Multi-hop Wireless Relay Using Bluetooth Specification

By

Richard K. Moy

Bachelor of Science in Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2001

Submitted To The Department Of Electrical Engineering And Computer Science In Partial Fulfillment Of The Requirements For The Degree Of

MASTER OF ENGINEERING
IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
AT THE
MASSACHUSETTS INSTITUE OF TECHNOLOGY

SEPTEMBER 2002

Author _____
Department of Electrical Engineering and Computer Science

Certified by _____
Professor Robert Morris
Assistant Professor, Massachusetts Institute of Technology
Thesis Advisor

Accepted by _____
Professor Arthur Smith
Chairman, Department Committee on Graduate Theses

Multi-Hop Wireless Relay Using Bluetooth Specification

by

Richard Moy

Submitted to the
Department of Electrical Engineering and Computer Science

September 18, 2002

In Partial Fulfillment of the
Requirements for the Degree of
Masters of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis presents BlueRelay, a novel protocol that extends the Bluetooth specification to support multi-hop wireless communication. Bluetooth is a wireless communication protocol originally designed for point-to-point communication within a small network called a piconet, where a master has centralized coordination over slave devices. The purpose of BlueRelay is to enable inter-piconet communication using existing Bluetooth mechanisms without any modifications to the Bluetooth link layer and its medium access control protocol. BlueRelay is a two-part protocol. First, the relay establishment process details how a slave device establishes communications with two master devices and acts as a forwarding node between two piconets. Second, the piconet switching scheme employs a Bluetooth mechanism called the HOLD mode to enable a slave to switch back and forth between two piconets. These two techniques allow Bluetooth devices to forward packets over multiple hops and beyond the communication range of the source's transmitter. BlueRelay is simulated in a Bluetooth simulator that was developed by IBM to simulate the Bluetooth protocol stack. By simulating the relay establishment process and the piconet switching scheme in the Bluetooth simulator, we demonstrate that multi-hop wireless communication can be achieved using the existing mechanisms of the Bluetooth specification. Simulation results show that relay establishment occurs under 10 seconds and a maximum of 300 kbps average throughput can be achieved for multi-hop wireless communication. End-to-end latency ranges from 15ms per hop to 300ms per hop, depending on the HOLD period.

Faculty Thesis Supervisor: Robert Morris
Title: Assistant Professor, Massachusetts Institute of Technology

ACKNOWLEDGEMENTS

Thank you to Dorothy Poppe, my supervisor at Draper Laboratory, for providing research guidance. Your advice has been invaluable in improving the quality of my work. I thank Professor Morris, my thesis advisor, for his time and patience and for his helpful suggestions and guidance.

I want to thank all my closest friends. You know who you are. Our times at Next House, Baker House, parties and formals, the weightroom, the Chinese Student Club, the Lion Dance Team, the Kennedy Biscuit Lofts, and in and around Boston are experiences that I will forever cherish.

I thank my wonderful sisters, Patricia and Cindy. You two are always there to listen to me, laugh with me, and keep me in check. Most importantly, I thank my parents. I sincerely appreciate all the sacrifices you have made throughout the years. Thank you for your endless love, support, and understanding in everything that I do.

TABLE OF CONTENTS

This page intentionally left blank.

LIST OF FIGURES

LIST OF TABLES

This page intentionally left blank.

Chapter 1

Introduction

The purpose of this thesis is to investigate the performance of a multi-hop wireless relay using the Bluetooth standard. Bluetooth is a single-hop wireless communication protocol designed for small networks called piconets. The challenge is to design techniques for inter-piconet communication using mechanisms defined in the Bluetooth specification without modifying the Bluetooth link layer or the medium access control protocol. This thesis presents BlueRelay, a novel protocol to enable inter-piconet communication. The techniques used to establish forwarding nodes and support inter-piconet communication are simulated in a Bluetooth simulator to evaluate the performance of a multi-hop wireless relay.

1.1 Motivation

The Bluetooth standard, which is designed specifically for short range, low-power wireless communication, has a medium access control (MAC) protocol that facilitates the construction of small

networks called piconets. A piconet is a centralized network controlled by a master node, which allocates transmission slots to all other nodes (slaves) in the piconet using a time-division duplex scheme. Bluetooth also employs a frequency-hopping sequence that allows multiple small networks to operate without interference. The FH sequences make it possible for the internetworking of multiple piconets. In the Bluetooth specification, however, the process of establishing and maintaining inter-piconet communication is not described in detail. Therefore, this thesis presents a protocol to enable inter-piconet communication using defined Bluetooth mechanisms without any modification to the Bluetooth Specification.

1.2 Related Work

Significant research has already been done with Bluetooth, particularly to extend the Bluetooth specification to establish internetworking piconets, called scatternets. Previous research includes evaluating scatternet size and scatternet formation algorithms.

The IBM India Research Laboratory has devised a clustering algorithm for the formation of scatternets in "Clustering Algorithms for Wireless Ad Hoc Networks." [10] The proposed algorithm is a two stage randomized algorithm that determines the minimum number of star-shaped clusters that are of maximum size.

In another research on "Proximity Awareness and Fast Connection Establishment in Bluetooth," the study discusses techniques for quickly establishing connections between a master and a slave. [11] The research shows that the time elapse in forming a connection between a pair of nodes can be reduced.

These research efforts show that scatternet formation is viable. However, scatternets are applicable only if they can scale well without interference. "Radio Network Performance of Bluetooth" studies the number of piconets that can be handled with acceptable performance. [12] Simulations were

donc with voice traffic to show that multiple piconets could be supported in the same area without performance degradation due to interference.

In another study, "Short Range Radio Based Ad-hoc Networking: Performance and Properties" simulated the performance of data and voice traffic within a Bluetooth piconet. [13] The piconets were placed under high load and bursty traffic conditions. It was concluded that Bluetooth piconets are capable of sharing both voice and data traffic in a single communication medium without performance degradation even under stressful load conditions. However, there are no published studies on the performance of traffic in scatternets.

Furthermore, simulations in past research do not model the layers of the Bluetooth communication protocol stack, which consists of the Baseband, Link Controller, and Link Manager. Previous simulations, done in Matlab and Java, do not take into account the details of the Bluetooth specification, such as inquiry and paging trains, backoff timers, slot sizes, and the maintenance of clock offsets. In this thesis, simulations are done in *ns-2* using an extension that implements the layers of Bluetooth.

1.3 Problem Description

The goal of this thesis is to propose a relay establishment process and piconet switching scheme for inter-piconet communication. The Bluetooth medium access control protocol is designed to employ a master-slave mechanism instead of distributed contention resolution as seen in the ALOHANET project and other radio packet networks. Therefore, communication in Bluetooth follows a strict master-slave scheme. Due to this restriction set by the Bluetooth specification, master-to-master and slave-to-slave communication is prohibited. Therefore, the path of a packet must alternate between master and slave nodes when traversing multiple nodes. An obvious starting point is to choose certain nodes to participate as relays to forward data.

1.3.1 Relay Establishment

While the basic idea is simple, there are challenging issues. First, there is a need for a relay establishment process where the master and slave devices connect to each other to form a single-chain relay of alternating master and slave nodes. This establishment process requires a slave node to establish connections with two masters, which is not a requirement in the Bluetooth specification. In fact, a slave that acts as a relay node decentralizes the communication coordination. [23] Master nodes no longer have centralized control of communication. In BlueRelay, slaves as well as masters are responsible for forwarding data.

1.3.2 Piconet Switching

Once the relay is formed, scheduling the transmission of data packets is an issue due to the time-division duplex property of Bluetooth that is not seen in traditional wireless channel scheduling. The original intent of the time-division duplex property was to eliminate contention for point-to-point communication between Bluetooth-enabled devices. The time-division duplex property, however, prevents a relay slave unit to simultaneously participate in two piconets. Instead, the relay slave unit must switch back and forth between two piconets. This requires a piconet switching scheme where the relay slave unit schedules when to leave one piconet to participate in another piconet.

1.4 Thesis Outline and Approach

In this thesis, BlueRelay is presented as a solution to enable inter-piconet communication over multiple hops. The relay establishment process and piconet switching scheme for Bluetooth devices are presented and evaluated.

First, Chapter 2 describes the basics of Bluetooth and how Bluetooth units establish point-to-point connections. The process of inquiry and paging is explained to illustrate how a master and a slave

form a communication link. Frequency-hopping and time-division duplex are discussed in detail to show how devices share the communication medium without interference or contention.

In Chapter 3, the protocol BlueRelay is presented. The relay establishment process to connect master and slave nodes in a linear relay is described. The BlueRelay scheme uses Bluetooth's mechanisms for detecting potential nodes and establishing links along a predetermined relay. After establishing a multi-hop relay, relay units follow a piconet switching scheme to control piconet switching on a periodic basis to forward data packets.

Chapter 4 presents the results and analysis of simulating relay establishment and piconet switching in the network simulator, *ns-2*. Using extensions to *ns-2* that IBM developed, Bluetooth multi-hop wireless relays were simulated in *ns-2*. The relay length, switching parameter, and packet size were varied to determine the impact on the relay establishment time, average throughput, and end-to-end latency for data packets.

Finally, Chapter 5 summarizes the project and discusses the contribution of the work towards extending the Bluetooth standard for multi-hop wireless communication. Possible future work is also discussed.

This page intentionally left blank.

Chapter 2

Bluetooth Specification

This chapter provides an overview of Bluetooth mechanisms that are leveraged to form a multi-hop relay. The Bluetooth mechanisms that are presented here are referenced from the Bluetooth Specification Book, which is copyrighted by the Bluetooth Special Interest Group. [1]

The original intent of Bluetooth was to connect accessories to mobile phones. Besides mobile phones, Bluetooth was developed to replace cables for devices in the home and office such as headsets, laptops, printers, and PDAs. [2] Of the current wireless standards that exist today, Bluetooth has the lowest transmitting power of 1mW with a range of 10 meters.

In Figure 2-1, the layers of the Bluetooth Protocol Stack are compared to with the Open Systems Interconnect (OSI) standard reference model. The Radio and Baseband layers of Bluetooth perform frequency modulation and demodulation for the transmission and reception of packets over the air. The Baseband and Link Control are responsible for framing, error checking, and correction of packets. The Link Control and Link Manager together set up, maintain, and tear down links. The Logical Link Control and Adaptation Protocol (L2CAP) provide management and control data flow at the host level. The L2CAP segments and reassembles the data into smaller pieces that fit into the

maximum baseband packet payload. The levels below the L2CAP layer are collectively the Bluetooth module.

The L2CAP and RFCOMM layers hide the complexities of the Bluetooth module from the application. The middle layers can accept many familiar data formats and protocols, package them, multiplex them, and pass them to the lower layers in a manner that matches the lower layers' capabilities. Thus, the management of over-the-air transmissions is hidden from the application by the abstraction of layers.

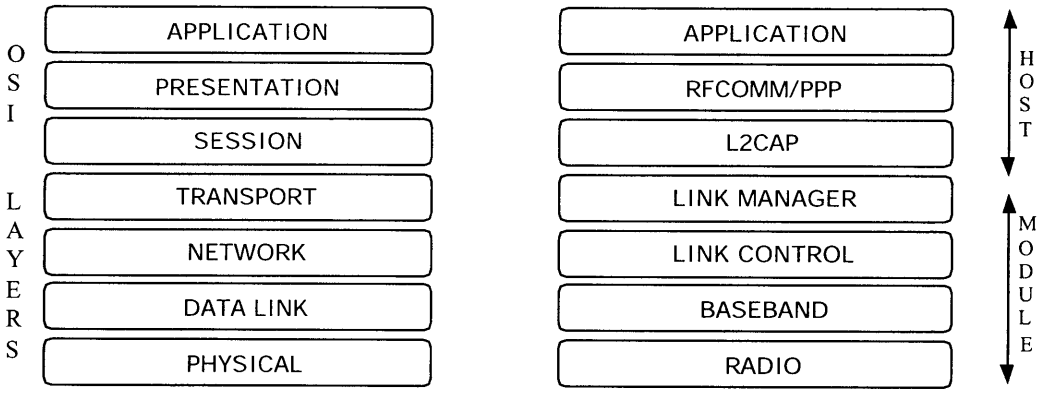

Figure 2-1. Comparison of OSI Layers to the Bluetooth Protocol Stack. [8]

For the remainder of the thesis, the Bluetooth Procotol Stack will be referenced to illustrate which layer is responsible for carrying out certain functions.

2.1 Piconet Description

The basic structure for a Bluetooth network is a piconet, a simple one-hop star network which contains one master unit and up to seven active slave units. Although any Bluetooth-enabled device can take on either role, there can be only one master in a piconet at any given time. The simplest piconet is two Bluetooth-enabled devices within radio range, and having one as the master and the other as a slave as shown in Figure 2-2.

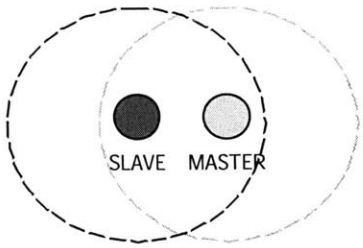

Figure 2-2. The simplest piconet contains one master and one slave that are within radio range of each other.

The master controls the communication of at most seven "active" devices on a single shared 1-Mbps communication link by having all the slaves in its piconet synchronized to the master's clock. Figure 2-3 shows a piconet in a star configuration where the slaves are distributed around the master unit.

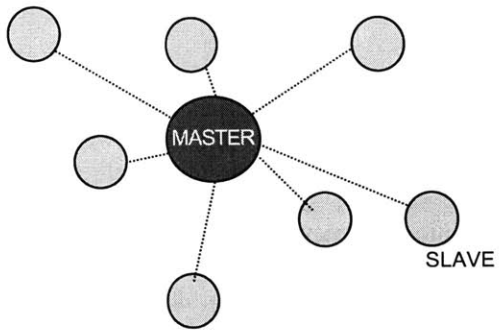

Figure 2-3. An example piconet in star configuration. This is a full piconet with the master controlling seven active slaves. [2]

Slave devices in a piconet are assumed to be within radio range with the master device. As shown in Figure 2-4 when two slave devices are out of range of each other, they can communicate through the master device. In effect, communication between two slave devices via a master device is a two-hop relay. Even if the locations of slave units are within radio range with each other, the strict master-slave mechanism does not allow slaves to directly communicate. The slaves can only communicate directly with a master, not to any other slave. Hence, the master is always in full control of the piconet.

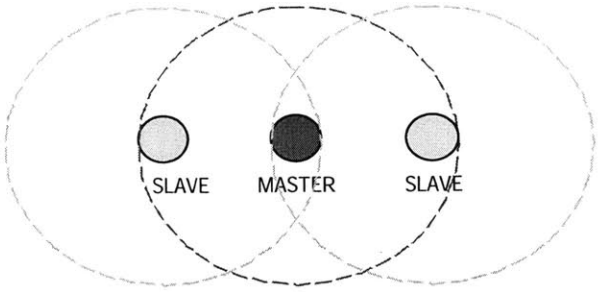

Figure 2-4. Two slave devices that are out of radio range of each other can still communicate via a master device. Note both slave devices must be within radio range of the master device.

A maximum of eight devices can be active in a piconet at once. Interference among devices in the same piconet is prevented by the time-division duplex property. To understand how contention is prevented between a master and its slave devices within a piconet, the time-division duplex (TDD) property is discussed in Section 2.2.

Multiple piconets can co-exist in the same area without inference due to Bluetooth's frequency-hopping scheme. The frequency-hopping scheme requires the Bluetooth device to spend very little time transmitting on any specific frequency. Hence, interference among piconets is minimized since the amount of time spent transmitting on any particular frequency is small. To understand the co-existence of multiple piconets in the 2.4 GHz range, an explanation of frequency-hopping spread spectrum is given in Section 2.3.

Section 2.4 describes the link establishment process between a master and a slave, which can be extended to establish multi-hop relays.

2.2 Time-Division Duplex for Shared Communication Channel

Time-division duplex (TDD) is the basis of the Bluetooth's medium access control protocol. TDD divides the communication into time slots, such that the even-numbered time slots are reserved for master transmissions and the odd-numbered time slots are reserved for slave transmissions. The TDD design avoids packet collisions within a piconet by giving the master centralized control. The

master is responsible for allocating transmission slots, effectively channel bandwidth, to the slaves in the piconet. After the master sends data or poll packet over the forward link, the subsequent slot is reserved for the slave to transmit data in the reverse link. [22] This poll-response policy restricts the slave "to speak when spoken to". [16] Thus a slave can never talk over the master, nor will slaves talk over each other.

As shown in Figure 2-5, the TDD requires the master unit to start its transmissions in even-numbered time slots only. The slaves can respond in odd-numbered time slots only. Each time slot is 625μs in length. With a constant bit rate transmission time of 1μs, each slot accommodates a maximum of 625 bits, or 78.125 bytes.

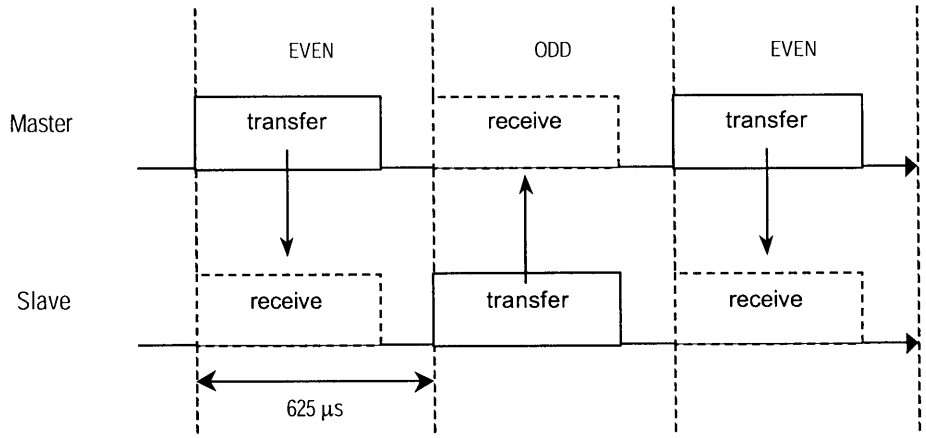

Figure 2-5. The time-division duplex scheme allocates even time slots for masters to transmit and reserves the odd time slots for the slave to transmit. A master and slave alternate transmit and receive slots to prevent contention. [1]

The TDD scheme makes multiple access straightforward. Figure 2-6 shows how a master and its slaves share a single TDD channel. The master provides a single point of coordination. The packet start is aligned with the slot start. Packets transmitted by the master or the slave may extend over a maximum of five time slots. The first two pairs of master-slave transmission packets are the poll-response packets between the Master and Slave 1. The third pair of master-slave transmission packets is packets transmitted between the Master and Slave 3. The fourth pair of master-slave transmission

packets is packet transmissions between the Master and Slave 2. Whenever a master transmits a packet to a slave in the piconet, the slave must respond in the following slot with an acknowledgement packet or a data packet. A slave cannot ignore a packet from a master unless the master sends a null packet, which requires no acknowledgement in return.

