this check effectively excludes all connected slaves. By forcing all nodes to revisit the BT-Init

procedure continually regardless of whether they will actually execute the procedure or not, we provide fault-tolerance in the event of disconnections and failures.

To proportionally distribute the master and slave role assignments, each node locally maintains a threshold value called p_{thres}. This value is adaptively adjusted as more slaves are connected to the piconet to increase the probability that the piconet master revisits BT-INIT and again initiates device discovery as a master. Meanwhile, disjoint nodes or masters of smaller piconets are probabilistically less likely to enter BT-INIT based on their local threshold value. The reasoning behind this is that we initially want a low Master/Slave (MS) ratio to favor the natural generation of larger piconets, as opposed to artifically moving nodes or merging piconets as done in [16]. Another threshold value b_{thres} is also used to control the number of bridge connections that are formed. However, unlike the adaptive nature of p_{thres}, the b_{thres} is set statically to η_1, where $\eta_1 = (0, 1]$. As described in Section 4.4, we initially set $p_{thres} = 0.08$ and $b_{thres} = 1$. p_{thres} is thereafter adaptively adjusted using a linear function while b_{thres} remains static.

```
BT-INIT()
 1  if !LT_ADDR
 2     then p_thresInit ← η0
 3          p_thres ← f_p(p_thresInit, s_count)
 4          b_thres ← η1
 5          p ← rand(0, 1)
 6          b ← rand(0, 1)
 7          if p < p_thres
 8             then BT_MASTER()
 9             else if s_count = 0
10                     then BT_SLAVE()
11             else if b < b_thres
12                     then BT_BRIDGE()
13  return
```

Figure 4.1: BTDSP initialization procedure BT-INIT

As mentioned earlier, the BT-INIT procedure is executed only by masters and disjoint devices. It utilizes a random variable p to determine the probabilistic outcome and consequently which sub-procedure a device will execute. The BT-MASTER inquiry procedure

(Figure 4.2) is executed by inquiring masters. Depending on its status, a device whose probabilistic outcome falls below the p_{thres} threshold, executes one of the inquiry scanning device procedures. A disjoint device enters the *INQUIRY_SCAN* state and scans for inquiries, while a piconet master instead may or may not, depending on the random variable b and the b_{thres} threshold value, execute the BT-Bridge (Figure 4.3) procedure and assigns a pure slave as a bridge designate to form a new bridge link. The BT-Slave procedure is simply the inquiry scan operation from the Bluetooth® specification [11].

Piconet Formation

The BT-Master procedure in Figure 4.2 is the core procedure for piconet formation. Devices that fail to enter BT-Master (and lack their own slaves) will enter the *INQUIRY_SCAN* state, indicated by BT-Slave, and attempt to connect to another master that is executing the BT-Master procedure. The BT-Master procedure operates in two rounds: the *inquiry round* followed by the *paging round*.

The inquiry round lasts for the duration of the *inquiryTO* interval or until a predetermined number of responses have been reached. Assuming that no SCO links are present, the *inquiryTO* is set to the scan interval of both the inquiry scan trains. Each inquiry scan train of 16 frequencies is covered in 2.56s [11]. We therefore set the *inquiryTO* to 5.12s.

For each response the inquiry scanning peer sends a Frequency Hopping Sequence (FHS) response packet. The packet contains the FHS (based on the slave's *BD_ADDR*) that the master should use for paging the device, as well as clock synchronization and device address information. When the master receives an FHS response packet it enqueues it and waits for additional responses for the remainder of the timeout interval.

The paging round follows the inquiry round if any responses are received. It lasts for the duration of the *pageTO* interval or until full piconet capacity has been reached. The *pageTO* is set to the default page timeout 5.12s, similar to *inquiryTO*. The response at the front of the queue is then dequeued and the device is paged. For each successful connection the Slave_{count} is also incremented.

```
BT-MASTER()
 1  Num_Inq ← 0
 2  Q ← EMPTY()
 3  while !inquiryTO and Num_Inq < MAX_INQUIRIES
 4  do INQUIRY()
 5     if INQ_RESP_FHS
 6       then Q.ENQUEUE(INQ_RESP_FHS)
 7            Num_Inq ← Num_Inq + 1
 8  while !pageTO and Num_Inq > 0 and s_count < 7
 9  do PageDev ← Q.dequeue()
10     PAGE(PageDev)
11     if CONNECTED(PageDev)
12       then BD_ADDR_pcnt ← lmp_scat_rep
13            if BD_ADDR_pcnt
14              then  /* Connected a bridge node */
15                    if !PICONET_LOOKUP(Bridge_Table, BD_ADDR_pcnt)
16                      then ADD(Bridge_Table, LT_ADDR, BD_ADDR_pcnt)
17                           s_count ← s_count + 1
18                      else  /* Node is already a bridge */
19                           DISCONNECT(PageDev)
20              else  /* Pure slave connected */
21                    s_count ← s_count + 1
22     Num_Inq ← Num_Inq − 1
23  return
```

Figure 4.2: BTDSP master procedure BT-MASTER

If a bridge connection is successfully formed, the bridge node sends a link manager PDU *lmp_scat_rep* that contains the *BD_ADDR* of the other inter-connected piconet master to each connected master. In this manner the masters can update their neighborhood information. In particular, the masters update their *Bridge_Table* to indicate bridge nodes and to which piconets those are connected.

Piconet Interconnection

Piconet masters that do not perform inquiries may or may not execute the BT-BRIDGE procedure (illustrated in Figure 4.3) based on the probabilistic outcome of the random value b and the threshold value b_{thres}. This procedure is the core of the actual interconnection of piconets. Each master uses a lookup table, called *Bridge_Table* (Table 4.1), that contains

```
BT-BRIDGE()
 1  if SIZE(Bridge_Table) < s_count
 2    then  /* Randomly pick a pure slave */
 3          Bridge_Designate ← Rand_Pure_Slave
 4  if Bridge_Designate
 5    then LMP_SCAT_INQ_SCAN(Bridge_Designate)
 6          while !pageTO
 7          do BD_ADDR_pcnt ← lmp_scat_rep
 8             if BD_ADDR_pcnt
 9               then ADD(Bridge_Table, Bridge_Designate, BD_ADDR_pcnt)
10  return
```

Figure 4.3: BTDSP bridge scan procedure BT-BRIDGE

the *LT_ADDR* of the bridge node and the *BD_ADDR* of the connected piconet's master. This table is used to track bridge connections between piconets, but also for our routing and security algorithms in Chapters 5 and 6 respectively.

The master first performs a lookup from the *Bridge_Table* and then randomly selects a slave that is not already in the table (a pure slave) as a bridge designate. If every slave is already in the *Bridge_Table*, then there are no pure slaves and the master will simply exit the procedure.

Table 4.1: Bridge_Table for scatternet formation

LT_ADDR	**Piconet**
001	$<BD\_ADDR_{piconet_A}>$
010	$<BD\_ADDR_{piconet_B}>$
011	$<BD\_ADDR_{piconet_C}>$
100	$<BD\_ADDR_{piconet_D}>$
101	$<BD\_ADDR_{piconet_E}>$
110	$<BD\_ADDR_{piconet_F}>$
111	$<BD\_ADDR_{piconet_G}>$

We propose the addition of the following two new Link Manager (LM) PDUs for scatternet formation:

- The *lmp_scat_inq_scan* PDU is sent from a master executing the BT-BRIDGE procedure to a bridge designate. It forces the slave to enter the inquiry scan state in between

polling slots. It does not require explicit acknowledgment; however, if a bridge connection is formed within the *pageTO* interval, an *lmp_scat_rep* PDU must be returned to *both* piconet masters.

- The *lmp_scat_rep* PDU is sent by the new bridge slave to each master and contains the *BD_ADDR* of the other connected piconet's master. Thereby, each piconet master can identify the neighboring piconets by their masters' addresses.

If the master finds a suitable candidate for a new bridge designate, it sends a link manager PDU *lmp_scat_inq_scan* to the node in the next master to slave polling slot. If an *lmp_scat_rep* PDU is returned within the timeout period, the slave's *LT_ADDR* and corresponding master's *BD_ADDR* are added to the *Bridge_Table*.

The BT-Bridge procedure is periodically executed whenever the piconet is either at full capacity or when the probabilistic outcome at the master falls below the threshold and consequently does not enter the BT-Master inquiry procedure. This ensures that the best possible scatternet connectivity is maintained, since an unexpectedly disconnected bridge node would put the piconet below capacity and the master would again, with high probability, enter the BT-Master inquiry procedure.

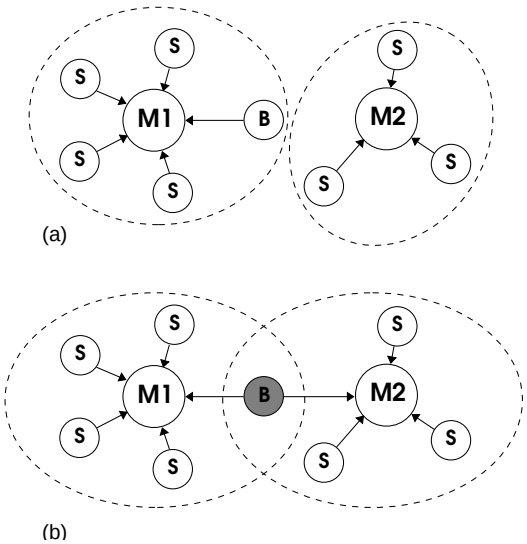


Figure 4.4: Example of a bridge node connection to form a scatternet

An example of a bridge connection is illustrated in Figure 4.4. Node *M1* is the master of piconet 1 and node *M2* is the master of piconet 2. Suppose initially that the node B is a pure slave of piconet 1, as illustrated in Figure 4.4(a). For illustration purposes we assume that *M2* enters BT-MASTER again to connect more slaves. Meanwhile, let us assume that *M1* enters the *BT_Bridge* procedure and selects node B as its bridge designate. *M1* thereafter sends a link manager command to B in the next polling slot. Upon receipt of the *lmp_scat_inq_scan* PDU, B enters the *INQUIRY SCAN* state. If the inquiry from *M2* reaches B, it immediately responds to *M2*. If a connection is successfully formed, the bridge node B returns an *lmp_scat_rep* PDU, containing the BD_ADDR of the *other* piconet, to each of the two connected masters. Both masters then update their *Bridge_Table* accordingly. The completed bridge inter-connection is illustrated in Fig. 4.4(b).

4.3.5 Fault Tolerance and Scatternet Maintenance

Through execution of BT-INIT periodically, our algorithm provides fault-tolerance in a distributed fashion. The scatternet formed is self-healing; nodes that are unexpectedly disconnected are automatically re-incorporated into the scatternet. Slaves that are disconnected lose their logical transport address, *LT_ADDR*, and can subsequently re-enter BT-INIT and either form or join a piconet. Similarly, masters that lose all their children will again have a low threshold value and with high probability join an existing piconet instead of forming a new one. Due to the simplicity of the approach, inter-piconet fault-tolerance is also provided by the same mechanism. Bridge nodes that lose one of their links can again become bridge designates and form bridge connections.

As an optimization to the basic fault-tolerance properties of the algorithm, we propose a *rapid link re-establishment* mechanism as a way to explicitly reconnect broken links. More specifically, when a link is lost due to wireless interference or temporary mobility, it is desirable to re-establish that particular link and not just incorporate the disconnected node elsewhere. Therefore, we propose a mechanism that attempts to explicitly re-establish previously

```
BT-INIT-REPAIR()
 1  if !EMPTY(Repair_Table)
 2    then  /* Existing links detected */
 3          clk ← CURRENTLOCALCLOCK()
 4          for  each Repair_Table as entry
 5          do if |clk − entry.ts| > repairTO
 6                then PURGE(entry)
 7                else  if |clk − entry.ts| > POLL_PERIOD
 8                         then if !entry.BD_ADDR
 9                                 then BT-PAGESCAN()
10                                 else  BT-PAGE(entry.BD_ADDR)
11  if !LT_ADDR
12    then p_thresInit ← η0
13          p_thres ← f_p(p_thresInit, s_count)
14          b_thres ← η1
15          p ← rand(0, 1)
16          b ← rand(0, 1)
17          if p < p_thres
18            then BT_MASTER()
19            else  if s_count = 0
20                    then BT_SLAVE()
21            else  if b < b_thres
22                    then BT_BRIDGE()
23  return
```

Figure 4.5: Pseudo code for BTDSP modified initialization procedure BT-INIT-REPAIR

existing links. This requires that each node keeps a soft state, *Repair_Table*, of recent links. Each table entry consists of a *BD_ADDR*, *LT_ADDR*, and a timestamp based on the local clock. We set the $POLL\_PERIOD$ parameter equal to one polling period, or $625\mu s * T_{poll}$ slots, where T_{poll} is the current polling interval set by the link manager. Consequently, we also add a timeout value, *repairTO*, and set it conservatively to $2 * POLL\_PERIOD$. The *repairTO* parameter is used to purge expired entries from the *Repair_Table*, while the $POLL\_PERIOD$ parameter is used to detect missing links.

In each master device we populate the table with the *BD_ADDR*, *LT_ADDR* of the link and current local clock value for each connected slave. Slaves and bridge nodes populate only the *BD_ADDR* and current clock value for one or two entries respectively, since only one master link exists for pure slaves and two for bridge nodes. The entries are updated at every

packet exchange with a new timestamp. In this manner entries older than the *repairTO* period can easily be purged. The reason for including *LT_ADDR* is to determine whether a device was a master or a slave along the broken link.

In order to provide rapid link re-establishment we must also modify the BT-INIT procedure to purge expired entries from the *Repair_Table* and directly enter the *PAGE* or *PAGE_SCAN* states, depending on whether the device is a master or a slave respectively. This is determined by checking if an *LT_ADDR* value was set for the specific entry. The new and modified procedure, BT-INIT-REPAIR, is shown in Figure 4.5.

In the modified initialization procedure, a maintenance operation has been included to purge entries older than *repairTO* from the *Repair_Table*. Entries that are between *POLL_PERIOD* and *repairTO* are detected as broken links. Slaves that encounter a broken link directly enter the *BT_PageScan* procedure, which is merely the *PAGE_SCAN* sub state. On the other hand, masters enter the *BT_Page* procedure and generate a device specific page hopping sequence before entering the *PAGE* sub state. In this manner broken links can explicitly be re-established.

4.4 Performance Evaluation

To evaluate the performance of our algorithm we conduct a performance evaluation study using the ns-2[1] network simulator and a customized version of the University of Cincinnati - BlueTooth (UCBT) extension module[69]. The UCBT module is based on [32] and [49], but includes a fully operational Bluetooth stack. We extend the UCBT module by including functionality for our BTDSP algorithm. In the first part we determine the best possible settings for the threshold parameters used in the BT-INIT procedure. In the second part we simulate the BTDSP algorithm using the settings chosen in part one and compare it with two existing approaches.

4.4.1 Parameter Optimization

Before evaluating network parameters and topology structure for BTDSP and reference algorithms, we study the threshold parameters p_{thres} and b_{thres} used in the BT-INIT procedure and heuristically determine the best possible settings.

The factors $p_{thresInit}$ and s_{count}, the piconet slave count, are the arguments used in function f_p to compute the value p_{thres}, as described in Equation 4.1.

$$p_{thres} = f_p(p_{thresInit}, s_{count}) \tag{4.1}$$

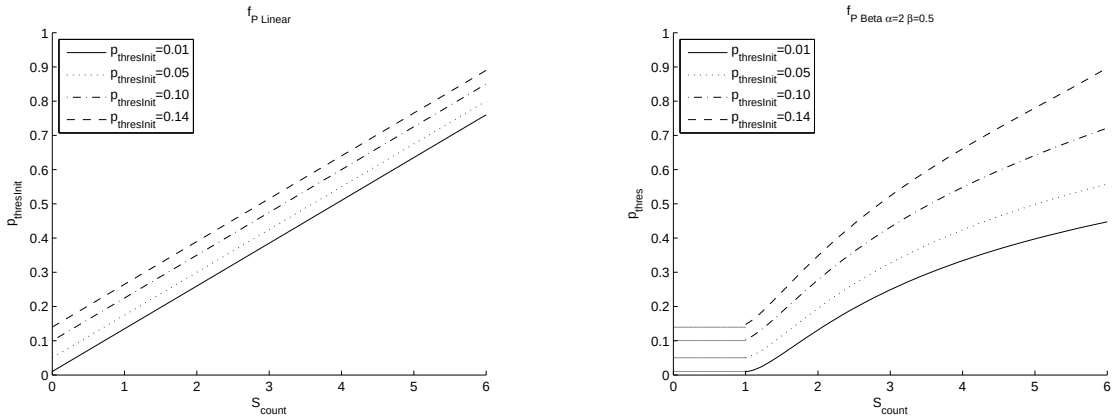
Evaluation of p_{thres} candidate functions

We independently evaluate three different functions for computing p_{thres} as follows:

- $f_{p_{Linear}}(p_{thresInit}, s_{count})$: Original linear function [59]
- $f_{p_{Beta_{\alpha=2,\beta=0.5}}}(p_{thresInit}, s_{count})$: Randomized incomplete beta function with $\alpha = 2$ and $\beta = 0.5$
- $f_{p_{Beta_{\alpha=50,\beta=4.5}}}(p_{thresInit}, s_{count})$: Randomized incomplete beta function with $\alpha = 50$ and $\beta = 4.5$

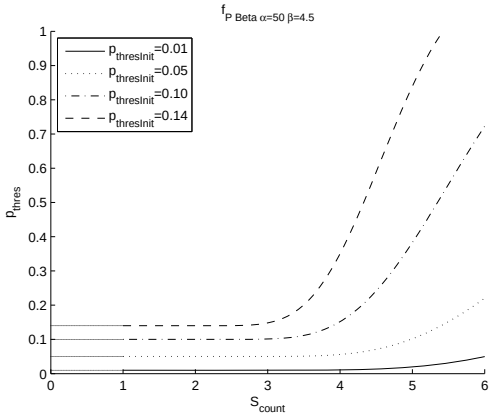
In addition to the original linear function $f_{p_{Linear}}$ [59], we also use two other functions, $f_{p_{Beta_{\alpha=2,\beta=0.5}}}$ and $f_{p_{Beta_{\alpha=50,\beta=4.5}}}$, based on the regularized incomplete beta function [58]. The reason for using the regularized incomplete beta function is that the parameters α and β can be altered to easily adjust the behavior of the function. Therefore, we can produce behaviors that allow us to evaluate how the growth of f_p affects the distribution of masters and slaves and the configuration of the resulting topologies. Other functions with similar behavior could have also been used in place of the regularized incomplete beta functions.

The three candidate functions for f_p are depicted in Figure 4.6. For clarity we do not display every level for $p_{thresInit}$ in Figure 4.6. The range of levels used for factor $p_{thresInit}$ are chosen to maintain the range of $f_p = (0, 1)$ for the regularized incomplete beta function, but



(a) $f_{pLinear}$

(b) $f_{pBeta_{\alpha=2,\beta=0.5}}$



(c) $f_{pBeta_{\alpha=50,\beta=4.5}}$

Figure 4.6: p_{thres}=f_p candidate functions

also to avoid excessive initial master role assignments. The value η_0 denotes the initial value assigned to the threshold $p_{thresInit}$ on line 3 of BT-INIT, where $\eta_0 = (0, 0.15)$. Also, the value η_1 denotes the initial value assigned to the threshold value b_{thres}, line 4 of BT-INIT, and is set to a constant value for each factor treatment, where $\eta_1 = (0, 1]$.

We first evaluate the original linear function, called $f_{p_{Linear}}$, for f_p. To validate the most appropriate setting for $p_{thresInit}$ we evaluate several levels where $\eta_0 = [0.01, 0.14]$[1].

This function is illustrated in Equation 4.2.

[1]In [59] the $p_{thresInit}$ was set to 0.10.

$$p_{thres} = f_{p_{Linear}}(p_{thresInit}, s_{count}) = p_{thresInit} + \begin{cases} \text{-}p_{thresInit} & \text{if } s_{count} = 7 \\ s_{count} \ / \ 8 & \text{if } 0 < s_{count} < 7 \\ 0 & \text{else} \end{cases} \quad (4.2)$$

We simulate the original algorithm from [59] with different levels for η_0, where $\eta_0 = [0.01, 0.14]$ and $s_{count} = [0, 7]$, to determine the value of $p_{thresInit}$ that results in a connected scatternet topology using only natural generation of piconets and inter-piconet bridge connections, as opposed to artifical piconet merging or node migration as seen in other algorithms.

