BLUETOOTH AUDIO AND VIDEO STREAMING ON THE J2ME PLATFORM

Submitted in fulfilment
of the requirements of the degree of

MASTER OF SCIENCE

of Rhodes University

Curtis Lee Sahd

Grahamstown, South Africa
September 2010

Abstract

With the increase in bandwidth, more widespread distribution of media, and increased capability of mobile devices, multimedia streaming has not only become feasible, but more economical in terms of space occupied by the media file and the costs involved in attaining it. Although much attention has been paid to peer to peer media streaming over the Internet using HTTP and RTSP, little research has focussed on the use of the Bluetooth protocol for streaming audio and video between mobile devices. This project investigates the feasibility of Bluetooth as a protocol for audio and video streaming between mobile phones using the J2ME platform, through the analysis of Bluetooth protocols, media formats, optimum packet sizes, and the effects of distance on transfer speed.

A comparison was made between RFCOMM and L2CAP to determine which protocol could support the fastest transfer speed between two mobile devices. The L2CAP protocol proved to be the most suitable, providing average transfer rates of 136.17 KBps. Using this protocol a second experiment was undertaken to determine the most suitable media format for streaming in terms of: file size, bandwidth usage, quality, and ease of implementation. Out of the eight media formats investigated, the MP3 format provided the smallest file size, smallest bandwidth usage, best quality and highest ease of implementation. Another experiment was conducted to determine the optimum packet size for transfer between devices. A tradeoff was found between packet size and the quality of the sound file, with highest transfer rates being recorded with the MTU size of 668 bytes (136.58 KBps). The class of Bluetooth transmitter typically used in mobile devices (class 2) is considered a weak signal and is adversely affected by distance. As such, the final investigation that was undertaken was aimed at determining the effects of distance on audio streaming and playback. As can be expected, when devices were situated close to each other, the transfer speeds obtained were higher than when devices were far apart. Readings were taken at varying distances (1-15 metres), with erratic transfer speeds observed from 7 metres onwards. This research showed that audio streaming on the J2ME platform is feasible, however using the currently available class of Bluetooth transmitter, video streaming is not feasible. Video files were only playable once the entire media file had been transferred.

Acknowledgements

First and foremost I would like to thank Lord Jesus Christ for guiding me through this thesis, and giving me the opportunity and the ability to undertake this research. I cannot thank you enough for everything you have done for me and continue to do for me. I would like to thank my parents, Mark and Lyndee, my sisters Sian and Simone, and my Blondie Bluu (Natasha Stevenson) for constantly encouraging me, and standing by me through thick and thin. Thank you for everything you have done for me, and sorry about all the times I was sitting in the room "just finishing one last bit of code", I really appreciate your patience. Thank you Sylvester (Wessie), Jado, Ray, Terence (Foxy), Taka, Andre, and Mikey for always having a keen interest in my research, and listening to me throwing ideas around (Some acceptable, and others outrageous - Many of which involved a hammer or a two story building and my laptop :) I would like to thank my sponsors, the Telkom Centre of Excellence at Rhodes University, funded by Telkom SA, Business Connexion, Verso Technologies, THRIP, Stortech, Tellabs and the National Research Foundation. Last but not least, I would sincerely like to thank my supervisor, Dr. Hannah Thinyane, for always giving me the benefit of the doubt and allowing me to carry on working even though at times I didn't have very much to show for it. Thank you Hannah for being the best supervisor anyone could wish to have. "Esse Quam Videri" - To be rather than to seem to be

Contents

1 Introduction **10**

1.1 Background 10

1.2 Problem Statement 11

1.3 Research outcomes 12

1.4 Project scope 12

1.5 Methodology 13

1.6 Structure of the document 13

2 Research Context **15**

2.1 Bluetooth 15

2.1.1 Bluetooth Overview 15

2.1.2 Bluetooth Architecture 16

2.1.2.1 Lower Layers of the Bluetooth protocol stack 17

2.1.2.2 Upper Layers of the Bluetooth protocol stack 17

2.1.3 Bluetooth Profiles 18

2.1.4 Bluetooth links and packets 20

2.1.5 Bluetooth networking 21

2.1.6 Bluetooth Enhanced Data Rate (EDR) 22

2.1.7 Bluetooth security 22

2.1.7.1 Discovering Bluetooth devices during communication 24

2.1.7.2 Bluetooth vulnerabilities 24

2.1.7.3 Pairing, Bonding and Trust 25

2.1.8 Current and future Bluetooth trends 25

2.1.9 Bluetooth challenges and constraints 26

2.2 Streaming 27

2.2.1 Streaming using HTTP . . . 28
2.2.2 Streaming with the Advanced Audio Distribution Protocol . . . 28
2.2.2.1 A2DP profile restrictions . . . 30
2.2.2.2 Audio Streaming between the source and the sink . . . 30
2.2.3 Other streaming architectures . . . 31
2.2.3.1 The RTSP, double buffering, and streaming architectures . . . 31
2.2.3.2 Audio streaming from hotspots to Bluetooth enabled devices 36
2.2.3.3 Video streaming over Bluetooth links with video compression 37
2.3 Media files used for multimedia streaming . . . 39
2.3.1 The MP3 standard . . . 40
2.3.2 The AMR standard . . . 40
2.3.3 The MPEG-4 standard . . . 41
2.3.4 The 3GPP standard . . . 41
2.3.5 The Waveform Audio File (WAV) standard . . . 42
2.3.5.1 WAV file header (RIFF type chunk) . . . 43
2.3.5.2 WAV chunks . . . 43
2.4 Summary . . . 44

3 Design and Implementation 45
3.1 Introduction . . . 45
3.2 Development environment . . . 45
3.3 Flow chart diagrams . . . 46
3.4 Technologies . . . 48
3.4.1 Sending device interaction . . . 48
3.4.2 Receiving device interaction . . . 48
3.4.3 The Symbian Operating System . . . 49
3.4.4 Java 2 Mobile (J2ME) . . . 50
3.4.5 The FileConnection API . . . 51
3.4.6 The Mobile Media API . . . 52
3.4.6.1 Compatibility of the MMAPI with MIDP 2.0 . . . 53
3.4.6.2 MMAPI Architecture . . . 54
3.4.7 Buffers . . . 56

3.4.8 Media playback control mechanisms 58
3.5 User interface and flow 58
3.6 Summary 60

4 Results and Analysis **62**
4.1 Introduction 62
4.2 Streaming benchmarks 62
4.2.1 Constant bitrate and Variable bitrate encodings 64
4.3 Apparatus 65
4.3.1 Performance metrics 65
4.3.2 File Compression 65
4.4 Experiment 1: RFCOMM vs L2CAP 66
4.4.1 Hypothesis 67
4.4.2 Methodology 68
4.4.3 Results and analysis 68
4.4.3.1 L2CAP - Device A to Device B 68
4.4.3.2 RFCOMM - Device A to Device B 69
4.4.3.3 L2CAP - Device B to Device A 71
4.4.3.4 RFCOMM - Device B to Device A 73
4.4.4 Summary 76
4.5 Experiment 2: Suitability of media types for streaming 77
4.5.1 Hypothesis 77
4.5.2 Methodology 77
4.5.3 Results and analysis 78
4.5.4 Summary 80
4.6 Experiment 3: Effects of distance on transfer speeds of L2CAP 81
4.6.1 Hypothesis 81
4.6.2 Methodology 81
4.6.3 Results and analysis 82
4.6.4 Summary 84
4.7 Experiment 4: Variation of transmission unit size 84
4.7.1 Hypothesis 84

4.7.2 Methodology . . . 85
4.7.3 Results and analysis . . . 85
4.7.4 Summary . . . 89
4.8 Experiment 5: Optimum transmission unit size for audio streaming . . . 89
4.8.1 Hypothesis . . . 89
4.8.2 Methodology . . . 90
4.8.3 Results and analysis . . . 90
4.8.4 Summary . . . 92
4.9 Experiment 6: Effects of dropped packets on audio playback . . . 92
4.9.1 Hypothesis . . . 93
4.9.2 Methodology . . . 93
4.9.3 Results and analysis . . . 94
4.9.4 Summary . . . 95
4.10 Discussion . . . 95
4.10.1 Summary . . . 96

5 Conclusion and Future Work 97

A Glossary of Acronyms 104

B L2CAP and RFCOMM speeds and times 107

C Link disconnection error prevalence at distance 109

D Audio quality ratings for transmission units 111

List of Figures

2.1 The Bluetooth Protocol Stack [1]. 16
2.2 Bluetooth Profiles [1]. 19
2.3 Bluetooth range extension through device hopping 21
2.4 Relationship between the Generic Access Profile and the A2DP profile [9]. . . 29
2.5 A2DP and its relationship with the audio source and sink [9]. 29
2.6 Encoding and decoding of audio stream [9]. 31
2.7 Architecture of proposed GPRS/3G audio book streaming [50]. 33
2.8 Double buffer and player implementation for AMR audio files [50]. 34
2.9 Original Piconet with 1 master node and 7 slave nodes [45]. 35
2.10 Partitioning of Piconet 1 into multicast and non-multicast groups [45]. 35
2.11 Video streaming over Bluetooth with compression, QoS, and Intermediate Protocols [56]. 37
2.12 Relationship between MPEG versions [32]. 41
2.13 Basic WAV file layout [51]. 43

3.1 Flow chart diagram (sending device side) 46
3.2 Flow chart diagram (receiving device side) 47
3.3 J2ME Configuration [57]. 51
3.4 Relationship between the MMAPI and the J2ME Environment [21]. 54
3.5 MMAPI States and Transitions [25]. 55
3.6 Testbed buffers and their relationships 57
3.7 59
3.8 60

4.1 Graph of average time, average speed, and normalized time (Device A to B) [L2CAP] 69

4.2 Graph of average time, average speed, and normalized time (Device A to B) [RFCOMM] 71
4.3 Graph of average time, average speed, and normalized time (Device B to A) [L2CAP] 72
4.4 Graph of average time, average speed, and normalized time (Device B to A) [RFCOMM] 74
4.5 Graph of Average time vs Distance 83
4.6 Average time vs Transmission unit size 86
4.7 Average speed vs Transmission unit size 88

List of Tables

2.1 Audio streaming with double buffers and Players. Adapted from [54] 32
2.2 File size comparison between WAV, MP3, and AMR. Adapted from [50]. . . . 32
2.3 Comparison of intermediate streaming protocols [56]. 39

4.1 Minimum and benchmark transfer speeds for multimedia streaming 64
4.2 Multimedia conversions 66
4.3 Comparison between RFCOMM and L2CAP 67
4.4 Average time and average speed obtained with the L2CAP protocol (Device A to B) 68
4.5 Average time and average speed obtained with the RFCOMM protocol (Device A to B) 70
4.6 Average time and average speed obtained with the L2CAP protocol (Device B to A) 72
4.7 Average time and average speed obtained with the RFCOMM protocol (Device B to A) 73
4.8 T-test of L2CAP protocol - Device A to Device B **vs.** Device B to Device A . . 75
4.9 T-test of RFCOMM protocol - Device A to Device B **vs.** Device B to Device A 75
4.10 T-test of L2CAP **vs.** RFCOMM 76
4.11 Suitability of media types to streaming 78
4.12 t-test of Method 1 vs. Method 2 - Population 79
4.13 t-test of Method 1 vs. Method 2 - Playback 80
4.14 Average time taken and average transfer speed at distance intervals 82
4.15 Transfer times and average speed with MTU variation 85
4.16 Optimum transmission unit size for audio streaming 91
4.17 Transfer speeds and playback quality ratings for increasing and decreasing distances 94

B.1 Average time, average speed and nomalized times obtained with the L2CAP protocol (Device A to B) 107
B.2 Average and normalized speeds and times obtained with the L2CAP protocol (Device B to A) 107
B.3 Average time, average speed and nomalized times obtained with the RFCOMM protocol (Device A to B) 108
B.4 Average and normalized speeds and times obtained with the RFCOMM protocol (Device B to A) 108

C.1 Distances of link disconnection occurence 109

D.1 Optimum transmission unit size for audio streaming 112
D.2 Inconsistent ratings for audio playback 112

List of Algorithms

4.1 Calculating the required bitrate 63
4.2 Calculating the expected time 85

Chapter 1

Introduction

1.1 Background

Across both the developed and developing world, phones are growing in popularity and have overtaken computers in terms of number of handsets available. Popularity has grown to such an extent that they are now the most popular ICT devices. A worldwide study undertaken in November 2009 of 24,000 respondents from 35 markets found that 86% of respondents owned a mobile phone while only 55% owned a desktop computer. Of those who owned a mobile phone, 55% used it for digital music, 42% use for transferring files and 42% make use of the Bluetooth capabilities of the phone [18]. Stemming from these changing usage patterns, a new activity is becoming increasingly more popular – sideloading. Unlike uploading and downloading, where data is transferred to or from a server, sideloading refers to the peer to peer sharing of information. This thesis is investigating the use of Bluetooth to support streaming of multimedia content between two mobile handsets using the Bluetooth protocol on the Java 2 Mobile Edition (J2ME) platform.