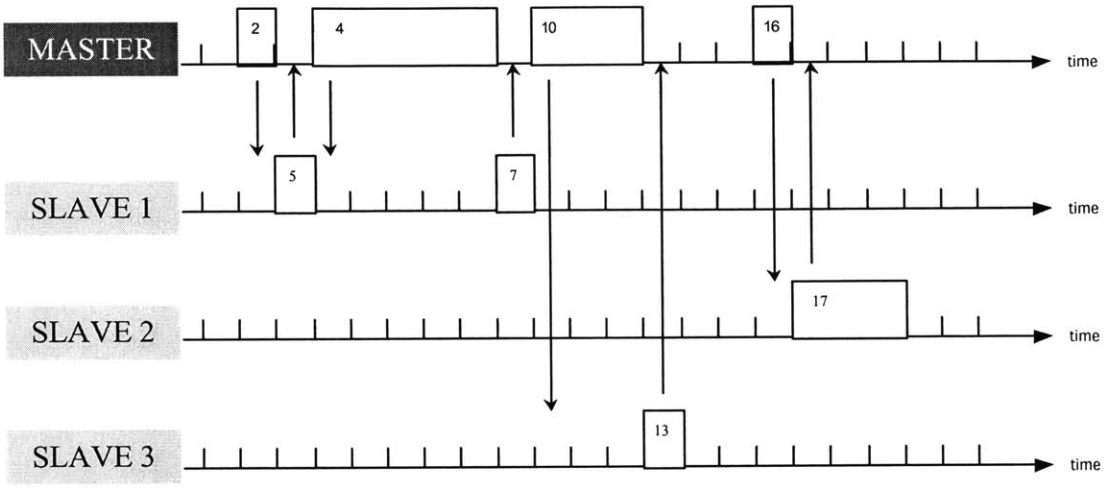

Figure 2-6. An example of a master managing three slaves in a time-division duplex channel. Only one slave at a time can communicate with the master. [22]

2.3 Frequency-Hopping Spread Spectrum

The Bluetooth transceiver is a frequency-hopping spread spectrum (FHSS) radio system operating over 79 1-MHz channels in the 2.4 GHz frequency spectrum. This discussion on how FHSS works is based on *Spread Spectrum Systems* by Robert Dixon. [9]

The use of frequency-hopping spread spectrum helps to reduce interference among piconets. The essence of spread spectrum communication is the expansion of the bandwidth of a signal, transmitting that expanded signal, and recovering the desired signal by remapping the received spread spectrum into the original information bandwidth. [9] By transmitting on a wide-band spread spectrum

signal, the signal appears to be noise-like signals. This is advantageous to avoid detection and interception.

The intent of the pseudo-random frequency-hopping pattern is to avoid interfering signals by not spending very much time on any specific frequency. In Figure 2-7, a transmitter hops between available frequencies according to a pseudo-random algorithm. The device transmits on one frequency for a certain period of time, then hops to another frequency and transmits again.

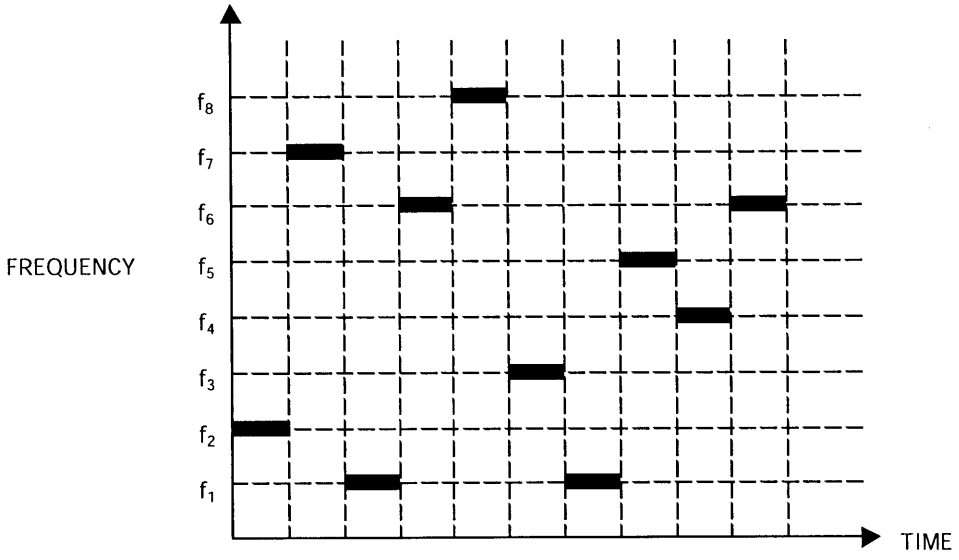

Figure 2-7. A frequency-hopping sequence changes frequency every time slot. Note only one frequency is employed at given time slot. [9]

If interference is present on any of the channels in the hopping pattern, the RF signal will experience interference when transmitting on that jammed frequency. As shown in Figure 2-8, such interference will be minimized by the small amount of time spent transmitting on the jammed frequency.

Let us consider a situation where there are three Bluetooth piconets in the same area operating under three unique frequency-hopping sequences. Since there are 79 frequencies available in Bluetooth and three channels are used at any given time, the probability of a collision is 3/79, or 0.038. [9]

In the case that two hopping patterns happen to transmit using the same frequency channel at the same time instance, that short period of data will be lost. However, such data losses will be minimal, as the transceivers will hop to another frequency in their unique sequence.

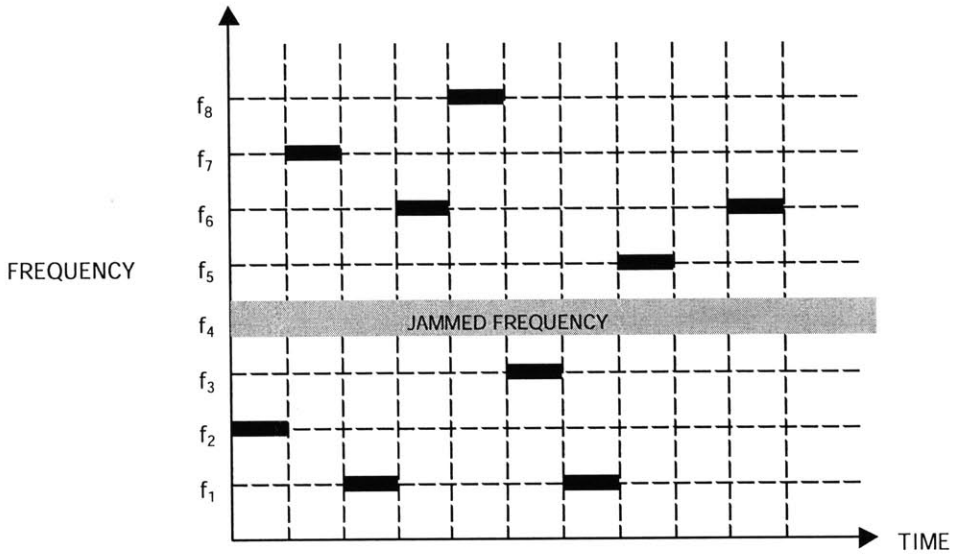

Figure 2-8. An example of a jammed frequency. As the frequency-hopping sequence pseudo-randomly hops among the 79 frequencies, transmissions on frequency f_4 will experience interference. Interference is momentary, however, as another frequency is used in the following time slot. [9]

Each frequency-hopping sequence is unique because the sequence is derived by the device address of the master. The phase in the hopping sequence is determined by the Bluetooth clock of the master. [1] Since each Bluetooth device address has a unique 48-bit address, the hopping pattern and phase is unique for each piconet.

During data transmission, the frequency-hopping sequence changes its transmission frequency each time a packet is transmitted. Since there are three different types of packet sizes, the transmission frequency can change every time slot or every five time slots.

In Figure 2-9, the frequency-hopping sequence is represented by

$$f[k] = 2402 + n\ MHz, \tag{2.1}$$

where n = 0 to 78. Thus, for the duration of one time slot, a single slot packet is transmitted on one frequency. For the case of a multi-slot slot packet, the frequency utilized will be the frequency as applied in the time slot where the packet transmission was started. [26]

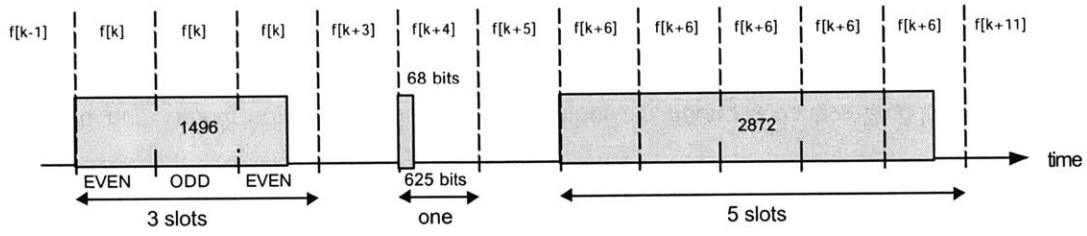

Figure 2-9. An example transmission of single-slot and multi-slot packets by a master. Note all packet transmission start on even time slots. [26]

For the 5-slot packet in Figure 2-9, the entire packet will be transmitted on frequency *f[k+6]*, the frequency the packet transmission had started.

The frequency-hopping sequence for data transmission is specifically referred to as the channel-hopping sequence in the Bluetooth specification. In addition to the channel-hopping sequence, there are four other types of frequency-hopping sequences. These four sequences are the basis for establishing communication between Bluetooth devices. They are the inquiry hopping sequence, the inquiry response sequence, the page hopping sequence, and the page response sequence. The following section will go into detail as to how these sequences are used to establish a connection between Bluetooth devices.

2.4 Link Establishment

The operational states for a Bluetooth device are the *standby*, *inquiry*, *page*, and *connected* states as shown in Figure 2-10. [1] The *standby* state is the default operational state. To move to the *connected* state, a master device goes through the *inquiry* and *page* states, while a slave device goes through the *inquiry scan* and *page scan* states. The purpose of the inquiry process is for a master node to discover the existence of slave devices and to collect information such as the device address and data on the native clock. The purpose of the page process is for a master to connect to potential slaves discovered during the previous inquiry procedure.

In order for devices to discover each other more easily, all devices operate on a well-known frequency-hopping sequence during the inquiry and page process. The inquiry and page hopping sequences are use only 32 frequencies, instead of all 79 frequencies that are used in the channel-hopping sequence after establishing a connection. [7] During the link establishment process, the potential master transmits packets changing their transmission frequency at a very rapid rate of 3200 hops/sec. Potential slaves, in contrast, change their listening frequency at a very slow 0.78125 hops/sec. The two different hopping rates increase the probability of the master finding a slave listening on any one of the channels. [25] The following sections go into detail as to how the frequency-hopping sequence is used during the inquiry and page states.

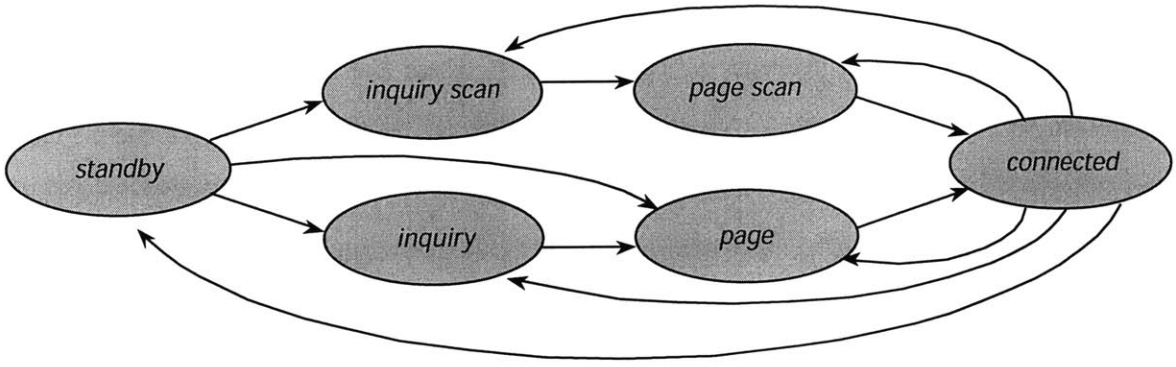

Figure 2-10. A state diagram of the possible operational states for a Bluetooth device during device discovery. The arrows indicate valid state transitions. [1]

2.4.1 Inquiry State

During the inquiry process, all Bluetooth devices use the same 32 frequencies, which are derived from the Bluetooth general inquiry access code (GIAC). The 32 frequencies derived from the GIAC are divided into two sets of 16 frequencies. These two sets of 16 frequencies are referred to as train A and train B. All potential masters use the 32 frequencies in the same order, however which frequencies fall in which train is dependent upon the phase shift derived from the master's clock.

The potential master uses these two frequency trains to search for potential slaves within radio range. As shown in Figure 2-11, the potential master uses both trains and switches trains at a much faster rate than the potential slaves.

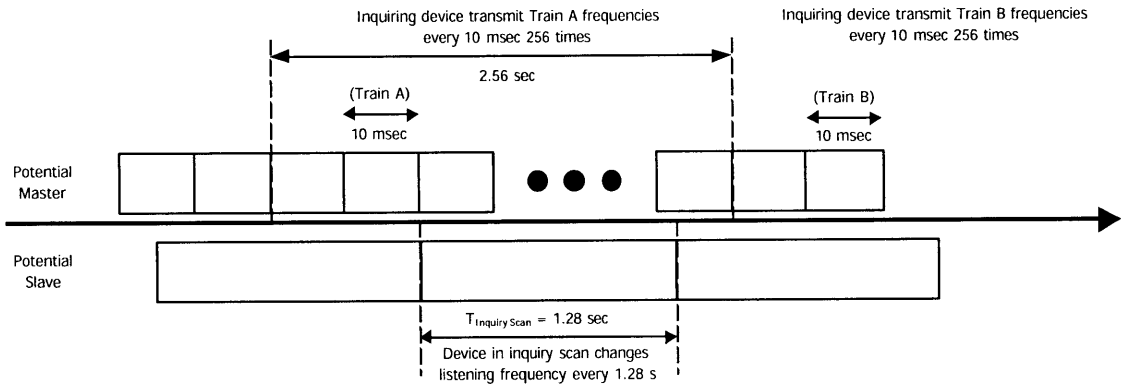

Figure 2-11. Inquiry frequency-hopping by a potential master, and Inquiry Scan frequency-hopping by a potential slave. [2]

The master will broadcast inquiry packets using the same train 256 times. Since the time to transmit one entire train is 10 milliseconds, a total of 2.56 seconds elapses before switching trains. Note that the potential slaves stay on the same frequency listening for 1.28 seconds.

During the 10ms transmission of a train, there are only 16 time slots available; eight transmit slots and eight receive slots. The master must transmit a 68-bit ID packet, containing an Inquiry Access Code (IAC), on all 16 frequency channels at least once. To accomplish this, two ID packets are transmitted in one slot on two different frequencies. In Figure 2-12, note that each transmit slot is

divided in half to accommodate the transmission of two ID packets on different frequencies. It takes 68μs to transmit a single inquiry packet, followed by a waiting period until the master can transmit the next inquiry packet. This makes the frequency hop rate 3,200 hops per second. After the two inquiry packets are transmitted on different frequencies, the potential master listens for a frequency-hopping synchronization (FHS) packet response on both frequencies in the event a potential slave was listening at on either frequency. The 366-bit FHS response packet contains the slave's Bluetooth device address, its clock, and information about when it enters its page scan state.

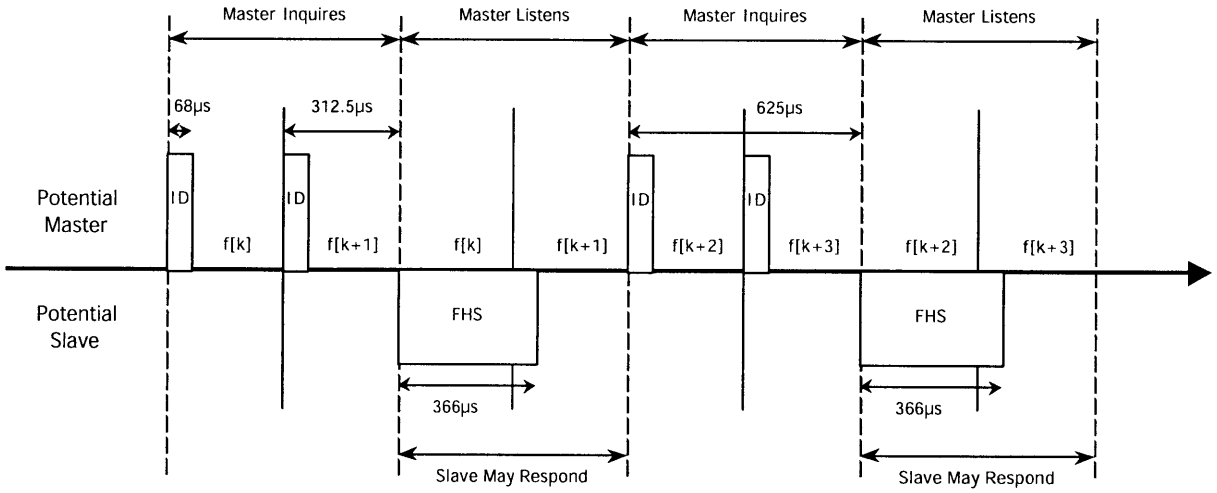

Figure 2-12. Transmission of Train by Inquiring Master and Response Window for Potential Slave. [2]

Bluetooth accounts for the possibility that multiple slaves can be in *inquiry scan* state listening to the same message from a potential master. See Figure 2-13. To prevent contention among the slaves, each scanning slave chooses a random back-off interval, $T_{back\text{-}off}$, between 0 and 1023 time slots, and waits to receive another ID packet from the master before responding with an FHS packet. Once the slave does respond, it enters the *standby* state where it will enter the *page scan* state. [1], [2], [4]

The time elapse for the master to match the frequency that the slave is currently listening is defined as the frequency synchronization delay. The minimum inquiry time for an inquiry operation is

2 slots, or 1.25ms. This is where the master transmits an inquiry message on the *f[k]* frequency, and the slave receives the inquiry packet in the first slot. The slave responds with a FHS packet in the following slot. A total of 2 slots are needed. However, this is highly unlikely as the slave will not respond after receiving the first inquiry message. Instead, the slave will wait a random number of slots, $T_{back\text{-}off}$, to avoid contention.