The distribution of the number of masters versus slave roles assigned during discovery in the BT-INIT procedure is correlated to the function value of p_{thres}. To assess the impact on the scatternet topology due to the behavior of the f_p function used to calculate p_{thres}, we also evaluate f_p using the regularized incomplete beta function [58] as an alternative to $f_{p_{Linear}}$.

The general form of the regularized incomplete beta function is illustrated in Equation 4.3 [58].

$$I_x(\alpha, \beta) = \frac{B_x(\alpha, \beta)}{B(\alpha, \beta)} = \frac{1}{B(\alpha, \beta)} \int_0^x t^{\alpha-1}(1-t)^{\beta-1} dt \quad (4.3)$$

Equation 4.4 [58] defines f_p in terms of the regularized incomplete beta function as $f_{p_{Beta}}$. By varying the parameters α and β in $f_{p_{Beta}}$ the behavior of $f_{p_{Beta}}$ can be adjusted.

$$p_{thres} = f_{p_{Beta}}(p_{thresInit}, s_{count}) = \frac{1}{B(\alpha, \beta)} \int_0^{p_{thresInit}+(1-\frac{1}{s_{count}})} t^{\alpha-1}(1-t)^{\beta-1} dt \quad (4.4)$$

As shown in Figure 4.6 we choose two different pairs of α and β values for $f_{p_{Beta}}$, $\alpha = 2, \beta = 0.5$ and $\alpha = 50, \beta = 4.5$, since these produce behaviors that control the growth of f_p in a manner that is suitable for comparison with $f_{p_{Linear}}$.

Simulation of p_{thres} candidate functions

We simulate our Bluetooth Distributed Scatternet Formation Protocol (BTDSP) algorithm with the BT-INIT values for $p_{thresInit}$ in the domain $\eta_0 = [0.01, 0.14]$. For the initial simula-

tion round the value of b_{thres} is set constant to 1.

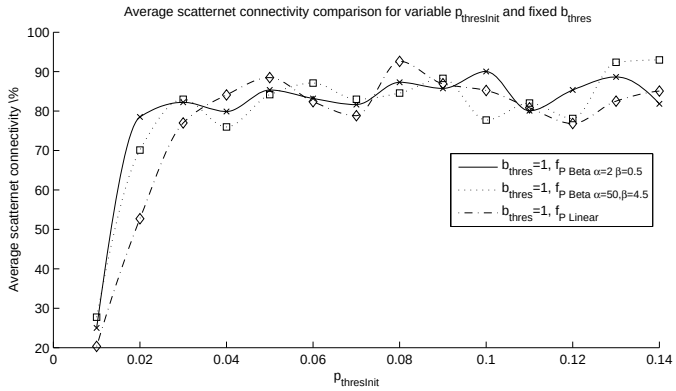
The simulation replications are performed in a limited area of 10 by 10 meters[2] with 32 nodes, randomly distributed within the region, for a duration of 300 seconds. It should also be noted that extra care has been taken to ensure that the Random Number Generator (RNG) used by the ns-2 network simulator is properly seeded, as described in [47], so that a non-deterministic behavior of the simulation is achieved for each replication.

Figure 4.7 summarizes the results of simulating each of the three candidate functions to determine the most suitable value for $p_{thresInit}$. We measure the scatternet connectivity by evaluating the percentage of nodes that are reachable by every other node. In other words, a fully connected scatternet is made up of a single connected component and contains no partitions. Due to the fact that node placements and resulting topologies are randomly generated in a non-deterministic fashion for each replication, the results are presented as mean and median values. From Figure 4.7 we conclude that each of the three candidate functions are characteristically different and produce higher degrees of scatternet connectivity when the level of the $p_{thresInit}$ parameter is set individually for each candidate function.

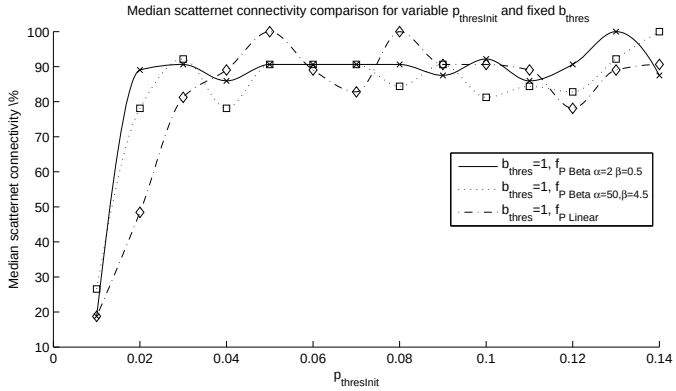
For $f_{p_{Linear}}$ we identify $p_{thresInit} = 0.5$ and $p_{thresInit} = 0.08$ as levels that produce significantly better scatternet connectivity. We choose a single level for each of the three functions to conduct the next round of simulations. For $f_{p_{Linear}}$ $p_{thresInit} = 0.08$ is selected due to a higher mean value. For $f_{p_{Beta_{\alpha=2,\beta=0.5}}}$ we note that $p_{thresInit} = 0.13$ distinctly results in better scatternet connectivity than any other levels. For $f_{p_{Beta_{\alpha=50,\beta=4.5}}}$ we emphasize that there is a trend of increased scatternet connectivity as the $p_{thresInit}$ level increases, and therefore we select $p_{thresInit} = 0.14$.

After determining a $p_{thresInit}$ value for each candidate function, we perform another round of simulation replications for which we fix the level for $p_{thresInit}$ at $\{0.08, 0.13, 0.14\}$, for $f_{p_{Linear}}$, $f_{p_{Beta_{\alpha=2,\beta=0.5}}}$, and $f_{p_{Beta_{\alpha=50,\beta=4.5}}}$ respectively, and study the effect of changing the level of the constant b_{thres} value as η_1, where $\eta_1 = (0, 1]$. The purpose of this round is

[2]We experiment with multi-hop topology formation in larger areas during the comparative simulation study in Section 4.4.2.



(a) Mean

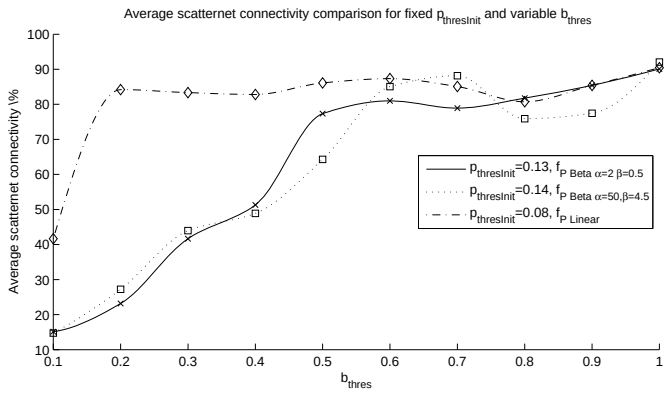


(b) Median

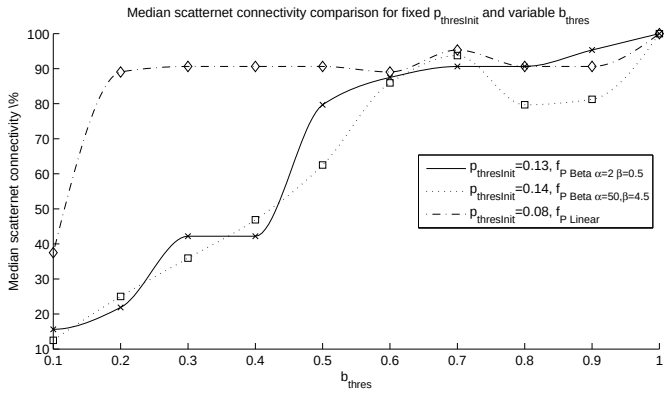
Figure 4.7: Scatternet Connectivity for p_{thres} candidate functions with variable $p_{thresInit}$

to determine if we can maintain acceptable scatternet connectivity, for each function and corresponding optimal $p_{thresInit}$ settings, with a lower level for the b_{thres} threshold value. The reasoning behind this experiment is to evaluate whether bridge designate assignments can also be done probabilistically, while maintaining scatternet connectivity without using artificial migration. The results from the simulation replications are summarized in Figure 4.8.

It should be noted from Figure 4.8 that $f_{p_{Linear}}$ produces relatively high scatternet connectivity, even at lower levels of b_{thres}, while the two regularized incomplete beta functions, $f_{p_{Beta_{\alpha=2,\beta=0.5}}}$, and $f_{p_{Beta_{\alpha=50,\beta=4.5}}}$, produce scatternets with extremely low connectivity for levels of b_{thres} in the bottom half of the range. We observe that b_{thres} is useful to prevent



(a) Mean



(b) Median

Figure 4.8: Scatternet Connectivity for p_{thres} candidate functions with variable b_{thres}

excessive bridge formation and can be useable as a performance tuning parameter. Hence, we conclude that $f_{p_{Linear}}$ with $p_{thresInit} = 0.08$ should be utilized over $f_{p_{Beta_{\alpha=2,\beta=0.5}}}$ and $f_{p_{Beta_{\alpha=50,\beta=4.5}}}$. Further, from Figure 4.8 we also note that specifically $b_{thres} = (0.2, 1]$ does produce scatternets with high connectivity. However, unless excessive bridge formation and Inter Piconet Scheduling (IPS) delays are predominant, $b_{thres} = 1$ should be used for the highest connectivity.

4.4.2 Comparative Simulation Study

To validate the performance of Bluetooth Distributed Scatternet Formation Protocol (BTDSP), we conduct a comparative simulation study of BTDSP, the Tree Scatternet Formation Protocol (TSF) by Tan *et al.* [27], and the algorithm by Law *et al.* [16] using the ns-2[1] sim-

ulator together with an extended version of the University of Cincinnati - BlueTooth (UCBT) extension module [69] and functionality from BlueHoc [32] and Blueware [49].

Single-hop Connectivity

To illustrate how these algorithms produce a scatternet in a *single-hop* environment, we simulate each algorithm with 32 disjoint devices in a limited operational area of 10 by 10 meters. To avoid the *border effect*, described in [6], we only use an effective simulation area of 7 by 7 meters to ensure single-hop connectivity. Each algorithm is simulated independently and each simulation run is replicated 500 times. The focus of this experiment is to evaluate how well each algorithm forms a scatternet topology from a set of disjoint, discoverable devices and compare the resulting scatternet connectivity. The connectivity of a scatternet is defined as the percentage of nodes that are reachable from every other node. The results are shown in Figure 4.9.

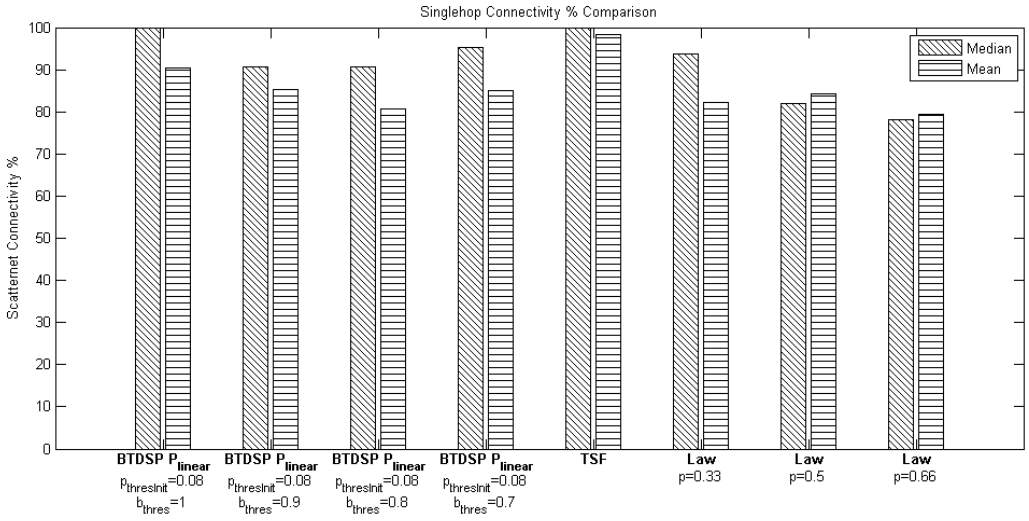


Figure 4.9: Comparison of Singlehop Scatternet Connectivity

We simulate BTDSP using the $f_{p_{Linear}}$ function with $p_{thresInit} = 0.08$, as determined in the previous section. In addition to $b_{thres} = 1$ we also include $b_{thres} = [0.7, 0.9]$ for comparison with the other algorithms. For completeness we also vary the threshold parameter p, where $p \in 0.33, 0.5, 0.66$, for the algorithm by Law *et al.*, as described in [16]. From Figure 4.9 it can

be seen that all instances of BTDSP provide as good or better mean scatternet connectivity as the algorithm by Law *et al.*. The main difference between these two algorithms is that BTDSP does not utilize centralized artificial migration to re-organize the topology. Although this can be beneficial in certain smaller and relatively static environments, as discussed in Chapter 3, this adds overhead and the algorithm is not capable of handling node failures. The algorithm by Law *et al.* performed the best with $p = 0.33$. Both BTDSP and TSF [27] are capable of providing complete scatternet connectivity, but the mean connectivity of the Tree Scatternet Formation Protocol (TSF) algorithm is better. We believe this is due to the probabilistic nature of BTDSP where a single role is selected, as opposed to alternating between master and slave states as done in TSF. However, the approach taken by TSF requires topology re-organization, which leads to higher formation delay and is described in more detail later.

Multi-hop Connectivity

To further illustrate the fundamental differences between the algorithms we also simulate all three in a multi-hop environment using a larger operational area of 20 by 20 meters. Similar to above, the effective operational area is somewhat smaller, 15 by 15 meters, to avoid the *border effect*. The results of the multi-hop comparison are displayed in Figure 4.10.

From Figure 4.10 it can been seen that BTDSP maintains high scatternet connectivity, while neither the algorithm by Law *et al.* nor TSF are capable of consistently producing a connected scatternet as the operational area increases beyond the single-hop range. This is expected behavior for the algorithm by Law *et al.* due to its centralized structure and reliance on leader election. Although TSF is capable of operating in a multi-hop environment it necessitates that the root nodes are within single-hop range of each other to be able to merge the subtrees. As indicated by the results from Figure 4.10 this is not always the case as TSF does not consistently produce a connected scatternet.

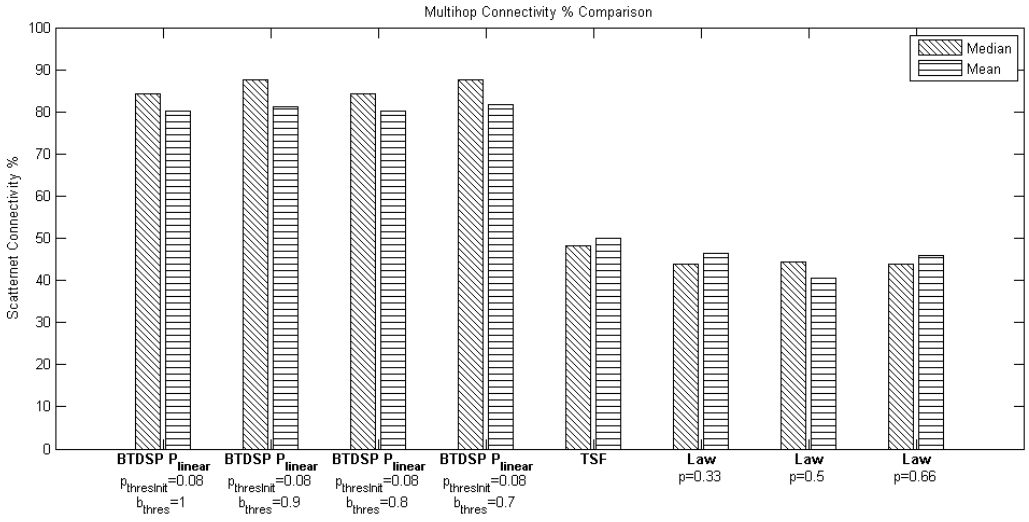


Figure 4.10: Comparison of Multihop Scatternet Connectivity

Formation Delay

We also compare the formation delay for BTDSP, TSF, and the algorithm by Law *et al.*. To provide a fair comparison we simulate each algorithm in a single-hop environment, due to the multi-hop connectivity discrepancies displayed in Figure 4.10. We simulate each algorithm with 4, 8, 16, 32, and 64 nodes; each replicated 100 times. The scatternet formation delay is defined as the time from when the formation is initiated until every node is incorporated into the scatternet and all phases are completed, including leader election, merging, and migration. The results of the comparison are displayed in Figure 4.11.

From Figure 4.11 it can be seen that BTDSP has consistently lower formation delay than both TSF and the algorithm by Law *et al.*. For both TSF and the algorithm by Law *et al.* this delay increases as the number of nodes increase, while the formation delay for BTDSP remains relatively constant. The reason for this is that as the number of nodes increases both TSF and the algorithm by Law *et al.* suffer additional overhead from coordination between sub-tree root nodes for tree merging and piconet merge and migrate procedures respectively, while BTDSP is completely distributed and does not employ any artifical migration.

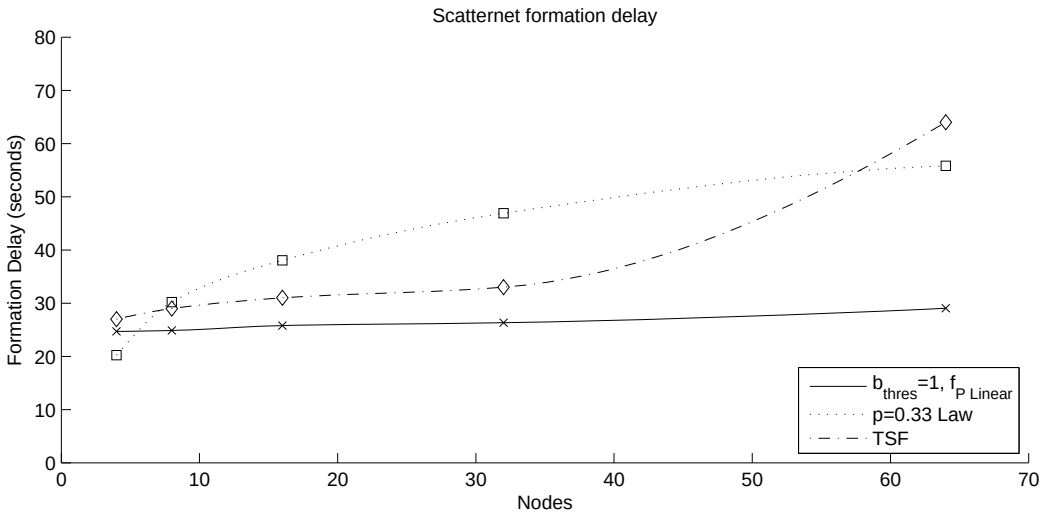


Figure 4.11: Comparison of Formation Delay

4.5 Summary

In this chapter we proposed a new algorithm for scatternet formation. Based on thorough analysis of previous approaches, our algorithm remedied some of their drawbacks. These drawbacks included the use of inefficient MS bridges, phase-divided algorithms, and inefficient grouping of nodes in piconets. Our BTDSP algorithm formed a scatternet in a single phase and accommodates late arriving nodes. The algorithm displayed two important self-healing properties: it allowed disconnected nodes to be quickly re-incorporated and healed scatternet partitions. The algorithm minimized the bridge switching delay by allowing bridge nodes to participate in only two piconets. Only pure slaves were used as bridge designates, which resulted in a flat 2-SSM scatternet topology that was free of bottleneck links associated with tree-based algorithms. It was completely distributed and worked in a multi-hop environment, meaning that it did not require that all devices were within transmission range of each other. We used heuristics to determine optimal settings for BTDSP and compared the topology formation to other existing solutions. BTDSP showed better or equivalent performance for single-hop scenarios but demonstrated a distinct advantage in multi-hop

scenarios.