Bluetooth is a low powered wireless protocol that provides connectivity between both fixed and mobile devices. Bluetooth was developed in order to abandon the use of wires and shift towards wireless transfer of data without any synchronization issues [41]. It is used in cell phones, laptop computers, televisions, television remote controls and various computer peripherals.[41].

With the advances in memory and processing capabilties of mobile devices, Bluetooth gradually became the lightweight protocol of choice for wireless transfer of media [41]. The introduction of cameras and media playback capabilities on mobile devices also established Bluetooth as an ideal protocol for the transfer of media [41]. Bluetooth also serves as a replacement for its wired media transfer counterpart, enabling technology such as Bluetooth handsfree kits and streaming of audio to enabled headsets.

At the same time there has been development on the software front, in the form of the Mobile Media API (MMAPI). The MMAPI allows programmers to utilize the multimedia capabilities of Java enabled mobile devices and enables the playback of different media formats: from the network (usually HTTP and RTSP streaming); from a local database (record store); from the local file system; or from a Java Archive (JAR) file. The MMAPI allows advanced control over the media, enabling playback, pausing, fast forwarding and rewinding media, as well as capturing audio and video and streaming radio over the network. To encourage device manufacturers to implement this Application Programming Interface (API) in their Java enabled devices, the MMAPI was designed as protocol and format agnostic which allows it to be compatible with any Java configuration [21]. The MMAPI provides a plethora of multimedia opportunities for Java enabled cell phones which implement it. Despite its numerous advantages, the MMAPI provides limited support for the streaming of audio and video over the Bluetooth protocol. The MMAPI accepts input from both local and remote media locations and provides support for a large subset of media formats. Most remote media is transferred via the Hypertext Transfer Protocol (HTTP) and the Real-Time Streaming Protocol (RTSP). There is little to no research pertaining to the streaming of audio and video across the Bluetooth protocol, and even less is known about this streaming between Bluetooth enabled mobile phones [21]. One of the key components of the MMAPI is the Player which enables the playback and control of media, but lacks the functionality required to play and control partially downloaded media [21].

1.2 Problem Statement

This study aims to investigate the viability of multimedia streaming between mobile handsets using the J2ME platform. This can be broken into four sub-problems:

1. Determine the current state of Bluetooth audio and video streaming on mobile phones. This was conducted in order to identify the most suitable approach to overcoming the limited support of the MMAPI for streaming across the Bluetooth protocol.

2. Design and implement a mobile testbed which should be modular in order to support an investigation into the most suitable Bluetooth protocol in terms of: ease of implementation; bandwidth capabilities; and audio and video streaming.

3. Once the testbed has been created, the viability of audio and video streaming is tested using the more suitable protocol determined from (2). The efficiency of streaming should be evaluated using multiple file formats of varying quality, in order to determine the optimium streaming file format and quality.

4. Once the applicable file format and quality is determined, buffering and playback optimization will be refined, thus overcoming intermittent playback and poor quality.

1.3 Research outcomes

This research aims to:

1. Evaluate the viability of audio and video streaming over the Bluetooth protocol on the J2ME platform.
2. Compare and contrast the Radio Frequency Communication (RFCOMM) and Logical link Control and Adaptation (L2CAP) protocols, focusing on such factors as: ease of implementation; transfer speeds and times; and suitability for audio and video streaming.
3. Determine which audio and video file formats are more suited to streaming in terms of the optimum distance and transmission unit size.
4. Optimize the playback of media from double buffers, resulting in continuous, good quality playback with minimal intermission.

1.4 Project scope

The scope of this project is limited to evaluating the viability of audio and video streaming with the J2ME platform between Symbian S60 devices. In this study, the devices used are the Nokia N95 8GB and the Nokia N82. Since this research is a proof of concept of audio and video streaming on the J2ME platform, results obtained are specific to class two Bluetooth devices which are explained in Section 2.1.1. This research only covers streaming of media through Bluetooth protocol input streams and does not use any streaming specific protocols such as the RTSP protocol. Only media formats supported by the above mentioned handsets have been included in this study and any media formats supported but not listed in the device manufacturers specifications (e.g Windows Media Format) have only been partially covered. The variation of transmission unit sizes have been limited to 10% decrements of the Maximum Transmission Unit (MTU), since it is unreasonable to test each and every transmission unit size from the MTU downwards. Although care was taken to minimize interference across all experiments, unwanted devices operating in the same frequency band as Bluetooth could adversly affect the experiments and their results.

1.5 Methodology

A literature review was initially conducted to determine the state of the art in Bluetooth multimedia streaming. The literature review led to the design and implementation of a mobile testbed which in turn enabled an evaluation of the viability of audio and video streaming on the J2ME platform. The evaluation was performed through the deployment and testing of eight media file formats: Waveform audio file (WAV), MPEG-1 Audio Layer 3 (MP3), Advanced Audio Coding (AAC), Windows Media Audio (WMA), Musical Instrument Digital Interface (MIDI), Adaptive Multi-Rate (AMR), 3rd Generation Partnership Project (3GP), and MPEG 4 (MP4). A comparison between the L2CAP and RFCOMM protocols was conducted to determine which protocol yielded the highest transfer speed. Once the preferred protocol had been found, an investigation into the most suitable media type for streaming was conducted. Thereafter an analysis was undertaken to determine the effects of distance on the transfer speed, since it is realistic to expect that multimedia streaming will take place at varying distances. The effects of transmission unit size variation on transfer speed led to the determining of the optimum transmission unit size which allowed for the most efficient streaming solution. The optimum transmission unit size was determined by user preference for three audio files: MP3, WAV and AMR. Each transmission unit size was given a rating out of ten (best) and the average of the these results then provided an insight into the optimum transmission unit size for each of the three media files. Once the optimum transmission unit size was determined, an experiment relating to the effects of dropped packets on the quality of streaming was considered necessary in order to completely evaluate the viability of audio and video streaming on the J2ME platform.

1.6 Structure of the document

This document consists of five chapters, four of which are outlined below:

Chapter 2 starts with an in depth explanation of the Bluetooth protocol; its history; architecture; profiles; networking; Bluetooth EDR; security; current and future trends; and challenges and constraints. Thereafter, an introduction to streaming with the HTTP protocol and the Advanced Audio Distribution Protocol (A2DP) are then given. This chapter then proceeds with a study of related work for audio and video streaming, focusing on topics such has double buffering and streaming architectures, as well as audio streaming from hotspots and video streaming over Bluetooth links with compression. Chapter 2 is concluded with an explanation of selected media types supported on the Nokia N95 8GB and the Nokia N82.

Chapter 3 begins with an introduction to the development environment which outlines the hardware and software components used throughout the experimentation process. The layout of the mobile testbed is then explained, along with diagrams to aid the understanding of the struc-

ture. This chapter then introduces the technologies involved in streaming on the J2ME platform, detailing the roles of the sending and receiving devices. An overview of the Symbian operating system, the J2ME environment, the File Connection API, and the MMAPI are then given. The significance of buffers and threads and their role in the mobile testbed is then detailed, along with an explanation of media playback and control mechanisms. The user interface and flow of execution are detailed by means of screenshots.

Chapter 4 begins with an overview of multimedia streaming benchmarks, followed by the effects of constant and variable bit rate encoding schemes. The general aparatus of streaming are then highlighted, which include Performance metrics, and file compression. A comparison between the RFCOMM and L2CAP protocols is conducted, followed by an overview of the various media types and their suitability for multimedia streaming. Thereafter, the effects of distance on the transfer speed of the L2CAP protocol is highlighted. Chapter 4 also deals with the effects of transmission unit size variation on transfer speed, as well as the optimum transmission unit size for audio and video streaming. This chapter is concluded with the effects of dropped packets on multimedia streaming.

Chapter 5 concludes this thesis by drawing conclusions and reviewing research outcomes, as well as highlighting possible future extensions to this research.

Chapter 2

Research Context

This chapter starts with an introduction to the Bluetooth protocol, along with a discussion of its architecture; profiles; links and packets; networking; Bluetooth EDR; security; current and future trends; and the challenges and constraints associated with this protocol. The topic of audio and video streaming is then discussed in terms of HTTP and A2DP, which is then followed by a review of related work pertaining to audio and video streaming over Bluetooth. Various media file formats and their internal structures are then discussed.

2.1 Bluetooth

Bluetooth is comprised of hardware and software components, and this chapter attempts to show the relationship between the software and hardware components by describing the various software layers and their associated profiles [47]. The Bluetooth layers show the levels at which the instructions from the software at the application level affect the Bluetooth hardware. Each software layer has associated profile(s) which provide services to developers and users. Different profiles are designed with different purposes in mind, such as networking, human and device interaction, and file transfer [47]. This section provides an overview of the Bluetooth protocol and its architecture.

2.1.1 Bluetooth Overview

As mentioned in Chapter 1, Bluetooth is a lightweight wireless protocol which allows seamless connection of devices and the subsequent transfer of media such as voice, music, and videos [6]. Bluetooth operates using radio waves, which are constrained in terms of transmission range. Bluetooth transmits information within a confined space, which is known as your Personal Area Network (PAN) and operates in the unlicensed 2.4 GHz Industrial, scientific, and medical

(ISM) band. There are three classes of Bluetooth transmitters which govern the strength of the Bluetooth signal. Class one has a range of 100m, class two has a range of 10m, and class three has a range of 1m. Since class one Bluetooth transmission consumes more power than class two Bluetooth transmission, it is commonly found in larger, fixed devices without any power constraints. Power consumption issues are especially prevalent in mobile devices and class two Bluetooth transmitters are therefore prioritized over class one transmitters, resulting in a transmission range of 10m [39]. Bluetooth makes use of 79 channels each spaced 1Mhz apart, which enables successful wireless transfer without interference. Interference avoidance is achieved through timeslotted frequency hopping, where one packet can use up to five time slots [41]. Each time slot is 0.625 ms in length. Bluetooth has a rate of 1600 hops per second between the 79 frequencies which minimizes the chance of interference [39].

2.1.2 Bluetooth Architecture

Bluetooth is a hardware based radio system with a software stack which allows for implementation across multiple devices and platforms. The Bluetooth protocol stack is central to the success of the Bluetooth protocol. As shown in Figure 2.1, the Bluetooth protocol stack is comprised of well defined layers, which delineate each layer's responsibility and allow for maximum flexibility.

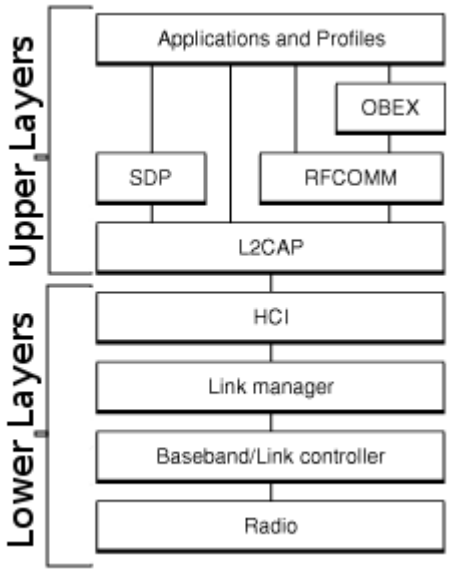


Figure 2.1: The Bluetooth Protocol Stack [1].

The remainder of this section details the lower and upper layers of the Bluetooth protocol stack.

2.1.2.1 Lower Layers of the Bluetooth protocol stack

The lower layers of the Bluetooth protocol stack consist of the Radio layer; the Baseband/Link controller layer; the Link manager layer; and the Host Controller Interface (HCI) layer. The radio layer is responsible for modulation and demodulation of data into radio frequency (RF) signals and is used to describe characteristics which the Bluetooth devices' transmitter and receiver components should possess. These characteristics include frequency tolerance, sensitivity level, and modulation characteristics [1]. The Baseband layer accepts this information from the radio layer and formats data appropriately before forwarding it to the Link manager. The Link controller is responsible for executing commands from the Link manager as well as establishing and maintaining the links from the Link manager [1]. The link manager is also responsible for translating HCI commands into Baseband level operations.