The maximum inquiry time as specified in the Bluetooth specification is one minute, at which time the inquiry is halted. Typically, in an error-free environment, four trains must be used. [1] Therefore, 10.24 seconds could elapse unless the potential master collects enough responses and aborts the procedure. However, piconet research has revealed that two trains are sufficient to complete the inquiry process. [11] Thus, the average inquiry time is 5.12 seconds.

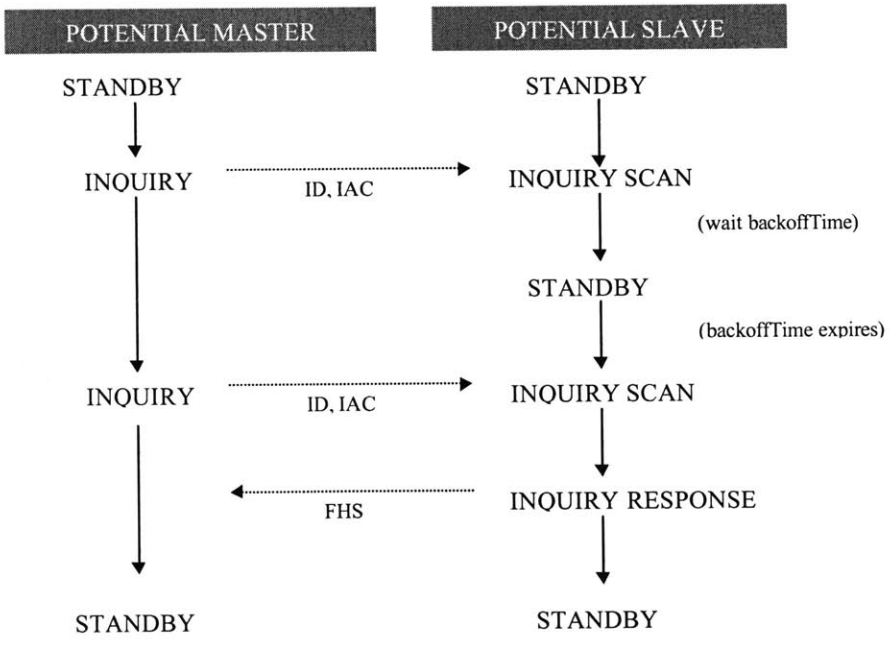

Figure 2-13. State transitions during the Inquiry and Inquiry Scan process. [4]

2.4.2 Page State

The potential master keeps track of how many responses it obtains during the Inquiry process. If the number of responses is greater than zero, the potential master enters the *page* state. In the *page* state, the master should have obtained during the Inquiry process the signaling information about when the slave will enter its *page scan* state. The master calculates the slave's Device Access Code (sDAC), which is derived from the slave's BD_ADDR address, and the master then transmits a sDAC packet that can be heard only by the corresponding receiver device.

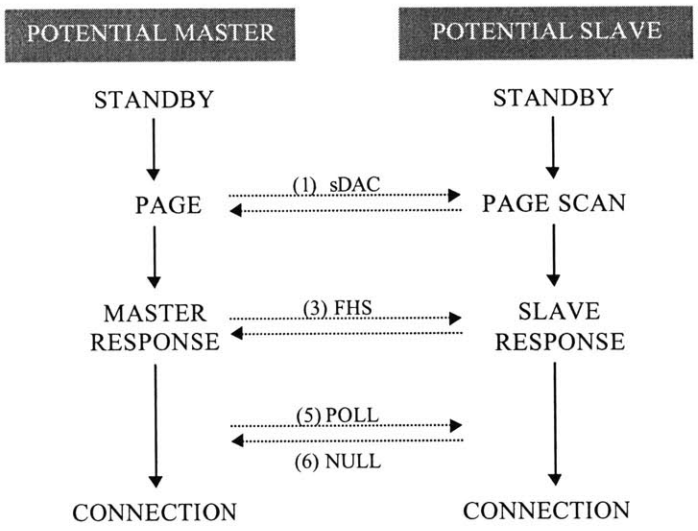

Figure 2-14. State transitions during the Page and Page Scan process. [4]

The slave listens to page packets for its own sDAC over an interval of 1.28 seconds. This window of 1.28 seconds is long enough for the slave to cover the frequency hops from one train of a paging master. Just as in inquiry, the slave slowly hops the different frequencies once every 1.28 seconds and the master hops once every 312.5μs. Eventually a match is found, and the slave transmits a response with its sDAC and changes to the *page response* state. [1]

The master returns a FHS packet transmission that includes its Bluetooth device address (BD_ADDR), its clock value for calculating the clock offset, and the slave's active member address (AM_ADDR) assignment. The time of transmission of this FHS packet identifies the start of the master's transmit slots in the piconet. The slave and master exchange a POLL and NULL packet to fully establish the master-slave connection and enter the *connected* state. [2]

With this paging scheme, the average time for connection should be 1.28 seconds. The average page time is shorter than the average inquiry time because the potential master has some knowledge about where the potential is listening during the page operation. The maximum page time before making connection is 2.56 seconds. To take the maximum time, both A and B train need to be repeated 128 times. For the case of minimum page time, see Figure 2-15.

The master transmits two page packets containing the sDAC at frequency *f[k]* and *f[k+1]* in the first slot. If the slave receives a page packet in the first slot as shown in Figure 2-15, then the slave will respond in the next slot. The master will respond with a FHS packet in the third slot, and the slave answers in the fourth slot. In total, four slots are needed for the minimum 2.5 ms.

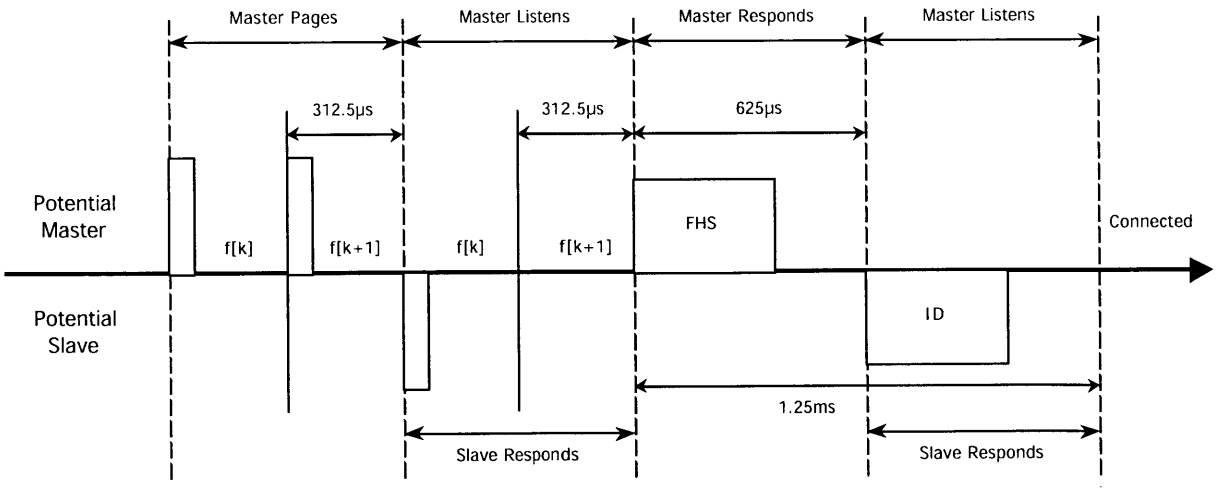

Figure 2-15. Page process can be completed in a minimum time of four time slots. [2]

2.4.3 Connection State

For a slave to be connected to the master, it must be assigned an AM_ADDR. It is a 3-bit address value temporarily assigned to a slave so that it can participate on the piconet. Every packet exchanged between the master and slave contains the AM_ADDR of the slave unit. But, the all zero value of this field is reserved for broadcast packets from the master to all slaves on the piconet.

When a master and slave have established a communication link, the two units can operate in one of four modes – ACTIVE, SNIFF, HOLD, or PARK. [1]

The default mode is ACTIVE, in which both master and slave actively participate on the channel by listening, transmitting, or receiving packets. All slaves in ACTIVE mode maintain an active member address (AM_ADDR) and must listen at the beginning of every even-numbered time slot to determine if the master is sending a packet with its AM_ADDR.

In SNIFF mode, a slave reduces its duty cycle during periods of low activity to conserve battery life. Instead of listening at the beginning of each even-numbered time slot, the slave listens at every T_{SNIFF} interval. [24] The slave in SNIFF mode is still an active member of the piconet and maintains its AM_ADDR.

In HOLD mode, a transceiver unit neither transmits nor receives information with its current piconet. During this time, the slave or master can be made free to do other things like sleep, scanning, paging, or inquiring. [24] It may even join another piconet by just changing the channel parameters such as the clock offset and frequency-hopping sequence. [22] When a slave enters HOLD mode, it is still considered an active member of the piconet and it retains its AM_ADDR. Prior to entering the HOLD mode, the master and slave agree on the time duration the slave remains in the HOLD mode. A unit can be held in HOLD mode for 625μs, up to 40.9 seconds. A timer is initialized with the hold timeout value. When the timer expires, the HOLD mode is released and the slave synchronizes to the traffic on the channel and waits for further master instructions.

In PARK mode, a slave enters a low-power mode of very little activity. It no longer participates on the piconet channel, so it gives up its active member address AM_ADDR. In place of the AM_ADDR, the slave is assigned two new addresses to be used while in PARK mode: 8-bit parked member address (PM_ADDR) and 8-bit access request address (AR_ADDR). The PM_ADDR is used when a master wants to unpark the slave. The AR_ADDR is used when the slave wants to unpark itself. The PARK mode also permits more slaves to be connected to a single master, a total of 255 to be exact. These parked slaves can be swapped in and out of ACTIVE mode. While in PARK mode, the parked slave wakes up at regular intervals to listen to a special beacon channel in order to re-synchronize and to check for broadcast messages. The slave remains synchronized to the channel so that when it wants to return to the piconet, it does not have to repeat the inquiry and page process.

2.5 Time Taken to Complete Inquiry and Paging Procedures

According to the Bluetooth Special Interest Group, the numbers in Table 2-1 summarizes the theoretical time taken to complete a typical successful inquiry and page operation, and thus the typical times taken to setup a Bluetooth connection. From Table 2-1, the time to set up a connection may be as long as 12.8 seconds. In most cases, the 6.4 seconds average connection time is acceptable.

Operation Type	Minimum Time	Average Time	Maximum Time
Inquiry	1.25ms	5.12s	10.24s
Page	2.5ms	1.28s	2.56s
Total	3.75ms	6.4s	12.8s

Table 2-1. The expected time for Inquiry and Page processes to complete. [1]

2.6 Packet Types

In this section, the packet types with the various payload sizes and features are summarized. Earlier in this chapter, ID, FHS, POLL, and NULL packets were described to illustrate the process of

establishing a communication link. In addition to these control packets, there are data packets supported by Bluetooth which differ in payload size, encoding scheme, and the type of connection link.

The two types of connection link are the Synchronous Connection Oriented (SCO) link or Asynchronous Connectionless Link (ACL). In this thesis, only ACL links are used in simulation. There are six different ACL packets: D(M|H)(1|3|5). DM stands for medium data speed using 2/3 Forward Error Correction (FEC) encoding of its payload. DH stands for high data speed and no FEC is used for the payload. (1|3|5) are size qualifiers referring to the number of slots occupied by the packet.

Below is a summary of the baseband packets discussed in this thesis.

TYPE	User Payload (bytes)	Slot Occupancy	FEC	Symmetric Max Rate(kb/s)
ID	Na	1	Na	Na
NULL	Na	1	Na	Na
POLL	Na	1	Na	Na
FHS	18	1	2/3	Na
DM1	0-17	1	2/3	108.8
DH1	0-27	1	No	172.8
DM3	0-121	3	2/3	258.1
DH3	0-183	3	No	390.4
DM5	0-224	5	2/3	286.7
DH5	0-339	5	No	433.9

Table 2-2. Packets and their characteristics according to the Bluetooth Specification. [1]

Chapter 3

BlueRelay

Although the Bluetooth specification details how piconet communication is established, it provides little insight to the formation of inter-piconet communication that would enable multi-hop communication. This chapter describes BlueRelay, a novel protocol built upon the Bluetooth framework. BlueRelay is presented here in two parts, the first part describing the relay establishment process and the second part describing the piconet switching scheme to enable the forwarding of data packets in a multi-hop relay. The goals of BlueRelay are:

1. To show that a slave can establish connections with two masters so that a chain of alternating master and slave nodes can be formed for a single path of communication over multiple hops.
2. To devise a piconet switching scheme that allows a slave node to communicate on two different links on a time division basis. This requires coordination by a forwarding slave unit since a slave cannot be active on two channels at the same time.
3. To leverage defined Bluetooth mechanisms summarized in Chapter 2, specifically the connection process (inquiry and page) for relay establishment and the HOLD mode for piconet switching.

The BlueRelay interface sits between the application layer and the L2CAP layer in the Bluetooth protocol stack as shown in Figure 3-1. The relay establishment procedure link and piconet switching schedule interact with the L2CAP layer interface which can access the capabilities in the lower layers to execute the individual link establishment and the piconet switching.

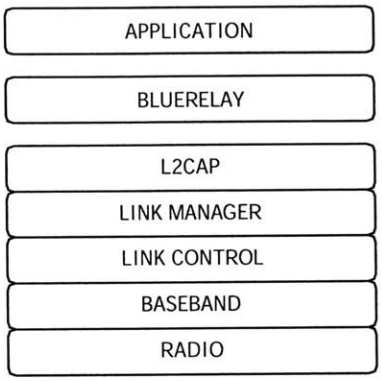

Figure 3-1. The BlueRelay interface sits between the Application and L2CAP layer of the Bluetooth Protocol Stack.

In simulation, the topology that the BlueRelay scheme forms is illustrated in Figure 3-2. It is an alternating chain of master and slave nodes. The slaves are responsible for coordinating communication between different masters, in essence different piconets. [3] BlueRelay also makes certain assumptions about the environment and the master and slave nodes.

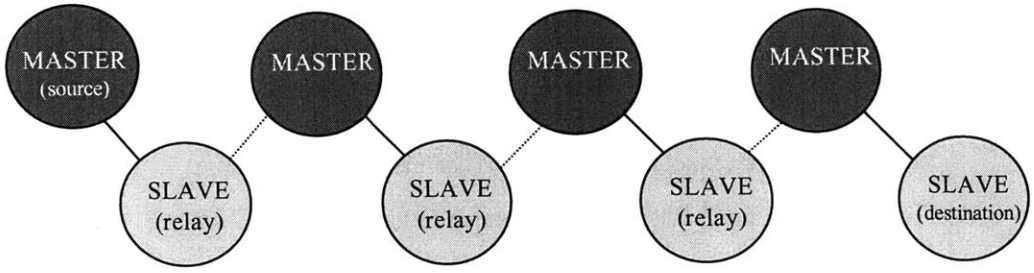

Figure 3-2. Alternating chain of master and slave nodes for multi-hop relay.

We assume the following for a multi-hop relay, such as the one shown in Figure 3-2:

1) Nodes are non-mobile, making the topology static.
2) The relay is a chain of alternating master and slave units.
3) There is only one path between the source and destination.
4) Each unit participates in the relay for the life of the network.
5) The source node and destination node can be either a slave or master device.
6) Each node is preconfigured to carry out either the master or slave role.
7) The Bluetooth Address, BD_ADDR, of all devices in the network is known prior to network establishment.
8) Slave nodes have prior knowledge of which master nodes to connect to. For example, upon powering up, a slave will have been assigned to respond to a pre-determined master node.
9) Master nodes, unlike the slave nodes, do not discriminate which slave devices to connect to.
10) Slaves act as the bridge unit to route data from one piconet to another. They are allowed to participate in two different piconets, while masters cannot.

Having prior knowledge of the Bluetooth addresses of devices and assigning slaves to respond to specific masters do not make BlueRelay an ad hoc protocol. Manual configuration and pre-assigning addresses is unattractive. In a real network, the manual configuration should be replaced by an automatic configuration scheme. However, the focus of this thesis is not ad hoc discovery. Therefore, BlueRelay simplifies the relay establishment process to focus on demonstrating that a slave can connect to two masters. By showing that a slave can connect to two masters, this proves that an alternating chain of master and slave nodes can be established and provides the multi-hop relay to examine inter-piconet communication.

Manual configuration can be accomplished within Bluetooth utilizing Inquiry Access Codes which have been left undefined by the Bluetooth specification. They were left to be defined by application developers for the purpose of adding new functionality to a protocol that leverages Bluetooth, such as BlueRelay. By pre-configuring the Bluetooth devices, a single-chain multi-hop wireless relay can be established to test the idea of inter-piconet communication.

3.1 Relay Establishment in BlueRelay

The purpose of relay establishment is to demonstrate that a slave can establish connections with two masters and maintain two different channel-hopping sequences. In the Bluetooth Specification, a slave is originally designed to connect to only one master in a piconet configuration. This design is insufficient for multi-hop relays. In BlueRelay, a slave must be able to establish connections with two masters in order to create an alternating chain of master and slave nodes. By successfully establishing an alternating chain of master and slave nodes, only then inter-piconet communication is possible.