Chapter 5

Hybrid Bluetooth Scatternet Routing

5.1 Introduction

Due to lack of a standard and differences between proposed scatternet formation methods, communication between nodes in a scatternet requires a routing protocol that is compatible with the underlying scatternet formation approach. In addition, it is common that devices are not aware of the identities of their peers and hence are not able to perform traditional, destination-address based route discovery. Devices often need just a route to *some* peer device that offers the requested service rather than a *specific* peer. Furthermore, as part of a scatternet the star shaped master-centric piconets with inter-connecting bridge nodes between them provide an inherent neighborhood substructure, which we believe should be taken advantage of for more efficient routing.

As a basis for our approach we have identified the following important criteria for an efficient scatternet routing protocol:

- Minimizing topology induced bottlenecks and switching overhead.
- Route resilience by avoiding dependency on specific bridge nodes.
- Efficient topology utilization between neighboring piconet clusters and the direct bridge links between them.
- *Hybrid route discovery*: both destination- and service-based.

- Reducing reactive overhead due to inefficient route request flooding.
- Caching of routes during route discovery and periodic route invalidation to prevent cache poisoning.
- Logical placement of scatternet routing functionality in the Bluetooth® protocol stack.

A novel feature of our scatternet routing protocol is *hybrid route discovery*: the ability to either discover a scatternet peer directly based on its destination address or based on a service that it offers. The service-based route discovery can be viewed as an extension of the Service Discovery Protocol [11], which by itself is used only within a piconet.

We base our approach on a 2-SSM flat scatternet topology to *minimize bottlenecks and switching overhead*. We use only Slave/Slave (SS) bridge nodes of degree two, meaning that they participate in exactly two piconets. This is discussed in more detail in Section 3.2.2.

Route resilience is important for fault tolerance and to prevent unnecessary route discovery operations due to the failure or unavailability of a specific bridge node. We observe that the star-shaped and master-centric piconet structure necessitates that each scatternet route must pass through the master of every intermediate piconet along a scatternet route. We also believe that an efficient scatternet routing protocol should be bridge link agnostic, meaning that any particular bridge node could be utilized to reach a next-hop piconet master. We therefore propose using *modified source routes* that only contains intermediate piconet masters, but not the specific Slave/Slave (SS) bridge nodes between them. This allows *any* available bridge node to be utilized when establishing a path between adjacent piconet masters. It should also be noted that this approach allows for potentially more than one bridge node between two piconets, for multiple path routing.

We also observe that *efficient topology utilization* should take advantage of the bridge node overlap between adjacent piconets. While a single piconet is limited to one master and up to seven active slaves, adjacent piconets provide a natural extension, or an *Extended Scatternet Neighborhood (ESN)*, between piconets that share common bridge nodes. An example ESN

is illustrated in Figure 5.1. Using this zone routing approach, we take advantage of the inherent topology to reduce the need for route discovery when devices communicate with nodes close to each other, or within the ESN. A proactive table-driven approach is used within the ESN and a reactive modified source routing approach is employed elsewhere. We also avoid complex zone maintenance operations by keeping the ESN zone radius static.

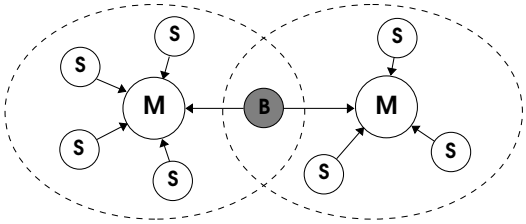


Figure 5.1: Extended Scatternet Neighborhood (ESN)

Route discovery must be performed for destinations or services that are not available within the ESN. However, to reduce routing load we do not utilize flooding to propagate route requests. Instead we employ a two-tiered *probabilistic gossiping* strategy to determine whether a route request should be forwarded to a neighbor. Instead of always forwarding a route request, route requests are forwarded with some probability p, where p is a threshold value. A higher probabilistic threshold value is used for sparser topologies to ensure sufficient scatternet coverage while a lower threshold value prevents excessive route request propagation for denser piconets with a higher degree of scatternet connectivity.

We also utilize route caching at piconet masters and periodically invalidate stale routes to prevent cache poisoning. Intermediate piconet masters that have lost connectivity to the next hop master can also attempt to repair the route locally if a different path to the destination or another, downstream, intermediate master along the route is available.

Perhaps the most important aspect of a functional scatternet routing protocol is logical placement in the protocol stack. None of the scatternet routing approaches presented in [7, 63, 15] have taken this into consideration and they merely discussed routing as a function of the Link Manager (LM). We believe that a scatternet routing protocol should be placed on top of the Logical Link Control and Adaptation Protocol (L2CAP) layer and thereby

allow application layer protocols to more easily incorporate scatternet routing.

By using our modified source routing approach we also eliminate routing loops, avoid having to use sequence numbers to determine route freshness at intermediate nodes, and reduce routing load. The tradeoff is larger overhead due to the inclusion of the modified source route in the routing header.

This chapter is organized as follows. In Section 5.2 we describe previous work related to scatternet routing. Section 5.3 establishes some necessary preliminaries and concepts before we present our routing algorithm in Section 5.4. The basic idea behind our algorithm is discussed in Section 5.4.1. Thereafter, we present the complete algorithm in Section 5.4.2. In Section 5.5 we evaluate our approach and present simulation results. Finally, we conclude the chapter in Section 5.6.

5.2 Related Work

In this section we describe previous work on routing in Bluetooth scatternets.

Bhagwat *et al.* [7] present the Routing Vector Method (RVM) for scatternet routing. They base their protocol on a hypothetical scatternet topology that uses only single-role slave bridges. A route vector is used to source route packets throughout the scatternet. It is made up of a series of alternating local piconet identifiers, and piconet LT_ADDR's. Packets are routed through a bridge node to the next locally identified piconet and from there to the piconet member with a corresponding LT_ADDR, and so on. Routes are found by flooding route discovery messages, to which replies are unicasted back along the reverse source route. The RVM approach does not consider many issues such as mobility and route resilience. To the best of our knowledge, this is one of the first attempts to address scatternet-specific routing issues and it does provide valuable insight.

Prabhu *et al.* [62] extend the RVM approach by also considering energy efficiency when determining routes. Their approach also uses a route vector, similar to RVM, and finds routes by way of route discovery. However, they incorporate two energy saving techniques:

power control and master-slave switch. The former allows for variable transmitting power, while the latter attempts to allieviate energy drainage of the master by sharing the master role among piconet members. They show that by considering energy efficiency, scatternet lifetime can be significantly extended.

To illustrate the complexity of the scatternet routing problem due to the topology constraints we also discuss two scatternet formation solutions that incorporate routing mechanisms. Sun *et al.* [45] present a tree-based scatternet topology, while Lin *et al.* [68] take a different approach and their solution forms a ring topology. In both these approaches routing is an intrinsic feature of the scatternet formation protocol. The ring-based topology allows for a simple routing mechanism using a token-based approach. However, it requires mechanisms to prevent orphan packets, token regeneration, and ring maintenance. In the tree-based topology, routing is simplified by making tree nodes aware of the identities of other nodes in their respective subtrees. Unanswered queries are directed toward the root. As with the ring-topology, the tree-based scatternets requires a strict topology and continuous maintenance in order to provide routes. Although routing is simple and part of the scatternet formation protocol, the approach used for formation is inefficient.

Liu *et al.* [73] builds scatternets on-demand along requested routes instead of first forming a single monolithic scatternet. Their approach incorporates the route discovery mechanism in the Bluetooth® inquiry process prior to link establishment. Thus, route discovery packets can be flooded without a pre-existing scatternet. A scatternet is thereafter formed in reverse along the route on which the route request has propagated, but merely for the duration of the traffic flow. This approach is suitable for sporadic communication, but requires complex scatternet scheduling when multiple scatternet-routes intersect.

Kapoor *et al.* [63] take advantage of the clustered structure of piconets and apply the Zone Routing Protocol (ZRP) [76] to Bluetooth® scatternets. The ZRP approach uses a proactive, table-driven part to reach destinations within the neighborhood zone and a reactive part, which floods route requests and is based on AODV route discovery, for all

other destinations. The authors note that the piconet structure automatically provides a one-hop neighborhood zone, but also allow the use of a varying routing zone radius. They also place their routing mechanism right above the Link Manager (LM) in the Bluetooth® stack. The protocol does not take mobility or topology formation into account, but present convincing arguments for hybrid solutions in Bluetooth® scatternets. This approach is also discussed in [71].

Huang *et al.* [15] propose a self-adaptive zone routing approach that adjusts the reactive routing zone using fuzzy logic. This approach adjusts the zone size to control the need for proactive route discovery, and reduces some of the control packet traffic due to flooding by preventing discovery packets from being forwarded to piconets that have no forward routes. This approach is well suited for scatternet routing, but does not have the combined hybrid features and robust inter-piconet traversal that our solution provides.

5.3 Routing Preliminaries

In this section we describe some preliminaries and concepts that we use in our algorithm. First, we describe some basic scatternet concepts. Thereafter, we define and describe the Extended Scatternet Neighborhood (ESN) and finally the idea behind probabilistic gossiping. We base our routing approach on a 2-SSM flat scatternet topology, where bridge nodes are allowed to participate only as slaves in exactly two piconets and inefficient Master/Slave (MS) bridges are never used [59]. This is also described further in Chapter 3. From here on we assume that such an underlying topology exists. We also assume that scatternet scheduling functionality exists, similar to [38, 56].

5.3.1 Scatternets

For routing in Bluetooth® scatternets it is assumed that a scatternet has been formed. The Bluetooth® specification[11] merely defines the concept of a scatternet but does not specify a protocol for forming scatternets. In contrast to ad hoc networks based on Direct

Sequence Spread Spectrum (DSSS), in which neighbors overhear any transmission within their range, Bluetooth® devices need to first discover each other, explicitly establish links, form piconets, and for the purpose of larger WPANs, subsequently inter-connect these into a scatternet before engaging in communication.

Numerous scatternet formation approaches have been proposed in literature [67, 16, 59, 24, 27, 74, 64]. In one way or other they all describe the formation of a scatternet topology; however, the approaches and resulting topologies vary significantly. Varying criteria are used to determine the number of bridge nodes, the bridge degrees, and other design decisions. The Master/Slave (MS) cluster-head structure imposed on piconets further differentiates the proposed solutions, as some let only slaves function as bridge nodes while others allow masters to have dual roles and participate as slaves in other piconets[1].

More details regarding scatternet formation can be found in Chapters 3 and 4.

5.3.2 Extended Scatternet Neighborhood (ESN)

In this section we first clarify how a network *hop* is defined in the context of piconets and scatternets, and then define the Extended Scatternet Neighborhood (ESN):

Definition 1. Piconet Hop

A physical link between any two Bluetooth® devices

Definition 2. 1-Scatternet Hop

A virtual hop (path) between any two piconet masters, which goes through a bridge node

Definition 3. Extended Scatternet Neighborhood (ESN) of a piconet

The 1-scatternet hop region formed by the union of all neighboring piconets that are connected through a common bridge node

[1]Nodes that participate as a master in one piconet and as a slave in others are referred to as Master/Slave (MS) bridges, while nodes with only slave roles are referred to as Slave/Slave (SS) bridges. The *bridge degree* further describes the number of piconets in which the bridge node participates.

An Extended Scatternet Neighborhood (ESN) of a piconet is the union of all neighboring piconets that are directly connected through a common bridge node, as illustrated in Figure 5.1. As we shall see later the ESN makes up our proactive routing zone. Within the ESN a local routing table at each piconet master is periodically updated with routes from adjacent masters. As is illustrated in Figure 5.2, this allows two slaves in adjacent piconets, four piconet-hops apart, to establish a route without using the reactive part of the routing protocol. For example, a route can be established between slaves S_A and S_B, along the route $S_A \leftrightarrow M_1 \leftrightarrow B \leftrightarrow M_2 \leftrightarrow S_B$, using only proactive ESN routing information.

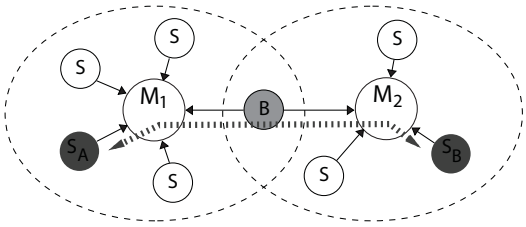


Figure 5.2: A 4-piconet hop route within the ESN

5.3.3 Probabilistic Gossiping

For the reactive part of our scatternet routing approach, instead of flooding route requests throughout the scatternet we employ an alternative strategy called probabilistic gossiping [40, 41]. As we describe in Section 5.4, our scatternet routing approach uses a hybrid zone routing strategy with a proactive component inside the ESN and a reactive component outside the ESN.

The idea behind gossiping is to limit propagation of routing control messages throughout the network. For some threshold value p, where $p < 1$, a node forwards the route request with probability p and discards it with probability *(1 - p)*. However, a potential problem with this approach is that in a sparse network, where a piconet master has few neighbors (low out-degree), the master might not propagate the route request to *any* of its neighbors and therefore the request may never reach its intended destination. To remedy this situation we employ a 2-tier gossiping scheme similar to [40]. However, unlike [40], we make a gossiping

decision for each forwarding node rather than either forwarding the route request to all forwarding nodes or none at all. In sparse topologies a threshold p_2, where $p_1 < p_2$, is used to increase the probability of route propagation. On the contrary, for denser topologies the value p_1 is used to prevent unnecessary propagation. Note that sparseness is a local decision made by each piconet master by determining whether its out-degree, or the number of connected bridges to other piconets, is less than a threshold value $degree_{thres}$.

In the next section we first describe the basic idea behind our scatternet routing approach and then present the algorithm in detail.

5.4 A Hybrid Bluetooth Scatternet Routing Algorithm

5.4.1 Basic Idea

The word *hybrid* in our Hybrid Bluetooth Scatternet Routing (HBSR) algorithm has dual meaning. It signifies both destination versus service-based discovery as well as zone routing with a proactive approach within the Extended Scatternet Neighborhood (ESN) zone and reactive gossiping-based route discovery outside the zone. In this section we describe the basic idea behind HBSR and summarize the main points of the algorithm.

In contrast to previous approaches, we do not incorporate scatternet routing as a function of the Link Manager (LM) but rather place routing functionality on top of the L2CAP layer in the Bluetooth® stack. HBSR is thereby capable of providing limited compatibility with existing application layer protocols, e.g. by intercepting requests from the Service Discovery Protocol (SDP) or inquiries directed to a former piconet member that has since moved and is now located elsewhere within the scatternet. Although scatternet formation procedures must be available in the LM, HBSR does not require the addition of any functionality to the lower layers of the Bluetooth stack, unless cross-layer scheduling and formation optimizations are needed. To take full advantage of scatternet communication, the application layer protocols should, however, be HBSR aware. By placing the HBSR routing header, which includes the modified source route, inside the L2CAP header, the overhead is significantly lower than

for a similar approach at the link layer where the packet sizes are much smaller. However, L2CAP channels must be established on each master to bridge link.

We observe that Bluetooth® scatternets are mainly designed for Wireless Personal Area Networks (WPANs) in which devices are not always aware of the identities of one another. Therefore, the traditional methodology of destination-based route discovery is often not as effective in scatternets as in other networks, particularly one-to-all broadcast networks where devices can use promiscuous listening to overhear communication between peers. On the other hand, devices often require services that could be provided by *any* scatternet peer if a route was available. In piconets the Service Discovery Protocol (SDP) is designed to allow devices to discover what services its piconet peers can provide. We extend this functionality to scatternets and incorporate it into our hybrid destination and service-based scatternet routing protocol. In our approach the target of a route discovery can either be an individual destination address or an SDP-style *U*niversal *U*nique *ID*entifier (UUID), corresponding to a service [10].

In addition to providing destination and service-based route discovery, our hybrid strategy also uses zone routing to divide the routing protocol into proactive and reactive regions. We call the proactive region the Extended Scatternet Neighborhood (ESN). From Definition 3 we have: *an ESN is the 1-scatternet hop region formed naturally by the union of all neighboring piconets that are connected through a common bridge node.* The simplest example of an ESN contains two piconets and is illustrated in Figure 5.1. Within an ESN adjacent piconet masters periodically exchange local piconet membership information through their common bridge nodes using the HBSR-ESN procedure, illustrated in Figure 5.3. Using this procedure piconet masters populate their local ESN routing tables with reachability and bridge connectivity information from adjacent piconets. An ESN routing table located at each piconet master contains a list of all devices in adjacent piconets, their *BD_ADDR*'s, as well as which of them have bridge connections to other piconets. In this manner a piconet master has reachability information for every node within its ESN, which is also accessible

to its slaves as the master functions as the gateway router to nodes outside the piconet.

The reactive part of the protocol utilizes a route discovery mechanism to disseminate both destination- and service-based route requests for destinations outside the ESN zone. We employ a probabilistic gossiping strategy, when necessary, to control propagation of route requests throughout the scatternet, similar to [40, 41].

When an intermediate piconet master receives a route request packet it first checks whether it can provide a valid reply to the request. We assume that there are not any non-cooperative or malicious nodes in the scatternet. In terms of when an intermediate master is allowed to reply, the main difference between destination and service-based route requests is that the UUID information is not shared between adjacent piconet masters due to the overhead involved. Thus, an intermediate master does not have UUID service information for ESN members unless a cache entry exists.

If an intermediate master is unable to reply to a route request, it determines whether to forward the route request packet based on gossiping probability p_i, where $\{i : 1 \leq i \leq 2\}$. By using two different gossiping probability values, p_1 and p_2 where $p_1 < p_2$, we assign a higher forwarding probability (p_2) to sparsely connected piconets to provide better propagation and a lower value (p_1) to dense piconets with more forwarding bridge candidates. The threshold parameter $degree_{thres}$ determines if a piconet is sparsely or densely connected. Suitable values for $degree_{thres}$ is discussed further in Section 5.5.3.

Even though our approach is based on source routing, each node along a scatternet route does *not* need to be included in the source route. Due to the specific scatternet topology constraints, we observe that it is not necessary to maintain hop-by-hop routes that include both master and bridge nodes. Instead we utilize the inherent hierarchical structure in Bluetooth® scatternets to differentiate between intra- and inter-piconet connections. In other words, routes between two adjacent piconets are not dependent on which bridge node is used when multiple common bridges are present. Therefore it is only necessary to use a *modified source route* consisting of the piconet masters along a scatternet route. This bridge

link agnostic approach increases route resilience and reliability and also simplifies scheduling since there is no need to wait for a specific bridge node to become available if more than one exists to the next-hop piconet.

Intermediate nodes aggressively store routes in their own route caches as they process route requests and route replies, similar to the Dynamic Source Routing (DSR) protocol [21]. However, for the purposes of storing only recently verified routes, intermediate nodes do not cache the entire modified source route. Instead, for routes in route reply packets, only the portion of the route between the destination piconet and the intermediate piconet master, along with a timestamp, is stored in the path cache. Similarly, for routes in route request packets, intermediate masters only cache the portion of the route between the source piconet master and itself. The local route cache consists of node-centric path cache entries and also includes local timestamps and the *BD_ADDR* of the initiator of the route request/reply. This information is used to invalidate old routes and prevent duplicate route replies.

In order to prevent cache pollution, routes in the cache are periodically invalidated by setting a stale flag. However, these entries are not immediately removed. Instead, the stale flag indicates that the route is old and requires that a gratuitous route request is sent along the existing source route toward the destination the next time it is needed. This pessimistic form of route maintenance minimizes cache pollution. If a route reply is successfully returned to the intermediate node, the route is made fresh again and the buffered packet is sent along the source route. If an invalidated route is not updated and is still marked stale during the next iteration of the periodic route cache maintenance, it is removed from the cache and a route error is returned.

For route replies due to service-based route requests, intermediate nodes also cache a mapping between the service UUID and the destination address for the provider of the service in a table called *ServiceUUID_Table*. This allows the node to reply to future service-based requests if a valid route to a destination node with a mapping for the requested UUID exists in the ESN or in the route cache.

5.4.2 Algorithm

In this section we describe the details of the Hybrid Bluetooth Scatternet Routing (HBSR) algorithm. We first describe the behavior of the proactive portion of the algorithm using periodic exchanges of topology information. Thereafter we describe the reactive portion of the algorithm that performs route discovery.