The Bluetooth specification defines two types of links which can be established between Bluetooth devices, namely: Synchronous Connection Oriented (SCO); and Asynchronous Connectionless (ACL). SCO is used for isochronous and voice communication e.g. bluetooth headsets, whereas ACL is used for data communication such as the exchange of electronic business cards (vcards). An SCO link provides reserved channel bandwidth and omits support for packet retransmission, which is redundant in voice communication[30]. ACL data packets have 142 bits of encoding information in addition to the payload which can be up to 2712 bits. This data encoding provides extra security and ensures seamless communication in an environment with multiple devices and plenty of interference [1].

The HCI acts as a boundary between the lower and upper layers of the Bluetooth protocol stack and is specifically useful when Bluetooth is implemented across two separate processors. A device may use the Bluetooth module's processor to implement the lower layers of the Bluetooth protocol stack (radio, baseband and link controller, and link manager) and its own processor to implement the upper layers of the Bluetooth protocol stack (L2CAP, RFCOMM, OBEX, and the profiles). In such a scenario, the lower portion is known as the Bluetooth module and the upper portion is known as Bluetooth host. Most Bluetooth devices are constrained in terms of memory and thus the module and host portions of the Bluetooth protocol stack are implemented on the same processor [1].

2.1.2.2 Upper Layers of the Bluetooth protocol stack

The L2CAP layer is the first of the upper layers in the Bluetooth protocol stack, and is responsible for: establishing connections across existing ACL links, and requesting an ACL link if one does not already exist; multiplexing between Service Discovery Protocol (SDP) and RFCOMM, thus enabling many different applications to utilize a single ACL link; and reformatting

and repackaging data packets from the upper layers in a form recognized by the lower layers.

L2CAP is a required component of every Bluetooth system, since it forms the base of communication between devices via logical channels [1]. Above the L2CAP layer are the SDP and RFCOMM layers. The SDP layer defines actions for sending and receiving Bluetooth devices. An SDP client communicates with an SDP server via a reserved channel on an L2CAP link, which ensures that clients are always able to discover services offered by servers. When the client wants to make use of services offered by the server, a separate connection is requested. An SDP server maintains its own SDP database which contains a list of services offered and specifies how each client is able to make use of each of these services. Each service record in the database is paired with a universally unique identifier (UUID) [8].

The RFCOMM layer is responsible for emulating the serial cable line settings and status of a serial port. This provides support for legacy serial port applications as well as the Object Exchange (OBEX) protocol. The OBEX protocol layer is situated above the RFCOMM layer and is responsible for defining objects (business cards, data, and applications) as well as providing a means of communication between two devices. An OBEX communication session commences when: the client sends SDP request packets to another device to determine whether that device is able to act as an OBEX server. If the device is able to act as an OBEX server, it will respond with an OBEX service record which contains the RFCOMM channel number which the client should use. Once the communication channel between the devices is established, the data is sent in request and response packets [8]. The applications and profiles layer contains the profiles depicted in Figure 2.2 and serves as a means for the profiles to communicate with the rest of the upper and lower layers [1].

2.1.3 Bluetooth Profiles

The Bluetooth specification defines a number of profiles which describe the tasks and capabilities of the Bluetooth protocol. Each profile specification contains information on the dependencies between profiles, the suggested user interface formats, and the particular components of the Bluetooth protocol stack to be used for the particular protocol. Figure 2.2 shows the various Bluetooth profiles:

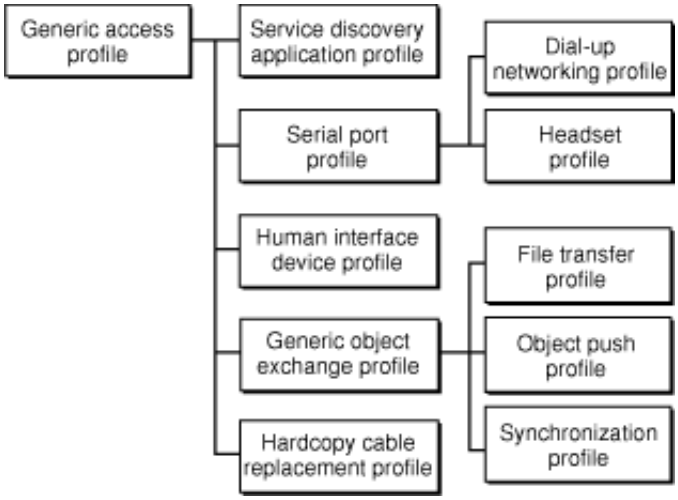


Figure 2.2: Bluetooth Profiles [1].

From the left of this Figure, the Generic Access Profile (GAP) is the parent of the profile hierarchy and serves as a base for all other profiles. The GAP defines which features should be offered on Bluetooth devices, as well as the means for discovering and connecting to other devices. The GAP provides a consistent means for establishing baseband links between devices, as well as basic user interface technology. The SDP, L2CAP, RFCOMM, and OBEX protocols communicate directly with the GAP, and the GAP in turn communicates with the appropriate profile [23].

The service discovery application profile defines a means for Bluetooth devices to discover services offered by other Bluetooth devices by implementing the SDP and the serial port profile defines serial port emulation for Bluetooth devices thus enabling legacy applications to utilize Bluetooth serial port capabilities. This profile makes use of the RFCOMM layer to perform serial port emulation [23].

The dial-up networking (DUN) profile describes how a data terminal device such as a laptop computer can use a gateway device such as a cell phone to access the Internet. The headset profile outlines how a Bluetooth enabled headset communicates with another Bluetooth device, thus becoming the other Bluetooth device's input and output. As illustrated in Figure 2.2, the dial-up networking profile and the headset profile extend the serial port profile.

The human interface device (HID) profile describes how to communicate with a HID class device. The L2CAP layer provides support for HID services.

The generic object exchange profile is a generic blueprint for other devices implementing the OBEX protocol, and it defines the roles of the client and server. This profile stipulates that the client should initiate all transactions. This profile does not define which objects are to be exchanged, nor does it define how these objects are to be exchanged. Which objects are to be exchanged and how they are to be exchanged are defined by the profiles which depend on

the GAP, namely: the object push, file transfer and synchronization profiles. The file transfer profile is dependent on the generic object exchange model, and it defines procedures needed to exchange files and folders instead of the limited objects supported by the object push profile. The file transfer profile also defines the roles of the client and server devices. The object push profile defines the roles of the push client and push server which must be analogous to the roles of the client and the sever of the generic object exchange profile. The object push profile only supports a limited number of formats, one of which is the vCard format. The synchronization profile is responsible for defining the roles of the client and server devices and provides a means for these devices to synchronize their personal information management (PIM) data, such as to-do lists and contacts, as well as emails and calendars. Automatic synchronization can take place when the devices to be synchronized discover each other, or when the user initiates a synchronization session. Finally, the hardcopy cable replacement profile describes how to send rendered data over a Bluetooth link to a device such as a printer[23].

2.1.4 Bluetooth links and packets

As described in Section 2.1.2.1, links can be established between Bluetooth devices using either ACL or SCO. Master-slave pairs make use of either one of these links in order to transfer data, and the link type between the master and slave is changeable throughout the session. Voice data is usually supported using SCO links which provides symmetrical, circuit switched, point-to-point connections. SCO links occupy two consecutive time slots, both up and down, and at fixed intervals. The data transfer rate of SCO links is 64 Kbps. Packet data on the other hand, is governed by ACL which supports symmetrical and asymmetrical packet-switched point-to-multipoint connections. Packets utilizing ACLs have a bandwidth of 723 Kbps in one direction and 57.6 Kbps in the other direction. ACL is suitable for Piconet management, since the master node controls the ACL bandwidth and also decides how much bandwidth to allocate to the slave nodes. Since ACL links support broadcast messages, this enables the master to communicate with all other slaves within the Piconet [3]. Both ACL and SCO provide support for retransmission errors. ACL data packets are protected by an Automatic Retransmission Query (ARQ) scheme. ARQ performs error checking after every packet reception which ensures that in the case of error detection, the sending unit is notified in the next packet (lost or faulty packets therefore occupy one time slot). Since SCO predominantly deals with voice packets, retransmission of lost or faulty packets is redundant, since voice data packets require sequential ordering. SCO however, implements a voice encoding scheme which guards against bit errors. Uncorrected errors result in increased background noise[3]. L2CAP is used for communication over ACL links, and is comprised of two basic modes, namely the basic mode; and the retransmission and control mode. In basic mode the L2CAP protocol has a default MTU of 672 bytes and a minimum MTU of 48 bytes. Whether the retransmission and control mode or the basic mode

is used, the lower layer Bluetooth BDR/EDR ensures reliability by configuring the number of retransmissions and the flush timeout. The default timeout period is 20 seconds.

2.1.5 Bluetooth networking

Apart from Bluetooth being used as a point-to-point connection protocol, it can be used to create PANs and mobile ad-hoc networks (MANET) through the implementation of Bluetooth Piconets and Scatternets. By making use of MANETs, the range of Bluetooth devices is increased considerably. An example of how Bluetooth MANETs could be used to extend the range of Bluetooth devices is illustrated in Figure 2.3:

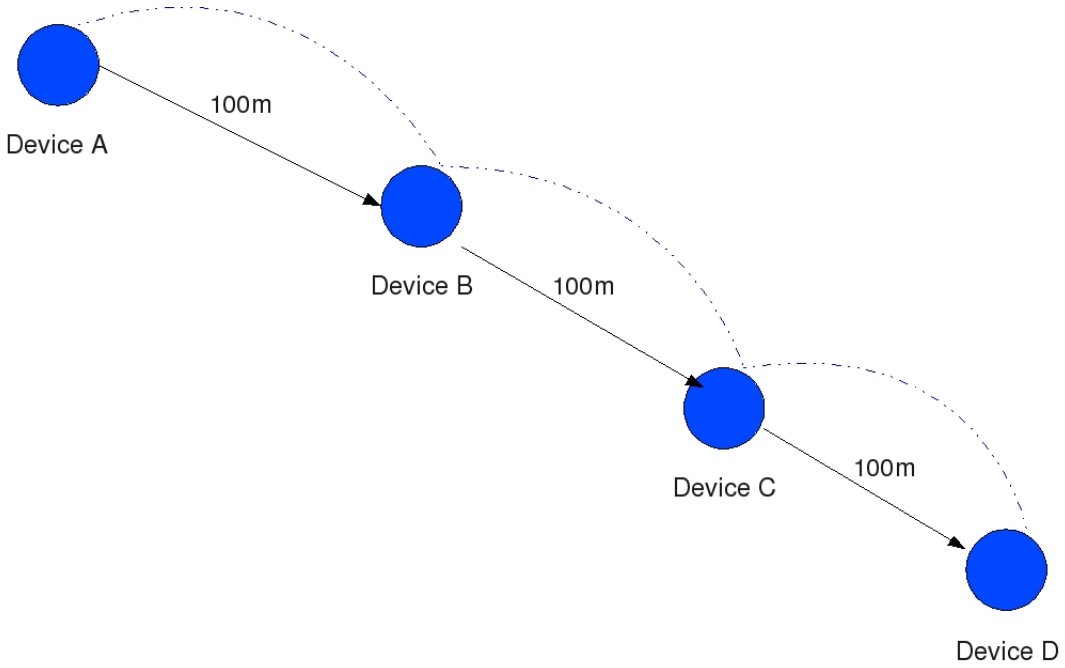


Figure 2.3: Bluetooth range extension through device hopping

In Figure 2.3 it is assumed that devices A-D are class one devices. Since the maximum range of class one Bluetooth devices is 100m, Device A would only be able to communicate with Device B. In order for Device A to send data to Device C, Device B would have to be used as a hop. By utilizing each node in the Bluetooth transmission range as a potential hop to a node which would otherwise be unreachable, the normal range of Bluetooth devices is extended. In normal point-to-point Bluetooth communications, Device A would only be able to communicate with Device B, but by making use of MANETs, Device A is able to communicate with Devices B, C and D. Clearly this type of setup would have bandwidth implications. If Device A was to to send data to Device D, then the data transfer rate would be slower than if Device A were to send the same data to another Device within range (Device B), since Devices B and C would have to be used as hops in order for the data to be transferred between Devices A and D. Without

some sort of security mechanism in place, the data sent by Device A destined for Device D would be visible to Devices B and C, which is clearly undesirable [35]. Section 2.1.7 outlines the security modes and services available to Bluetooth devices. Section 2.1.7 also discusses the various threats which could affect distributed Bluetooth networks.