In this section, relay establishment is described. BlueRelay assumes that nodes are pre-configured to be either a master or a slave because a master and a slave must follow different algorithms in establishing a connection to the relay during the topology formation process.

Master nodes in BlueRelay follow these steps to establish a multi-hop relay:

Given: The required number of inquiry responses is set to two.
Inquiry timeout is set to 10.24 s, which is max time master can spend in Inquiry mode.
Step 1: Enter *Standby* state.
Step 2: Enter *Inquiry* state; Time out occurs in 10.24 seconds.
Broadcast ID packet and listen for FHS response packet from potential slaves.
Step 3: If number of FHS responses is greater than zero,
then enter *Page* state and choose one slave to establish connection.
Otherwise, return to Step 1.
Step 4: If master is designated as an end node of a relay, then go to Step 6.
Step 5: If master node has two active slaves in its piconet, then go to Step 6.
Otherwise, go to Step 1.
Step 6: Master is considered connected in relay.

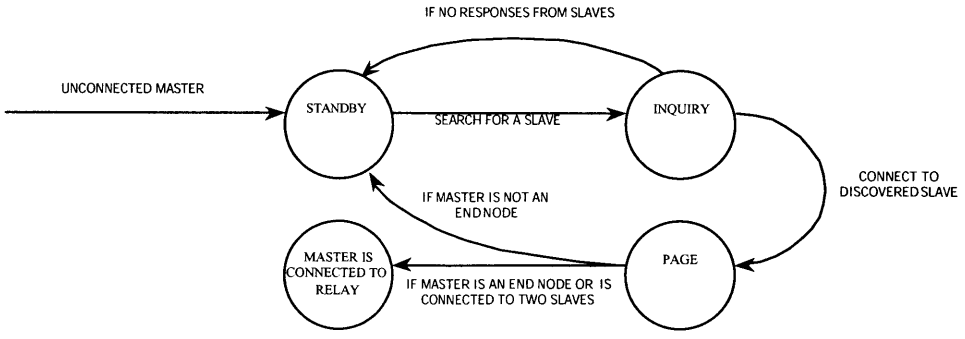

Figure 3-3. State diagram showing valid transitions for a master node during the connection establishment process.

Slave nodes in BlueRelay follow these steps to establish a multi-hop relay:

Given: The 48-bit Bluetooth Address(es) of the master device(s) which to connect.

Step 1: Enter *Standby* state.
Step 2: Enter *Inquiry Scan* state; Listen for broadcast packet from masters.
Step 3: If broadcast packet is from a master which the slave is pre-assigned to connect to, then enter *Page Scan* state, listen for page packet from pre-assigned master, respond to that master, and establish a connection with that master. Otherwise, go to Step 1.
Step 4: If slave is designated as an end node of a relay, then go to Step 6.
Step 5: If slave is an active member in two different piconets, then go to Step 6. Otherwise, enter HOLD mode and go to Step 1.
Step 6. Slave is considered connected in relay.

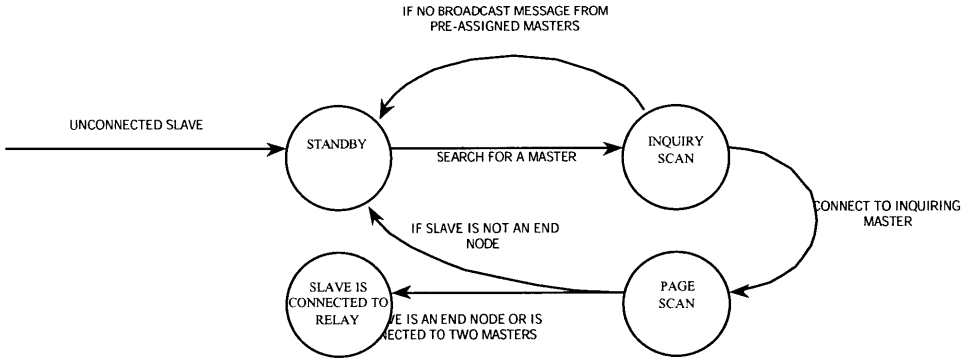

Figure 3-4. State diagram showing valid transitions for a slave node during the connection establishment process.

For the purpose of readability, the following terms are defined. A slave node acting as a forwarding node is called a bridge slave since it "bridges" communication between two piconets. A primary piconet is defined as the first piconet which a "bridge" slave connects to during the discovery process. The secondary piconet is defined as the second piconet which a "bridge" slave connects to during the discovery process. After the discovery process, a "bridge" slave should be connected to different master nodes and considered active members in the two different piconets.

Hence, there are three possible node designations in BlueRelay: master, slave, and bridge slave. A master acts as any master would in a piconet. A slave participates in only one piconet, such as the units on the end of a relay. A bridge slave acts as the forwarding node between two piconets, enabling inter-piconet communication.

Figure 3-5 illustrates the time elapse of the relay establishment process for three master nodes and two slave nodes. The nodes on the extreme ends of the relay form only one connected link with its neighbor. All other nodes establish two connected links with its neighbors.

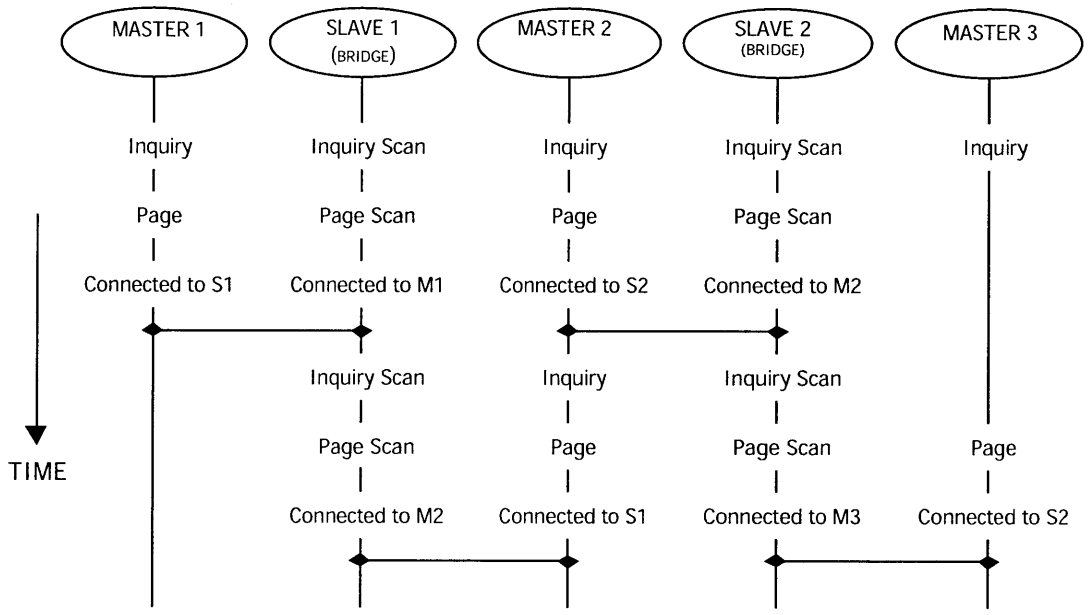

Figure 3-5. The state transitions for three master nodes and two slave nodes during the topology formation process. Slave 1 forms a connection with master 1, then master 2. Slave 2 forms a connection with master 2, then master 3.

3.2 Piconet Switching in BlueRelay

Once in the *Connected* state, a bridge slave is active in both piconets alternately. The bridge slave can request to leave one piconet momentarily to join the other piconet. A bridge slave can switch between piconets because the Bluetooth protocol allows a node to maintain more than one set of clock

information. Hence, a bridge slave maintains the frequency-hopping sequence, the clock offset, and the device address of the master devices in the primary piconet and the secondary piconet. To participate on the proper frequency-hopping channel, a bridge slave employs the proper clock offset to obtain the correct phase to connect to the desired master device. However, resynchronizing a bridge slave to a master incurs overhead since two piconets are not synchronized by time or frequency. Consequently, bandwidth is wasted during the switching process. In the following sections, the challenges, the operation of piconet switching, and the overhead incurred by the switching process, are explained.

3.2.1 Challenges of Piconet Switching

Understanding how an individual node properly switches parameters to realign time slots and communicate with the desired neighbor is just one issue that makes piconet switching a challenge. Another concern is the frequency of piconet switching, which cannot occur too frequently or infrequently. Frequent switching would result in high overhead of control packets being sent to synchronize a slave with the desired master. Infrequent switching would increase packet latency as the packet incurs a longer waiting time at each hop. There is also the matter of synchronizing the piconet switching in the relay. If a master has two bridge slaves that enter HOLD mode at the same time and return to the piconet at the same time, the two bridge slaves would have to share the piconet bandwidth. To better utilize the piconet bandwidth, the master should alternately communicate with its two bridge slaves.

3.2.2 Justification of HOLD Mode

Of the possible mechanisms of Bluetooth, the HOLD mode is most appropriate to enable the activation and deactivation of communication links. The two other connection modes that are supported by Bluetooth, SNIFF and PARK, were not preferred for the following reasons.

A slave, when in PARK mode, has to listen to special periodic master transmissions, called beacon messages, to remain synchronized to the master. Also the process of parking and unparking a slave incurs more overhead than the HOLD mode because it wastes more time slots to request and enter PARK mode.

A slave in SNIFF mode is also not an attractive solution, since a slave in SNIFF mode has to periodically listen to master transmissions to remain synchronized to the master. This requires a bridge slave to constantly switch channel parameters to acknowledge the periodic transmission. Consequently, this restriction does not allow a slave to completely leave a piconet for a fixed period of time to fully participate in another piconet.

Thus HOLD mode is appropriate for piconet switching. It allows a bridge slave to be inactive for a fixed time interval from one master to contend with another master without interruption. Moreover, when a communication link between a master and slave is on hold, the Bluetooth specification does not allow a master to poll the inactive slave until the HOLD time expires and returns to the master. This reduces overhead because fewer control packets are transmitted than compared to the SNIFF and PARK modes.

3.2.3 Slave Nodes Utilize HOLD Mode

Although the Bluetooth specification allows both master and slave nodes to request entering the HOLD mode, masters in BlueRelay are not allowed to request entering the HOLD mode. In BlueRelay, only slave nodes that act as bridge nodes are allowed to request entering the HOLD mode. The justification is that master nodes already handle coordination of communication within a piconet. Masters should not be responsible for inter-piconet communication. That task should be delegated to the slave since the slaves are the nodes responsible for bridging piconets, switching between piconets, and maintaining two sets of channel parameters. Since slaves are responsible for forwarding data, slaves should be the nodes requesting the HOLD mode to leave a piconet when it so desires.

With the understanding that slave nodes in BlueRelay are responsible for managing the HOLD mode, the following describes how a "bridge" slave specifically requests a HOLD mode and switches from one piconet to another piconet. Recall from the relay establishment process, a primary piconet is defined as the first piconet which a "bridge" slave connects to during the discovery process. The secondary piconet is defined as the second piconet which a "bridge" slave connects to during the discovery process. It is obvious that a slave at the end of a relay would make only one connection to a primary piconet. It would remain connected to its primary piconet at all times and never have to connect to a secondary piconet. Therefore, the remainder of the discussion is for the case of a slave acting as a "bridge" node to connect to its primary and secondary piconet. Since the clocks of the primary and secondary masters are not synchronized, the slave unit must maintain two clock offsets. When the appropriate clock offset is added to its own native clock, the slave can synchronize with either the primary or secondary master clock.

3.2.4 Scheduling HOLD Mode To Enable Inter-Piconet Communication

According to the Bluetooth specification, a connection between two Bluetooth devices can be placed in HOLD mode for a specified amount of time, hold_time. During this hold_time, no packets can be transmitted from the master. Knowing that communication with one piconet has temporarily ceased, a slave can join another piconet. It is important to note that the activities of a device during the hold_time are up to the device to decide.

When a bridge slave wants to enter the HOLD mode for its primary piconet, the slave must request to enter the HOLD mode. Figure 3-6 illustrates the request process. [1] The Link Manager layer of the slave device sends a request message, LMP_hold_req(*hold_time, hold_instant*). This LMP_hold_req is encapsulated in a baseband packet and sent over the air to the master. The Link Manager on the receiving side will process the LMP_hold_req and respond with an LMP_accepted message, which is encapsulated and sent to the slave that requested the HOLD mode.

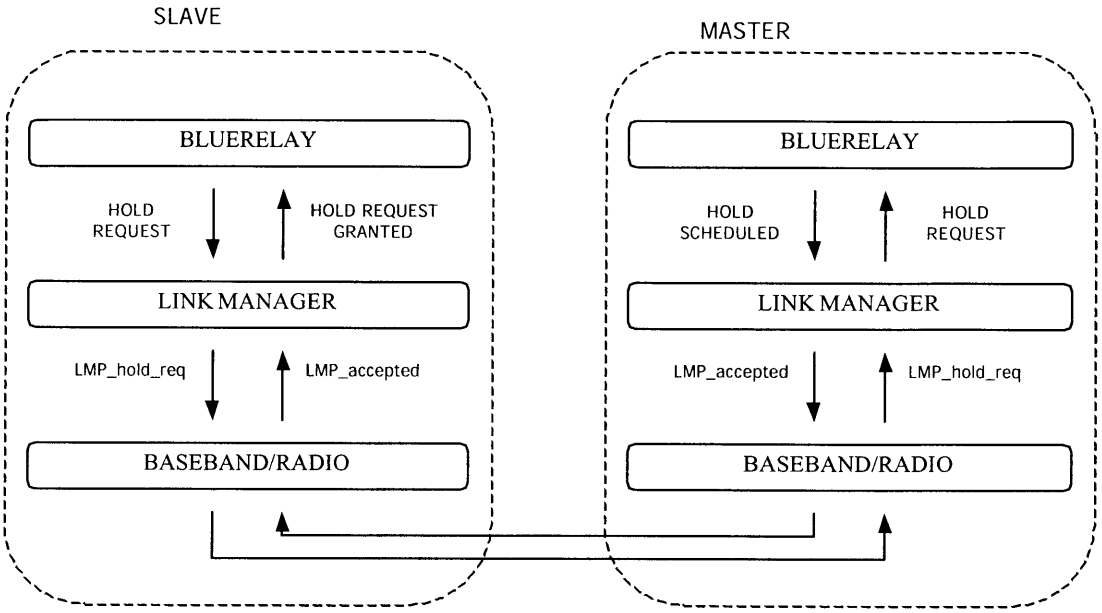

Figure 3-6. Scheduling of HOLD mode: Slave requests HOLD mode. Master accepts and responds. Slave receives acknowledgement that the request has been accepted. [1]

In the hold request, the *hold_time* and *hold_instant* are specified. The *hold_time* is the amount of time that a master and slave are to be in HOLD mode. The *hold_instant* specifies the instance at which the hold will commence. This instant is chosen by the slave to be at least nine slots into the future of the current value of the master's clock. The nine slots are required by the Bluetooth specification to allow for the worst-case scenario that a master and a slave both need to transmit a 5-slot packet while the HOLD request is negotiated. [28]

A bridge slave manages piconet switching by cycling through these eight steps:

1. Slave in ACTIVE mode with primary piconet for a duration of hold_time.
2. Slave and Master A (of primary piconet) negotiate HOLD mode for the future.
3. Slave enters HOLD mode with primary piconet; Master A does not poll slave.
4. Slave switches clock parameters to synchronize with Master B.
5. Slave in ACTIVE mode with secondary piconet for a duration of hold_time.
6. Slave and Master B (of secondary piconet) negotiate HOLD mode for the future.
7. Slave enters HOLD mode with secondary piconet; Master B does not poll slave.
8. Slave switches clock parameters to synchronize with Master A.

By following the piconet switching schedule, a bridge slave switches periodically from one piconet to another as shown in Figure 3-7. When a bridge slave does leave a piconet, communication ceases between the bridge slave and that piconet's master. To ensure that each master timeshares the bridge slave equally, the bridge slave requests the same hold duration for both piconets. This keeps the piconet switching predictable.

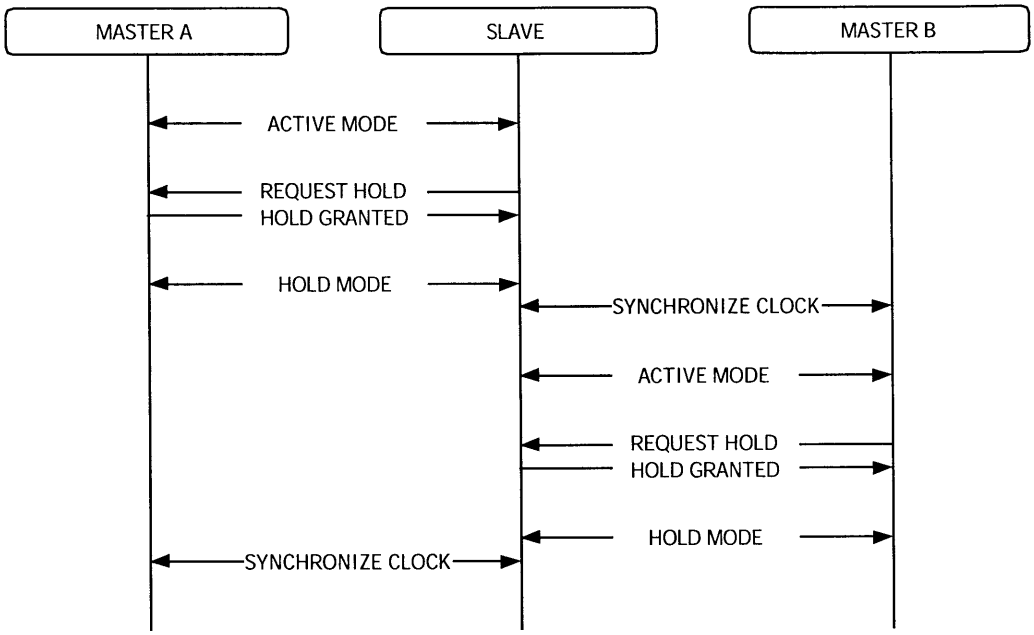

Figure 3-7. One complete cycle of piconet switching for inter-piconet communication. During the ACTIVE mode with master A, the slave schedules a HOLD mode for master A. During the HOLD mode with master A, the slave is in ACTIVE mode with master B. The slave schedules a HOLD mode for master B before the HOLD mode with master A expires and the slave has to return to master A.