Proactive ESN Maintenance

As mentioned in Section 5.3, we assume that a scatternet has been formed with protocols such as Bluetooth Distributed Scatternet Formation Protocol (BTDSP) from Chapter 4 and that efficient Inter Piconet Scheduling (IPS) functionality exists, such as the Maximum Distance Rendezvous Point (MDRP) algorithm [56] or the Dichotomized Rendezvous Point (DRP) algorithm [70]. Further, a local parameter t_{esn} is used to determine the frequency of the periodic ESN membership exchanges between neighboring masters.

```
HBSR-ESN(slaves, esn_masters, seq)
 1  if t_esn expired  and !LT_ADDR
 2    then seq ← seq + 1
 3         for  each m_i in esn_masters
 4         do for  each s_k in slaves
 5            do if slaves[s_k].bridge = 1 and
 6                  slaves[s_k].piconet ≠ m_i
 7                  then bit_mask[k] ← 1
 8                  else  bit_mask[k] ← 0
 9               bit_mask[0] ← 0
10            HBSR-ESN-UPDATE(src = M_local, dst = m_i, slaves, bit_mask, seq)
11  return
```

Figure 5.3: HBSR ESN update procedure HBSR-ESN

Every t_{esn} seconds piconet masters periodically exchange connectivity information with adjacent piconet masters using the HBSR-ESN procedure (Figure 5.3). Each update includes the list of slaves within the piconet, as well as connectivity information for the slaves and identification of which of them are bridge nodes to other piconets. The bridge connectivity information is conveyed using an 8-digit bit mask. Each entry in the *esn_master* list corre-

sponds to an adjacent piconet master and the *slaves* list contains the list of all slaves in the piconet.

The HBSR-ESN procedure works as follows. Upon expiration of the t_{esn} timer piconet masters execute the HBSR-ESN procedure (the *LT_ADDR* check prevents non-piconet masters from entering the procedure). For each adjacent piconet master in the *esn_masters* list, a different bit mask is created to indicate bridge connections to piconets other than the destination piconet master. The bit mask is also used to determine bridge out-degree for the gossiping threshold values in the HBSR-DISCOVERY procedure (Figure 5.5). Thereafter, the list of slaves and the bit mask are then scheduled for transmission to the adjacent piconet master across the next available bridge link on line 10 of the HBSR-ESN procedure.

The HBSR-ESN update procedure is executed proactively by all piconet masters. Upon disconnect, restart, or other node failure resulting in loss of ESN information, a piconet master will simply re-advertise its membership information to neighbors with 0 as the sequence number. That requires neighboring masters to flush its current membership information for the failed peer master and prevents neighbors from discarding updated membership information from a peer when a failure results in loss of previous sequence number.

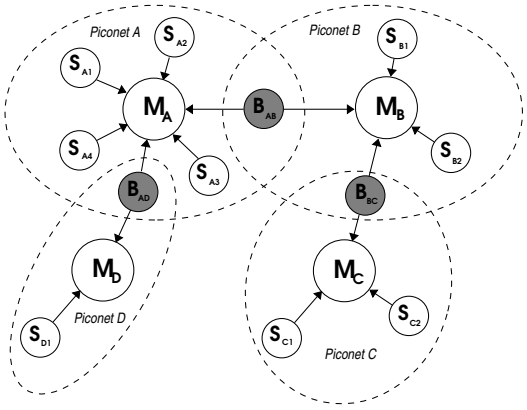


Figure 5.4: Example ESN Routing Zone

After each update the piconet master updates its ESN routing table with the new information. An example of an ESN routing table for a piconet master M_A is given in Table 5.1.

For the purpose of illustration, let us consider the ESN routing table based on the topology in Figure 5.4.

Table 5.1: Piconet master M_A's ESN Routing Table

PicoID	**BD_ADDR**	**SEQ**	**B**	**M**	**F**
B	$<BD\_ADDR_{M_B}>$	seq_{M_B}	*0*	*1*	*0*
B	$<BD\_ADDR_{S_{B_1}}>$	seq_{M_B}	*0*	*0*	*0*
B	$<BD\_ADDR_{B_{AB}}>$	seq_{M_B}	*1*	*0*	*0*
B	$<BD\_ADDR_{S_{B_2}}>$	seq_{M_B}	*0*	*0*	*0*
B	$<BD\_ADDR_{B_{BC}}>$	seq_{M_B}	*0*	*0*	*1*
D	$<BD\_ADDR_{M_D}>$	seq_{M_D}	*0*	*1*	*0*
D	$<BD\_ADDR_{S_{D_1}}>$	seq_{M_D}	*0*	*0*	*0*
D	$<BD\_ADDR_{B_{AD}}>$	seq_{M_D}	*1*	*0*	*0*

Note that an ESN routing zone extends only 1-scatternet hop from any given piconet, so that nodes in piconet C (Figure 5.4) are excluded from piconet master M_A's ESN.

Table 5.1 contains six columns: *PicoID*, *BD_ADDR*, *SEQ*, *B*, *M*, and *F*. *PicoID* is a local identifier given to each adjacent piconet, which is is uniquely assigned to each entry in a piconet master's ESN routing table that has the *M* flag set. The *M* flag is set to indicate that the entry corresponds to an adjacent piconet master.

In addition, the *B* flag is set for table entries corresponding to nodes B_{AB} and B_{AD} to indicate that they have a bridge connection to the masters of the piconets with *PicoID* *B* and *D* respectively. The node B_{BC} also has the *F* flag set, which indicates that it is a bridge to a foreign piconet. This is a useful feature to prune the list of potential next-hop nodes for route discovery requests. An adjacent piconet without foreign (meaning other piconets outside the ESN) bridge connections would not be able to propagate the request further anyway.

Every new ESN update from an adjacent piconet master also includes an increasing sequence number, which invalidates previous entries for that *PicoID*. Note that due to our bridge link agnostic approach, specifically for the purposes of providing multiple inter-piconet routes, multiple entries with the same *PicoID* can have the bridge flag *B* set to indicate a bridge connection.

ESN Optimization

As a further optimization to the ESN update process, three different types of ESN messages are used to reduce the amount of data that needs to be exchanged:

- *ESNFull*: Complete update initially and every esn_{count} updates
- *ESNUpdate*: Refresh sequence number and, if any, add new entries
- *ESNFlush*: Flush all existing entries

The *ESNFull* type is the basic message that is used to initially populate the ESN neighbor's ESN routing table, as well as every esn_{count} updates to ensure up-to-date information. A value of esn_{count}=10 has been used for simulation, but can be modified as necessary depending on whether the topology is relatively static or not. It should be noted that when a piconet master node is first incorporated into a scatternet, briefly disconnected, or otherwise lose its state, the initial ESN exchange must be of type *ESNFull* and have the sequence number set to 0, so that the peer piconet master recognizes that the peer lost its previous state since sequences numbers are otherwise non-zero.

The *ESNUpdate* type can simply contain an updated sequence number to refresh existing entries or it can also contain new entries, in which case existing entries are implicitly updated with the new sequence number. Since all ESN information is unique to a piconet and proatively pushed by the piconet master (and identified in the neighboring master's ESN table by the *PicoID* parameter), there is no risk of introducing inconsistent data as long as no malicious nodes exist in the scatternet.

The *ESNFlush* type is used when a piconet master is experiencing significant topology changes, excessive routing or processing load, in preparation for mobility, depleted energy supply, or otherwise wants to inform its peer piconet master that existing information should be removed from the ESN table.

Route Discovery

Whenever a route request is initiated, or received from a neighboring piconet master, to either a destination or for a service, the HBSR-DISCOVERY procedure (illustrated in Figure 5.5) is executed by the piconet master. This procedure handles all route discovery operations, except route errors, and encompasses both destination and service-based route requests, as well as route reply generation.

We describe the HBSR-DISCOVERY procedure by refering to the line numbers shown in Figure 5.5. First, a check to determine that only piconet masters execute the procedure is performed on line 1. Then, lines 2-22 of HBSR-DISCOVERY determine whether a valid route is already available locally, either by way of the local master being the destination piconet master itself, the destination being a neighboring piconet master in the ESN, or a route being available in local route cache.

More specifically, on lines 5 or 6 the partial route from the source is set. When the master itself is the source of the discovery, the *rte_to_src* is empty (line 5). Otherwise, when the current master is an intermediate node the partial route from the source is set to $route_{partial}$ (line 6), which is the existing partial route gossiped in the HBSR-DISCOVERY procedure from the previous *1-scatternet hop* piconet master nodes on lines 56-57. Next, if the target of the route discovery is a service (which is the case when a UUID is specified but a destination address is not), lines 7-10 determine whether there is already an existing mapping between the service UUID and a *BD_ADDR* in the local *ServiceUUID_Table*. This table caches UUID and *BD_ADDR* pairs from route request and reply packets. If a match is found, the *BD_ADDR* of a node that provides the service corresponding to the requested UUID is assigned to BD_ADDR_{DST} on line 8. If a match is not located, then the local SDP server is queried on line 9 for an intra-piconet service provider node. The reason that these calls are not reversed is that table lookups are a lot faster than the SDP queries and entries from previous SDP queries are cached in the *ServiceUUID_Table* for future use anyway.

Whether a BD_ADDR_{DST} is passed to the procedure (destination-based) or assigned

```
HBSR-DISCOVERY(p_1, p_2, degree_thres, BD_ADDR_local, BD_ADDR_SRC, BD_ADDR_DST, UUID ← NIL, route_partial ← NIL)
 1  if !LT_ADDR
 2    then if BD_ADDR_DST = NIL and UUID = NIL
 3           then return "No Destination BD_ADDR or Service UUID for Route Requests"
 4         if BD_ADDR_SRC = BD_ADDR_local or BD_ADDR_SRC ∈ LocalPiconet
 5           then rte_to_src ← NIL
 6           else rte_to_src ← REVERSE(route_partial)
 7         if BD_ADDR_DST = NIL and UUID in ServiceUUID_Table
 8           then BD_ADDR_DST ← LOOKUP-SERVICE-DEST(ServiceUUID_Table[UUID].list)
 9           else if BD_ADDR_DST = NIL and Local_SDP_Query(UUID)
10                  then BD_ADDR_DST = BD_ADDR_local
11         if BD_ADDR_DST ≠ NIL and BD_ADDR_DST not in rte_to_src and BD_ADDR_local not in rte_to_src
12           then if BD_ADDR_DST = BD_ADDR_local or BD_ADDR_DST ∈ LocalPiconet
13                  then route ← REVERSE(rte_to_src) + BD_ADDR_local
14                  else if BD_ADDR_DST in ESN_Table
15                         then rte_to_dest ← ESN_Table[BD_ADDR_DST].route
16                  else if BD_ADDR_DST in Route_Cache and Route_Cache[BD_ADDR_DST].fresh = 1
17                         then rte_to_dest ← Route_Cache[BD_ADDR_DST].route
18                if rte_to_dest
19                  then if BD_ADDR_local not in rte_to_dest and ( each hop_i in rte_to_src) not in rte_to_dest
20                         then route ← REVERSE(rte_to_src) + BD_ADDR_local + rte_to_dest
21                         else invalid ← TRUE
22           else invalid ← TRUE
23         if invalid ≠ TRUE and (route or UUID ≠ NIL)
24           then if rte_to_src = NIL and route
25                  then return route
26                  else if UUID ≠ NIL and route
27                         then SEND_SREP(BD_ADDR_local, rte_to_src[0].BD_ADDR, route, UUID)
28                              return "Service Reply Sent"
29                         else SEND_RREP(BD_ADDR_local, rte_to_src[0].BD_ADDR, route, BD_ADDR_DST)
30                              return "Route Reply Sent"
31           else if invalid ≠ TRUE
32                  then non_sink ← ⟨⟩
33                       next_hops ← ⟨⟩
34                       for each e in ESN_Table
35                       do if e.F = 1
36                            then non_sink[e.PicoID] = TRUE
37                       for each e in ESN_Table
38                       do if e.M = 1 and non_sink[e.PicoID] = TRUE
39                            then next_hops[e.PicoID] = e.BD_ADDR
40                       if BD_ADDR_SRC ≠ BD_ADDR_local
41                         then for each hop_i in rte_to_src
42                                do if hop_i.BD_ADDR in next_hops
43                                     then next_hops ← next_hops − ⟨hop_i⟩
44                       degree ← LENGTH[next_hops]
45                       if degree = 0
46                         then return "No Valid Next Hops for Route Requests"
47                         else if degree < degree_thres
48                                then p ← p_2
49                                else p ← p_1
50                       num_sent ← 0
51                       for each n_i.BD_ADDR in next_hops
52                       do r_n ← rand(0, 1)
53                          if r_n ≤ p and BD_ADDR_local not in rte_to_src
54                            then route_partial ← REVERSE(rte_to_src) + BD_ADDR_local
55                                 if BD_ADDR_DST ≠ NIL
56                                   then SEND_RREQ(BD_ADDR_local, n_i, route_partial, BD_ADDR_DST)
57                                   else SEND_SREQ(BD_ADDR_local, n_i, route_partial, UUID)
58                                 num_sent ← num_sent + 1
59                       if num_sent > 0
60                         then return "num_sent Route Requests Sent"
```

Figure 5.5: HBSR discovery procedure HBSR-DISCOVERY

on lines 7-10 (service-based request and cached mapping), line 11 performs a loop detection to ensure that BD_ADDR_{DST} and BD_ADDR_{local} do not appear in the partial route from the source node already, in which case the route discovery is marked *invalid* on line 22 to prevent propagation of routing loops. Unless the route discovery is invalid, lines 12-17 attempt to locate an existing route to the destination node's piconet master from either the local node, an ESN entry, or a route cache entry if available. First, the destination address is checked for a match against the local master, which does not add any *1-scatternet hops* to the modified source route other than the current piconet master itself as the destination. Then, the ESN table and the route cache are both checked for a route to BD_ADDR_{DST} on lines 14 and 16 respectively. If either of them contain a route to BD_ADDR_{DST}, denoted *(rte_to_dest)*, lines 18-21 perform additional loop detection to ensure that neither the existing node, BD_ADDR_{local}, nor any of the nodes already in the partial route from the source are in the route to the destination, which would cause a routing loop.

If no routing loops are detected, a complete modified source route, denoted *route*, is constructed by appending the reverse of the route to the source from the current node, denoted REVERSE*(rte_to_src)*; the current node's BD_ADDR, denoted BD_ADDR_{local}; and the route to the destination from the current node, denoted *rte_to_dest*, on line 20. It should be noted that by convention all routes are stored with the source as the first entry and the destination, or intermediate node if partial, as the last entry.

If a valid route to the destination is available on line 23, then lines 24-30 determine where the reply is sent. If the current piconet master or a node in the local piconet is the source of the route discovery, then the route is directly returned to the current piconet master on line 25. Otherwise, a reply packet must be sent toward the source of the discovery. Line 26 determines if the discovery is for a service UUID, in which case a reply, including the UUID and the modified source route to the destination piconet offering the service, is unicasted to the previous-hop piconet master and on along the modified source route back toward the source on line 27. Similarly, if the route discovery was initiated for a specific destination, a

reply is unicasted back to the source on line 29. Note that the two procedures SEND_RREP and SEND_SREP, on lines 27 and 29 respectively, merely symbolize the generation of a route reply packet, for destination- and service-based requests respectively, and transmission to the previous hop ESN piconet master.

If a route reply can not be returned directly and the loop detection mechanism did not mark the discovery *invalid*, then additional route discovery operations are performed on lines 31-60 of HBSR-DISCOVERY to propagate the route request further. The first step in this process is to enumerate the piconets within the local ESN routing table that have at least one member with the *F* (*foreign*) flag set. This means that those neighboring ESN masters have at least one bridge connection to a piconet master that is not within the ESN; hence the connotation *foreign* piconet. This is done on lines 34-36 where each adjacent master entry is set to TRUE in the *non_sink* list when the *F* flag is set. Thereafter, on lines 37-39 the *BD_ADDR* for each entry found in the previous step is added to the *next_hop* list. Piconets that are in the *next_hop* list are next-hop candidates for finding a route to the destination, since line 14 established that the destination node is not within the ESN (in which case a route reply would have already been returned on lines 27 or 29).

If the route request packet was received from a neighbor (and not initiated), the master must also prune the previous hop neighbor as well as every other neighbor along the partial modified source route from consideration, so that the route request is not propagated back to the neighbor it was received from. This step is performed on lines 40-43.

The piconet master sets the *degree* on line 44. The degree is the number of forwarding nodes that are available after piconets with no foreign connections and nodes already in the modified source route have been pruned. It is used to determines the gossiping value, on lines 47-49, using the $degree_{thres}$ parameter as a threshold. The value of the parameter $degree_{thres}$ for determining sparse or dense connectivity is discussed further in Section 5.5.

We utilize a technique called gossiping to propagate the route request packets throughout the scatternet. By using gossiping instead of flooding, we can reduce the routing overhead

while still ensuring with high probability that the scatternet is covered and the destination can be found. On lines 51-58, each of the remaining forwarding nodes are gossiped based on either p_1 or p_2 as the probabistic threshold, depending on the number (degree) of forwarding nodes and the $degree_{thres}$ threshold value. For successful gossips, determined on line 53, either a service or a destination-based route request is propagated through the common bridge node to the next hop ESN piconet master. A loop detection is also performed on line 53 to ensure that is current piconet master is not already in the partial route from the source. Note that the two procedures SEND_RREQ and SEND_SREQ, on lines 56 and 57 respectively, merely illustrate the generation of a route request packet and transmission to the next hop ESN piconet master.

Route Reply

Route replies are generated by both destination nodes and intermediate nodes, on lines 27 or 29 of HBSR-DISCOVERY, depending on whether the request was for a service or destination respectively, and unicasted back along the reverse modified source route to the initiator. Due to the master-centric design of the algorithm, piconet masters reply to route requests on behalf of their piconet members. A piconet master is allowed to reply to a *destination-based* route request if:

- It is itself the target of the route request
- A node within its piconet is the target of the route request
- A node within its ESN is the target of the route request
- It has a valid route in its route cache to the target of the route request

For *service-based* route requests a piconet master is allowed to reply if:

- It provides the service sought in the route request itself
- Based on local SDP information, a node within its piconet provides the service sought in the route request

- An entry from the Service UUID cache matches the service sought in the route request and a valid route exists in its ESN table or in the route cache

Mappings between UUIDs and destination addresses allow intermediate masters to reply to service-based requests that otherwise would have been re-forwarded.

Route Maintenance

Aggressive route caching is used to reduce the need for route discovery outside the ESN zone. Each piconet master maintains a local route cache in which it stores routes that it receives from route request and route reply packets. The route cache is implemented as a node-centric path cache, and includes local timestamps and the *BD_ADDR* of the initiator of the route request/reply. This information is used to invalidate old routes and prevent duplicate route replies.

When an intermediate node receives a route request the partial modified source route from the source piconet master to the current piconet master is added to the route cache. When route reply packets reach intermediate nodes on the reverse modified source route, the complete modified source route is split and added to the route cache relative to the current node.

Each entry in the route cache consists of a route, a timestamp, and a source *BD_ADDR*. The route is always added relative to the master itself with the originator of the packet as the last hop. For example, if intermediate node C receives a route reply containing the modified source route $E \leftrightarrow D \leftrightarrow C \leftrightarrow B \leftrightarrow A$ from a node E, in response to a route request from node A, it adds the route $C \leftrightarrow D \leftrightarrow E$ to its route cache along with the current timestamp and the *BD_ADDR* of node E as well as the partial modified source route $C \leftrightarrow B \leftrightarrow A$ along with the timestamp and the *BD_ADDR* of node A. Periodic route cache maintenance is also performed to ensure that cached routes are fresh. To prevent cache pollution, a *stale* flag is set for routes older than t_{cache} seconds. After another t_{cache} seconds, stale entries are removed from the cache.

Route Error

When sending data packets along the *modified source routes*, the loss of an intermediate node along the route can cause abruption of the data flow, unless the intermediate node, at the point of next hop failure, has another route, either as its ESN neighbor or as another fresh route cache entry in which the destination is an intermediate node. When this occurs the intermediate piconet master, at the point of next hop failure, sends a route error reply packet back to the source node to trigger a new route discovery, similar to AODV [13]. As the route error passes through intermediate nodes it invalidates route cache entries for the destination.