2.1.6 Bluetooth Enhanced Data Rate (EDR)

There are a number of factors which determine the efficiency of audio and video streaming, and one such factor is the availability of bandwidth, or the data transfer rate. The data transfer rate is dependent on the Bluetooth modulation mode of which there are two: the mandatory basic rate; and the optional EDR. The basic rate uses shaped binary FM modulation to minimize transceiver complexity and the EDR uses Phase-shift keying (PSK) modulation which has two variants, namely: Differential Quadrature Phase Shift Keying (pi/4-DQPSK) and Differential Phase Shift Keying (8DPSK). The gross air data rate for basic rate modulation is 1 Mbps. EDR using pi/4-DQPSK has a data rate of 2 Mbps and EDR using 8DPSK has a data rate of 3 Mbps [7].

The key characteristic of EDR is that the modulation scheme is changed within the packet and the access code and packet header are transmitted using the basic rate 1 Mbps Gaussian Frequency Shift Keying (GFSK) modulation scheme. The synchronization sequence, payload, and trailer sequence are transmitted using the 2 Mbps or 3 Mbps EDR PSK modulation scheme [7].

2.1.7 Bluetooth security

Bluetooth security is divided into three security modes: nonsecure; service level enforced security; and link level enforced security. In nonsecure mode the Bluetooth device does not implement any security measures. In service level security mode, two Bluetooth devices can establish an unsecured ACL link. Authentication and authorization as well as optional encryption are initialized when an L2CAP connection oriented or connectionless channel request is made. The difference between service level security and link level security is that in the latter, security procedures are initiated before channel establishment [48].

The Bluetooth specification outlines three basic security services, namely: Authentication; authorization; and confidentiality. Authentication verifies the identity of communicating devices, but does not provide user authorization. Authorization controls resources by ensuring that a device is authorized to use a service and thus has the necessary permissions to do so. Confidentiality ensures that information is kept secret and that only authorized users have access to that information. The Bluetooth specification outlines four security modes which

ensure the security of the Bluetooth protocol. The National Institute of Standards and Technology (NIST) recommend that organizations carry out the following activities to protect their Bluetooth networks and devices: use the strongest Bluetooth security mode available for their Bluetooth devices; address the use of Bluetooth technology in organizational security policies and then change procedures and device settings to reflect these policies; ensure that Bluetooth users are made aware of policies [37].

Morrow [38] outlines the three categories of threats in distributed networks, namely: Disclosure threat; Integrity threat; and denial of service (DoS) threat. Disclosure threats occur when there is a leakage of information from the target machine, resulting in unauthorized access by eavesdroppers. Integrity threats occur when information is deliberately altered in order to confuse and mislead others. DoS threats occur when access to a service by authorized users is denied or severely limited [4]. An example of a DoS attack would be to make multiple connections with the target device, thus limiting its bandwidth. The most well known and basic method for securing Bluetooth is by rendering the device non-discoverable. When a device is in discovery mode it appears on the search list of searching Bluetooth devices, and if discovery mode is disabled, the device will no longer be visible to searching Bluetooth devices. Even though the device is invisible, other devices are still able to connect to the invisible device if they have the MAC address, which is often the case when two devices were previously paired [4].

The GAP outlines the structure of Bluetooth security and since this profile is the root of the profile hierarchy, it provides security for each child profile. The user is largely influential in the strength of Bluetooth security, since they control how the device will implement its discovery and connectivity options. According to Bialoglowy [4], these options can be divided into the following three categories:

- Silent – The PAGE SCAN and INQUIRY SCAN modes will be unentered, and thus no connections to this device will be possible. This device simply monitors other Bluetooth traffic.
- Private – The device will periodically enter the PAGE SCAN mode, enabling it to accept connections if its BD_ADDR is known, but the device will never enter the INQUIRY SCAN mode, thus making it invisible to other Bluetooth devices.
- Public – The device enters both the PAGE SCAN and INQUIRY SCAN modes, which renders it both visible and connectible.

The following sub-sections outline the various ways in which a Bluetooth device can be compromised and the security mechanisms in place to prevent such compromisation.

2.1.7.1 Discovering Bluetooth devices during communication

One of the major downfalls of the current Bluetooth specification is that the Bluetooth address is unencrypted, regardless of whether the user chooses to encrypt the rest of the communication session. Bluetooth has a pseudo random frequency hopping sequence of 1600 hops per second which provides a reasonable level of protection for the Bluetooth address, but if someone had to discover the pseudo random sequence, both of the communicating devices Bluetooth addresses would be visible. When two devices are communicating using the RFCOMM protocol, both devices have to grant permission to one another before communication can continue. With the L2CAP protocol, a mobile phone simply accepts a connection request without the need for confirmation from the user. With a known Bluetooth address and an L2CAP connection, a device can be compromised [4].

2.1.7.2 Bluetooth vulnerabilities

The most common Bluetooth vulnerabilities are bluejacking, bluesnarfing, and the backdoor attack. These vulnerabilities fall into discoverable and non-discoverable modes.

Bluejacking and bluesnarfing operate best when the device which they are attacking is in discoverable mode. Bluejacking is the process of sending messages and business cards to Bluetooth enabled devices. This is mainly used for promotional purposes and does not involve any altering of the data on the phone, but too many messages sent to the phone can in fact render the phone inoperable, which is another example of a DoS attack. Bluejacking is not possible when devices are set to non-discoverable mode. Bluesnarfing is a method of accessing a mobile phone and copying all of the contacts, calendars and any other information stored in the phone's memory and is significantly more difficult when a device is in non-discoverable mode.

The backdoor attack is a vulnerability which can be exploited when the device is in discoverable or non-discoverable mode, and involves establishing a paired connection with another Bluetooth device, ensuring that the paired connection is not shown in the paired connections list. This essentially means that the secretly paired Bluetooth device has access to all the services offered to paired devices such as modems, ability to send Short Message Service messages (SMS) as well as access to GPRS services [4]. There are a number of vendor issues where devices have not been properly tested for Bluetooth security flaws. Some of these vendor issues can be found in Nokia and Sony Ericsson cell phones where the execution of two simple commands in most Linux distributions will enable the retrieval of the devices phone book [4].

Some Bluetooth devices pose as fake access points, to which other Bluetooth devices would connect, thus allowing L2CAP connections to take place without user authentication. Once an L2CAP connection is established, the fake access point then has access to the connected device's services. Social engineering is another common way which is used to gain

unauthorized access to Bluetooth devices. Social engineering is the the art of convincing someone through coercion or deceit to reveal confidential information which could lead to compromisation of the target's machine/device [4].

As with any networked environment, Bluetooth creates the opportunity for viruses to be spread across phones. The Cabir mobile worm took advantage of the vulnerability in the Bluetooth implementation of several Nokia and Sony Ericsson phones. The virus was not dangerous, as it simply drained the phones battery. Another virus was encountered in Japan in 2001 where it denied users the ability to make emergency calls [4].

2.1.7.3 Pairing, Bonding and Trust

Pairing between Bluetooth devices occurs when two devices enter a common Personal identification number (PIN), which in turn is used to get a link key to authenticate a Bluetooth session. Authentication can occur automatically if both Bluetooth devices have the same stored PIN and derive the same link key for authentication. Alternatively, an application can request both users to enter a PIN manually.

Once devices are paired, the link keys can either be stored and then used for future authentication sessions, or the keys can be discarded resulting in the pairing process being repeated upon device reconnection. Bonding can expire immediately after link disconnection (not bonded), or it can expire after a certain time period has elapsed (temporarily bonded) or continue until explicitly terminated (permanently bonded). When devices are paired with one another, there is a vulnerability period when the PIN is used to form the link keys (a random number is sent over the air). If an eavesdropper is able to obtain this random number, the PIN can then be guessed, resulting in a possible derivation of one of the link keys. Trust applies to the authorization of a device to access the services hosted on another device. A trusted device is previously authenticated and based on this authentication, has authorization to access various services. Untrusted devices may be authenticated, but an action such as entering a password may have to occur in order for that device to be authorized to access the services on another device [38].

2.1.8 Current and future Bluetooth trends

Since Bluetooth is a low cost, low power, and radio based technology, it is ideal for use in PANs. A PAN allows a user to have a small, simple mobile network available at all times. Bluetooth is ideal for PANs since there are no cables and no complex settings are needed to setup network infrastructure. Some of the current uses of Bluetooth are: wireless dial up networking between mobile devices and computers; Bluetooth controlled human interface devices such as mice and remote controls; exchanging of contact information and other data between Bluetooth enabled devices.

According to industry analysts, the popularity of Bluetooth and PANs is increasing which provides more opportunities for device manufacturers [1]. In 2003 version 1.2 of the Bluetooth specification was released and provides the following improvements over version 1.1:

- Enhanced quality of service (QOS) which allows human-interface devices to send and receive data when needed.
- An improved frequency hopping algorithm which increases communication reliability and interference from other wireless devices.

The Bluetooth SIG has envisaged many future functions for Bluetooth, some of which are the following: the three-in-one phone which can be used as an intercom at the office (with no telephony charge), as a portable phone at home, and as a mobile phone when on the move (cellular charge). The Bluetooth SIG also envisaged a Bluetooth automatic synchronizer, where data between a desktop computer, notebook, mobile device and Personal digital assistant (PDA) is synchronized as soon as these devices are within range of one another. This would enable calendars and contacts to be in sync with each other across multiple platforms [24].

2.1.9 Bluetooth challenges and constraints

In order for Bluetooth to remain a low cost protocol and also accomodate for the low powered nature of this protocol, a number of challenges and constraints are encountered. Haartsen et al. (1998) [24] outline a list of Bluetooth requirements and some of the challenges met while attempting to meet these requirements:

- **Bluetooth should support both voice and data and the quality of connection should be equivalent to phone line quality**. This is especially important to users who are accustomed to good quality connections, and to voice recognition software where the accuracy of the voice is imperative for the correct functioning of the software.
- **Ability to establish ad-hoc connections**. Since Bluetooth is a highly mobile protocol, and is often found in rapidly changing environments, it is necessary for Bluetooth devices to be able to detect other Bluetooth devices in range. These Bluetooth devices should be able to connect to multiple Bluetooth devices while simultaneously accepting multiple connections.

- **Bluetooth should have the ability to withstand interference from other devices in the same frequency range**. Since Bluetooth operates in the 2.4GHz range, it encounters interference from a plethora of devices operating in the same unlicensed frequency range. An example of such a device is a microwave oven.
- **Similar security to its cable counterpart**. Even though Bluetooth is short range, it still requires security mechanisms, as users do not want their confidential data to be intercepted.
- **Bluetooth radio modules should be small enough to integrate into a wide variety of devices**. Bluetooth radio modules should be able to fit into wearable devices such as watches, smart badges, mobile phones, and headsets.
- **Bluetooth radios should consume minimal power**. This is vitally important, since most Bluetooth radios will be powered by the battery of the device in which they are present, and the battery is primarily intended for the device itself, and not the Bluetooth radio.
- **Encourage ubiquitous deployment of the technology**. The Bluetooth SIG is designing an open specification which defines the various layers of the Bluetooth protocol.

One of the main drawbacks of Bluetooth as a wireless communication protocol is that it is constrained in terms of bandwidth. The available bandwidth for all Bluetooth version 1.1 connections on a single link is 720 Kbps, which equates to 90 KBps. Some applications allocate the entire link bandwidth to a single connection which results in two negative consequences: Performance of other connections is hindered; and when human interface devices such as Bluetooth keyboards and mice are being used the available bandwidth for the application is limited. The constraints of QOS devices require a certain amount of bandwidth to be available for the operation of these devices [55].

With an overview of the various components involved in audio and video streaming on the J2ME platform, the next section introduces various streaming protocols and covers related work pertaining to audio and video streaming.