A bridge slave begins switching back and forth from one piconet to another immediately after establishing connections with its primary and secondary masters. Each time a bridge slave returns to a piconet, it is important that the master be expecting the slave's return to facilitate resynchronization. If the master is still in HOLD mode and not expecting a slave's return, an undesirable situation occurs

where the slave will be in HOLD mode for both piconets. Eventually, however, the HOLD mode will timeout and the slave will be able to resynchronize.

To prevent a bridge slave from being in HOLD mode in both piconets simultaneously, a bridge slave should schedule to leave one piconet at the same instance that the other piconet is expecting its return. In Figure 3-7, whenever the bridge slave schedules a HOLD mode with Master B, the bridge slave must schedule the HOLD mode to commence just as the HOLD mode with Master A expires. This ensures a bridge slave to be in constant communication with a master at any given time.

Like slaves, a master can potentially be on HOLD with both bridge slaves in its piconet. Such a situation where a master is sitting idle waiting for slaves to return is undesirable. Moreover, when both slaves return to the master, the two slaves will have to split the available bandwidth since the master can only communicate with one slave at a time. This is poor utilization of bandwidth. It would be optimal to have one slave at a time to be ACTIVE with the master to utilize the full bandwidth.

3.2.5 Synchronization of Piconet Switching

In general, only relays with two bridge slaves or more are cases where piconet switching needs to be synchronized. For a simple two-node network, there is only one master and one slave. Since the slave is not a bridge slave, no piconet switching occurs. For a three-node network, no piconet switching occurs either. A slave-master-slave relay is essentially a piconet, so the two slaves stay in constant connection with the master. In a master-slave-master relay, the slave node is a bridge slave and does piconet switching to alternately communicate with the masters. However, no synchronization of piconet switching is required since there is only one bridge slave. Even in a 4-node network and a 5-node network as shown in Figure 3-8, there is only one bridge slave. Again, a network consisting of only one bridge slave does not have to be concerned with synchronizing the piconet switching schedule in the relay.

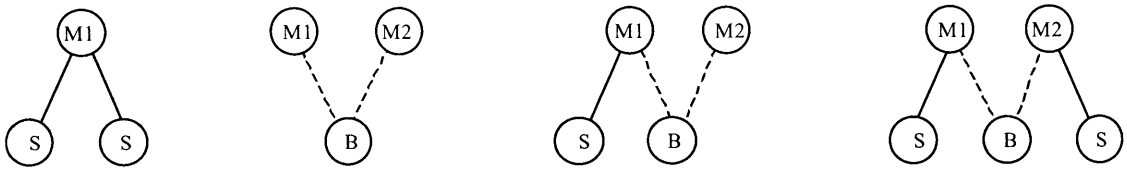

Figure 3-8. These relay topologies have either one bridge slave or none at all. The solid lines connecting a master and an end-node slave denote a constant connection. The dotted lines denote alternating communication between a master and a bridge slave. Note that no synchronization of piconet switching is required when only one bridge slave exists in the relay.

Only for relays with two or more bridge slaves does synchronized piconet switching become an issue. It is these cases where a master node can be in HOLD mode simultaneously with two bridge slaves. To correct this and synchronize piconet switching, the masters are responsible for adjusting the start times when scheduling a HOLD mode.

The master corrects this by delaying the start time for HOLD mode by $T_{stagger}$ slots. To calculate $T_{stagger}$, the overlap time must be determined. The overlap time, Δt_{start_time}, is the difference of the start times of the two most recent HOLD modes. Let x_{start_time} be the time stamp of the most recent HOLD mode, let y_{start_time} be the time stamp of the second most recent HOLD mode, and let n_{hold_slots} be the number of time slots for the hold interval. A master calculates the Δt_{start_time} and $T_{stagger}$ as follows.

$$\Delta t_{start\_time} = x_{start\_time} - y_{start\_time} \tag{3.1}$$

$$T_{stagger} = \begin{cases} \frac{1}{2} n_{hold\_slots}, & \Delta t_{start\_time} = 0 \\ 0, & \Delta t_{start\_time} \le n_{hold\_slots} \\ ceiling\left(\frac{\Delta t_{start\_tim} - n_{hold\_slots}}{2}\right), & \Delta t_{start\_time} > n_{hold\_slots} \end{cases} \tag{3.2}$$

Equation 3.2 determines the amount of time slots to stagger the requested HOLD start time. If the bridge slaves happen to be entering the HOLD mode at the same exact time, then the master staggers the start time of one of the bridges by one-half the hold interval. If the bridge slave that is

requesting the HOLD mode is leading the other bridge slave, then the stagger time is zero. The responsibility of reducing the overlap time is given to the bridge slave that is lagging. Since the start time of the HOLD mechanism can only be delayed, it is appropriate that only the lagging bridge slave delays its start time to increase its lag time behind the leading bridge slave. Hence, the leading bridge slave stretches its lead until the time difference is exactly one hold interval, at which point the master has synchronized its bridge slaves.

Note how the $T_{stagger}$ is one-half the overlapping time. The master does not stagger the start time by the entire overlapping time. Consider the case where a bridge slave is lagging by one slot. If the master were to delay the lagging bridge slave by $(n_{hold\_slots} - 1)$ slots, then the lagging bridge slave would be away from its other piconet for $2n_{hold\_slots}$ slots. This is a relatively long time since the other piconet is expecting a slave to return in one hold interval. Therefore, the master does not stagger the start time by the entire overlapping time. Instead, the lagging bridge slave is delayed by one-half the overlapping time. Hence, a bridge slave leaves its masters for no longer than $1.5n_{hold\_slots}$ slots.

This piconet synchronization scheme works for the worst-case scenarios where neither bridge slave is lagging one another, where one bridge slave is lagging by one time slot, and where one bridge slave is lagging by $(n_{hold\_slots} - 1)$ time slots.

Figure 3-9a is an example of how two bridge slaves, B1 and B2, are simultaneously ACTIVE in the same piconet, whereas B2 and B3 have synchronized piconet switching. The master M2 schedules slave B1 to enter HOLD in T_{guard} slots and schedules the slave B2 to enter HOLD in $T_{guard} + T_{stagger}$ slots, where $T_{stagger} = \frac{1}{2} n_{hold\_slots}$. When slave B2 switches to its other piconet with Master M3, its HOLD interval will encroach upon the slave B3. Master M3 will then have to calculate its $T_{stagger}$ and add that offset to the HOLD mode start time for the lagging bridge slave. Every round of piconet switching will propagate the $T_{stagger}$ offset down the relay. Eventually, the periodicity will converge and all piconet switching throughout the network will be synchronized together as shown in Figure 3-9b. As a result, parallel communications can occur, thereby increasing the overall throughput of the relay.

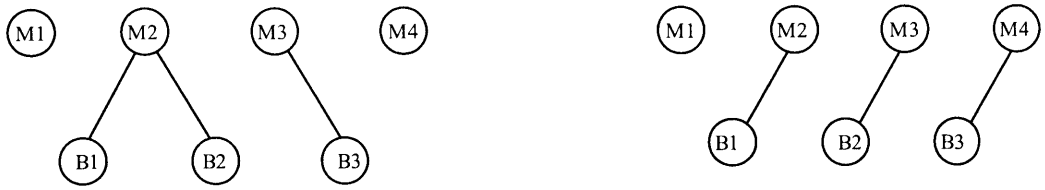

Figure 3-9. (a) Bridge slaves B1 and B2 are simultaneously ACTIVE with master M2. Slave B2 needs to offset its start time of its HOLD mode. And Slave B3 needs to adjust its HOLD mode start time accordingly. (b) Slaves B1, B2, and B3 have synchronized their piconet switching such that no two slaves are simultaneously ACTIVE in the same piconet.

The number of rounds to synchronize piconet switching in a relay depends on two factors, the duration of the HOLD mode and the length of the relay. Intuitively, a longer relay requires additional rounds of piconet switches to propagate the $T_{stagger}$ offset. Also, a longer hold duration requires more rounds of piconet switches for the overlapping time, $\Delta t_{start\_time_n}$ to reduce to zero.

For the worst case scenario which a master node is entering HOLD mode at the same instant with both of its bridge slaves, the master requires R rounds to coordinate the piconet switching.

$$R = ceiling\left(\log_2 n_{hold\_slots}\right) \tag{3.3}$$

R is the maximum number of rounds of piconet switches needed for a master to coordinate the piconet switching such that its bridge slaves are in alternate communication. For the $T_{stagger}$ offset to propagate to the remainder of the relay, N_{rounds} is the number of rounds for a relay to have synchronized piconet switching.

$$N_{rounds} = \left(n_{bridge\_slaves} - 1\right)R \tag{3.4}$$

N_{rounds} is a multiple of R rounds, where n_{bridge_slaves} is the number of bridge slaves in the relay. Equation 3.4 accounts for the worst case scenario that each subsequent bridge slave downstream is offset by one entire HOLD duration. This provides an upper bound for when piconet switching becomes synchronized.

3.2.6 Alignment of Time Slots

Each time a bridge slave switches back and forth between piconets, the slave must synchronize with the clock of the targeted master. Recall that the slave maintains two offsets, one for each of the its two masters. By adding the appropriate clock offset to its own native clock, a slave can estimate the timing of a master's clock.

Figure 3-10 shows an example of a bridge slave aligning time slots with one master and then again with its other master. The slot boundaries of two masters do not match, just as shown in the figure. Therefore, during a piconet switch, a device has to wait at least until the next even slot begins in order to be able to participate in the current piconet. The realignment could potentially cost up to two time slots.

The two slot overhead cost prevents a master and slave from communicating immediately after returning from HOLD mode. Consequently, piconet switches should not occur too often, since it could cause unnecessary wasted time slots. Conversely, a low switching frequency results in high forwarding delays over several hops. In simulation, the tradeoff between high switching frequency and low switching frequency will be analyzed in terms of throughput and delay.

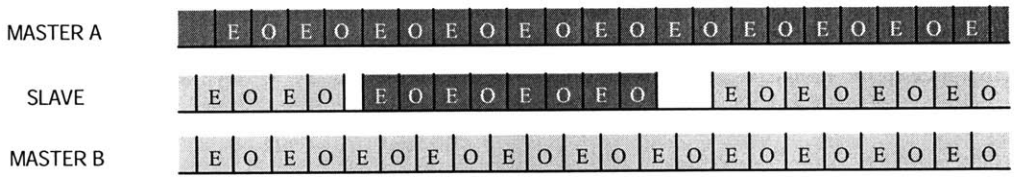

Figure 3-10. Piconet switching. Slave switches channel parameters from Master B to Master A and waits less than a slot to synchronize. A slave may take up to two time slots to synchronize with a master as seen with the Slave synchronizing with Master B.

3.3 Routing Policy

The single-chain relay simplifies the routing of packets, and the piconet switching dictates the timing of the routing. This research does not consider implementing any particular routing algorithm, such as Dynamic Source Routing [20] or Ad Hoc On-Demand Distance Vector Routing [27]. These algorithms which cope with mobility at the expense of bandwidth overhead are not suitable for Bluetooth since baseband packets are relatively very small and network broadcasts would burden the low bandwidth capacity of Bluetooth networks. [5, 6, 17]

In BlueRelay, the loop-free relay provides only one path for data packets to traverse. Nodes have either one or two connection links. For nodes that have two links, the incoming data arrives on the inbound link. If the data needs to be forwarded, it is queued in a buffer and awaits to be sent out on the other link, the outbound link. The designation of inbound link and outbound link is relative to the baseband packet. The nodes at the end of a relay have only one link, which acts as both an inbound and an outbound link.

When a baseband packet is received, the local node checks the data packet if the packet has arrived at its destination. If the local node is not the packet's destination, then the packet is forwarded on the outbound link.

When a packet is received at each hop, the node responds with an acknowledgement. The Bluetooth specification requires the receiver to acknowledge a packet arrival in the next time slot. If the receiving node does not respond with an acknowledgement, the last node that forwarded the packet then has the responsibility of notifying the source node of a failed transmission.

If the devices on the communication link have no additional data to send, the master will send a POLL packet and the slave device will send a NULL packet in response. After this POLL-NULL exchange, the master and slave will be silent for the remainder of the HOLD duration.

This naïve packet routing scheme includes a simple routing table that associates an outbound link with each node in the relay. BlueRelay, as implemented in simulation, does not support a broadcast policy to populate a routing table because routing algorithms are not a focus of this research. Instead,

BlueRelay requires each node to have prior knowledge of all the nodes in the relay before powering up. At startup, each node already has a pre-configured routing table containing all the nodes in the relay and their associated outbound link.

3.3.1 Intra-Piconet Routing

A packet that is routed through a master node is called intra-piconet routing. Intra-piconet packets arrive from one slave and are redirected to the other slave in the piconet. Only the active member address, AM_ADDR, in the packet header must be updated to reflect the packet's new destination within the piconet.

3.3.2 Inter-Piconet Routing

Inter-piconet routing involves a packet to be routed on one frequency-hopping sequence and forwarded on another frequency-hopping sequence. The bridge slave that manages this forwarding must modify the packet access code and the packet header before sending out the packet to a different piconet.

The access code must be modified because it is derived from the master address receiving the packet. Hence, the access code must be changed to reflect the Bluetooth device address, BD_ADDR, which is receiving the packet.

The header format also needs to be updated since it contains the active member address and the header error check field. Recall that the bridge slave maintains an active member address, AM_ADDR, for each piconet to connect. Thus, the AM_ADDR must be modified to reflect the bridge slave's respective AM_ADDR in the piconet that is receiving the forwarded data. The HEC also needs updating since it is generated by the upper address portion of the master's Bluetooth device address. Thus, the HEC field is recalculated with the appropriate BD_ADDR of the master that is receiving the forwarded data.

3.4 Expected Latency

The delay that a packet incurs at each hop is dependent on two factors: the hold duration and the two-slot overhead for the realignment of time slots. We do not account for the additional delay from the $T_{stagger}$ offset because it is assumed that piconet switching synchronization occurs in a relatively short time compared to the life of the network. Once piconet switching is synchronized, each master should be communicating with one bridge slave at any instant. With communication occurring in parallel, a packet can be forwarded to its destination as quickly as the piconet switching occurs.

Once piconet switching achieves steady state, then the packet latency is a function of the hop count, the hold duration, and a constant two-slot overhead for the realignment of time slots.

$$latency(n_{hold\_slots}, n_{hops}) = \left((n_{hops} - \tfrac{1}{2})n_{hold\_slots} + 2n_{hops}\right)(625\mu s / slot) \qquad (3.5)$$

In Equation 3.5, the number of HOLD time slots, n_{hold_slots}, is the largest contributor to packet latency. The two-slot overhead for the realignment of time slots becomes negligible when n_{hold_slots} is two magnitudes of order or larger. Essentially, latency is a linear function of the number of hops.

For example, a packet transmitted across three hops in a relay with piconet switching every 20 time slots experiences an estimated latency of 35 milliseconds. A packet transmitted across three hops in a relay with piconet switching every 500 time slots experiences a significantly longer delay of 0.785 seconds.

This simple comparison shows that choosing the appropriate hold time is important. For delay sensitive packets, a short hold time should be used to forward packets quickly as possible over multiple hops. Long hold times are advantageous, however, for bandwidth intensive applications. In the next section, the expected throughput is analyzed to compare the bandwidth offered by long hold times and short hold times.

3.5 Expected Throughput

The throughput in BlueRelay is a function of the hold time. Assuming piconet switching is synchronized in a relay, the expected throughput in a multi-hop relay can be expressed as follows:

$$throughput = \frac{floor\left(\frac{n_{hold\_slots} - 4}{slots_{pkt} + slots_{ack}}\right)(payload)}{2(n_{hold\_slots})(625\mu s)} \tag{3.6}$$

where n_{hold_slots} = the length of hold time in units of time slots,
$slots_{pkt}$ = the length of baseband packet in units of time slots,
$slots_{ack}$ = the length of an acknowledgement message, typically one time slot,
payload = the number of bits in the payload of the baseband packet.

Equation 3.6 accounts for the four slots of overhead associated with piconet switching. The floor function ensures that only a whole number of packets can be transmitted during the hold interval. The piconet switching, however, has the largest effect on the efficient utilization of bandwidth because the relay nodes are switching back and forth between its two connected nodes. The bridge slaves switching back and forth between two piconets. Likewise, the masters are switching back and forth between two bridge slaves. The throughput in the relay is one-half the available bandwidth. Hence, the expression in Equation 3.6 is divided by 2 to account for the piconet switching.

For a 20-slot hold time and a 5-slot packet, the expected throughput is 217kbps. For a 500-slot hold time and a 5-slot packet, the expected throughput is 355kbps. The simple comparison demonstrates that a long hold time achieves a higher throughput than a short hold time because the longer hold time dominates the overhead. For a 20-slot hold time, the overhead has a more noticeable effect per hold interval, which reduces the throughput by 40%.

Chapter 4

Results and Analysis

In this chapter, the performance of relay establishment and piconet switching is evaluated from the results of simulation and is compared to expected calculations. Simulations of the relay establishment process measure the link establishment delay, the relay establishment delay, and the number of control packets sent during the inquiry and page process. The number of hops in a relay is examined to determine its impact on the average throughput and on the average end-to-end latency. Analysis is also done to determine how piconet switching with various HOLD intervals affects packet delay and throughput.

The results show that individual point-to-point link establishment occurs within 4 seconds. Relay establishment can be achieved under 10 seconds. Inter-piconet communication is shown to offer 300 kbps throughput. The largest restriction on throughput is due to the bridge slaves having to switch back and forth between two piconets, which effectively reduces the available bandwidth by one-half. Piconet switching with 20-slot hold times imposes a 15-millisecond latency per hop. This is suitable for delay sensitive applications while long HOLD durations, such as 100 milliseconds or longer, are better suited for bandwidth intensive applications. The largest contribution to latency is the HOLD duration, not the overhead time slots incurred to negotiate the HOLD mode.