Route errors are not used as part of the discovery process, but merely as a response to a broken modified source route during data packet transmissions. It operates similar to a route reply, in that it is source routed along the reverse of the modified source route back to the source, but instead of providing a route it triggers a new HBSR-Discovery from the source node.

Loop Freedom Correctness Proof

HBSR provides routing loop-freedom by preventing the introduction of *routing loops* due to multiple instances of the same node in the modified source route. The following illustrates the proof of routing loop-freedom in HBSR.

Proof. The proof is by contradiction. Suppose that there exists a modified source route with multiple instances of an intermediate piconet master forming a routing loop, which is returned as valid from HBSR-Discovery. Any intermediate piconet master, the source, or destination node, splits the modified source route in two parts: the partial route from the source from the current node, *rte_to_src*, and the partial route to the destination from the current node, *rte_to_dest*, with zero or more entries in each part. The routing loop must therefore be contained in either *rte_to_src* or *rte_to_dest*. For the first part of the proof, we assume that the routing loop is in *rte_to_src*. However, that is a contradiction since lines 11

and 53 ensure that any intermediate node that is added to the partial route from the source is not already in *rte_to_src*. For the second part of the proof, we assume that the routing loop is in *rte_to_dest* returned from either an ESN tables entry or a route cache entry. However, that is a contradiction since line 19 ensures that neither the current intermediate node nor any entry in *rte_to_src* are already in *rte_to_dest*. ∎

5.5 Performance Evaluation

To evaluate the performance of the Hybrid Bluetooth Scatternet Routing (HBSR) algorithm we developed custom functionality for HBSR using the UCBT extension module [69] and the ns-2 [1] network simulator. We specifically evaluate the usefulness of the ESN zone routing approach, the appropriate selection of an Inter Piconet Scheduling (IPS) algorithm, route acquisition delay, and performance tuning of gossiping parameters.

5.5.1 Extended Scatternet Neighborhood (ESN)

Due to the topology constraints imposed by the scatternet formation approach (specifically the BTDSP formation algorithm in our case) and the piconet structure, Bluetooth® scatternet topologies are much different than traditional DSSS ad hoc networks in how they related to hybrid (proactive versus reactive) zone routing, as only bridge nodes form connections to next-hop piconet masters. We evaluate the topology coverage of the ESN zone to determine the effectiveness of the proactive HBSR-Esn procedure in terms of how many destination nodes within the scatternet are reachable from any given source node by an ESN routing table lookup rather than invoking the reactive HBSR-Discovery procedure.

We utilize BTDSP and the ns-2 [1] network simulator to form random 2-Slave/Slave Mesh (SSM) scatternet topologies with 8, 16, 32, and 64 nodes respectively; each replicated 100 times. Thereafter we compute the all-pairs shortest paths (in terms of 1-scatternet hops) between any two nodes in the scatternet topology. As illustrated in Figure 5.2, any destination *within one 1-scatternet hop* is reachable within the Extended Scatternet

Neighborhood (ESN) of the source node's piconet master. Therefore, any node within one 1-scatternet hop[2] of the source node can be reached without invoking the HBSR-Discovery procedure. The results are illustrated in Figure 5.6.

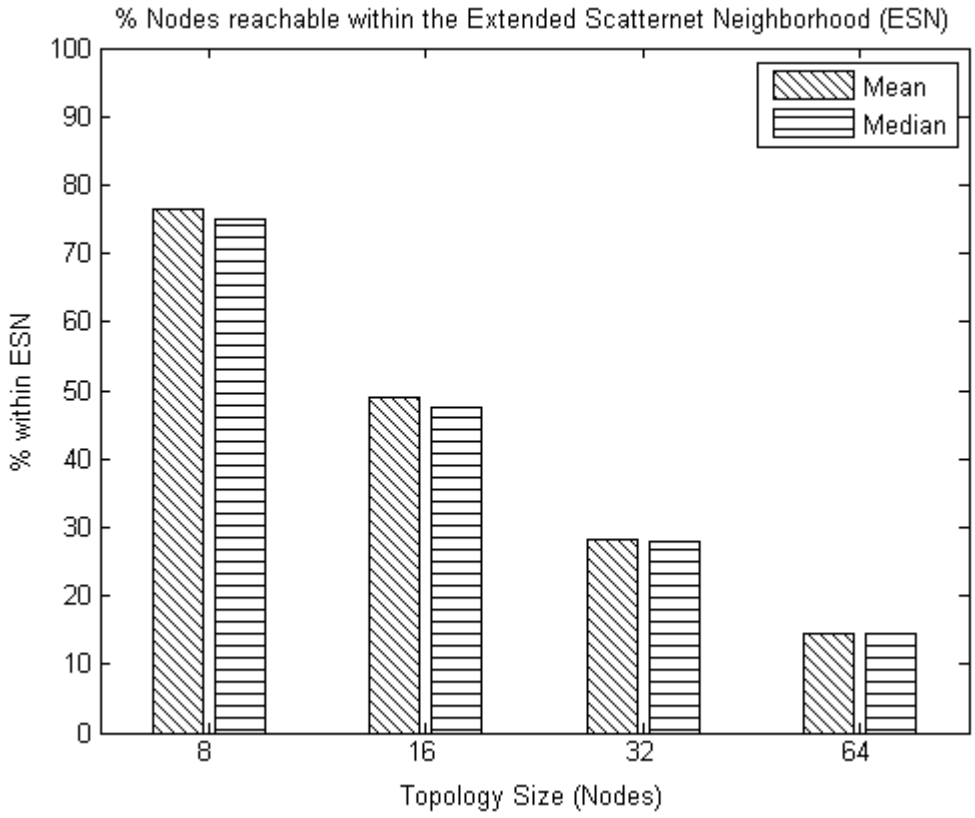


Figure 5.6: Extended Scatternet Neighborhood (ESN) Node Reachability

From Figure 5.6 it should be noted that for scatternet topologies of 16 nodes or less, on average, about 50% of nodes within the scatternet are reachable without need for route discovery. For scatternet topologies with 32 nodes or less, on average, about 30% of nodes are directly accessible by way of ESN routing table lookups. As Bluetooth scatternets are most suitable for WPAN applications with around 30 nodes or less, as opposed to large-scale WLANs where IEEE 802.11 is more suitable, the hybrid HBSR approach eliminates the need for a third or more of the route discovery operations for such topologies.

[2]As illustrated in Figure 5.4, if both the source and the destination nodes are either pure slaves or bridges to piconets other than *between* the source and destination piconets, a *1-scatternet hop* could be 4 hops at the link level.

5.5.2 Route Acquisition Delay

To evaluate the performance of the HBSR route discovery process and determine the most suitable Inter Piconet Scheduling (IPS) algorithm to use, we study the route acquisition delay, or in other words the amount of time before a route reply is returned to the HBSR-Discovery initiator. We compare the performance of the HBSR-Discovery process in conjunction with the Maximum Distance Rendezvous Point (MDRP) algorithm by Johansson *et al.* [56] as well as the Dichotomized Rendezvous Point (DRP) algorithm by Wang *et al.* [70]. These algorithms are also described further in Section 3.2.6. It should also be noted that, as suggested in [3] and [22], that the Exhaustive Round-Robin (ERR) Intra Piconet Scheduling (IRPS) algorithm performs significantly better than any other polling scheme in terms of both delay and fairness, so ERR is used for local piconet polling.

To evaluate route acquisition delay and scheduling delay we run the HBSR-Discovery procedure from a fixed source node to random destination on connected 32-node 2-SSM topologies with between two and six bridge nodes. Each iteration is ran using both the Maximum Distance Rendezvous Point (MDRP) and Dichotomized Rendezvous Point (DRP) IPS algorithms. It should be noted that although gossiping is available in the HBSR algorithm, this particular simulation study is constructed to evaluate delay and differences between the Inter Piconet Scheduling (IPS) algorithms without confounding with the gossiping parameter(s) and therefore $p = 1$ is used.

Figure 5.7 illustrates the route acquisition delay for the reactive HBSR-Discovery mechanism used for route discovery. It should be noted that caching of modified source routes are not considered, which would reduce the acquisition delay as intermediate nodes could respond when past HBSR-Discovery invocations have already produced a valid route from the intermediate node to the destination. The mean route acquisition delay is illustrated by solid lines, while the minimum and maximum outliers for HBSR with either Maximum Distance Rendezvous Point (MDRP) or Dichotomized Rendezvous Point (DRP) are also illustrated.

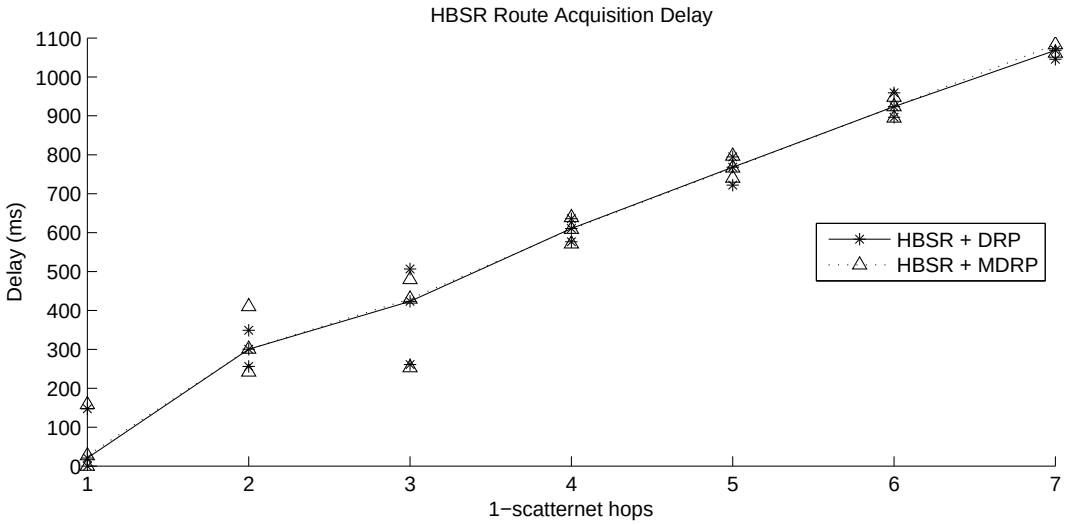


Figure 5.7: HBSR Route Acquisition Delay

From Figure 5.7 it can be seen that there is a significant delay incurred for destinations outside the ESN zone that require route discovery. The reason for this is two-fold: the need for buffering of packets at intermediate nodes due to specific rendezvous points where peer nodes have to be synchronized and the need to fragment L2CAP frames and transmit multiple packets as the modified source routes grow longer. However, it should be noted that using the *modified source route* approach instead of hop-by-hop source routes requires only n entries, where n is the number of piconet masters along the route, as opposed to $n + (n - 1)$ for complete source routes including bridge nodes. We plan to address the modified source route overhead for data packets by adding flow IDs in our future work. It does show, however, that HBSR is more suitable for denser and more compact topologies than for long scatternet paths, in which case a specialized on-demand approach such as [73] might be more suitable.

It can also be seen from Figure 5.7 that the mean route acquisition delay for nodes within 1-scatternet hop is larger than it would be if only ESN table lookups were performed. The reason for this is that in some dense topologies that contain piconet with high bridge degrees, complete ESN updates had not been propagated yet when the HBSR-Discovery procedure

was initiated in the simulation. This does illustrate the resilient behavior of HBSR as routes can be found using the reactive approach even within the ESN zone, if incomplete or no information is available.

It should be noted that significantly fewer iterations of results were available for routes longer than five 1-scatternet hops, since the BTDSP algorithm is not designed to form scatternet topologies with long chains of small size piconets.

Also, as shown in Figure 5.7 there were no significant differences noted between the performance of HBSR using either the Maximum Distance Rendezvous Point (MDRP) or Dichotomized Rendezvous Point (DRP) IPS algorithms.

5.5.3 Parameter Optimization

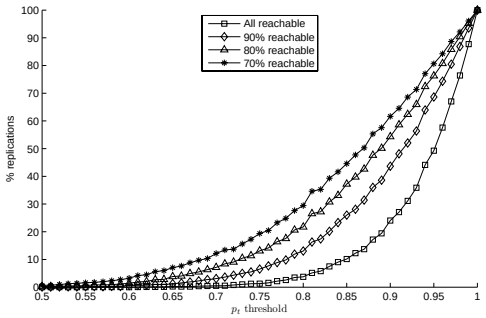
In general MANETs, as investigated in [40], when gossiping of route discovery messages is employed a node either "gossips" and re-broadcasts the message to all its neighbors or does not propagate the message to anyone. We employ a different approach for HBSR in which each piconet master determines individually, for each bridge node that is a forwarding candidate (meaning that it has a another *foreign* bridge connection), whether to "gossip" or not. This approach takes into consideration that Bluetooth® scatternets are relatively sparse and small in size, compared to the extremely large and much denser general ad hoc networks discussed in [40] and [41].

To allow for gossiping to be used by HBSR we must determine the gossiping probability threshold values, p_1 and p_2, as well as the outgoing degree of connectivity $degree_{thres}$ parameter; all used in the HBSR-Discovery procedure, illustrated in Figure 5.5. In piconets with a bridge out-degree less than $degree_{thres}$, the larger threshold parameter p_2 is used, while in piconets with $degree_{thres}$ or higher bridge out-degree p_1 is used.

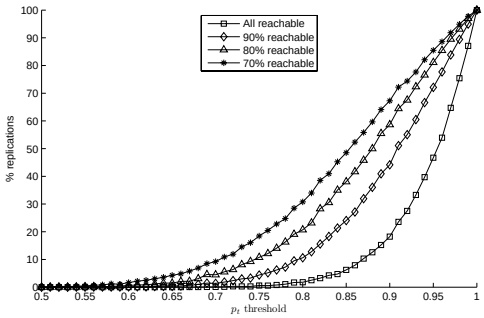
For each experiment in this section we simulate reachability from a random single-source to all other nodes in 73 actual BTDSP generated 32 node topologies, with a 2.16 mean bridge degree, using a modified breadth-first search approach that incorporates gossiping.

For each topology each treatment of factors is replicated 100 times. To avoid unnecessary confounding in the results of the experiments we assume that there is no radio interference and no link failures.

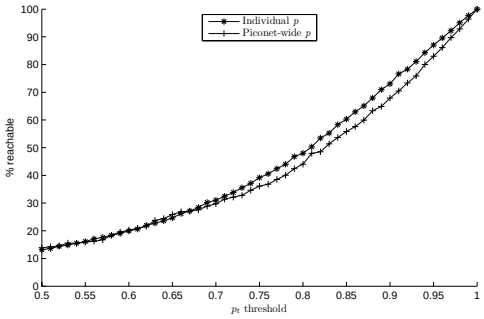
Unlike the simulation study in [40], which examines extremely large and dense traditional omni-directional MANETs[3], our experiments focus on more realistic Bluetooth® WPAN topologies. This allows us to investigate the effectiveness of gossiping for actual BTDSP scatternet topologies. As discussed in [40], the bimodal behavior of gossiping is only evident in extremely large topologies, which are not applicable to Bluetooth® scatternets, so that is not discussed in this simulation study.



(a) Piconet-wide p: Nodes reachable for % of replications



(b) Individual p: Nodes reachable for % of replications



(c) Piconet-wide p vs. Individual p: Mean reachability comparison

Figure 5.8: HBSR gossiping node reachability with single threshold

To investigate the effects of the *individual probability* forwarding method (gossiping de-

[3]Theoretical MANETs with 1000 or more nodes and connectivity degrees as large as 10 or higher [40].

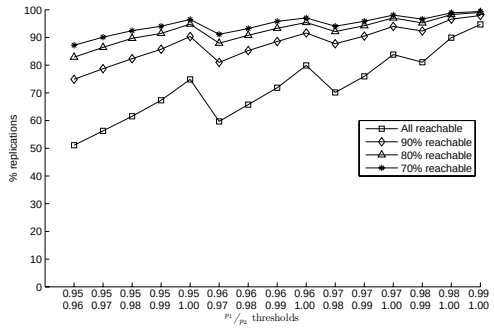
termined for each forwarding candidate separately) compared to the *piconet-wide probability* forwarding method, we conduct an experient based on actual BTDSP topologies, as previously discussed herein. The results are illustrated in Figure 5.8.

Figure 5.8(a) and Figure 5.8(b) illustrate the percentage of replications, using a single probability threshold value p_t, in which: all nodes were reached, 90% of nodes were reached, 80% of nodes were reached, and 70% of nodes were reached. Figure 5.8(c) illustrates the mean reachability percentage across all replications and compares using individual p and piconet-wide p forwarding methods, where p refers to a pseudo-random probability value generated either for each forwarding candidate or piconet-wide respectively. It should be noted that, although the mean reachability is only marginally higher using the individual p method, a higher percentage of replications provide better reachability using individual p forwarding methods for $p_t > 0.90$. For the scatternet topologies investigated, reachability using both methods was poor for $p_t < 0.90$ and marginal for $p_t < 0.95$. Therefore, for the purpose of HBSR, we focus on gossiping threshold values where $p_t > 0.95$.

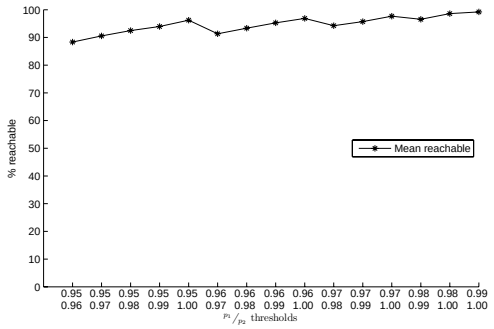
As described in Section 5.4.1, HBSR utilizes a two-threshold scheme, where $p_1 < p_2$. The larger threshold, p_2, is utilized for sparse topologies, where the number of forwarding candidate bridge nodes, in an individual piconet, is less than $degree_{thres}$, and p_1 is used when there are at least $degree_{thres}$ forwarding candidate in a piconet.

We investigate suitable values for p_1, p_2, and $degree_{thres}$ that provide high reachability while reducing the number of route request forwarding messages. Figures 5.9(a), (c), and (e) illustrate the reachability percentage across replications for (p_1, p_2) pairs, while maintaining the invariants $p_1 < p_2$ and $p_i > 0.95$, where $\{i : 1 \leq i \leq 2\}$, for $degree_{thres}$ 2, 3, and 4 respectively. Figures 5.9(b), (d), and (f) simply illustrate the mean reachability for $degree_{thres}$ 2, 3, and 4 respectively.

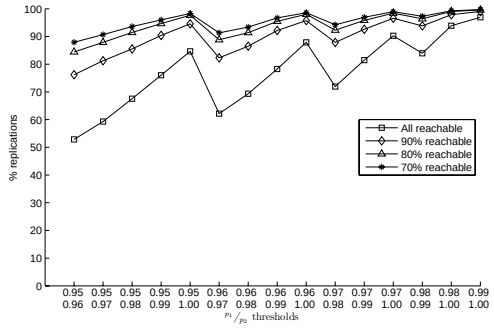
It should be noted from Figure 5.9 that for reachability in the 70^{th} percentile, there are not significant differences between $degree_{thres}$ 2, 3, and 4. However, for the 90^{th} percentile and above, it can be seen from Figures 5.9(a), (c), and (e) that the choice of $degree_{thres}$



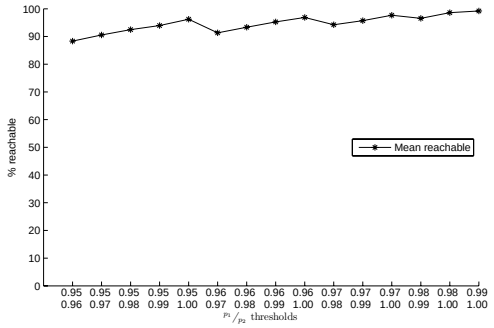
(a) $degree_{thres}$ = 2: Nodes reachable for % of replications



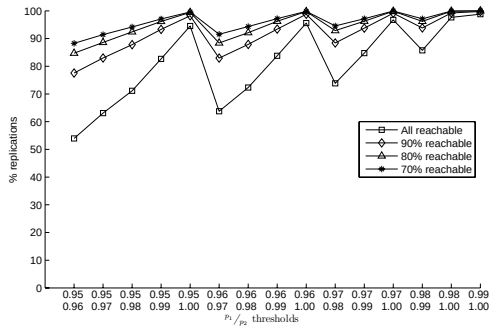
(b) $degree_{thres}$ = 2: Mean reachable



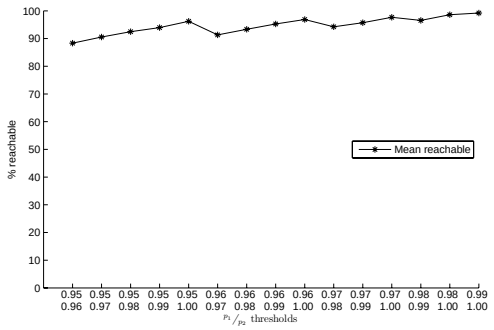
(c) $degree_{thres}$ = 3: Nodes reachable for % of replications



(d) $degree_{thres}$ = 3: Mean reachable



(e) $degree_{thres}$ = 4: Nodes reachable for % of replications



(f) $degree_{thres}$ = 4: Mean reachable

Figure 5.9: HBSR gossiping node reachability with two thresholds, p_1 and p_2

value does affect reachability.