2.2 Streaming

Streaming is the process of transferring data from source to destination where the destination device decodes the data before all the data has been transferred [13]. With mobile devices being constrained in terms of bandwidth and memory, streaming via a lightweight protocol such as

Bluetooth alleviates memory constraints, and serves as an ideal platform for minimizing bandwidth usage. This section describes different protocols that are commonly used for streaming.

2.2.1 Streaming using HTTP

The HTTP protocol is used for retrieval of remote data and facilitates communication between a client and a server. When an audio file is to be played over HTTP, it is often streamed between the client and server, leading to the suitability of HTTP as a streaming protocol for media. Even though HTTP streaming is possible it results in sub-standard performance due to the fact that HTTP depends on the Transmission control protocol (TCP) which prioritizes reliable packet delivery over the speed at which packets are delivered. The RTSP protocol is based on the User datagram protocol (UDP) which is a connectionless protocol and prioritizes speed over reliability. Even though the latter scenario is favoured, the RTSP protocol is seldom supported on mobile phones, and HTTP is therefore more widely implemented on the the J2ME platform [22].

2.2.2 Streaming with the Advanced Audio Distribution Protocol

The Advanced Audio Distribution Protocol (A2DP) is a Bluetooth profile which is dedicated to audio streaming. A2DP defines the protocols and various procedures necessary for distribution of high quality audio on ACL channels. This profile differs from narrow band voice distribution on SCO channels.

One of the main uses for A2DP is the streaming of audio content from music sources to Bluetooth enabled speakers or servers. A2DP focuses primarily on audio streaming, while the Video Distribution Profile (VDP) focuses on streaming video content. An ideal situation would be the combination of both of these profiles thus enabling high quality multimedia streaming between Bluetooth enabled devices [9].

Section 2.1.3 outlined the various Bluetooth profiles of which A2DP is a member. A2DP is dependent on the GAP profile and the Generic audio/video Distribution Profile (GAVDP). GAVDP defines the necessary procedures for setting up audio/video streaming, whereas A2DP defines the procedures specific to the setup of audio streaming. Figure 2.4 shows the relationships and dependencies of the A2DP profile:

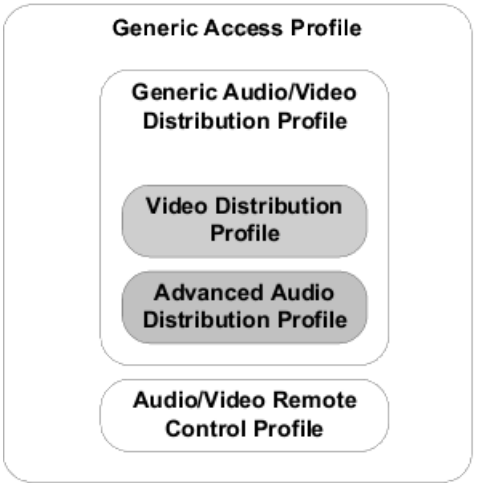


Figure 2.4: Relationship between the Generic Access Profile and the A2DP profile [9].

Apart from A2DPs dependence on other profiles, it also depends on protocols and entities which can be seen in Figure 2.5:

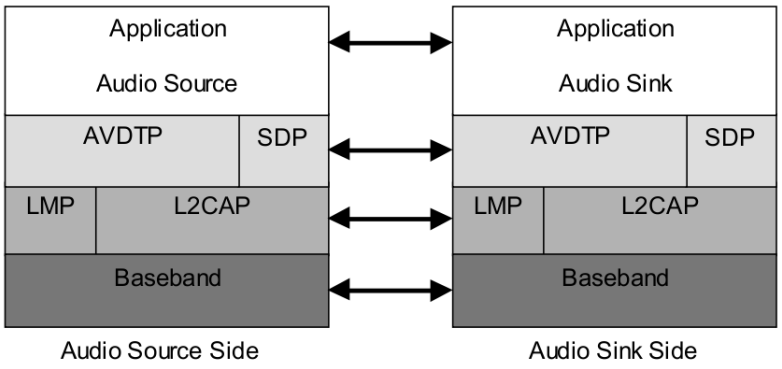


Figure 2.5: A2DP and its relationship with the audio source and sink [9].

As shown in Figure 2.5, SDP, Link Management Protocol (LMP), L2CAP and Baseband are all Bluetooth protocols which are responsible for data transmission. A2DP is based on the GAVDP protocol which in turn uses the Audio/Video Distribution Transport Protocol (AVDTP) as the underlying transport mechanism for audio streaming [9]. In order for communication between two devices implementing the A2DP profile to occur, there has to be a source (SRC) device and a sink (SNK) device. The SRC is the source of the audio content to be streamed and the SNK is the destination of the streamed audio. At the application level, data is processed by the SRC, after which it is passed to the AVDTP transport protocol and onto the L2CAP and Baseband layers where the data is packetized and transmitted to the SNK device. The SNK device receives the transmitted audio data through the Baseband layer. The data is then passed to the upper layers until it eventually reaches the application layer of the SNK where it is played.

2.2.2.1 A2DP profile restrictions

A2DP is responsible for the setup, control and manipulation of audio streams between the SRC and SNK. According to the A2DP specification of 2007 [9], there are a few restrictions on this profile:

- It does not support synchronized point-to-multipoint connections, meaning that it can only stream audio to one device at a time
- There is a 37ms delay between the SRC and the SNK due to radio signal processing, buffering and encoding/decoding of the audio stream. This delay can be reduced, but is implementation dependent
- The audio data rate should be smaller than the usable Bluetooth bitrate to allow for re-transmission schemes to reduce packet loss (if any)
- The profile does not include any content protection method
- Content protection is performed at the application level and not at the Bluetooth link level

2.2.2.2 Audio Streaming between the source and the sink

Before audio streaming between the SRC and SNK can commence, both devices need to setup a streaming connection. During this setup procedure the devices select the most suitable streaming parameters, namely: application service capability, and transport service capability. Application service capability consists of audio codec and content protection capabilities. Transport service capability is provided by AVDTP which allows the manipulation of streaming packets more efficiently.

Once the streaming connection has been established and the *Start Streaming* procedure has been initiated, both the SRC and SNK transition from the Open state to the Streaming state in which the SRC is able to send audio through the *Send Audio Stream* procedure and the SNK is able to receive audio through the *Receive Audio Stream* procedure.

With the *Send Audio Stream* procedure the SRC can if desired, encode the data into a certain format in the signaling session. The application layer then encodes the data into the defined media payload format. The encoded audio data is then passed down to the AVDTP which in turn passes the audio data to the L2CAP transport layer. The SNK then receives the audio stream through the L2CAP transport layer and passes it to the AVDTP layer which

decrypts the data (if encrypted) and then finally passes this audio stream to the application layer where it is decoded and played [9].

This process can be seen in Figure 2.6:

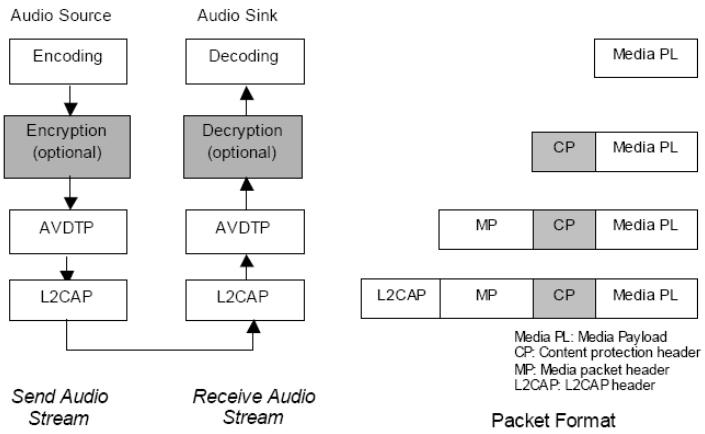


Figure 2.6: Encoding and decoding of audio stream [9].

2.2.3 Other streaming architectures

Media streaming has been attempted by a number of researchers using various protocols and methods for streaming optimization. The RTSP protocol as well as the double buffering technique are widely used throughout streaming on the J2ME platform. This section describes the implementation of streaming over 3G and GPRS networks with various media files, using double buffering. This section also covers audio streaming from hotspots to Bluetooth enabled devices and gives an overview of video compression over Bluetooth links.

2.2.3.1 The RTSP, double buffering, and streaming architectures

Vazquez-Briseno and Vincent [54] propose an adaptable system architecture which implements media streaming. The components of their architecture are the streaming server, the multicast proxy, and the mobile client. The streaming server and client are compatible with 3rd generation partnership project (3GPP) standards and make use of the SDP, RTSP, and Real-time protocol (RTP) protocols and the AMR audio standard. The architecture takes the limitations of mobile devices into consideration and has thus been designed in an efficient manner.

Since most mobile phones do not support the RTSP protocol, the client implementation on the mobile device enables the use of RTSP on mobile devices which formerly didn't support this protocol. Their architecture streams AMR audio files between the RTSP server and the mobile client. One of their contributions to audio streaming is that they manage to stream

partially downloaded AMR music files. AMR music files are comprised of a number of frames, with each frame containing a 1 byte header and 20ms of audio data. By sending these frames in small RTP packets, the Player in the MMAPI is led to believe that it is playing the entire AMR audio file, when in actual fact it is only playing part of the file. A drawback of this approach is that two Players and two buffers are used for such an implementation, which results in jittery playback of these audio files. The process of double buffering and multiple Player implementations is indicated in Table 2.1:

Table 2.1: Audio streaming with double buffers and Players. Adapted from [54]

Time	Buffer 1	Buffer 2	Player 1	Player 2
T_1	Receiving			
T_2		Receiving	Playing Buffer 1	
T_3	Receiving			Playing Buffer 2
T_4		Receiving	Playing Buffer 1	

The client begins receiving the AMR data in RTP packets, at which point the Player commences with playback of the data contained in the the first buffer (after a certain number of packets have been received). While the first Player is playing the received audio data, the second buffer is receiving other parts of the AMR file. After the first Player has played the first portion of audio data, the second Player then commences with playback of the audio contained in the second buffer. While the second Player is playing the audio, the first buffer is receiving another part of the AMR file. This process is repeated until the entire audio file has been transferred and played [54].

Sillén and Nordlund [50] implemented a similar system which streams audio books to mobile phones using GPRS and 3G. A streaming server is used and the format of the streamed media is AMR. They decided to use AMR as the audio format for streaming, since AMR files are smaller than other audio types, and without a quality compromise. Table 2.2 compares WAV, MP3, and AMR audio files in terms of size:

Table 2.2: File size comparison between WAV, MP3, and AMR. Adapted from [50].

File Type	File Size (MB)	% of WAV
WAV	12	100
MP3	1.618	13.5
AMR	0.492	4.1

From Table 2.2 it can be seen that AMR and MP3 files are approximately 4.1% and 13.5% the

size of the WAV file respectively. With such vast differences in size, it can be seen that file choice is critical to the efficiency of streaming. The components of their proposed prototype are the streaming server, a servlet, a database, the GPRS or 3G network and a mobile phone. Figure 2.7 is an overview of their system:

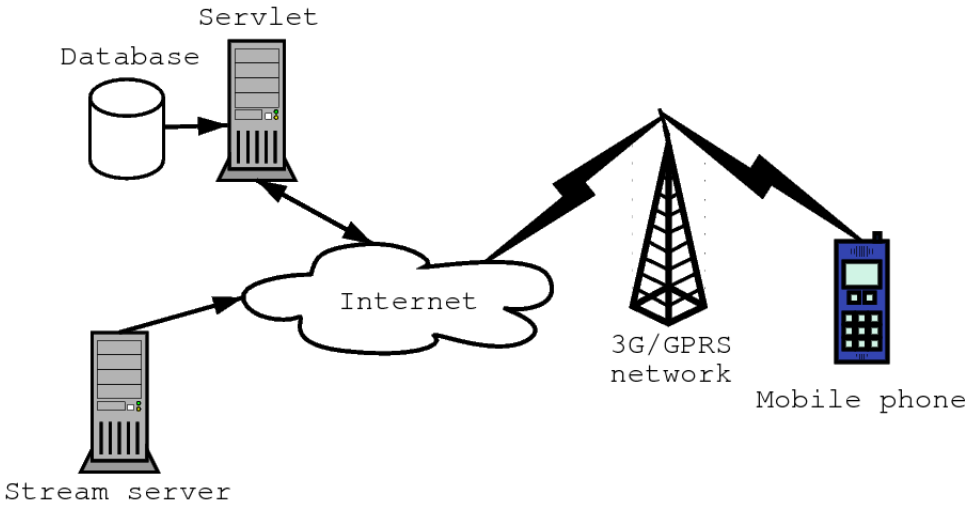


Figure 2.7: Architecture of proposed GPRS/3G audio book streaming [50].