4.1 Bluetooth Extension to Network Simulator *ns-2*

To assess the performance of BlueRelay, the relay establishment and piconet switching are simulated in the network simulator *ns-2* using a Bluetooth simulation extension model. [15, 18] *ns-2* is an object oriented, discrete event driven network simulator developed at UC-Berkeley. The Bluetooth extension to *ns-2* was developed by IBM India Research Lab. [18] The extension to *ns-2* supports the features of the Bluetooth specification as discussed in Chapter 2. For example, time-division duplex, frequency-hopping, master-slave slot allocation, and device discovery are implemented according to the Bluetooth specification. The extension simulates the Bluetooth layers: Baseband, Link Control, Link Manager, L2CAP, and the host layer. Each layer has methods to interface with the layers above and below. Figure 4-1 shows the 5 layers, the interface methods, and message exchanges that IBM implemented in *ns-2*.

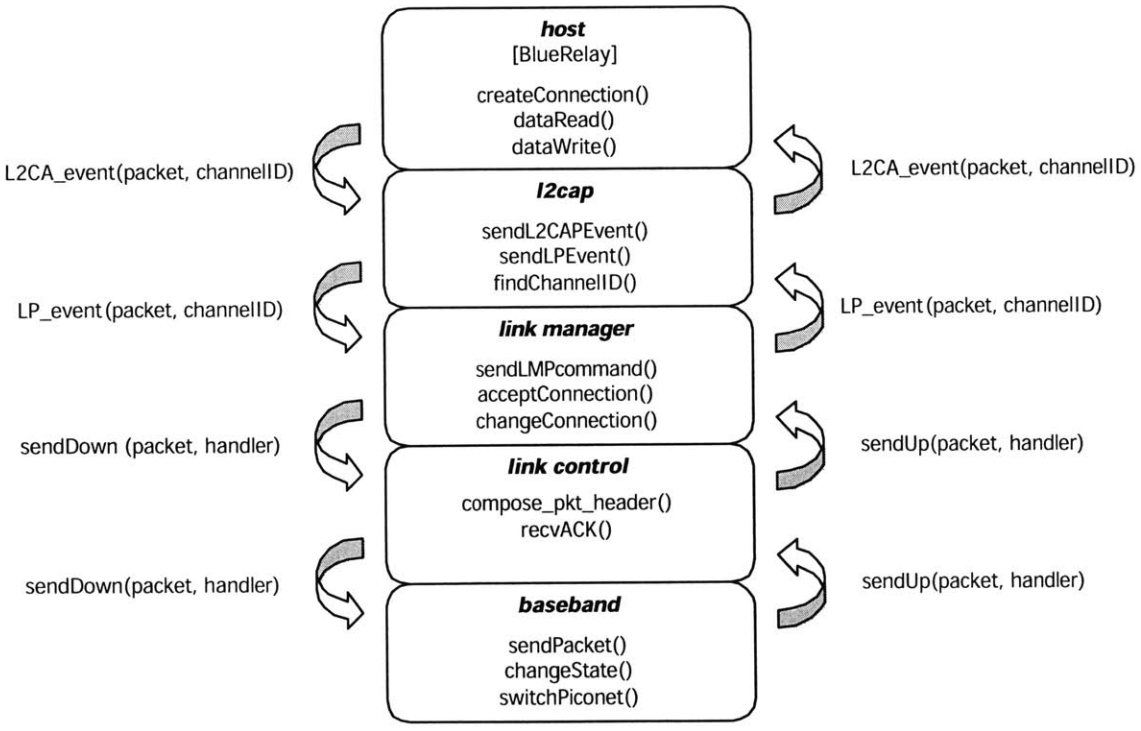

Figure 4-1. C++ Modules Emulating the Bluetooth Protocol Stack. Methods inside the block diagrams carry out tasks specific to that layer. Methods associated with arrows provide the interface to higher and lower layers. [18]

The host class was modified to support the simulation of BlueRelay. In the host class, slave nodes designated as a bridge slave are directed to establish connections with two masters, instead of just one master. A function was also added to the host class to schedule the piconet switching as described in Chapter 3. Each time a master node received a HOLD request, the overlapping time, Δt_{start_time}, is calculated. Δt_{start_time} is the difference in the time stamps of the two most recent HOLD start times. Based on Δt_{start_time}, the offset can be calculated and added to the start time of the lagging bridge slave. In the baseband class, a function was written to lookup an array of Bluetooth Addresses to determine the next-hop address for forwarding packets.

4.2 Simulation Scripts

To configure simulation scenarios of BlueRelay, Tcl scripts were written to include parameters about the network as summarized in Table 4-1.

Parameter	Description
NumDevices	The total number of devices in the network; all masters plus all slaves.
BD_ADDR	The Bluetooth Device Address for node.
Transport	The transport layer to be used: UDP.
Xposition	The x-coordinate for device (meters).
Yposition	The y-coordinate for device (meters).
Application	The traffic model used: CBR.
NumResponses	The maximum number of inquiry responses a master receives from slaves.
InqScanOffset	The backoff time slave waits after receiving its first inquiry message.
InqTimeout	The maximum time a master can spend in inquiry state.
IAC	The field which assigns a slave to a master.
HoldTime	The length of time device is to be in HOLD mode.
WaitToHold	The length of time device waits before commencing HOLD mode.
StartTime	The start time for a device.
SimulationTime	The duration of simulation run.

Table 4-1. Parameters for BlueRelay simulation scenarios.

When a Tcl configuration script is run, node instances are created. These nodes are immediately assigned their Bluetooth Device Address and designated either a master or a slave node.

Slaves are pre-assigned to connect to certain masters as specified in the Inquiry Access Code (IAC) field. For these simulation scenarios, the IAC fields must be assigned correctly to force slaves to connect to a master resulting in the desired single-chain network topology. The resulting topology is a relay with alternating master and slave nodes. Once the relay topology is formed, the piconet switching occurs in the manner as detailed in Chapter 3.

Each simulation is run with a channel capacity of 1Mbps and a radio transmission of 10 meters. Traffic flows are simulated with the Constant Bit Rate application model in *ns-2*. The maximum data payload of 339 bytes is transmitted per packet, which is the maximum size allowed by Bluetooth baseband packets that do not require segmentation and reassembly at the L2CAP layer. Each simulation is run for 120 simulation seconds. This duration is long enough for observing device discovery and 300 to 8000 rounds of piconet switching depending on the hold period. Scenarios ranging from two to eight nodes are simulated.

The output of the Bluetooth simulations shows each packet received during inquiry and paging procedures. Once the complete packet is received, the destination node, the end-to-end delay, packet size, and the simulation clock instance are reported. By processing these output files, the amount of time spent in inquiry and page mode for each device can be determined. From these numbers, we can derive the aggregate time taken to complete relay establishment. The time stamps of each received packet also can be processed to determine single hop delay, end-to-end delays, and data throughput.

4.3 Results of Relay Establishment

The performance of relay establishment is dependent on several factors. They include the inquiry processing time, the page processing time, the connection process for master nodes, and the connection process for slave nodes. [19] In this section, these factors are analyzed to determine the efficiency of establishing communication links and the overall relay establishment delay.

4.3.1 Establishment of Point-to-Point Communication Links

In Chapter 2, the inquiry and page process are discussed and the time for a round of inquiry and paging to complete is expected to be 6.4 seconds, according to the Bluetooth Special Interest Group. To verify this estimated number, simulations were run to determine how quickly a node could perform inquiry and page to establish a connection with another remote device.

The simulations were performed with the number of nodes ranging incrementally from two nodes to eight nodes. For each linear topology, the simulation output captured time stamps of when inquiry and page messages were received, as well as the number of messages that were sent. We define the link establishment delay as the time taken for a node to establish its first communication link with another node. This metric shows how quickly a node can establish a connection with another node. At the start of simulation, nodes are already designated as either a master or a slave. Therefore, nodes are assumed to start in their appropriate state, Inquiry or Inquiry Scan.

In these scenarios, slaves are configured to connect to only one master. The nodes in these simulations do not make a second connection with another master. Figure 4-2 shows the average time a slave takes to form a single connection with a master. The results of this simulation show that a slave takes ~4 seconds on average to connect to a master. This is significantly lower than the estimated time of 6.4 seconds.

One explanation is that a master node does not spend as long in Inquiry state as expected. Recall that during Inquiry, transmitting two trains requires a time elapse of 5.12 seconds. However, transmitting two full trains may be unnecessary. It is possible to end the Inquiry process earlier if a slave responds to the master during the first train, Train A. Then the master will never have to transmit on its Train B frequencies. This effectively reduces the time spent in Inquiry state by half.

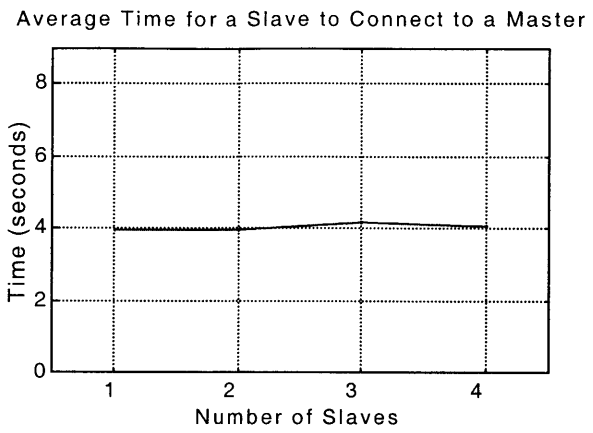

Figure 4-2. The average time taken for a slave to establish a connection to a master is 4 seconds in simulation. This is lower than the estimated time of 6.4 seconds.

Even if Train A is exhausted, the master will transmit on Train B frequencies for a short time. The reason is that the simulator does not introduce interference. Therefore, the slave should successfully receive an Inquiry message shortly after the master begins to cycle through its Train B frequencies. Recall that a train repeats every 10ms. Hence, the slave should receive an Inquiry message within 10ms. The slave will wait for some random backoff interval, up to a maximum 0.64 seconds. Then the slave will listen for a second Inquiry message from the master and then respond.

This explains why, in simulation, a slave does not take the full 5.12 seconds to respond to a master. Instead, a slave responds to a master's inquiry within 3 seconds. Since the page process takes an additional 1 second, this explains why the simulation outputs show that connection establishment delay is ~4 seconds, and not 6.4 seconds.

4.3.2 Establishment of Forwarding Links

The amount of time to establish one connection link $T_{connection}$ in simulation is 4 seconds. Since all relay nodes in BlueRelay must form two connection links, the time to establish two connections should be simply $2T_{connection}$.

To determine how long a slave establishes two connections, scenarios were configured such that each slave connected to two masters. Each slave listens for both masters simultaneously. The first connection to a master is regarded as the primary master. The slave will return to Standby and enter Inquiry Scan again to listen for its secondary master and establish a second connection.

Simulations show that the average time for a slave to connect to two masters is ~8 seconds, as shown in Figure 4-3. This result is reasonable since it takes 4 seconds for two remote devices to form a single connection.

In scenarios with more than one slave, two slaves will attempt to scan for the same master simultaneously. When two slaves do receive an inquiry message broadcasted from the same master, both slaves will backoff for some random interval of time to avoid contention. The random backoff interval ranges from 0 to 0.64 seconds, and it forces a slave to wait before it can resume scanning for inquiry messages from the same inquiring master again. Simulation results show that this backoff mechanism has a negligible effect on connection setup delay as more nodes are added to the relay.

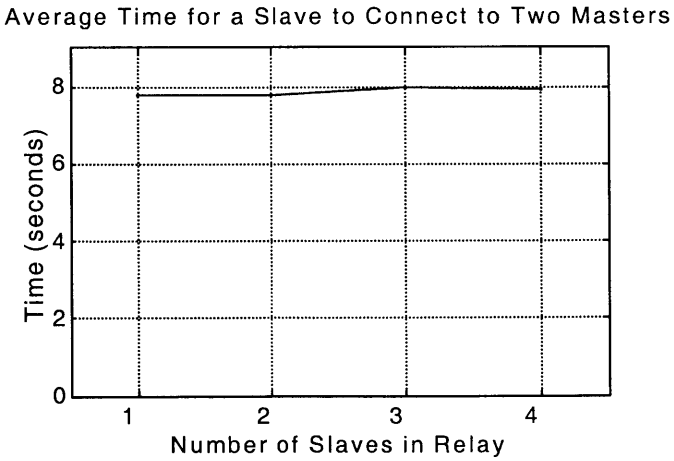

Figure 4-3. The average time taken for a slave to connect to two masters is 8 seconds.

Pre-assigning masters to slaves has a two fold purpose. It reduces the connection establishment delay and forces the desired topology to be created. If slave and master roles were not pre-assigned, then two nodes may happen to enter Inquiry. By being in the same state, the two devices will not discover each other. Instead, they will have to wait for the Inquiry state to timeout in 10.24 seconds at which point one device hopefully switches to Inquiry Scan for another round of device discovery to occur. Waiting for two devices to enter the correct complimentary states can potentially incur a long connection delay. Hence, the pre-assignment of roles removes this aspect of uncertainty during device discovery.

Moreover, pre-assignments force slaves have to wait for an inquiry message from a particular master. This requires slaves to respond to the appropriate masters, resulting in a network with a linear topology. Without this prior knowledge, slaves would be free to connect to whatever master, and the resulting topology would not be a relay.

Although the connection establishment delay remains constant for varying relay lengths, the amount of packets being transmitted over the air is not constant. Figure 4-4 shows that the total number of messages transmitted by all the masters in a network during the inquiry and page process is a function of the relay length. The total number of inquiry packets and page packets are average totals over 20 simulation runs.

Master nodes send significantly more inquiry packets than page packets due to the differences in function of the two processes. The purpose of the inquiry process is for the master to discover the existence of neighboring devices, their addresses, their clock offsets, and their page scanning parameters. This information facilitates the page process, such that a master can quickly synchronize with a known slave and create a connection. Essentially, inquiry messages are broadcasted in the hopes that a slave happens to be listening on the same frequency. Page messages, on the other hand, are addressed packets targeted at a particular a slave. Hence, less page messages are sent since the master is directing packets addressed to a destination while inquiry messages are broadcasted to unsynchronized

receivers. The difference in the number of packets sent during discovery reflects the amount of time spent in the inquiry and page process.

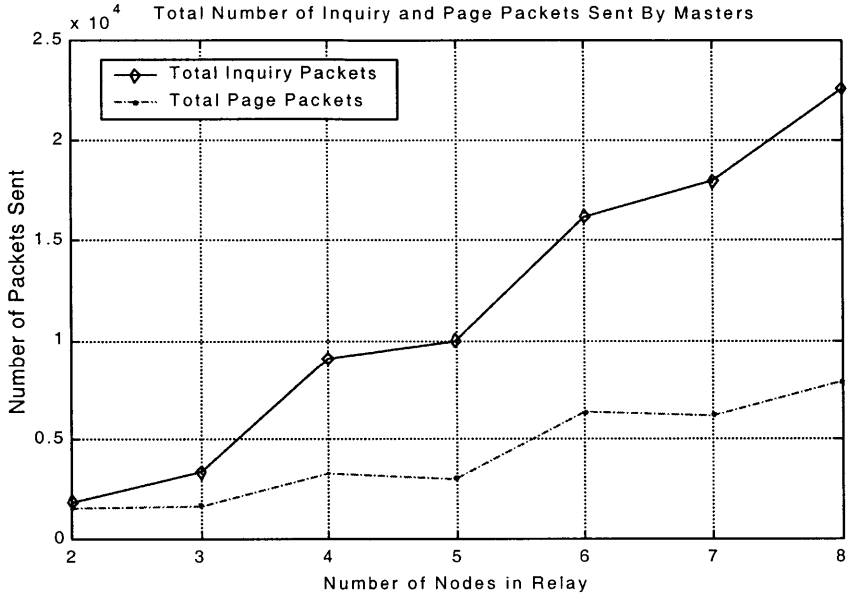

Figure 4-4. A comparison of the total number of inquiry packets and the total number of page packets sent by all the masters in the relay.

To see how the simulation results matched expectations for the number of Inquiry and Page packets sent, we refer to the Bluetooth Specification and Bluetooth SIG expected Inquiry and Page times. Recall from Table 2-1, a master is expected to spend 5.12s in Inquiry state and 1.28s in Page state. During the Inquiry and Page states, a master is generating 16 packets per 10 milliseconds. Thus, a master node generates 8,192 packets during the Inquiry state and 2,048 packets during the Page state.

For each master node in a relay, the number of Inquiry and Page packets should increase by a multiple of 8,192 and 2,038 respectively. Figure 4-4 shows that a jump in the number of messages occurs when a master is added to the relay for the 4-node, 6-node, and 8-node relay. However, the simulation results show that the number of Inquiry and Page packets sent are significantly lower than

estimates. This follows the fact that the time spent in Inquiry and Page state was 33% less in simulation than expected.

The graph also shows that a master sends more packets when the number of slaves is greater then the number of masters. For relays of 2 or 3 nodes, there is only one master node. It is reasonable that a master would send double the inquiry messages when trying to connect to two slaves instead of just one slave because two slaves are likely to be listening on different frequencies. Therefore, a master has to spend more time in Inquiry state to wait for the second slave to respond. For 4- and 5-node relays, the masters in the 5-node relay send more inquiry messages than in the 4-node relay. This is because the 5-node relay has an extra slave node to connect while the 4-node does not. The same behavior happens for 6- and 7-node relays. The addition of a slave to a relay requires a master to send more inquiry messages. When a master is added to a relay, the number of page messages as well as the number of inquiry messages increase because a master generates broadcast messages while slaves do not.

While masters generate thousands of packets during the link establishment process, slaves handle significantly far less messages because slaves do not broadcast messages. As shown in Figure 4-5, the number of inquiry and page messages handled by slaves is a linear function. Slaves generate packets on only two occasions: when an inquiry message's frequency matches its own scanning frequency; or when a page message arrives.