Since BTDSP forms topologies with mean bridge degree between 2 and 3, we exclude $degree_{thres}$=4 from consideration due to the fact that p_1 would rarely be used. Based on comparison between Figures 5.9(a) and 5.9(c) there is a significant difference in reachability

between $degree_{thres}$=2 and $degree_{thres}$=3. Therefore, we heuristically choose $degree_{thres}$=3. We identify p_1=0.96 and p_2=1.00 as parameter choices that provide reachability in the 90^{th} percentile for most replications and complete reachability for 85% of replications.

It should be noted that setting p_2=1.00 equates to using simple flooding of route requests for piconets with less than three forwarding candidates. However, as illustrated by the results in Figure 5.8 and Figure 5.9 it is deemed a necessary compromise in order to ensure sufficient topology coverage for route discovery operations in Bluetooth® scatternets.

5.6 Summary

In this chapter we presented the Hybrid Bluetooth Scatternet Routing (HBSR) algorithm, which allowed scatternet nodes to discover peers by either destination address or by a service *U*niversal *U*nique *ID*entifier (UUID). We also considered the underlying topology constraints and defined the Extended Scatternet Neighborhood (ESN) as the 1-scatternet-hop region formed by the union of adjacent piconets. Furthermore, our hybrid approach utilized a proactive approach within the ESN and used a reactive *modified* source-routing approach outside the ESN. The modified source-routes were bridge link agnostic and made up of only intermediate piconet masters, so that changes in individual bridge links did not affect the entire scatternet route. The routing protocol was placed above the L2CAP layer to allow easier access to routing functionality for application layer protocols. However, the inclusion of source routes and need for fragmentation of L2CAP segments due to larger HBSR packets also incurred a significant route acquisition delay for destinations outside the ESN zone. As more than 50% of nodes are within the ESN zone for scatternet topologies of 30 nodes or less formed by BTDSP, the proactive component of the hybrid approach is very beneficial as route discovery operations are inhibited by difficult scheduling coordination in general purpose scatternet topologies. The addition of gossiping for route discovery outside the ESN zone was used to reduce routing overhead, but provided marginal benefit except for in dense

topologies where piconets had many bridge connections.

Chapter 6

Security in Bluetooth Scatternets

6.1 Introduction

Bluetooth® security features are described in the specification[11] at several layers of the protocol stack as well as in the Generic Access Profile (GAP)[8]. However, only security for direct links within a piconet is covered. We describe the current security features from the specification in detail in Section 6.2.2. The specification outlines both authentication and encryption procedures for Bluetooth®. For increased security each master-slave link can use a distinct link key or a piconet-wide, broadcast-friendly link key. Either way, the scope of authenticity and confidentiality for Bluetooth® communication remains within a single piconet.

By extending Bluetooth® functionality to scatternets we also introduce some security implications. Increasing the connectivity between nodes offers intruders more ways to compromise devices. Therefore, we specifically address scatternet security in this chapter. We present an algorithm for secure communication between adjacent piconets within an Extended Scatternet Neighborhood (ESN) and for securing a piconet in Section 6.3. Before describing the algorithms we first analyze current security measures from the Bluetooth® specification and related work, and discuss problems and potential threats. Thereafter, we extend the current Bluetooth® security model to include scatternet communication.

The remainder of this chapter is organized as follows. In Section 6.2 we overview general security issues in wireless ad hoc networks and describe the details of existing security features

in the Bluetooth® specification. We also discuss other proposed solutions and how they apply to Bluetooth® security. In Section 6.3 we describe two scenarios to provide link level security in scatternets. Section 6.3.1 presents the two scenarios and how they extend the piconet security model to secure scatternets. Each of the two scenarios are presented in Sections 6.3.2 and 6.3.3 respectively. Thereafter, we conclude the chapter in Section 6.4.

6.2 Security Preliminaries

6.2.1 Concepts

Network security is a crucial topic that is often overlooked and never sufficiently addressed. The problem of maintaining a secure system, or protecting *confidentiality* and *integrity* while ensuring *availability*, is difficult in both Wireless Personal Area Networks (WPANs) and general wireless ad hoc networks. Wireless devices are generally resource poor, which makes it difficult to implement strong security mechanisms. Most Bluetooth® devices have limited processing power and battery capacity. Therefore, a general purpose security mechanism must adhere to strict resource constraints. For instance, public key cryptography is much more computationally intensive than symmetric solutions and is therefore not always suitable for wireless devices. On the other hand, private key cryptography uses the same key for both encryption and decryption, so secret keys must be exchanged securely a priori. Although security is an important issue in any system, there are specific security concerns for Bluetooth® WPANs. In Section 6.2.2 we describe the methods currently used for security within the scope of the Bluetooth® specification [11]. Section 6.2.3 further describes security threats and attacks that are specific to Bluetooth® WPANs.

6.2.2 Overview

Key generation

The Bluetooth® specification defines the SAFER+ block cipher algorithm for encryption and link key generation [20]. Encryption of packet payloads is performed in hardware by the

Table 6.1: Keys used for Bluetooth® security

Key name	Key type	Algorithm	Description
K_{init}	Initialization key	E_{22}	*Single session temp key*
K_A	Unit key	E_{21}	*Semi permanent unit key*
K_{master}	Master key	E_{22}	*Non contributory key (point-to-multipoint)*
K_{AB}	Combination key	E_{21}	*Contributory key (point-to-point)*
K_c	Encryption key	E_3	*Derived from current link key*

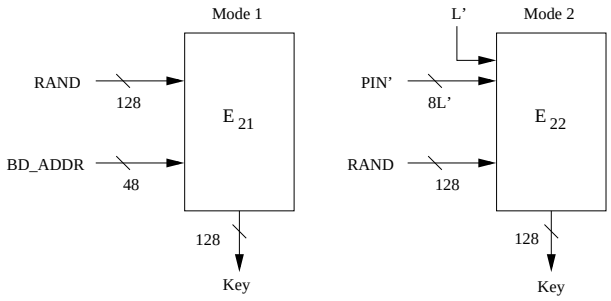


Figure 6.1: E_{21} and E_{22} key generation

E_0 stream cipher encryption engine using four Linear Feedback Shift Registers (LFSRs) [11]. Bluetooth® authentication keys are 128-bits in length but can be shortened to comply with cryptographic export restrictions. Before discussing the details of the authentication and encryption procedures, we discuss the necessary components for Bluetooth® security.

Keys are generated for encryption and authentication procedures using several different algorithms. The algorithms and the corresponding generated keys are illustrated in Table 6.1. A key can either be *temporary*, valid only during a session, or *semi-permanent*. A semi-permanent key is stored in non-volatile memory and is seldom changed.

To initiate authentication and generation of subsequent link keys, an initialization key K_{init} is generated first. The E_{22} algorithm generates this key from a random value *IN_RAND*, the *BD_ADDR* of the initiator, and a Personal Identificant Number (PIN)[11]. Figure 6.1[1] illustrates the E_{21} and E_{22} algorithms for key generation. Since there is no trusted third party to authenticate devices, the Bluetooth® specification assumes that there is a short

[1]Illustration derived from Bluetooth® Core v1.2 specification [11] page 783. Copyright 2003 © Bluetooth SIG [30].

shared secret (PIN) available. Another possibility is to pre-program units with a shared link key instead of the initialization key K_{init}. This is called bonding or pre-pairing. Once a shared initialization link key has been derived, the units can generate a new link key and authenticate each other. Each pair of communicating peers (a master and a slave) can generate a *contributory* combination key K_{AB}. This is done using the E_{21} algorithm from an old key and random numbers from both devices as follows. Each of the peers generates random numbers, LK_RAND_A and LK_RAND_B. Thereafter, each of them generates a key share from their BD_ADDR and the random number:

$$LK\_K_A = E_{21}(LK\_RAND_A, BD\_ADDR_A)$$
$$LK\_K_B = E_{21}(LK\_RAND_B, BD\_ADDR_B)$$

Each device transfers the exclusive-or (XOR) of the old key and its key share. The peer can then derive the LK_K random value of the other device from the exclusive-OR of the received value and the old key. Thus, the new combination key is calculated as $K_{AB} = LK\_K_A \oplus LK\_K_B$.

Alternatively, if a master wants to broadcast data packets to all slaves it can temporarily switch to a common K_{master} key. This is done by the master using the E_{22} algorithm from two random numbers. The K_{master} key is non-contributory, so it has to be distributed to the slaves. This is performed securely using the E_{22} algorithm as well. The master computes a 128-bit overlay from the current link key and a random value. It then sends the random value, the exclusive-OR of the overlay, and the new K_{master} key, call it C, to the peer. Thereafter, the slaves can compute the same overlay from the received random number $RAND$ and the old key. The slaves then derive K_{master} from the exclusive-OR of the received value C and the overlay. If the K_{master} is used to generate the encryption key K_c, then all slaves are capable of decrypting the contents of a data packet regardless of whether the data packet is broadcasted or addressed to a specific slave.

Each device also has a semi-permanent unit key K_A that is generated when the device is manufactured. Although it is semi-permanent, the K_A unit key almost never changes. The

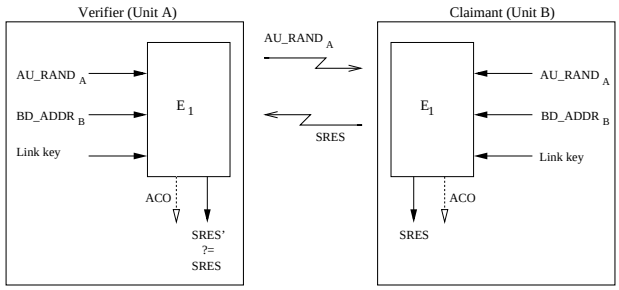


Figure 6.2: Challenge-response authentication

unit key is unique to each device and together with the *BD_ADDR* identifies the device. In earlier versions of the specification the unit key is described as a potential link key. However, as mentioned in [46], exchanging the unit key compromises the identity of a device and makes it vulnerable to impersonation attacks. Therefore, from here on we assume that the unit key K_A is **never** shared with another device.

Authentication

After the peers have generated a shared secret link key they must mutually authenticate each other before generating the encryption key. The authentication procedure is based on a challenge-response scheme between a verifier and a claimant. This is illustrated in Figure 6.2[2]. The verifier first generates a random value AU_RAND_A and sends it to the claimant. The claimant produces a signed response value *SRES'* from AU_RAND_A, the link key, and its *BD_ADDR*, using the E_1 Message Authentication Code (MAC) algorithm and sends it back to the verifier. The verifier performs the same calculation and compares its signed response *SRES* to the received *SRES'* value. After the devices switch roles and successfully perform the same procedure the mutual authentication is complete. In Bluetooth®, authentication and the generation of a link key is called *pairing*[12]. Paired devices have authenticated each other and share a secret key. When this shared key is stored it is called *bonding*[8].

[2]Illustration derived from Bluetooth® Core v1.2 specification [11] page 773. Copyright 2003 © Bluetooth SIG [30].

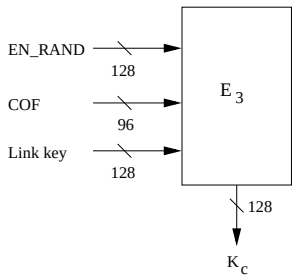


Figure 6.3: E_3 Encryption key generation algorithm

Encryption

After the participants have been properly authenticated, a cipher key K_c is generated. The generation of the encryption key K_c is shown in Figure 6.3[3]. The key length of K_c is 128-bit, but a shortened key K_c' can be derived if necessary to comply with cryptographic export regulations. To ensure a high level of security the link key should be temporarily re-generated. A combination link key K_{AB} is only valid for communication during the lifetime of a session, so it is not vital that combination keys are re-generated as frequently. On the other hand, if a master key is used as the link key then re-generation is important to ensure Perfect Forward Secrecy (PFS). That means that that the shared secret is used only to derive a single key, so that new participants are not able to decrypt old transmissions and old participants are not able to determine new keys. Anytime a new slave joins or an old slave is disconnected, the master key should be re-generated and distributed only to current piconet members.

Once an encryption key K_c has been generated, the payload of data packets can be encrypted. Bluetooth® uses an cipher stream encryption engine E_0 that takes as inputs the encryption key K_c, the *BD_ADDR*, a 128-bit random number, and the 26 Least Significant Bits (LSBs) of the master's clock. The input values are shifted into four Linear Feedback Shift Registers (LFSRs) and then combined in the *Summation Combiner* Finite State Machine (FSM) to produce the cipher stream K_{cipher}[11]. This generates a different cipher stream for

[3]Illustration derived from Bluetooth® Core v1.2 specification [11] page 783. Copyright 2003 © Bluetooth SIG [30].

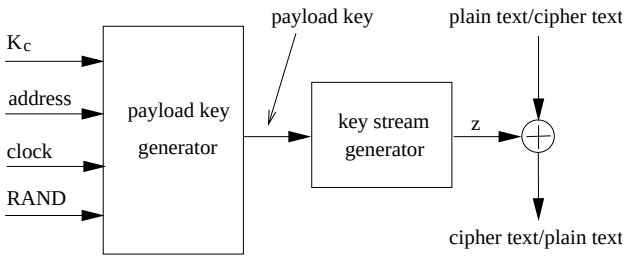


Figure 6.4: E_0 Cipher stream encryption engine

each packet since the clock value is incremented in every time slot. The generation of the stream cipher is illustrated in Figure 6.4[4]. It is then exclusive-OR'd with the packet payload to produce the ciphertext or plaintext.

There are three security modes defined for Bluetooth® in the Generic Access Profile (GAP)[8]. Mode 1 is inherently insecure and does not allow authentication or encryption at all. Security mode 2 requires that devices are authenticated and encryption is used once channels that require security have been established. Thus, the initial connection establishment is not secure in mode 2 either. In the third security mode, security is enforced down to the link layer and the authentication is performed during the connection establishment. Therefore, connection between devices in security mode 3 can be restricted to devices that have previously been paired with each other. During pairing, devices exchange link keys and are said to be *bonding*[8]. Bonding in Bluetooth® means that devices exchange and store a link key for that particular peer. The bonding peer is identified by the Device Access Code (DAC) received during the paging procedure. Hence, for pre-paired devices we can restrict connections from forming with any other devices. We will discuss this further in Section 6.3.

6.2.3 Security Threats

It is extremely difficult to ensure security in a wireless ad hoc network. The characteristics of wireless communication make such a system much more susceptible to *eavesdropping* and

[4]Illustration derived from Bluetooth® Core v1.2 specification [11] page 763. Copyright 2003 © Bluetooth SIG [30].

communication interference such as *signal jamming* [42, 46, 37]. Although the Frequency Hopping Spread Spectrum (FHSS) scheme used in Bluetooth® WPANs inhibits such attacks, it is still not without risk. An intruder that is able to obtain the Frequency Hopping Sequence (FHS) of a piconet can certainly engage in such an attack since all communication still takes place over the air.

The decentralized and ever changing topology of an ad hoc network makes it even more difficult for a device to authenticate the identity of a peer. Stajano and Anderson propose a form of imprinting that they call the *resurrecting duckling* security policy [25]. The idea behind their policy is similar to a duckling that recognize its mother when it is first born. A device imprints the identity of the first peer that sent it a secret key. Their approach provides additional authentication, but it is not applicable to our scenario.

Devices in WPANs are generally also resource poor. Battery exhaustion attacks[25] are possible when an adversary prevents the device from going to sleep or into a battery saving mode.

Jakobsson and Wetzel describe attacks on the Personal Identificant Number (PIN) values used for authentication in [46]. The Bluetooth® specification [11] states that the PIN length can be between 1 and 16 octets. If it is too short then an adversary can exhaustively search all PIN combinations and obtain the initialization key, since the random value in E_{22} is sent in clear text. To alleviate this problem an exponential backoff algorithm is described in the Bluetooth® specification and used to prevent an adversary from repeatedly guessing PIN. However, this could actually be counter productive if an adversary launches a man-in-the-middle attack, as discovered by Jacobsson and Wetzel [46]. If the adversary obtains a transcript of a challenge-response session, it could repeately compute initialization keys offline from PIN guesses. The verification could thereafter be compared to the transcript. The backoff would therefore be beneficial to the adversary, since more time would be available to guess the PIN. Although these attacks are only feasible for short PIN values, it corroborates that a reasonably long PIN is necessary.

We assume that all participants that require authentication and subsequent data encryption share at least a common password, i.e. a Bluetooth® PIN value, for key generation.

6.3 Inter-Piconet Security

In this section we describe security models and approaches to provide security associations between adjacent piconets in a scatternet using inherent Bluetooth® security features.

6.3.1 Secure Scatternet Models

The lack of fixed infrastructure makes proper authentication difficult in Bluetooth® WPANs. We assume that any devices that require strong authentication have already been bonded and pre-paired.

We envision that the following two scenarios would be suitable for scatternet communication and benefit from efficient security solutions:

- **Conference room scenario**: *A large group of participants (more than 8 connected devices) need to form a secure scatternet based on a predetermined shared secret.*
- **Personal Area Network scenario**: *An individual requires that his or her personal devices form secure connections, while still maintaining connectivity to a non-secure public scatternet.*

In the first scenario we assume that the entire scatternet shares a common secret. For example, at a medical center a group of surgeons meet in a conference room. Each surgeon keeps all his or her patient journals in a Bluetooth® enabled Personal Digital Assistant (PDA). If participants want to share confidential medical information, they have to ensure that no adversary outside the room is eavesdropping and can intercept the data. It would not be practical for each surgeon to walk around and securely transfer the journal to each one of his or her colleagues. Therefore, the solution is to form a scatternet that connects all PDAs in the room. In this type of environment they could simply write a password on

the white board and use it as the PIN. Regardless of how the PIN is communicated, the underlying assumption is that the shared secret exists. Note that if an adversary discovers the PIN then the security of the system is compromised. In both proposed security scenarios we show that even though an adversary successfully obtains the PIN and authenticates him- or herself, no old keys are ever disclosed. We maintain PFS even though an adversary could have compromised the scatternet.

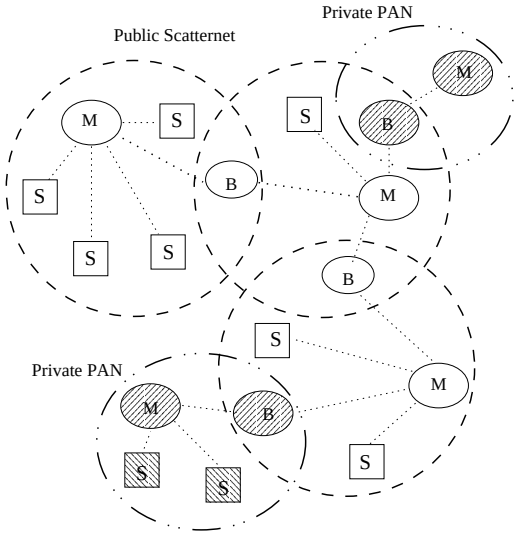


Figure 6.5: Private PAN piconets connected to the scatternet

The second scenario is somewhat different, but also important for Bluetooth® scatternets. As Bluetooth® devices become more and more prevalent, we envision that an individual will carry several Bluetooth® enabled devices. Thus, each individual assumes that his or her devices can communicate securely amongst each other while maintaining connectivity to the rest of the (insecure) scatternet. For example, suppose that our networked person wanted to add funds to his or her digital wallet. In this case the notebook computer, which is equipped with a custom authentication hardware module and has sufficient processing power, connects to an Internet bank though a nearby Bluetooth® access point. Although this part of the transaction is insecure at the Bluetooth® link layer, it should be protected by application level strong authentication and encryption. After successfully completing the transaction, the user wants to transfer the funds to his or her digital wallet, which requires a secure

link layer Bluetooth® connection. The basic idea is that the individual's own devices are connected in a secure piconet, which we call a *Private PAN*. These Private PANs are then connected to the rest of the scatternet through a regular bridge node. We define the *Private PAN* as a secure piconet connected to an insecure scatternet. A picture of a Private PAN as part of a scatternet is shown in Figure 6.5.