The audio books are stored on the streaming server and information regarding these audio books is stored in the database. The Servlet accesses the audio book information from the database and sends it to the mobile phone. The mobile phone then receives the audio book and its corresponding information and proceeds with playback of the streamed media. Their prototype application also makes use of the RTSP, RTP and SDP protocols, and the AMR audio data is contained in the RTP packet.

Sillén and Nordlund [50] came to the conclusion that the MMAPI provides limited support for streaming and needs to realize the entire audio file before being able to commence with playback. They made use of the double buffer method where one buffer continues to receive the audio data while the other buffer is used for media playback. The problem they encountered was that there was a slight break in playback when switching from one buffer to another. To minimize the effects of breaks in audio playback, they ensured that the player switched between buffers when there were breaks in the sentence. Figure 2.8 shows how double buffers are used for audio streaming of AMR audio files:

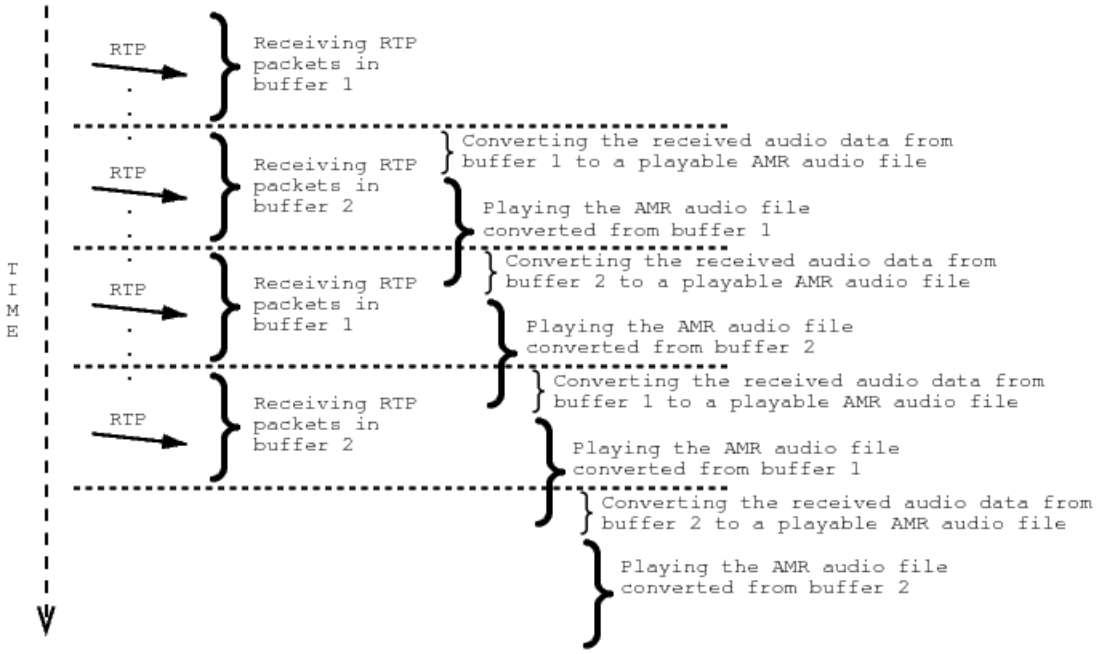


Figure 2.8: Double buffer and player implementation for AMR audio files [50].

With MP3 files being smaller in file size than traditional CD audio, it is becoming more popular to stream these lower bandwidth audio files to multiple receivers. There is no support for streaming to multiple devices with built in quality of service support. Even though the Bluetooth Special Interest Group (SIG) [8] recently announced a new feature known as the Enhanced Data Rate (EDR) [10] which increases the data rate aggregation up to 3 Mbps, there is still no multicast support, and round robin scheduling is still prevalent [31].

With these limitations of the Bluetooth protocol, Pinkumphi and Phonphoem [45] propose the following improvements to the Bluetooth protocol which firstly improve the polling mechanism of Bluetooth by using a prioritized round robin scheduling system where real-time audio packets receive a higher priority than regular data packets. Secondly, they propose the combination of the Bluetooth broadcast mechanism with Piconet partitioning techniques.

Their proposed protocol for multicast streaming is separated into three phases: real-time audio notification phase; piconet partitioning phase; and the streaming phase. The way in which the real-time audio notification phase works is through the use of Multicast Notification Packets (MNP). If there is a master or a slave node in the Piconet which possesses a real-time audio file and wants to stream this audio file to multiple nodes, then this node will send an MNP packet to the master who will then broadcast this MNP packet to all other nodes. Any slave that wants to receive the streaming content has to send a request packet to the master node. Only one multicast flow is allowed in a Piconet, so the first device that sends the MNP packet will be served [45].

If more than one device returns a request packet after receiving the MNP, then all devices will change their current state to the partitioning phase. In this phase the current Piconet (P1) is dismantled in such a way that only the multicast source and joining nodes remain in P1 and the non-multicast nodes (nodes which didn't reply with request packets) form a new Piconet (P2). The node with the least traffic throughput becomes the master node of P2 and also serves as the bridge node between P1 and P2. This partitioning can be seen in Figures 2.9 and 2.10:

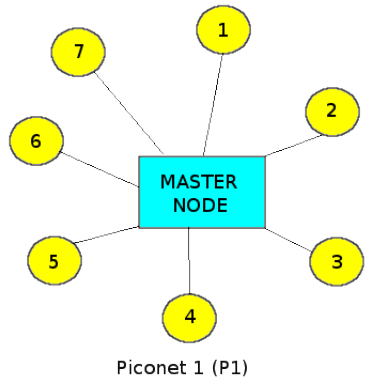


Figure 2.9: Original Piconet with 1 master node and 7 slave nodes [45].

After the transmission of the MNP and request packets, the proposed protocol partitions P1 as follows:

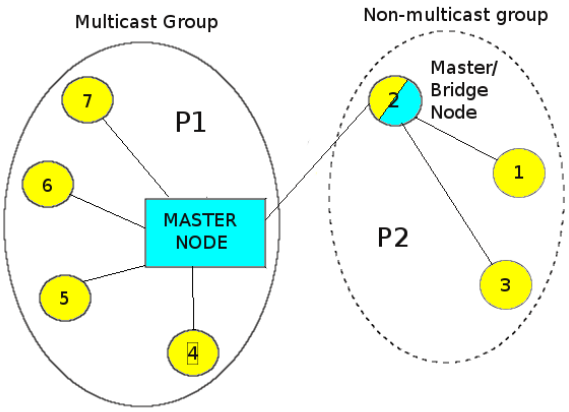


Figure 2.10: Partitioning of Piconet 1 into multicast and non-multicast groups [45].

The reason why certain nodes have been grouped under the multicast group and others in the non-multicast group is due to nodes 4, 5, 6, and 7 from the original P1 sending request packets

in response to the MNP packet, and nodes 1, 2 and 3 not sending any request packets. It can also be seen that node 2 in P2 is the master node and also the bridge node (for communication between P1 and P2).

After the partitioning of nodes into the multicast and non-multicast groups, all devices enter the streaming phase, in which the multicast source node starts to multicast real-time audio to the master in P1. For this phase, high priority DH5 (5-slot packet) packets are used. Once the master receives these audio packets it then makes use of the Bluetooth Network Encapsulation Protocol (BNEP) layer to multicast all these audio packets to the nodes in P1. Nodes other than the multicast source node are able to transmit using low priority packets such as DH3 (3-slot) packets [45].

By partitioning all these nodes into multicast and non-multicast groups, the overhead of Piconet switching is minimized. If a node from the multicast group no longer wants to receive the streamed audio then it can request that the master of P1 allocate it to P2, and if a node from P2 wants to receive audio then it can make a request to the master of P2 to switch it to P1 [45].

2.2.3.2 Audio streaming from hotspots to Bluetooth enabled devices

Bilan [5] proposes a system where audio is streamed over ACL links from a hotspot or access point to a Bluetooth enabled device. As mentioned in Section 2.1.2.1 there are two types of Bluetooth links: SCO and ACL. Since ACL has a higher tolerance for bandwidth intensive applications Bilan [5] decided it was more suitable than the traditional SCO link. Bilan [5] outlines three Bluetooth profiles which are suited to audio streaming: the A2DP profile; the Local Area Network (LAN) profile; and the BNEP protocol.

The A2DP profile defines a set of procedures for streaming audio over the L2CAP layer. The SRC streams RTP packets to the SNK across the L2CAP channel by utilizing mandatory or user defined codecs. The A2DP profile is dependent on the GAP; the GAVDP; and the AVDTP specification. The AVDTP profile is the underlying transport mechanism for streaming audio and provides basic services such as signalling information, stream management and compression of RTP headers. The A2DP profile specifies four codecs: sub-band codec (SBC); MPEG codec; AAC codec; and the ALTRAC codecs. Each of these codecs has its own advantages and disadvantages with regard to bandwidth usage, and quality and processing delays. SBC has been proven to be a suitable codec for streaming high quality voice over a Bluetooth link [43]. PcStats [43] showed that is is possible to stream music from a compact disk player over a Bluetooth link to a set of speakers in real-time. According to Bilan [5], the success of audio streaming over a Bluetooth link is largely dependent on the implementation of the Bluetooth stack.

The LAN profile functions by means of the point-to-point (PPP) protocol over an RF-

COMM channel. The profile specifies two components: a client which accesses services (DT); and an access point or gateway which hosts these services (LAP). If more than one DT is connected to the LAP then the available bandwidth is decreased significantly, but in the case of low bitrate codecs, there is sufficient bandwidth for multiple DTs to be connected in full duplex communication with the LAP. If a DT wants to connect to the LAP, the DT is the master node and the LAP is the slave node upon initial communication. The LAP then reverses the roles resulting in the DT becoming the slave node and the LAP the master node. The LAP specifies three roles for using PPP: LAN access for a single Bluetooth device; LAN access for multiple Bluetooth devices; and PC to PC communication through PPP over serial cable emulation.

The BNEP specification outlines the procedures necessary to send common protocols such as IP over Bluetooth links. In order for these packets to be transmitted over the Bluetooth link they need to traverse each Bluetooth stack layer. As each layer is traversed, the header for each layer is added to the packet, starting with the Ethernet header and ending with the L2CAP header. The receiving application traverses the layers of the received packet and converts them back to the original audio format.

2.2.3.3 Video streaming over Bluetooth links with video compression

Xiaohang [56] analyzed the research conducted in the field of video streaming over Bluetooth links by looking at metrics such as video compression; QoS control; and intermediate protocols. Figure 2.11 shows the different components and the layers over which these components operate:

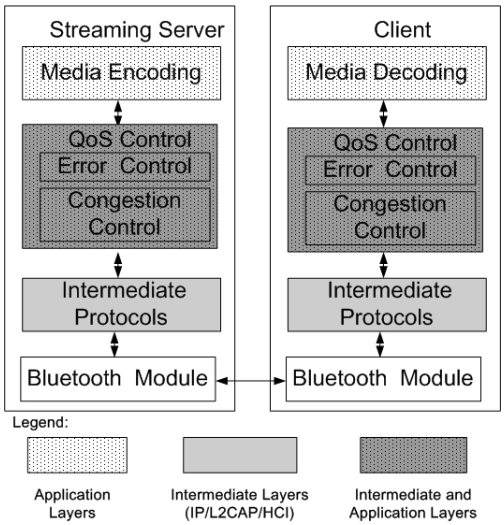


Figure 2.11: Video streaming over Bluetooth with compression, QoS, and Intermediate Protocols [56].

Due to the real-time nature of video streaming over wireless mediums, there exist bandwidth,

delay, and loss requirements. Video compression, QoS and the intermediate protocols help to overcome these challenges presented by streaming video over wireless mediums such as Bluetooth. Video compression aims to reduce the redundancy within video files, which improves the efficiency of streaming over Bluetooth. After QoS modules have optimized the Bluetooth link, the compressed video stream is then partitioned into packets matching the protocol of the intermediate layer (L2CAP, HCI, IP). The segmented video stream is then sent to the Bluetooth module for transmission. The client's Bluetooth module then receives the packets, reassembles these packets in accordance with the intermediate layer protocols and forwards them to the decoder for decompression. QoS control can further be divided into congestion control and error control. Congestion control deals with the regulation of the transmission rate according to link status and QoS requirements. Error control, however, deals with the improvement of video quality in the presence of packet loss. Three intermediate protocols are predominantly used: HCI; L2CAP; and IP [56].