During the inquiry process, a slave will receive two inquiry messages and send one inquiry response message. A slave receives two inquiry messages because it ignores the first successfully received inquiry message and waits for a back-off time interval before scanning for the second inquiry message. During the page process, a slave will respond to the first successfully received page message. The total number of page messages and the total number of inquiry messages handled by a slave is a ratio of 2:3. The results of simulation as shown in Figure 4-5 matches expectation and demonstrate that the slaves operate correctly in simulation.

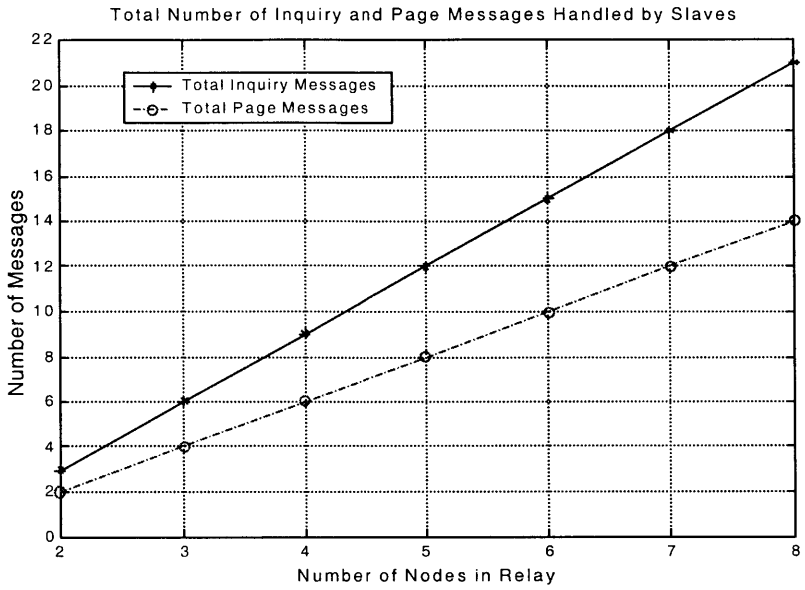

Figure 4-5. A comparison of the total number of inquiry messages and the total number of page messages the slaves in the relay handle.

4.3.3 Relay Establishment Delay

The time stamps associated with these inquiry and page messages are important for analysis. Each time a slave receives a successful inquiry message in simulation, we know when and which slave is about to change to the page state. We watch for that slave to successfully respond to a master with a page response packet to determine when a connection has been established.

The time spent in the Inquiry and Inquiry Scan state dominates the overall time to establish connections. This fact is illustrated in Figure 4-6, which shows the average time taken to form a complete relay. Twenty simulations were run for each relay length, ranging from two to eight nodes inclusive.

All nodes enter either Inquiry or Inquiry Scan state at the start of simulation. The last node to join the relay determines the relay formation delay. If the last node to join is an end node, then the relay formation delay is determined when it finishes its page process. For nodes responsible for making two

connections, the relay formation delay is determined by the node that took the longest time to establish two connection links. A relay is considered connected when the last free node has created its required connections successfully.

For the scenario of one master and one slave, a connection is established quickly in 4 seconds, which is consistent with the result from the experiments in Figure 4-2.

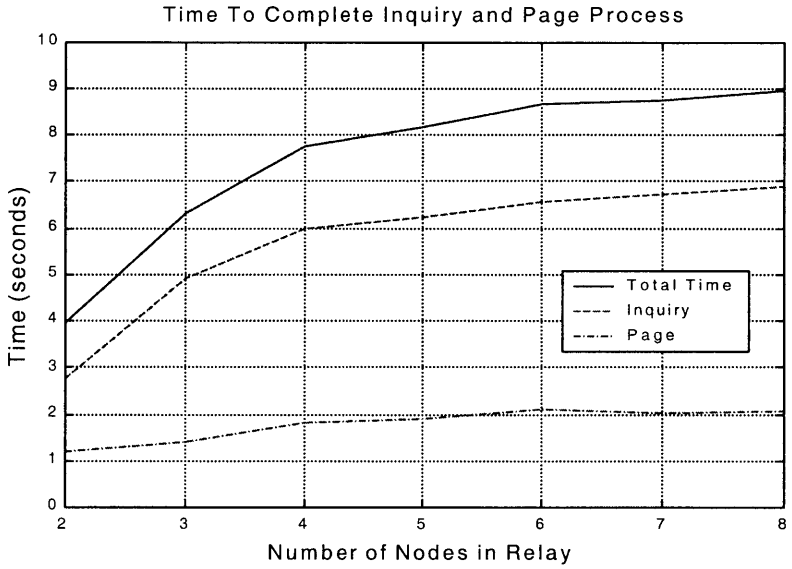

Figure 4-6. The average time elapse to complete the inquiry and page process for various length relays.

In the case of one master and two slaves, the two slaves connect to the master on average in 6 seconds. The slave-master-slave configuration is essentially a piconet, and the results suggest that connections in a piconet can be created faster. In the experiments associated with Figure 4-3, eight seconds were required for a slave to connect to two masters. This difference in connection delay for these two different configurations seems reasonable. A slave attempting to connect to two different masters is synchronizing with two different frequency-hopping sequences while the slaves in a piconet are synchronizing to only one frequency-hopping sequence. Synchronizing with a frequency-hopping

sequence is already a complex task. Therefore, synchronizing to two frequency-hopping sequences logically takes longer than just one sequence.

Figure 4-6 shows that the average relay establishment occurs under 10 seconds for relays with eight nodes or less. The relay establishment delay is longer than the average 8 seconds that a bridge slave takes to establish connections with two masters because the relay establishment delay measures when the last free node has connected successfully to the relay.

The relay establishment simulations demonstrate that a bridge slave can establish connections with two masters and participate in two piconets. This is significant for enabling inter-piconet communication and forwarding data packets over multiple hops.

4.4 Results of Piconet Switching

In evaluating the effects of piconet switching, we ran simulations to measure how piconet switching affects latency and throughput in a relay. Obviously, end-to-end latency is a function of hop count. Through simulation, however, the HOLD duration is shown to dominate the latency. The overhead associated in negotiating the HOLD mode is negligible for a long HOLD duration. Therefore, an optimal length of HOLD time is critical in reducing communication latency between two node pairs. The simulation results also show throughput ranges from 180 kbps to 300 kbps depending on the HOLD duration.

4.4.1 Single-Hop Data Rate

To determine the bandwidth of a single-hop transmission, one-directional and bi-directional flows between two devices were simulated in an 8-node relay. Each CBR application with UDP as the transport was started between a master and a bridge slave after the 8-node relay had been fully established. Three different packet types were used for data transmission: 1-slot, 3-slot, and 5-slot packets. Each bar in Figure 4-7 represents 10 simulation runs. The piconet switching, the number of flows, and the packet size all have an effect on throughput.

Single-directional flows have a higher throughput than bi-directional flows due to the nature of the time-division duplex scheme. For one-directional flows, a source blasts multi-slot data packets in one direction and receives a single-slot acknowledgement packet from the destination node. Hence, the distribution of slots is asymmetric. In bi-directional flows, the two nodes have to timeshare the TDD channel equally. Therefore, the master-to-slave and slave-to-master slots are symmetric and the available bandwidth in the TDD channel is equally shared between the master and slave thereby reducing the throughput.

For single-slot packets, one-directional and bi-directional throughputs are the same. Multi-slot data packets, however, benefit from one-directional flows by sending more slots of data per DATA-ACK cycle, thus achieving a higher throughput than bi-directional flows.

The single-hop throughput in a relay is roughly one-half the bandwidth available in a piconet. In the relay, the piconet switching scheme effectively cuts the available bandwidth in half due to the periodic switching back and forth between two piconets. The switching forces a slave to split its time spent in each piconet. Thus, the single-hop data transmission between a master and a slave is put on hold half of the time so that the slave can leave to join another piconet.

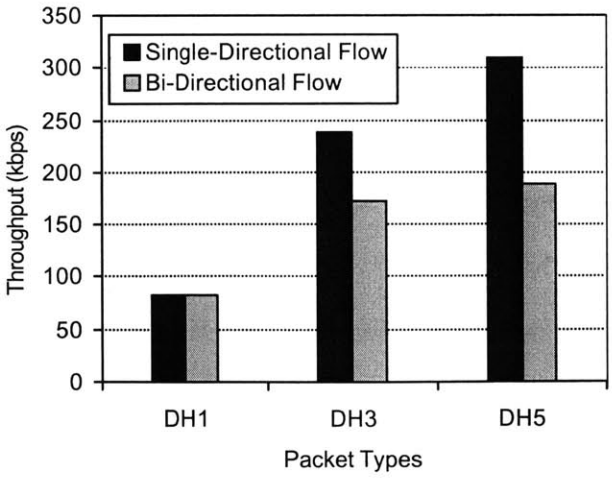

Figure 4-7. Comparison of Half-Duplex and Full-Duplex Data Rate over Single Hop for 1-, 3-, and 5-slot packets.

To see how simulation results match expectations, we refer to the Bluetooth Specification for single-directional and bi-directional data rates in a piconet. For single-directional flows, the expected data rates are 723 kbps, 585 kbps, and 172 kbps for 5-slot, 3-slot, and 1-slot packets respectively. For bi-directional flow in a piconet, the expected data rates are 434 kbps, 390 kbps, and 172 kbps for 5-slot, 3-slot, and 1-slot packets respectively. Ideally, all these values should be reduced by one-half when piconet switching is taken into account.

From simulation, the single-directional and bi-directional throughputs are less than the expected data rates. In simulation, the single-directional throughputs are 310kbps, 238kbps, and 82kbps. The bi-directional throughputs are 188kbps, 176kbps, and 82kbps. The overhead can be attributed to the following factors in the simulator. A master node randomly leaves the *Connected* state to return to *Inquiry* state to inquire for more slave nodes. This is a Bluetooth functionality that the IBM extension implemented in the simulator. Also, each received packet is randomly dropped according to a probability function that the Bluetooth simulator applies to determine the forwarding error rate vs. distance. The wasted time slots for negotiating a HOLD mode also contribute to the overhead. These factors contribute to the reduction in throughput. However, the piconet switching scheme is, by far, the largest overhead incurred, reducing the available throughput by one-half.

4.4.2 Multi-Hop Data Rate

In this section, we analyze how the number of hops in a relay affects the available throughput. A range of single-hop to 7-hop relays is simulated. A single CBR application with UDP as the underlying transport protocol transmits data from one end of the relay to the other end using 1-slot, 3-slot, and 5-slot packets. The HOLD duration is set at 100 time slots, so a slave spends 100 time slots per piconet as it switches back and forth between piconets.

Just as in the single-hop case, the available bandwidth is effectively reduced by one-half for multi-hop transmissions due to piconet scheduling. The bridge slaves that are responsible for forwarding data must spend half its time in one piconet to receive packets and the other half in another

piconet to forward the data. Hence, multi-hop throughput is one-half the available bandwidth. This assumes that all the nodes in the relay are switching piconets in synchronization such that all nodes are either receiving or forwarding data simultaneously.

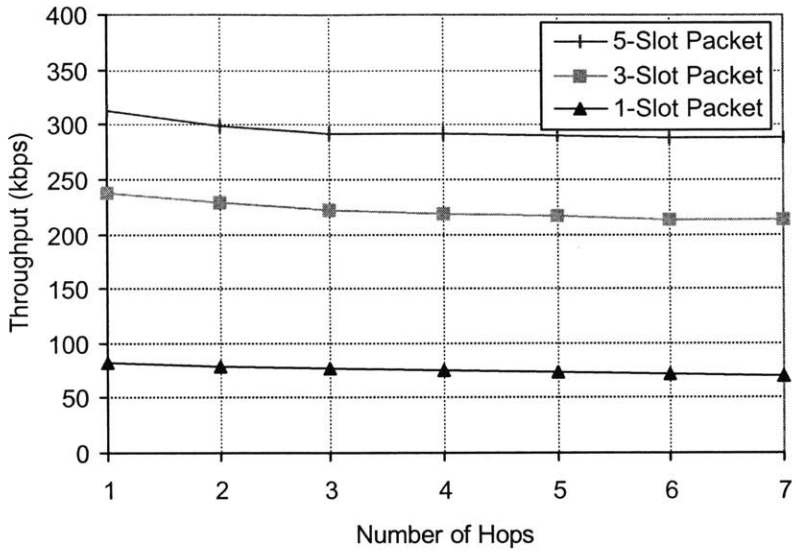

Figure 4-8. Average throughput of single CBR flow for various packet types.

The three trends in Figure 4-8 show the throughput for 1-slot, 3-slot, and 5-slot packets. Ideally, the throughput for 1-slot, 3-slot, and 5-slot packets should be 86kbps, 292kbps, and 362kbps, respectively. However as the relay increases, each additional hop increases the delay for a packet to reach its destination. If the next-hop happens to be a master, the master may randomly leave the *Connected* state to inquire for more slaves as specified by the Bluetooth Specification. Also, packets can be dropped at each node based on a forwarding error rate vs. distance probability function in the simulator. Hence, simulation runs show that the throughput decreases for all three trends as the relay increases in length.

4.4.3 End-to-End Latency

Latency is the time elapse from when a data packet is presented at the source node until it reaches its destination. To see how piconet switching affects the end-to-end latency of packets, a single CBR application using UDP as transport is simulated in an 8-node relay. To reduce the latency perceived by each packet, only single-slot packets are transmitted but not the multi-slot size packets. We vary the hold duration and the number of hops between the source and destination of the CBR application. Ten simulations were run against each hold time of 20, 50, 100, 200, and 500 slots.

The delays of packets were measured only when the piconet switching in the relay achieved steady state. Piconet switching in a relay achieves steady state depending on the length of the hold time. According to Equation 3.4, a 500-slot hold time requires at maximum of 8.5 seconds to synchronize piconet switching in an 8-node relay. For the shorter hold times, less than 8.5 seconds is needed to synchronize piconet switching. In simulation, piconet switching synchronization occurred within 5 seconds or less.

Packets that are transmitted, while piconet switching is being synchronized in the relay, experience greater delays than packets that are transmitted after piconet switching has achieved steady state. The increased delays are due to the additional stagger time, $T_{stagger}$, that is added to the hold start_time. Although the $T_{stagger}$ helps to correct the synchronization of piconet switching, the stagger time delays the start_time and forces packets to wait longer before being forwarded to the next hop. Hence, packets incur a longer delay than it would if the piconet switching were synchronized.

Simulation shows that once piconet switching achieves synchronization, then the packet latency increases linearly as the hop count increases. In Figure 4-9, the results from simulation are plotted against expected trends from Equation 3.5 for the different hold lengths. The latency values from simulation closely match expectations. However, some of the data points are slightly larger than expected. This can be attributed to the two-slot overhead for negotiating the HOLD request and the fact that a master node is allowed to momentarily leave the *Connected* state to enter the *Standby* state. Another source of delay occurs just before the HOLD mode is initiated. If less than two slots are

available before the HOLD mode is scheduled to start, there are not enough time slots to send a single-slot packet and receive a single-slot acknowledgement. In such situations, a single-slot packet is not sent to prevent the packet from being dropped when piconet switching occurs. Instead, the packet misses the current round of transmission. It is queued and it waits for the next round of transmission.

Despite these sources of delay, most of the packet latency is imposed by the hold time. The simulation results do follow the expected trends and demonstrate that different hold lengths have varying effects on packet latency. The effects of long hold lengths are more noticeable as the hop count increases since a packet incurs a delay of one hold period per hop. This is due to packets being queued at the forwarding node until the next round of piconet switching occurs.

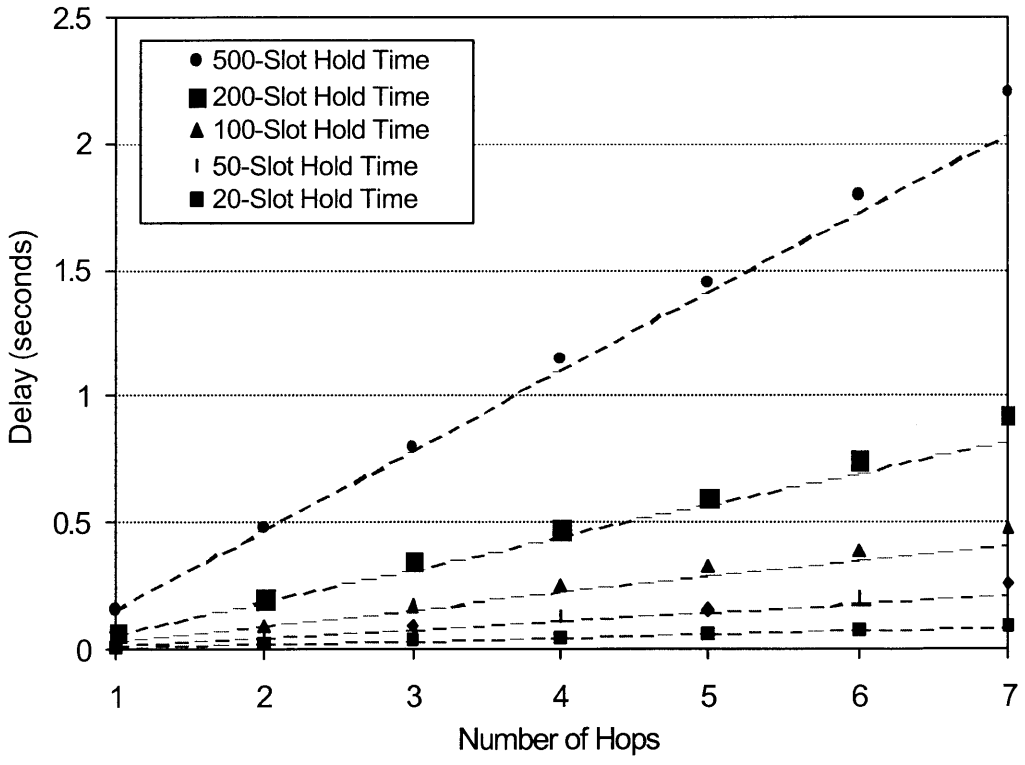

Figure 4-9. Average End-To-End Latency of a CBR Packet in an n-hop Relay.