Each of the two scenarios described above has different security requirements. In the conference scenario our main objective is to establish secure links between adjacent piconets in secure Extended Scatternet Neighborhoods (ESNs). We assume that knowledge of the fixed PIN is sufficient for membership. Analogous to intra-piconet link key generation, the objective in this case is to establish secure inter-piconet links. In the next section we further describe the rationale behind this idea. On the other hand, in the second case we wish to ensure that an adversary that potentially already belongs to the scatternet is not able to compromise a single secure Private PAN piconet. The Private PAN consists of devices by a single owner or entity. These devices are assumed to have been previously pre-paired and bonded in a non-hostile environment. We further enforce strict point-to-point link keys with the tradeoff that the master is unable to broadcast in the Private PAN piconet. Periodic re-generation of the bondings (meaning combination link keys) is also advisable to curtail cipher attacks.

6.3.2 General Password-based Secure Scatternet

The basic idea behind the creation of secure scatternet links is to first form an ordinary scatternet from devices that share a common secret. Before describing the algorithm in detail we first establish some initial assumptions; each device that participates in the secure scatternet must be *discoverable* and *pairable*[8].

As defined in the Generic Access Profile (GAP)[8] there are three security modes in Bluetooth®. Notably, since security mode 1 lacks both authentication and encryption support, a device in security mode 1 will not be able to authenticate itself and participate in

secure communication. Our security requirements for the conference scenario fit well into **security mode 2**. In security mode 2 authentication and encryption procedures are handled by the link manager after the link has been established but prior to channel establishment. The main difference between security mode 2 and 3 is that in security mode 3 the authentication and encryption procedures are conducted during the connection establishment. For the purpose of the general password-based secure scatternet, the security modes are functionally equivalent, so either mode 2 or 3 can be used.

Phase one: Scatternet formation

The initial phase of the secure scatternet creation utilizes our BTDSP scatternet formation algorithm from Chapter 4. We assume that every participating device possesses the Personal Identificant Number (PIN) a priori to formation. Each device then executes the initialization procedure to probabilistically determine whether to perform device discovery as a master and attempt to form a piconet or as a slave and scan for connections. The initialization procedure utilizes a threshold value, local to each device, that is increased proportionally to the number of slaves in a piconet. Thereby, the probability distribution between master and slave discovery, for disjoint devices, is heavily weighted to slaves. As masters connect more slaves, the threshold is proportionally increased and they attempt to either connect additional slaves or form bridge connections. Our Bluetooth Distributed Scatternet Formation Protocol (BTDSP) forms a flat scatternet topology that is fault-tolerant and does not require that all devices are within transmission range of each other. Fault-tolerance is achieved by ensuring that every device periodically executes the initialization procedure. Slaves do not continue execution, but periodically retry in case they are disconnected. In the same way, new bridge connections are formed to heal potential scatternet partitions as pure slaves become available to the piconet master. More details about scatternet formation are available in Chapters 3 and 4.

After the master has successfully paged a slave and formed a link, the Link Manager (LM) sends an LMP_host_connection_req Packet Data Unit (PDU). This initiates the connection

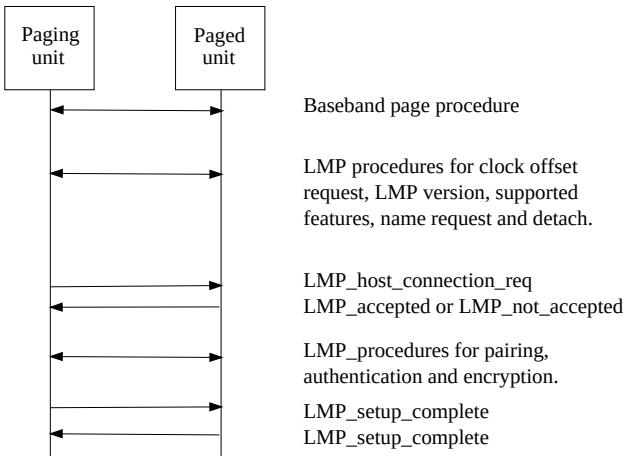


Figure 6.6: Secure connection establishment

setup as well as authentication and encryption procedures. The link establishment procedure is shown in Figure 6.6[5]. Based on our underlying assumption from section 6.2.3, each participating node has a preset PIN. Therefore, devices will fail to form connections if the LMP-authentication and pairing are not successful. The direct consequence from this is that only devices that possess the correct PIN are able to join the scatternet. Therefore, we effectively form a scatternet consisting of only devices with knowledge of the PIN.

Phase two: Inter-piconet link key establishment

After phase one we have formed a scatternet from devices that shared the PIN. Each point-to-point link in the piconets has an associated combination link key. As described in Section 4.3.4, each piconet master in the scatternet stores a table, *Bridge_Table*, that contains the *LT_ADDR* of each bridge node and the *BD_ADDR* of the corresponding piconet master.

We based the key establishment on a password-based key exchange similar to [52]. Our modified algorithm uses the pre-existing E_2 key generation algorithms and the E_1 Message Authentication Code (MAC) algorithm. The generated keys are shown in Figure 6.7 and the algorithm is illustrated in Figure 6.8.

The algorithm works as follows. The initiator A computes its key share SC_K_A from

[5]Illustration derived from Bluetooth® Core v1.2 specification [11] page 209. Copyright 2003 © Bluetooth SIG [30].

The initiator A computes the following quantities:

$SC\_K_A = E_{21}(SC\_RAND_A, BD\_ADDR_A)$
$K_{A\_tmp} = E_{22}(RAND_A, BD\_ADDR_A + PIN)$
$K_{A\_auth} = SC\_K_A \oplus K_{A\_tmp}$
$SRES_A = E_1(SC\_K_A, BD\_ADDR_B, RAND_A)$

$K'_{B\_tmp} = E_{22}(RAND_B, BD\_ADDR_B + PIN)$
$SC\_K_B = K_{B\_auth} \oplus K'_{B\_tmp}$
$SRES'_B = E_1(SC\_K_B, BD\_ADDR_A, RAND_B)$

The respondent B computes the following quantities:

$SC\_K_B = E_{21}(SC\_RAND_B, BD\_ADDR_B)$
$K_{B\_tmp} = E_{22}(RAND_B, BD\_ADDR_B + PIN)$
$K_{B\_auth} = SC\_K_B \oplus K_{B\_tmp}$
$SRES_B = E_1(SC\_K_B, BD\_ADDR_A, RAND_B)$

$K'_{A\_tmp} = E_{22}(RAND_A, BD\_ADDR_A + PIN)$
$SC\_K_A = K_{A\_auth} \oplus K'_{A\_tmp}$
$SRES'_A = E_1(SC\_K_A, BD\_ADDR_B, RAND_A)$

Figure 6.7: Inter-piconet key exchange values

INTER-PICONET-KEY-EXCHANGE()
1 $A \rightarrow B : RAND_A, K_{A\_auth}$
2 $B \rightarrow A : RAND_B, K_{B\_auth}, SRES'_A$
3 $A \rightarrow B : SRES'_B$

Figure 6.8: Inter-piconet authenticated key exchange procedure

a random value SC_RAND_A and the BD_ADDR_A. A first generates a temporary key K_{A_tmp} from another random value $RAND_A$ and the augmentation of its BD_ADDR_A and the PIN. Thereafter, A produces the authentication key: $K_{A\_auth} = SC\_K_A \oplus K_{A\_tmp}$. The authentication key is generated so that B can derive A's key share securely. A also produces a signed response value $SRES_A$ that will be compared to B's challenge-response value $SRES'_A$ for authentication.

In the first step of the algorithm, in Figure 6.8, A sends B the random value $RAND_A$ and the authentication key K_{A_auth}. If B has possession of the correct PIN it can compute

K'_{A_tmp}, since it received $RAND_A$ and knows the BD_ADDR of A. B then computes A's key share: $SC\_K_A = K_{A\_auth} \oplus K'_{A\_tmp}$. B also calculates the signed response $SRES'_A$ to send back to A, for the challenge-response authentication. B then generates its own $RAND_B$ and authentication key K_{B_auth} in the same manner as A. After B sends $RAND_B$, K_{B_auth} and $SRES'_A$ in step 2, A can authenticate B based on $SRES'_A$ and compute B's key share SC_K_B. In the last step A computes B's signed response $SRES'_B$ and sends it back to B.

The inter-piconet shared key can now be computed by both parties as follows:
$SC\_K_{AB} = SC\_K_A \oplus SC\_K_B$.

Each piconet master performs the key exchange procedure for *each* piconet master in its Extended Scatternet Neighborhood (ESN). Thereby all ESN connections can be encrypted. As we describe in Section 4.3.4, each piconet master stores a table *Bridge_Table* that contains the LT_ADDR of each bridge node and the BD_ADDR of the corresponding piconet master. For the purposes of link level security within ESNs, we redefine the table to include an extra column to hold the inter-piconet link key. The new *Bridge_Table* is illustrated in Table 6.2. The master can now encapsulate a packet destined for a neighboring piconet in the payload of an encrypted packet to the bridge node of the corresponding piconet. After decrypting the payload, the bridge node re-encrypts it with the combination link key between itself and the corresponding piconet master. In this manner we can securely exchange data between adjacent piconets.

Table 6.2: Redefined Bridge_Table for link-level scatternet security

LT_ADDR	**Piconet**	**Key**
000	$<BD\_ADDR_{master_A}>$	$< SC\_K_{AX} >$
001	$<BD\_ADDR_{master_B}>$	$< SC\_K_{BX} >$
010	$<BD\_ADDR_{master_C}>$	$< SC\_K_{CX} >$
011	$<BD\_ADDR_{master_D}>$	$< SC\_K_{DX} >$
100	$<BD\_ADDR_{master_E}>$	$< SC\_K_{EX} >$
101	$<BD\_ADDR_{master_F}>$	$< SC\_K_{FX} >$
110	$<BD\_ADDR_{master_G}>$	$< SC\_K_{GX} >$
111	$<BD\_ADDR_{master_H}>$	$< SC\_K_{HX} >$

6.3.3 Private PAN Security

As described in section 6.3.1 the second scenario is fundamentally different from the one described in the previous section. A Private PAN is a secure piconet connected to an insecure scatternet. We first define the requirements for a Private PAN. We have assumed previously that devices in a Private PAN have bonded and exchanged semi-permanent link keys a priori. It is also necessary that each device is in **security mode 3** to authenticate the pre-pairing before the connection is established. In contrast to the secure scatternet model, Private PAN are initially disjoint piconets that form exclusive connections between pre-paired entities. Thus, Private PAN devices are not required to be *discoverable*. However they are required to be *pairable* in order to create the bonding. Further, these devices do not engage in normal device discovery but are immediately paged by the Private PAN master device. This is possible since the bonding provides the master with the Device Access Code (DAC) of each peer device. Private PAN device connections and channel establishment are therefore similar to that of a normal piconet, although authenticated and encrypted at the link level and restricted to pre-paired devices.

The incorporation of Private PANs into the insecure scatternet requires a new link manager command. In 4.3.4 we defined the LMP PDU lmp_scat_inq_scan. It is used exclusively in the master to slave direction to initiate bridge scanning. We can utilize this for Private PANs as well. The Private PAN master designates a bridge designate to reply to inquiries from other piconet masters. Once the bridge node has been connected to a piconet, it returns a lmp_scat_rep PDU to the Private PAN master. This PDU contains the *BD_ADDR* of the corresponding master so that the master can update its *Bridge_Table*. The bridge node is also required to send a reply PDU to the other master as well. For this purpose we define a new PDU lmp_restrict_rep that notifies the corresponding piconet master of the *BD_ADDR* of the Private PAN master. In contrast to the regular response, this PDU notifies the other master's LM that it is connected to a Private PAN and should not attempt to exchange Extended Scatternet Neighborhood (ESN) information or route traffic through this bridge

link.

6.4 Summary

In this chapter we extended Bluetooth® security to scatternets. We also defined two scenarios: the conference room secure scatternet and the Private Personal Area Network (PAN) scenario. The former creates secure links within an ESN, while the latter allows devices to communicate securely within a Private PAN piconet that is connected to an insecure scatternet. We proposed an algorithm for establishing shared link keys between piconet masters and also introduced the concept of Private PAN secure piconets.

Chapter 7

Conclusions and Future Work

7.1 Problems Addressed and Solutions Proposed

In this dissertation we address the problem of scatternet communication. We emphasize that although scatternets are conceptually defined in the Bluetooth® specification there are no algorithms presented within the specification for the four main components of a scatternet communication protocol; namely formation, routing, Inter Piconet Scheduling (IPS), and security.

More specifically, a scatternet must be explicitly formed for a Bluetooth® device to communicate with more than 7 peer devices. A scatternet is essentially made up of piconets, each with a single master and up to 7 slaves, in which some devices, called inter-piconet bridge nodes, interleave their participation in multiple piconets on a time-sharing basis. Coordination between piconet masters (of when and for how long the inter-piconet bridge devices participate in each member piconet) is also necessary and can be done through an Inter Piconet Scheduling (IPS) algorithm. Further, a routing algorithm that is specifically designed to be compatible with both the formation and the IPS algorithms is needed for devices to communicate in the scatternet. As the inter-piconet bridge nodes perform the function of gateways between member piconets, data packets must be routed through the bridges nodes in a store-and-forward manner. Another important problem for scatternet communication is that of protecting data confidentiality and providing security in Bluetooth® scatternets, as wireless ad hoc networks, including Bluetooth® Wireless Personal Area Networks (WPANs),

susceptible to attacks such as eavesdropping, resource exhaustion, and communication interference.

As there are many existing approaches in literature that describe formation of Bluetooth® scatternets, part of our contribution include presenting six scatternet topology models, establishing criteria for efficient formation of scatternets within these topology models, and classifying scatternet formation approaches into single-hop, multi-hop, and optimized topologies.

We present a fault-tolerant scatternet formation algorithm, called Bluetooth Distributed Scatternet Formation Protocol (BTDSP), that forms a 2-Slave/Slave Mesh (SSM) scatternet topology. The main characteristics of BTDSP include: distributed operation, without reliance on a leader or coordinator; use of Slave/Slave (SS) bridge nodes, without any Master/Slave (MS) bridges; bridge degree limited to two participating piconets; support for incremental arrivals and re-incorporation of disconnected nodes, by avoiding protocol phase divisions; and no reliance on artificial merging or migration of piconet nodes. We emphasize that BTDSP operates in both single-hop and multi-hop (where all devices are not within radio vicinity) scenarios, and that due to the distributed nature of the protocol (and without artificial merging and migration of piconet nodes) the scatternet formation delay remains relatively constant as the scatternet size increases.

We also present a hybrid scatternet routing algorithm, called Hybrid Bluetooth Scatternet Routing (HBSR), with a dual meaning of *hybrid*. HBSR is a hybrid zone routing algorithm with a proactive zone, called the Extended Scatternet Neighborhood (ESN). In the reactive region outside the ESN on-demand route discovery is performed using a probabilistic gossiping approach. We utilize a source routing approach in which *modified source routes*, consisting of merely the intermediate piconet masters along the route, are used with bridge connections between piconet masters retrieved from the local ESN routing table. The second meaning of hybrid in HBSR corresponds to the ability to discover a route to either a destination, by its BD_ADDR, or to a device that offers a service, by a service *U*niversal *U*nique

*ID*entifier (UUID). Hybrid Bluetooth Scatternet Routing (HBSR) is also, unlike previous scatternet routing approaches, layered on top of the Logical Link Control and Adaptation Protocol (L2CAP), which simplifies future incorporation of scatternet communication support for higher layer protocols. We emphasize that, due to the topology constraints placed on Bluetooth® scatternets, it is essential that the routing algorithm is directly compatible with both the formation approach and the Inter Piconet Scheduling (IPS) approach. Hybrid Bluetooth Scatternet Routing (HBSR) is designed for scatternet topologies formed by BTDSP and is compatible with both the Maximum Distance Rendezvous Point (MDRP) [56] and the Dichotomized Rendezvous Point (DRP) [70] Rendezvous Point (RP)-based Inter Piconet Scheduling (IPS) algorithms. We note that although probabilistic gossiping, used to reduce the routing load for the reactive part of Hybrid Bluetooth Scatternet Routing (HBSR), can be effective is dense topologies, most scatternet topologies for which Hybrid Bluetooth Scatternet Routing (HBSR) is designed are relatively small and, in most instances, too sparse to take full advantage of the benefits of gossiping without adversely affecting route request propagation coverage.

To extend Bluetooth® security to scatternets we present two scenarios for secure scatternet communication: the conference room scenario, in which all participating inter-piconet links are secured, and the Private PAN piconet, which allows a secure piconet to connect to an insecure scatternet. We specifically extend existing security procedures from the Bluetooth® specification to allow secure exchange and establishment of inter-piconet shared symmetric keys between peer Extended Scatternet Neighborhood (ESN) piconet masters.

7.2 Future Work

As part of our future work we plan to focus on integration of the formation, scheduling, routing, and security components into a single protocol. We also plan to add optimization for newer versions of the Bluetooth® specification. Specifically we aim to further optimize scatternet functionality for use in high-speed Bluetooth® topologies. In addition, we also would

like to incorporate the proposed algorithms into an existing protocol stack, such as BlueZ[18], and utilize a testbed of Bluetooth devices to further improve scatternet functionality.

Specifically for scatternet formation we would like to further improve the topology formation by device classification using the Extended Inquiry Response, so that devices with more processing power and less energy constraints are prioritized for master roles. Further, we strive to provide even tighter integration with the scheduling and routing components to identify and reduce some inefficient topology constructions. In order to improve fault-tolerance, we plan to investigate methods to share stateful information from the piconet master with slaves so that a slaves can easily replace the failed master in the event of node disconnection.

We would like to improve the coordination between the routing and the Inter Piconet Scheduling (IPS) components as to reduce the routing propagation delay for destinations outside the Extended Scatternet Neighborhood (ESN). We also plan to investigate whether different approaches to Inter Piconet Scheduling (IPS) can be used to further reduce propagation delay. In order to lower the overhead from the inclusion of the modified source routes, we strive to investigate implementation of flow IDs that are cached at intermediate nodes. Adding multiple bridge nodes between any two piconets provides multiple routing paths, but it adds also additional switching overhead. We therefore plan to add specific mechanisms for controlling under which circumstances multiple bridge nodes between any two piconets are added. Segmentation of service-based route discovery by service UUID classes is also part of our future work. We also plan to investigate ways, such as the incorporation of a third gossiping threshold value, for further reducing routing overhead while maintaining sufficient topology coverage for route discovery.

To further improve security in Bluetooth® scatternets, we plan to add support for Security Mode 4, for Service Discovery Protocol (SDP) security, as well as investigate incorporation of additional security procedures. We are specifically interested in the ability to authenticate any scatternet peer without reliance on fixed infrastructure or key shares from a large number

of peers, as well as providing end-to-end encryption of data packets using existing Bluetooth® hardware.

Bibliography

[1] *ns2 - Network Simulator*. `http://www.isi.edu/nsnam/ns/`.

[2] A. Aggarwal, M. Kapoor, L. Ramachandran, and A. Sarkar. Clustering algorithms for wireless ad hoc networks. In *Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 54–63, Boston, Massachusetts, USA, August 2000.