Streaming via HCI maximizes the bandwidth usage, since video bit-streams are directly packetized and passed down to the baseband layer at which point they are transmitted. Overhead is also reduced as internal operations are exposed to the lower baseband layer. The size of the HCI packets is dependent on the buffer size containing the video bit-stream. Segmentation and reassembly is not handled by the HCI protocol, and thus needs to be taken care of by the application layer, which can contribute significantly to the load requirements of the host system. This is especially prevalent with MP4 which is a highly compressed format and demands high processing requirements. Another disadvantage of HCI is that the current Bluetooth protocol does not permit HCI to be run without L2CAP [56].

The L2CAP layer hides the low level details of the Bluetooth protocol and allows existing applications to run over Bluetooth links without much modification. L2CAP provides support for segmentation and reassembly, but is less efficient than the HCI protocol because of the extra bits needed for packet encapsulation. Streaming via the L2CAP protocol is defined by three profiles, namely: AVDTP; AVCTP; and GAVDP. Through the use of these profiles, a sender could stream RTP packets to a receiver across L2CAP without the need for video codecs [56].

Streaming via IP relies on the bridging of TCP/IP and Bluetooth, and its main advantage is that it can be used transparently with protocols such as RTP. Bluetooth streaming via IP can be achieved by using LAP or BNEP. LAP defines the procedures needed to setup a PPP connection over RFCOMM and allows the IP packets to flow across the link. BNEP deals with the traversal of packets through the TCP/IP stack layers, the Ethernet layer and finally through L2CAP [56].

The main factor to take into account with the intermediate protocols is the trade-off between efficiency in terms of encapsulation overhead and implementation complexity. Table 2.3 outlines the intermediate protocols and their characteristics:

Table 2.3: Comparison of intermediate streaming protocols [56].

	HCI	L2CAP	IP
Logical OSI Layer	Link	Link	Network
Implementation complexity	High	Medium	Low
Modified part	Hardware	Software	Software
Segmentation and reassembly support	No	Yes	Yes
Point-to-multipoint support	No	Yes	Yes
Bluetooth Profile	N/A	AVDTP	LAP or BNEP
Encapsulation overhead	RTP	L2CAP + AVDTP + RTP	L2CAP + LAP + RTP/BNEP + IP + UDP + RTP

As illustrated in Table 2.3, L2CAP and IP seem to be more suited to video streaming than HCI. Usually, the higher the protocol in the OSI protocol stack, the easier the implementation thereof. IP would be an easier implementation choice because it is higher in the protocol stack than both HCI and L2CAP. The L2CAP and IP protocols are modified at the application level, whereas the HCI protocol is modified at the hardware level. The encapsulation overhead is critical to the efficiency of packet transmission, and thus the least number of protocols which encapsulate a packet the better. From Table 2.3 it can be seen that the HCI protocol has the lowest encapsulation overhead, followed by L2CAP and then IP. With the aforesaid it can be deduced that the IP protocol is the most efficient intermediate protocol as its logical OSI layer is higher than that of the L2CAP protocol; and its implementation complexity is lower than that of the L2CAP protocol [56].

Section 2.3 introduces the media file formats used in assessing the viability of J2ME as a platform for media streaming over the Bluetooth protocol. These media file formats are introduced and their internal structure explained.

2.3 Media files used for multimedia streaming

Vazquez-Briseno and Vincent [54] and Sillén and Nordlund [50] both came to the conclusion that AMR audio files were the most efficient for audio streaming. Sillén and Nordlund [50] compared the MP3, WAV and AMR formats for the purposes of streaming audio books, and found that AMR was best suited to voice audio streaming when compared to MP3 and WAV. With the majority of related work referring to audio formats such as MP3, WAV and AMR, it was decided to give an overview of these formats for reasons explained in Section 4.5. The MPEG-4 and 3GPP standards are common video formats found on mobile devices, and it was thus decided to provide an overview of these two video standards.

2.3.1 The MP3 standard

MP3 is a highly compressed audio format which was standardized in 1991 by the MPEG which was formed in 1988 by the ISO/IEC organizations. Their intention was to develop generic coding standards of moving pictures, audio, and systems containing both. In 1991 the first phase of the work was complete, resulting in MPEG-I. This standard defines three layers with increasing levels of complexity, namely: Layer-I, Layer-II, and Layer-III. Layer-III is the most complex and provides the highest quality when the same bitrate is used [20].

MPEG-2 was developed to correct the shortcomings of the MPEG-1 standard. The main difference between the two standards is that MPEG-2 introduced multi-channel coding which supports up to six channels as opposed to the two channels supported by stereo in MPEG-1. Another difference between the two standards is that MPEG-2 introduced support for lower sampling frequencies which improves performance at lower bitrates [20].

2.3.2 The AMR standard

The AMR codecs were developed by the European Telecommunications Standards Institute (ETSI) for GSM cellular systems, and is now a mandatory codec for 3G cell phone networks. The AMR codec has bitrates ranging between 4.75 kbps and 12.2 kbps, and has a sampling rate of 8000 Hz. Speech or audio is contained in 20 ms frames [26].

The Adaptive Multi-Rate Wideband (AMR-WB) is also a multi-mode speech codec which was developed by 3GPP to be used in cell phone networks. It has bitrates ranging from 6.6 kbps to 23.85 kbps. The sampling frequency used is 16000 Hz and speech or audio is contained in 20 ms frames. This means that each AMR-WB frame represents twice has many speech samples as an AMR frame [26].

The AMR and AMR-WB file formats were originally designed for circuit switched mobile radio systems. This can be attributed to their flexibility and robustness. These formats are also ideal for real time services such as speech and streaming of audio over packet switched networks (Internet) [26].

These codecs were designed with robustness in mind and the multi-rate/multi-mode capability of these codecs enables them to preserve high speech quality in varying transmission conditions. Mode adaptation enables a session to find the balance between speech compression rate and error tolerance, and divides the overall bandwidth between speech data and error protective coding. Mode adaptation is performed by the decoder informing the encoder which mode is most suitable. The mode change signal is known as Codec Mode Request (CMR). CMRs are sent as part of the speech frames, and there is no need for out-of-band signalling [26].

2.3.3 The MPEG-4 standard

MPEG-4 Version 1 was approved in December 1998 and version 2 was approved in 1999. Each version follows on from the next, which means that Version 3 is backward compatible with Version 2 and Version 2 is backward compatible with Version 1. Existing tools and techniques are never replaced in subsequent versions, instead new profiles and tools are added to the current ones. Figure 2.12 depicts the relationship between the MPEG versions:

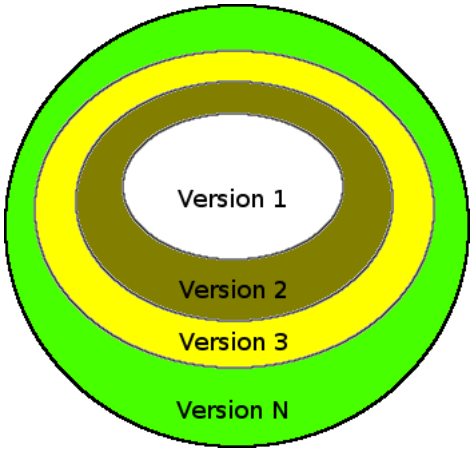


Figure 2.12: Relationship between MPEG versions [32].

One of the major advantages of MPEG is that MPEG encoded files are considerably smaller than audio and video files encoded using alternate standards. The reason for this is due to the highly sophisticated compression techniques of the MPEG compression algorithm [32]. The MPEG-4 standard does not define or implement its own transport layers, instead it uses an ammended MPEG-2 transport stream and also provides support for transport over the Internet Protocol (IP) [46]. Through the use of the Delivery Multimedia Integration Framework (DMIF) [42], the MPEG-4 standard enables developers to abstract from the transport mechanisms and focus on the application layer. DMIF enables support for homogeneous and mobile networks.

2.3.4 The 3GPP standard

3GP was defined by the 3GPP in 1998 and the Third Generation Partnership Project 2 (3GPP2). These two bodies involve the cooperation of several countries regarding the standardization of mobile phone formats. Their aim is to enable consistent delivery of multimedia formats over 3G networks [2].

3G enabled cell phones are becoming commonplace, and provide higher data transfer speeds than previous standards. This next generation of mobile technology brings with it, a file format

known as 3GP which defines video files and enables multimedia streaming on 3G networks, thus enabling users to share and watch video over networks [2].

There are two international standards for the 3GP file:

1. 3GPP - which uses the file extension .3gp, and is compatible with GSM enabled cell phones.
2. 3GPP2 - which uses the file extension .3g2, it is compatible with CDMA enabled cell phones.

3GP is a multimedia container format, which allows several different types of files to be housed within this container. This container allows audio, video and bitrate information to be stored together [2].

2.3.5 The Waveform Audio File (WAV) standard

The WAV file format is a native Windows file format which is used for storing digital audio data. WAV files use the standard Resource Interchange File Format (RIFF) which groups file contents into chunks, with each containing its own header and data. Each chunk header contains the type of chunk and the size of the chunk, thus allowing programs which don't recognize the chunk to skip over it and continue processing other chunks. Certain chunks contain sub-chunks, and each chunk should be a multiple of two bytes. For the purposes of simplicity, only the most common WAV file chunks are mentioned [51]. Figure 2.13 below provides an overview of the WAV file layout:

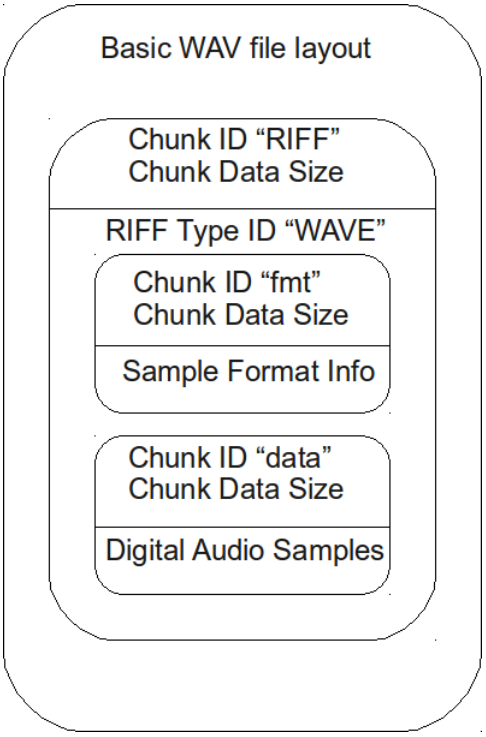


Figure 2.13: Basic WAV file layout [51].

2.3.5.1 WAV file header (RIFF type chunk)

WAV file headers follow the RIFF standard header with size 8 bytes. The header begins with a chunk ID equal to “RIFF”; followed by the chunk size, which is equal to the file size minus the 8 bytes used for the header. The RIFF chunk determines the type of resource found in the RIFF chunk, which for WAV files is always equal to “WAVE”. The RIFF chunk is followed by WAV chunks which define the audio [51].

2.3.5.2 WAV chunks

There are a number of defined WAV chunks. Most WAV files contain only 2 of these chunks, namely: the Format Chunk; and the Data Chunk. These two chunks are needed to describe the format of the audio samples. Although it is not mandatory, the format chunk should be placed before the data chunk. If the format and data chunks were reversed, the whole data and format chunks would have to be streamed before playback could commence [51].

2.4 Summary

This chapter provided a thorough introduction to the technologies necessary for media streaming and the platform on which these components run, as well as the Bluetooth protocol and its constituent parts. This chapter then illustrated the media file formats associated with media streaming, which aids in the implementation of the mobile testbed described in Chapter 3. The various media file formats discussed in this section have their own strengths and weaknesses, and even though each have been designed with varying purposes in mind, most media file formats conform to a similar structure, namely the "header/frame" structure. The next chapter describes the design and implementation of the mobile testbed used for experimentation.