For relays operating with a 20-slot hold time, packets experience an estimated 13-millisecond delay per hop. The simulation values match expectation for the 20-slot trend. For example, a packet transmitted over five hops should incur an estimated 62 milliseconds according to Equation 3.5. Simulation shows that the 5-hop latency is 65 milliseconds, which matches calculation. Applications such as real time audio, which have strict delay requirements and are sensitive to delay, would benefit from a short hold time. Significant end-to-end delays, even if infrequent, can render a time sensitive packet useless when it arrives too late at its destination. Therefore, delay sensitive applications would perform well with short hold times, such as 20 time slots.

Piconet switching with long hold intervals is better suited for bandwidth intensive applications, such as file transfers. Bandwidth intensive applications do not have worst-cast delay requirements and can cope with longer packet delays. Therefore, bandwidth intensive applications employ longer hold times to minimize the overhead cost incurred during piconet switching.

4.4.4 Effect of the HOLD Mode

To understand how varying the hold period can affect the overall bandwidth, we re-run the simulation of a single CBR application with UDP as the underlying transport protocol in an 8-node relay. Instead of using DH1 packets in these trials, DH5 packets, the packet with the largest available payload, are used to maximize the throughput. As before, hold times of 20, 50, 100, 200, and 500 slots are simulated.

Figure 4-10 shows the average throughput for various hop distances. By examining only packets that are generated after the piconet switching in the relay has achieved steady state, we obtain a true measure of the throughput for each hop. The single hops have a higher throughput because the destination or source can be a slave end node. For slaves at the end of a relay, they do not coordinate piconet switching. Consequently, slaves at the end of a relay achieve a higher throughput than bridge slaves do because bridge slaves lose bandwidth to the overhead of piconet switching.

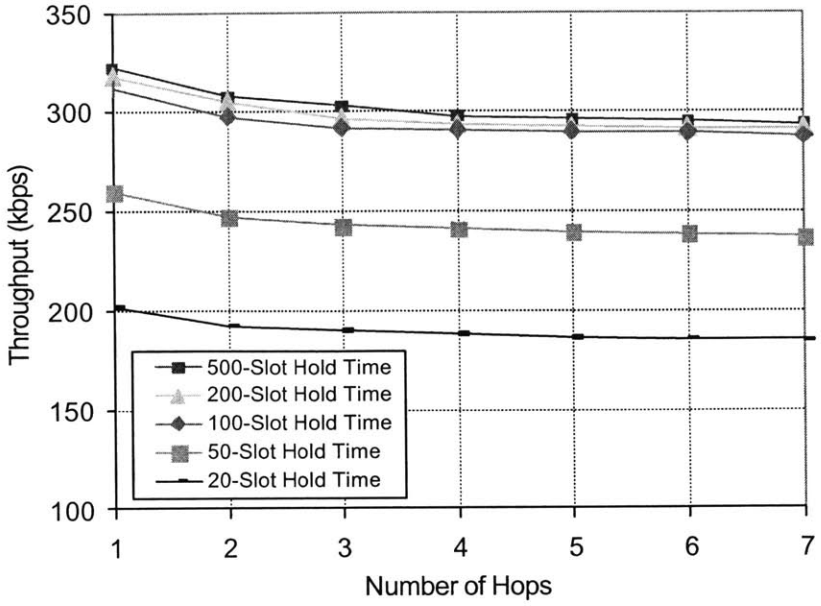

Figure 4-10. Average throughput for single CBR flow for various hold durations.

Examining the trends, the 20-slot hold interval has the lowest average throughput, 180kbps, of all the hold intervals simulated. On the opposite spectrum, the 200- and 500-slot hold times provide the highest average throughput, 290kbps. Interestingly, the trends show that increasing the hold duration beyond 100 time slots does not significantly improve the average throughput. This is because long hold durations dominate the overhead cost of piconet switching, making the overhead negligible. To recall, piconet switching includes negotiating the HOLD mode and realigning the time slots. Thus the overhead cost consists of one slot for requesting the HOLD mode, one slot for responding to accept the HOLD mode request, and two slots of guard time to allow a slave and master to realign their slot boundaries. In all, an overhead of four slots is negligible when the hold times are 100 slots or more.

For a short hold period such as 20-slots, however, the overhead cost associated with piconet switching has a more noticeable effect on throughput. The four slots of overhead significantly reduce

the number of available time slots in a 20-slot interval for transmitting data. Consequently, the 20-slot hold time is not utilized well.

Equation 3.6 estimates the throughput to be 217kbps for 20-slot hold intervals. Simulation shows that the 180kbps throughput is 83% of the estimate. Examining the other trends, the estimate throughput for 50-slot, 100-slot, 200-slot, and 500-slot hold times are 303kbps, 347kbps, 347kbps, and 355 kbps respectively. The simulation results are all approximately 80% of the estimates. The calculated throughput does not take into account the fact that the simulator drops received packets according to a forwarding error rate vs. distance probability function. The Bluetooth simulator also simulates the master node randomly leaving the *Connected* state to enter *Inquiry* state to inquire for more slave nodes.

Although the simulation throughputs are lower than estimates, the trends in Figure 4-10 demonstrate how frequent piconet switching would not be suitable for bandwidth intensive applications. Simulation shows that the 20-slot hold interval is 60% of the throughput offered by the 100-slot or longer hold intervals.

In summary, there are tradeoffs for both frequent and infrequent piconet switching. Short hold durations have low throughput and low latency. High hold durations achieve higher throughput but incur higher latencies. Neither extreme of hold durations offers an optimal piconet switching for efficient utilization of bandwidth. Therefore, choosing the appropriate hold mode for efficient piconet switching depends on the type of application traffic, bandwidth intensive or delay sensitive.

This page intentionally left blank.

Chapter 5

Conclusions

This thesis proposes and evaluates BlueRelay, a novel protocol for relay establishment and piconet switching of a multi-hop, single-chain, wireless network using existing Bluetooth features. The current Bluetooth specification, however, limits Bluetooth devices to only basic networks called piconets. The challenges in extending the piconet to support inter-piconet communication are unique to Bluetooth and not seen in traditional wired networks. Specifically, the strict master-slave mechanism, the time-division duplex scheme, and the frequency-hopping sequence all create complexity in establishing a multi-hop network.

5.1 Summary

The proposed relay establishment process leverages defined Bluetooth mechanisms for device discovery. The inquiry and page procedures are used to discover remote devices nearby, determine the

frequency-hopping sequence, and establish a point-to-point communication link. Master nodes that act as a forwarding relay are required to form connections with two slaves. Each bridge slave is also explicitly required to form communication links with two masters and maintain clock information about both masters in order to alternately participate in the two piconets. The bridge slaves in this proposed relay topology are responsible for piconet switching, and masters are responsible for synchronizing piconet switching.

Since relay nodes must timeshare between two neighboring devices, the proposed piconet switching scheme employs the HOLD mode feature of Bluetooth in order to schedule the forwarding of packets between piconets. The HOLD mode start time is adjustable to allow piconet switching to align itself throughout a relay so that all nodes in the relay can receive and forward data in parallel. The forwarding of packets is distributed where masters and slaves are responsible for forwarding data transmissions with their next-hop neighbors.

The proposed relay establishment process and piconet switching scheme were evaluated in a Bluetooth simulator in *ns-2*. The Bluetooth simulator, developed by IBM, included most aspects of the Bluetooth protocol stack. The simulation results demonstrate that the relay topology and piconet switching can be implemented within the existing Bluetooth specification. The relay establishment as proposed has a low establishment delay of 10 seconds. This is an acceptable delay for wireless network scenarios such as an office board meeting, whereas it would be inappropriate for an automated tollbooth where vehicles are passing through in a matter of milliseconds.

The performance of the piconet switching is dependent on the HOLD mode duration and the type of packet transmitted. The simulation trials demonstrate that a 100-slot hold interval or longer achieve an average throughput of 280-300 kbps. On the other end of the spectrum, a 20-slot hold interval achieves a throughput of 180kbps, or 60% of the throughput offered by the long hold intervals. The experiments confirmed that the hold time is the largest contributor to packet latency. Hence, long hold times are not suitable for delay sensitive applications. Instead, a short hold time of 20-slots incurs low end-to-end packet delays of approximately 15ms per hop. This low latency is appropriate for delay

sensitive applications. Short hold intervals, however, increase the frequency of piconet switching as well as the associated overhead cost. As a result, piconet switching scheme that employs the shortest possible hold interval reduces the effective throughput by as much as 40%. Hence, the choice in hold interval depends on whether the nature of the application is bandwidth intensive or delay sensitive.

5.2 Future Work

BlueRelay is a customized solution for linear topologies. It does not support true interpiconet communication called scatternets. However, the results from this thesis provide insight as to the average delay and packet latency for multiple hop Bluetooth networks.

The proposed relay establishment and piconet switching scheme can be extended to scatternets. If each master node were to connect to one or more slaves, the BlueRelay topology would be a true scatternet. The relay would act as a backbone for the slaves acting as leaf nodes. Bridge slaves would act as the forwarding nodes and the master would allocate its bandwidth among the bridge slaves and piconet slaves. The bridge slave would employ piconet switching, maintain two clock offsets, and request the HOLD mode just as described in BlueRelay.

With more slaves in a piconet, however, a master has more than one inbound and outbound links to manage. An intrapiconet scheduling needs to be implemented that gives the bridge slaves more priority over the other slaves. An adaptive scheduler that monitors the communication activity of each slave would impart fairness and poll slaves that have shown more activity on the channel. [14, 21] Since bridge slaves would naturally be the most active slaves in any piconet due to its function as a forwarding node, an adaptive scheduler would proactively poll the bridge slave more frequently.

The piconet switching in BlueRelay needs modification because it does not scale well as the number of hops between a source and destination increases. The piconet switching as described in BlueRelay schedules period hold intervals. As revealed in simulations, frequent piconet switching increases overhead and significantly reduces throughput. In contrast, infrequent piconet switching

wastes bandwidth when a bridge slave is participating in a piconet with no data to transmit or receive. In such cases, a bridge slave should be able to leave the piconet to join another piconet that may have data to transfer.

A more robust piconet switching schedule would be able to adjust the duration of the HOLD mode for each piconet switch. BlueRelay already adjusts the start time of the HOLD mode to align piconet switching throughout the relay. By adjusting the hold interval, throughput would be further increased. If bridge slaves could maintain a moving average of the traffic rate with both masters, a bridge slave can schedule the future hold duration with each piconet based on past activity. This would be a better utilization of bandwidth and effectively increase the average throughput.

These enhancements to BlueRelay are possible future areas of research. All the mechanisms are defined for an efficient transfer of data packet and switching of piconets at the baseband level. However, the Bluetooth specification lacks sufficient detail in the higher layers to support inter-piconet communication. Until the next version of the Bluetooth specification addresses the support for scatternet communication, the functionality will have to be managed at the host layer for now.

5.3 Conclusion

The ideas presented here can be extended to scatternet designs. The results of this thesis are a part of ongoing academic and industry research to extend Bluetooth to support inter-piconet communication. There has been much discussion on the aspects of piconet performance. Presently, there are few designs proposed for Bluetooth to operate in a large ad-hoc network. However, this is a challenging problem due to the inherent properties of Bluetooth, namely the centralized time-division duplex scheme and frequency-hopping sequence.

The goal of this thesis was to investigate the feasibility of using defined Bluetooth features to support a multi-hop relay. Simulations in *ns-2* demonstrated that the relay establishment process and piconet switching of BlueRelay can be implemented within the Bluetooth specification. The results are initial findings on the performance of multi-hop communication for Bluetooth devices. The

performance values demonstrate that Bluetooth devices can potentially support a scatternet and can forward packets beyond the communication range of the source's transmitter.

This page intentionally left blank.

References

[1] "Specification of the Bluetooth System Version 1.1," http://www.bluetooth.com/, February 2001, Bluetooth Special Interest Group.

[2] Brent A. Miller and Chatschik Bisdikian, *Bluetooth Revealed: The Insider's Guide to an Open Specification for Global Wireless Communications*, Prentice Hall, 2001.

[3] Theodoros Salonidis, Pravin Bhagwat, Leandros Tassiulas, and Richard LaMaire, "Distributed Topology Construction of Bluetooth Personal Area Networks," in Proceedings IEEE INFOCOM, Anchorage, AK, April 2001, http://www.ieee-infocom.org/2001/paper/785.pdf.

[4] Godfrey Tan, Allen Miu, John Guttag, and Hari Balakrishnan, "Forming Scatternets from Bluetooth Personal Area Networks," MIT Laboratory for Computer Science, October 2001. MIT-LCS-TR-826. http://citeseer.nj.nec.com/515523.html.

[5] Pravin Bhagwat and Adrian Segall, "A Routing Vector Method (RVM) for Routing in Bluetooth Scatternets," in 6th IEEE International Workshop on Mobile Multimedia Communications (MOMUC), San Diego, CA, November 1999, http://citeseer.nj.nec.com/bhagwat99routing.html.

[6] W. Heinzelman, A. Chandrakasan, and H. Balikrishnan, "Energy-Efficient Routing Protocols for Wireless Microsensor Networks," in Proceedings 33rd Hawaii International Conference on System Sciences (HICSS 2000), January 2000, www.ee.ucla.edu/~curts/papers/MILCOM01.pdf.

[7] Ken Stack, "Bluetooth Piconet and Scatternet," http://www.anywhereyougo.com, November 2000.

[8] Cathal McDaid, "Infotooth Bluetooth Tutorial: Protocol Stack, Radio, Baseband, LMP, HCI, L2CAP, RFCOMM, SDP, Profiles and more," http://www.palowireless.com/bluetooth.

[9] Robert C. Dixon, *Spread Spectrum Systems with Commercial Applications*, John Wiley & Sons, Inc., 1994.

[10] Lakshmi Ramachandran, Manika Kapoor, Abhinanda Sarkar, and Alok Aggarwal, "Clustering Algorithms for Wireless Ad Hoc Networks," in *Proceedings of Dial M for Mobility 2000* (held in conjunction with MobiCOM 2000), 2000.

[11] Theodoros Salondis, Pravin Bhagwat, and Leandros Tassiulas, "Proximity Awareness and Fast Connection Establishment in Bluetooth," IEEE 2000, pp. 141-142, http://opensource.nus.edu.sg/projects/bluetooth/others/proximity.pdf.

[12] Stefan Zurbes, Wolfgang Stahl, Kirsten Matheus, and Jaap Haartsen, "Radio Network Performance of Bluetooth," IEEE 2000, pp. 1563-1567, http://opensource.nus.edu.sg/projects/bluetooth/others/btradioperf.pdf.

[13] Per Johansson, Niklas Johansson, Ulf Korner, Johannes Elg, and Goran Svennarp, "Short Range Radio Based Ad-Hoc Networking: Performance and Properties," *1999 IEEE International Conference*, pp. 1414 -1420 vol.3, http://opensource.nus.edu.sg/projects/bluetooth/others/icc99-bt1.pdf.

[14] M. Shreedhar and G. Verghese, "Efficient Fair Queueing Using Deficit Round Robin," In *Proceedings of ACM SIGCOMM*, Boston, August 1995, pp.231-242, http://www.stanford.edu/class/ee384x/Papers/shreedhar.pdf.

[15] ns-2 Network Simulator, http://www.isi.edu/vint/nsnam/, 2000.

[16] Jennifer Bray and Charles Sturman, *Bluetooth: Connection Without Cables*, Prentice Hall, 2001.

[17] Josh Broch, David Maltz, David Johnson,Yih-Chun Hu, and Jorjeta Jetcheva, "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols," in the *Proceedings of the Conference on Mobile Computing and Networking*, Dallas, TX, 1998, http://citeseer.nj.nec.com/broch98performance.html.

[18] Bluetooth Extension to *ns-2*, http://oss.software.ibm.com/developerworks/opensource/bluehoc/, 2001.

[19] Ching Law, Amar Mehta, and Kai-Yeung Siu, "Performance of a New Bluetooth Scatternet Formation Protocol," in the *ACM Symposium on Mobile Ad Hoc Networking and Computing*, Long Beach, CA, October 2001, http://citeseer.nj.nec.com/468009.html.

[20] David B Johnson and David A Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," *Mobile Computing*, Volume 353, 1996, http://citeseer.nj.nec.com/johnson96dynamic.html.

[21] Haiyun Luo, Paul Medvedev, Jerry Cheng, and Songwu Lu, "A Self-Coordinating Approach to Distributed Fair Queueing in Ad Hoc Wireless Networks," *INFOCOM 2001*, pp.1370-1379, http://citeseer.nj.nec.com/article/luo01selfcoordinating.html.

[22] Timothy J. Shepard, "A Channel Access Scheme for Large Dense Packet Radio Networks," in the *Proceedings of ACM SIGCOMM*, Stanford University, August 1996, ftp://ftp.tapr.org/ss/shepard.aug96.pdf.

[23] Timothy J. Shepard, "Decentralized Channel Management in Scalable Multi-Hop Spread-Spectrum Packet Networks," MIT Laboratory of Computer Science 1995, MIT-LCS-TR-670, ftp://ftp.tapr.org/ss/MIT-LCS-TR-670.pdf.

[24] Sumit Garg, Manish Kalia, and Rajeev Shorey, "MAC Scheduling Policies for Power Optimization in Bluetooth: A Master Driven TDD Wireless System," IBM India Research Laboratory, 2000, http://www.research.ibm.com/irl/projects/pervasive/vtc00_4.pdf

[25] Theodoros Salonidis, Pravin Bhagwat, Leandros Tassiulas, and Richard LaMaire, "Distributed Topology Construction of Bluetooth Personal Area Networks," IEEE Infocom 2001, http://www.ieee-infocom.org/2001/paper/785.pdf.

[26] Jaap Haartsen, "The Bluetooth Radio System," *IEEE Personal Communications*, February 2000, http://www.cs.huji.ac.il/~postPC/Wireless_and_Bluetooth/Haartsen_Radio2000.pdf.

[27] Charles E. Perkins and Elizabeth M. Royer. "Ad hoc On-Demand Distance Vector Routing," in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, LA, February 1999, pp. 90-100, http://www.cs.ucsb.edu/~eroyer/txt/aodv.ps.

[28] David Kammer, Gordon McNutt, Brian Senese, Jennifer Bray. *Bluetooth: Application Developer's Guide: The Short Range Interconnect Solution*. Rockland, MA: Syngress Publishing, Inc., 2002.