[3] A. Capone, M. Gerla, R. Kapoor. Efficient polling schemes for bluetooth picocells. In *IEEE ICC 01, Helsinki, Finland*, volume 7, pages 1990–1994, June 2001.

[4] A. Racz, G. Miklos, F. Kubinszky, and A. Valko. A pseudo random coordinated scheduling algorithm for bluetooth scatternets. In *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 193–203, Long Beach, California, USA, 2001.

[5] S. Basagni and C.Petrioli. Multiphop scatternet formation for bluetooth networks. In *Proceedings of the IEEE 55th Vehicular Technology Conference*, volume 1, pages 424–428, May 2002.

[6] Christian Bettstetter. On the minimum node degree and connectivity of a wireless multihop network. In *MobiHoc '02: Proceedings of the 3rd ACM international symposium on mobile ad hoc networking & computing, New York, NY, USA*, pages 80–91, June 2002.

[7] P. Bhagwat and A. Segall. A routing vector method (RVM) for routing in bluetooth scatternets. In *IEEE International Workshop on Mobile Multimedia Communications (MoMuC '99)*, pages 375–379, 1999.

[8] Bluetooth SIG, `http://www.bluetooth.org/docs/`. *Bluetooth generic access profile*, February 2001.

[9] Bluetooth SIG, `http://www.bluetooth.org/docs/`. *Bluetooth personal area networking profile*, revision 0.95a edition, June 2001.

[10] Bluetooth Special Interest Group, `http://www.bluetooth.org/assigned-numbers/`. *Bluetooth standard assigned numbers*.

[11] Bluetooth Special Interest Group, `http://www.bluetooth.com`. *Bluetooth specification*, February 2001.

[12] J. Bray and C. F. Sturman. *Bluetooth: connect without cables*. Prentice Hall, 2001.

[13] C. E. Perkins, E. M. Belding-Royer, and S. Das. *Ad hoc on demand distance vector (AODV) routing*. IETF Internet Draft, `http://www.cs.ucsb.edu/~ebelding/txt/aodvid.txt`, November 2002.

[14] C. F. Chiasserini, M. A. Marsan, E. Baralis, and P. Garza. Towards feasible topology formation algorithms for bluetooth-based WPANs. In *36th Hawaii International Conference on System Science (HICSS-36), Big Island, Hawaii*, January 2003.

[15] C-J. Huang and W-K. Lai and S-Y Hsiao and H-Y Liu. A self-adaptive zone routing protocol for bluetooth scatternets. *Computer Communications*, 28(1):37–50, January 2005.

[16] C. Law, A. K. Mehta, and K. Siu. Performance of a new bluetooth scatternet formation protocol. In *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc2001)*, Long Beach, California, USA, October 2001.

[17] F. Chun-Choong and C. Kee-Chaing. BlueRings - bluetooth scatternets with ring structures. In *IASTED International Conference on Wireless and Optical Communication (WOC 2002), Banff, Canada*, July 2002.

[18] Open Source Community. BlueZ - official linux bluetooth protocol stack. `http://www.bluez.org`.

[19] F. Cuomo and T.Melodia. A general methodology and key metrics for scatternet formation in bluetooth. In *Proceedings of IEEE GLOBECOM 2002, Taipei, Taiwan*, volume 1, pages 941–945, November 2002.

[20] Cylink Corporation. "SAFER+ encryption algorithm". `http://www.cylink.com/library2/downloadbody.htm`.

[21] D. B. Johnson, D. A. Maltz, Y. Hu, and J. G. Jetcheva. *The dynamic source routing protocol for mobile ad hoc networks (DSR)*. IETF Internet Draft, `http://www.monarch.cs.rice.edu/internet-drafts/draft-ietf-manet-dsr-07.txt`, February 2002.

[22] D. Miorandi, A. Zanella, and G. Pierobon. Performance evaluation of bluetooth polling schemes: an analytical approach. *Mobile Networks and Applications (MONET)*, 9(1):63–72, 2004.

[23] C. de Morais Cordeiro and D. P. Agrawal. *Ad hoc and sensor networks: theory and applications*. World Scientific, 2006.

[24] F. Cuomo, G. Di Bacco, T. Melodia. SHAPER: a self-healing algorithm producing multihop bluetooth scatternets. In *Proceedings of the IEEE Globecom 2003, San Francisco USA*, December 2003.

[25] Frank Stajano and Ross Anderson. "The resurrecting duckling: security issues for adhoc wireless networks". In *Security Protocols, 7th International Workshop Proceedings*, 1999.

[26] G. Tan, A. Miu, J. Guttag, and H. Balakrishnan. Forming scatternets from bluetooth personal area networks. Technical Report MIT-LCS-TR-826, Massachusetts Institute of Techonology, `http://lcs.mit.edu/`, October 2001.

[27] G. Tan, A. Miu, J. Guttag, and H. Balakrishnan. An efficient scatternet formation algorithm for dynamic environments. In *IASTED International Conference on Communications and Computer Networks, Boston, MA*, November 2002.

[28] G. V. Zaruba, S. Basagni, and I. Chlamtac. Bluetrees - scatternet formation to enable bluetooth-based ad hoc networks. In *IEEE International Conference on Communications (ICC2001)*, pages 273–277, 2001.

[29] H. Garcia-Molina. Elections in a distributed computing system. *IEEE Transactions on Computers*, 31(1):48–59, January 1982.

[30] Bluetooth Special Interest Group. Bluetooth SIG member web site. `http://www.bluetooth.org`.

[31] Gy. Miklos, A. Racz, A. Valko, and P. Johansson. Performance aspects of bluetooth scatternet formation. In *Proceedings of the First Annual Workshop on Mobile Ad Hoc Networking and Computing*, pages 147–148, MobiHoc, 2000.

[32] IBM developer Works, IBM Research India, `http://oss.software.ibm.com/bluehoc/`. *Bluehoc: bluetooth network simulator.*

[33] IEEE, `http://grouper.ieee.org/groups/802/15/`. *IEEE 802.15 working group for WPANs.*

[34] IEEE, `http://www.ietf.org/html.charters/manet-charter.html`. *IEEE mobile ad hoc networking (MANet) working group.*

[35] IEEE, `http://standards.ieee.org/getieee802/`. *IEEE 802.11 Standard*, 1999.

[36] J. Yun, J. Kim, Y. Kim, and J. S. Ma. A three-phased hoc network formation protocol for bluetooth systems. In *Proceedings of the 5th International Symposium on Wireless Personal Multimedia Communications (WPMC '2002)*, December 2002.

[37] Jean-Pierre Hubaux, L. Buttyan and S. Capkun. "The quest for security in mobile ad hoc networks". In *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2001)*, 2001.

[38] R. Kapoor, A. Zanella, and M. Gerla. A fair and traffic dependent scheduling algorithm for bluetooth scatternets. *Mobile Networks and Applications*, 9(1):9–20, February 2004.

[39] L. Har-Shai, R. Kofman, G. Zussman, and A. Segall. Inter-piconet scheduling in bluetooth scatternets. In *Proceedings of the OPNETWORK 2002 Conference*, August 2002.

[40] L. Li, J. Halpern, and Z. Haas. Gossip-based ad hoc routing. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, June 2002.

[41] X-Y. Li, K. Moaveninejad, and O. Frieder. Regional gossip routing for wireless ad hoc networks. *Mobile Networks and Applications*, 10(1-2):61–77, February 2005.

[42] Lidong Zhou and Zygmunt J. Haas. "Securing ad hoc networks". *IEEE Network Magazine*, 16(6):24–30, November/December 1999.

[43] M. Kalia, D. Bansal, R. Shorey. Data scheduling and SAR for bluetooth MAC. In *Proceedings of the IEEE 51st Vehicular Technology Conference Proceedings (VTC 2000-Spring) Tokyo, Japan*, volume 2, pages 716–720, May 2000.

[44] M. Kalia, S. Garg, and R. Shorey. Scatternet structure and inter-piconet communication in the bluetooth system. In *IEEE National Conference on Communications New Dehli, India, 2000*, 2000.

[45] M. Sun, C. Chang and T. Lai. A self-routing topology for bluetooth scatternets. In *Proceedings of I-SPAN 2002, Manila, Philippines*, May 2002.

[46] Markus Jakobsson and Susanne Wetzel. "Security weaknesses in bluetooth". In *Proceedings of the RSA Conference*, 2001.

[47] Martina Umlauft and Peter Reichl. "Experiences with the ns-2 network simulator - explicitly setting seeds considered harmful". In *5th IEEE International Conference on Industrial Informatics*, Pomona, California, April 2007. ACM.

[48] J. Misic and V. B. Misic. Bridges of bluetooth county: topologies, scheduling and performance. In *IEEE Journal of Selected Areas in Communications*, volume 21 of *Wireless Series, Special issue on Wireless LANs and Home Networks*, pages 240–258, February 2003.

[49] MIT Laboratory for Computer Science : Networks and Mobile Systems, `http://nms.lcs.mit.edu/projects/blueware`. *Blueware: bluetooth simulator for ns.*

[50] Robert Morrow. *Bluetooth operation and use.* McGraw-Hill, 2002.

[51] C. Siva Ram Murthy and B.S. Manoj. *Ad hoc wireless networks: architecture and protocols.* Prentice Hall, 2004.

[52] N. Asokan and P. Ginzboorg. "Key agreement in ad-hoc networks". In *Proceedings of Nordsec'99*, November 1999.

[53] N. Golmie, N. Chevrollier, and I. ElBakkouri. Interference aware bluetooth packet scheduling. In *Proceedings of IEEE GLOBECOM*, pages 2857–2863, November 2001.

[54] N. Johansson, F. Alriksson, and U. Jonsson. JUMP mode - a dynamic window-based scheduling framework for bluetooth scatternets. In *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Long Beach, California, USA, October 2001.

[55] P. Johansson, M. Kazantzidis and M. Gerla. Bluetooth: an enabler for personal area networking. *IEEE Network*, 15(5):28–37, September/October 2001.

[56] P. Johansson, R. Kapoor, M. Kazantzidis, M. Gerla. Rendezvous scheduling in bluetooth scatternets. In *Proceedings of ICC 2002, New York City, New York*, pages 318–324, April 2002.

[57] P. Wang, K. M. Alzoubi, and O. Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. In *Proceedings of IEEE INFOCOM 2002, New York, NY*, pages 1597–1604, July 2002.

[58] K. Pearson. *Tables of incomplete beta functions*. Cambridge University Press, 2nd edition, 1968.

[59] K. Persson and D. Manivannan. A fault-tolerant distributed formation protocol for bluetooth scatternets. *International Journal of Pervasive Computing and Communications (JPCC)*, 2(2):165–176, June 2006.

[60] K. E. Persson, D. Manivannan, and M. Singhal. Bluetooth scatternets: criteria, models and classification. *Ad Hoc Networks*, 3(6):777–794, November 2005.

[61] C. Petrioli and S. Basagni. Degree-constrained multihop scatternet formation for bluetooth networks. In *Proceedings of the IEEE Globecom 2002, Taipei, Taiwan*, November 2002.

[62] B. J. Prabhu and A. Chockalingam. A routing protocol and energy efficient techniques in bluetooth scatternets. In *IEEE ICC'2002, New York, NY*, pages 3336–3340, April-May 2002.

[63] R. Kapoor and M. Gerla. A zone routing protocol for bluetooth scatternets. In *IEEE Wireless Communications and Networking (WCNC'03)*, pages 1459–1464, March 2003.

[64] S. Baatz, C. Bieschke, M. Frank, C. Kuhl, P. Martini, and C. Scholz. Building efficient bluetooth scatternet topologies from 1-factors. In *Proceedings of the IASTED International Conference on Wireless and Optical Communications, WOC 2002, Banff, Alberta, Canada*, July 2002.

[65] I. Stojmenovic. Dominating set based bluetooth scatternet formation with localized maintenance. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS '02)*, pages 148–155. IEEE Computer Society, April 2002.

[66] T. Salonidis, P. Bhagwat and L. Tassiulas. Proximity awareness and fast connection establishment in bluetooth. In *First Annual Workshop on Mobile and Ad Hoc Networking and Computing, 2000. MobiHOC 2000*, pages 141–142, 2000.

[67] T. Salonidis, P. Bhagwat, L. Tassiulas, and R. LaMaire. Distributed topology construction of bluetooth personal area networks. In *Proceedings of Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM 2001)*, volume 3, pages 1577–1586, April 2001.

[68] T. Y. Lin, Y. Tseng, K. Chang, and C. Tu. Formation, routing, and maintenance protocols for the BlueRing scatternet of bluetooths. In *Proceedings of the 36th Hawaii International Conference of System Sciences, Big Island, Hawaii*, January 2003.

[69] University of Cincinnati, `http://www.ececs.uc.edu/~cdmc/ucbt/`. *UCBT - bluetooth extension for ns2*.

[70] Q. Wang and D. P. Agrawal. A dichotomized rendezvous algorithm for mesh bluetooth scatternets. *Ad Hoc Sensor Wireless Networks*, 1(1):65–88, March 2005.

[71] J. P. F. Willekens. Ad hoc routing in bluetooth. In *Proceedings of the 6th International Conference on Protocols for Multimedia Systems*, pages 130–144. Springer-Verlag, 2001.

[72] J. Wu and H.L Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proceedings of the 3rd ACM international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 7–14, August 1999.

[73] Y. Liu, M. J. Lee, T. N. Saadawi. A bluetooth scatternet-route structure for multihop ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 21(2):229–239, February 2003.

[74] Z. Wang, R. J. Thomas, and Z. Haas. Bluenet - a new scatternet formation scheme. In *35th Hawaii International Conference on System Science (HICSS-35), Big Island, Hawaii*, January 2002.

[75] W. Zhang and G. Cao. A flexible scatternet-wide scheduling algorithm for bluetooth networks. In *IEEE International Performance, Computing, and Communications Conference (IPCCC)*, 2002.

[76] Z.J. Haas and M. R. Pearlman. *The zone routing protocol (ZRP) for ad hoc networks*. IETF Internet Draft, `http://wnl.ece.cornell.edu/Publications/draft-ietf-manet-zone-zrp-02.txt`, June 1999.

[77] S. Zurbes. Considerations on link and system throughput of bluetooth networks. In *11th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2000)*, volume 2, pages 1315–1319, 2000.

[78] G. Zussman and A. Segall. Capacity assignment in bluetooth scatternets - optimal and heuristic algorithms. *ACM/Kluwer Mobile Networks and Applications (MONET)*, Special Issue on Advances in Research of WPAN and Bluetooth Enabled Networks, 2003.

Acronyms

ACL Asynchronous Connectionless

AFH Adaptive Frequency-Hopping Spread Spectrum

AODV Ad hoc On-demand Distance Vector

BIAS Bluetooth Interference Aware Scheduling

BNEP Bluetooth Network Encapsulation Protocol

BTCP Bluetooth Topology Construction Protocol

BTDSP Bluetooth Distributed Scatternet Formation Protocol

CDMA Code Division Multiple Access

CG Communicating Group

CDS Connected Dominating Set

CL Connection-less

CO Connection-oriented

DAC Device Access Code

DH Data - High Rate

DIAC Dedicated Inquiry Access Code

DM Data - Medium Rate

DRP Dichotomized Rendezvous Point

DRPB Dichotomized Rendezvous Point Broadcasting

DS Direct Sequence

DSSS Direct Sequence Spread Spectrum

DSR Dynamic Source Routing

DV Data/Voice Bluetooth packet type

DUN Dial-up Networking

EDR Enhanced Data Rate

EID Extended ID

ERR Exhaustive Round-Robin

ESN Extended Scatternet Neighborhood

FEC Forward Error Correction

FH Frequency Hopping

FHSS Frequency Hopping Spread Spectrum

FHS Frequency Hopping Sequence

FSM Finite State Machine

FSS Flexible Scatternet-wide Scheduling

GAP Generic Access Profile

GFSK Gaussian Frequency Shift Keying

GIAC General Inquiry Access Code

GN Group Ad Hoc Network

HBSR Hybrid Bluetooth Scatternet Routing

HCI Host Controller Interface

HV High-quality voice Bluetooth packet type

ISM Industrial, Scientific, Medical

IAC Inquiry Access Code

IPS Inter Piconet Scheduling

IEEE Institute of Electrical and Electronics Engineers

IRPS Intra Piconet Scheduling

IR Infrared

KFP K-Fairness Scheduling Policy

L2CAP Logical Link Control and Adaptation Protocol

LAA Load Adaptive Algorithm

LAN Local Area Network

LC Link Controller

LFSR Linear Feedback Shift Register

LM Link Manager

LMP Link Manager Protocol

LSB Least Significant Bit

LIAC Limited Inquiry Access Code

LM Link Manager

LMP Link Manager Protocol

LWRR Limited and Weighted Round Robin

MAC Message Authentication Code

MANET Mobile Ad Hoc Network

MB-OFDM Multi-Band Orthogonal Frequency Division Multiplexing

MBps Megabytes Per Second

Mbps Megabits Per Second

MDRP Maximum Distance Rendezvous Point

MIS Minimum Independent Set

MS Master/Slave

MSM Master/Slave Mesh

MSR Master/Slave Ring

NAP Network Access Point

PAN Personal Area Network

PCSS Pseudo-Random Coordinated Scatternet Scheduling

PDA Personal Digital Assistant

PDU Packet Data Unit

PIN Personal Identificant Number

PFS Perfect Forward Secrecy

PP Priority Polling

PRR Pure Round Robin

PSK Phase Shift Keying

RNG Random Number Generator

RB Random Backoff

RF Radio Frequency

RP Rendezvous Point

RVM Routing Vector Method

RSSI Received Signal Strength Indicator

SCO Synchronous Connection-Oriented

eSCO Extended Synchronous Connection-Oriented

SDP Service Discovery Protocol

SPM Single Piconet Model

SS Slave/Slave

SSM Slave/Slave Mesh

SSR Slave/Slave Ring

SIG Special Interest Group

TDD Time Division Duplex

TDMA Time Division Multiple Access

TH Tree Hierarchy

TSF Tree Scatternet Formation Protocol

TTL Time To Live

UCBT University of Cincinnati - BlueTooth

UUID *U*niversal *U*nique *ID*entifier

UWB Ultra-Wideband

QoS Quality of Service

WG Working Group

WLAN Wireless Local Area Network

WPAN Wireless Personal Area Network

ZRP Zone Routing Protocol

Index

Vita

Karl E. Persson
Date of Birth: 05/03/1978
Place of Birth: Ludvika, Sweden

- **EDUCATION**
  - M.S., Computer Science, August 2005. University of Kentucky
  - B.S., Computer Science, May 2001. University of Kentucky
- **CERTIFICATIONS**
  - Certified Six Sigma Black Belt #6878, March 2008. American Society for Quality (ASQ)
  - Certified Six Sigma Green Belt #1048, June 2007. American Society for Quality (ASQ)
- **PUBLICATIONS**
  - **Journal publications:**
    - **Persson, K.**, and D. Manivannan "A fault-tolerant distributed formation protocol for bluetooth scatternets", International Journal of Pervasive Computing and Communications, Volume 2, Issue 2, June 2006, pp. 165-176
    - **Persson, K.E.**, D. Manivannan, and M. Singhal. "Bluetooth scatternet formation: criteria, models and classification", Ad Hoc Networks, Volume 3, Issue 6, November 2005, pp. 777-794, Elsevier.
  - **Conference publications:**
    - **Persson, K.**, and D. Manivannan. "Hybrid bluetooth scatternet routing", UIC '09: Proceedings of the 6th International Conference on Ubiquitous Intelligence and Computing. 7 -10 July, 2009, Brisbane, Australia.
    - **Persson, K.**, and D. Manivannan. "Distributed self-healing bluetooth scatternet formation", Proceedings of the 2004 International Conference on Wireless Networks. June 21-24, 2004, Las Vegas, NV, USA.
    - **Persson, K.**, D. Manivannan, and M. Singhal. "Bluetooth scatternet formation: criteria, models, and classification", Proceedings of the 1st Annual IEEE Consumer Communications and Networking Conference, 2004, January 5-8, 2004, Las Vegas, NV, USA.

* **Persson, K.E.**, and D. Manivannan. "Secure connections in bluetooth scatternets", Proceedings of the 36th Annual Hawaii International Conference on System Sciences, 2003. January 6-9 2003, Big Island, HI, USA.

Karl E. Persson