Chapter 3

Design and Implementation

3.1 Introduction

This chapter describes the design and implementation of the mobile testbed which was created to investigate the viability of Bluetooth streaming between mobile devices. It begins with an overview of the development environment, which outlines the various platforms; APIs and hardware specifications used for the development of the mobile testbed. The layout of the mobile testbed is described with the aid of diagrams, which are then followed by an introduction to the technologies used in the mobile testbed, with the roles of the sending and receiving devices being detailed. Overviews of the Symbian operating system, the J2ME environment, the FileConnection API and the MMAPI are then given. The significance of buffers and threads are then highlighted, with particular attention being paid to the layout of buffers in the mobile testbed, along with media playback control mechanisms. Finally, the interaction between the user interface and the technologies are then explained.

3.2 Development environment

For the purposes of evaluating current development methods, and future expansions to this research, it is necessary to outline the environment in which the testbed was developed. The operating system used was Ubuntu Linux (Kernel 2.6.31-15-generic) AMD 64bit, running on an Intel(R) Core(TM)2 Duo CPU T5800 2.00GHz with 4GB RAM. The chosen development platform was the Netbeans IDE version 6.7.1 and the language used for the mobile testbed was J2ME CLDC (version 1.1)/MIDP (version 2.0). In order to integrate the necessary capability for the testbed, a number of optional APIs were included: FileConnection API (version 1.0) [29] for media file access and serialization; the Bluetooth Wireless Technology API (version 1.1)

[33] for transmission and receipt of streams; and MMAPI (version 1.1) [16] for media playback and control. The testbed runs on the Symbian operating system version 9.2, S60 3rd Edition, Feature Pack 1. Both the Nokia N95 8GB and the Nokia N82 have the same Symbian firmware version and feature pack. Throughout this dissertation, all tests were performed on these two devices. For consistency and clarity, these devices will be referred to as Device A and Device B respectively.

3.3 Flow chart diagrams

The mobile testbed was developed in order to evaluate the feasibility of audio and video streaming on the J2ME platform. To gain a better understanding of the APIs involved, their functions, and how they are combined to stream audio and video, Figures 3.1 and 3.2 illustrate the flow charts for the sending and the receiving devices respectively. The purpose of these flow charts is to depict the major processes involved in media streaming, so all events relating to user input have been omitted for simplicity's sake.

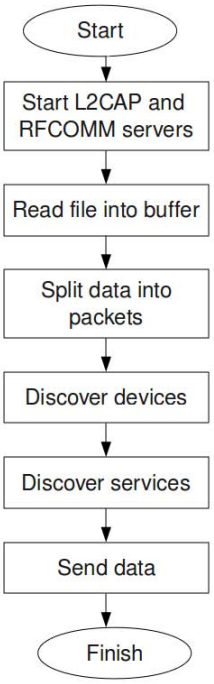


Figure 3.1: Flow chart diagram (sending device side)

The flow of events gives an insight into the structure of the mobile testbed. As can be seen in Figure 3.1 the L2CAP and RFCOMM servers are started, followed by the media file being read into a buffer via the FileConnection API. After the serialization of the media file has taken

place, it is split into packets which correspond to the transmission unit sizes of Devices A and B. Device discovery is then initiated. When the receiving device has been found and selected, it is queried about the required services. The user is then able to send the packetized media.

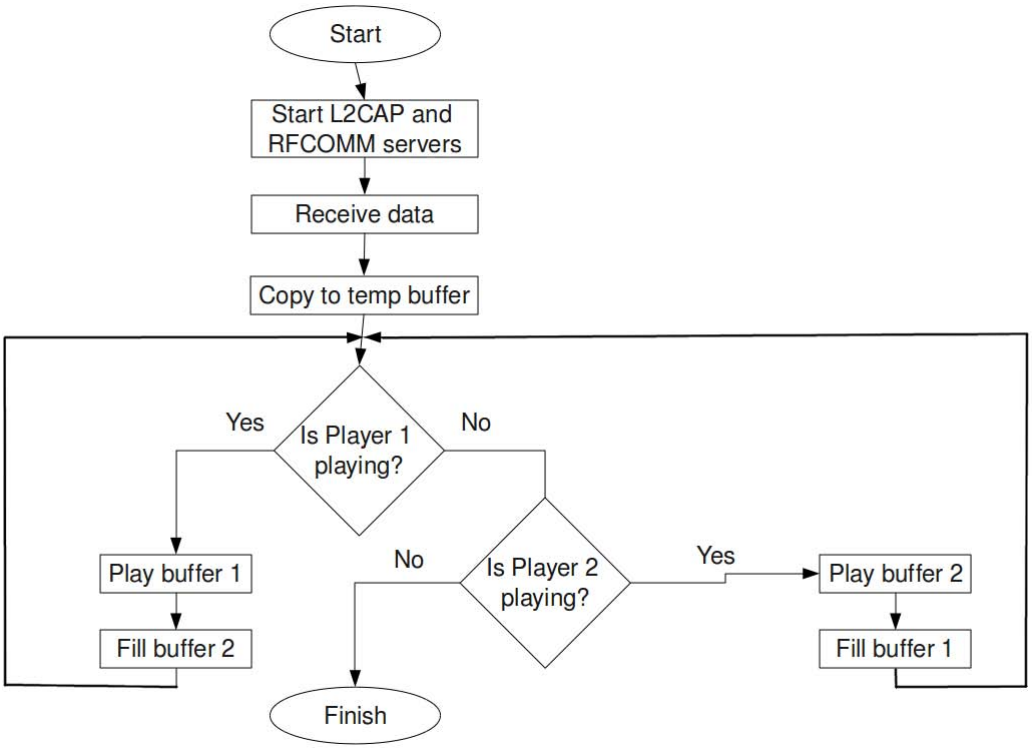


Figure 3.2: Flow chart diagram (receiving device side)

As described in subsection 2.2.3, the most successful architecture implemented by Vazquez-Briseno and Vincent [54] and Sillén and Nordlund [50] involved two buffers and two players. It was therefore decided to implement multimedia streaming using a similar technique. As can be seen in Figure 3.2, the L2CAP and RFCOMM servers are started upon application initialization, and receive data (if any). The benefit of starting the L2CAP and RFCOMM servers upon application initialization is for the receiving device to advertise its services. Once the user initiates the playback sequence, all received media is copied to a temporary buffer (which enables the receiving buffer to handle realtime data); while one Player is playing media, another buffer is being populated with data from the temporary buffer. More information regarding the functioning of the buffers is described in Subsection 3.4.7.

3.4 Technologies

This section describes the technologies used in the implementation of the mobile testbed. The processes involved in streaming on the client and server sides of the mobile testbed are documented, and the relationship between the J2ME environment and the Symbian operating system is highlighted.

3.4.1 Sending device interaction

From the flow chart diagram in Figure 3.1 it can be seen that there are a number of distinct processes needed in order for the client to begin streaming the media file. The mobile testbed consists of one application which hosts the functionality of both the sending and receiving devices, and thus each instance of the application can act as either the sender or the receiver, depending on the user's choice. The process therefore begins with the initialization of the L2CAP and RFCOMM servers, followed by the media file being read into a buffer. Since the media file resides on the local file system, it needs to be accessed before any streaming can commence. This process of accessing the media file and preparing it for streaming is outlined in Subection 3.4.5. In order to transmit data over the L2CAP protocol, data first needs to be packaged into packets which conform to the minimum and maximum transmission unit sizes of this protocol. A method in the mobile testbed accepts the serialized data, and 'converts' it into packets matching the size of the MTUs of the L2CAP protocol on that specific device. Thereafter, devices and their associated services are discovered, at which point the user can initiate the transmission of the serialized and appropriately packaged data. Section 3.4.5 is only associated with the processes on the sender's side of the mobile testbed.

3.4.2 Receiving device interaction

Both the sending and receiving devices begin with the initialization of the L2CAP server. The receiver continuously listens for incoming data, and upon receipt, this data is copied between a number of buffers, the process of which is highlighted in Section 3.4.7. When a sufficient amount of data has been received (at least 100KB), and the user has pressed the play button (Figure 3.8c), media playback can commence. Subsection 3.4.6 is only associated with the processes of the receiving device.

3.4.3 The Symbian Operating System

The first Symbian OS released was v6.0 in 2001, and mobile phones based on this OS were 'open' meaning that users could install their own applications. Development of the Symbian OS continued over the next few years until 2005 when Symbian OS v9.0 was released . This new version of the Symbian OS provided benefits such as more security and considerable cost savings to manufacturers through the use of standard tooling and applications. These mobile devices would often run for months if not years and thus the OS was designed to deal with such rigorous requirements. The Symbian OS is an event based OS, where the CPU is released when not being directly used by the OS [14].

Before modern OSs for mobile phones emerged, applications had to be developed for each individual platform. J2ME was an option for achieving cross platform compatibility of applications. Since the emergence of OSes for cell phones, the ability to develop applications for a wide variety of devices has become far less complicated than before. The two main OSes for cell phones are the Symbian OS and the Binary Runtime Environment of Wireless (BREW) [27]. Even though Symbian is the more widespread OS in use with a market share of 47.2% [11], plenty of debate still remains as to which OS is more advantageous and will dominate the market. Both OSes provide support for the Mobile Information Device Profile (MIDP) [44] profile and also offer functionality beyond the scope of J2ME [14].

The Symbian OSes management and user interface rendering of Java MIDlets is identical to native applications on this platform. Symbian provides the following benefits to Java MIDlets: MIDlets are managed in the same way as native applications in terms of installation, execution and removal; MIDlets make use of the native user interface components, which are efficient and simplify the control of consistent user interfaces between MIDlets and the native applications. Heap sizes and number of characters contained in text boxes or text fields are unlimited; Record Management Systems (RMS) have no limitations and any number of records can be stored, retrieved and altered. The Symbian OS was clearly designed with the user (consistent user interfaces and execution cycles) as well as the developer (unlimited heap sizes and flexible record stores) in mind. Native colour depth is supported and there is one Virtual Machine (VM) instantiation per MIDlet suite, which is unusual for a mobile phone OS [19]. With the Symbian operating system handling J2ME and native applications in a similar manner, development of J2ME applications on this platform provide speed, reliability and consistency across devices.

3.4.4 Java 2 Mobile (J2ME)

J2ME provides a flexible, robust environment for applications running on cell phones, personal digital assistants and printers. J2ME provides support for user interfaces; security; built in network protocols; and networked and offline applications that can be downloaded dynamically [52]. There are two types of J2ME configurations: the Connected Limited Device Configuration (CLDC), and the Connected Device Configuration (CDC). CLDC is the smaller of the the configurations, and the MIDP is built on top of this configuration. CLDC was designed for devices with limited system resources such as cell phones, personal digital assistants and two-way pagers. These devices usually have a minimum of 128 KB to 512 KB of memory. MIDP offers the core functionality to mobile devices such as the user interface, networking, storage, and application management [34].

CLDC devices typically have memory of between 160 KB and 512 KB and a 16-bit or 32-bit processor with a clock speed of 16 MHz or higher. CLDC devices provide a lower power supply level and consume less power than their CDC counterparts. Due to the limited power and processing capabilities of these devices, user interfaces (if any) and bandwidth are limited.

Most CLDC devices consist of lower end cell phones, low end PDAs, and Radio Frequency Identification (RFID) devices. It can be seen that CLDC devices are constrained in terms of power, processing capability, memory capacity and network bandwidth and connections, and thus the standard Java Virtual Machine (JVM) is inappropriate. Instead a cut down version of the JVM is used, known as the Kilobyte Virtual Machine (KVM). These KVMs run on a number of platforms, such as Symbian OS, Palm OS, Windows Mobile, and Embedded Linux [57].

CDC devices typically possess a 32 bit processor with 2 MB of Random Access Memory (RAM) and 2.5 MB of Read Only Memory (ROM). The battery life of CDC devices is considerably better than that of CLDC devices, and network connectivity is reliable. With the increased processing capabilities of CDC devices, multiple user interface designs are possible, which provides an advantage for CDC devices over CLDC devices in terms of user interaction and aesthetic appeal. Devices which make use of CDC include high-end PDAs and cell phones. Clearly it can be seen that CDC is considerably more powerful than CLDC. Unlike CLDC, CDC makes use of the standard JVM.

CLDC and CDC provide profiles which enable the functioning of standard Java APIs in such environments with limited resources. There are three profiles which exist for CDC: The Foundation Profile (the core for the other two profiles), the personal basis profile and the personal profile. CLDC on the other hand provides two profiles: The MIDP and the PDA profile. Both the CLDC and CDC J2ME configurations can be seen in Figure 3.3:

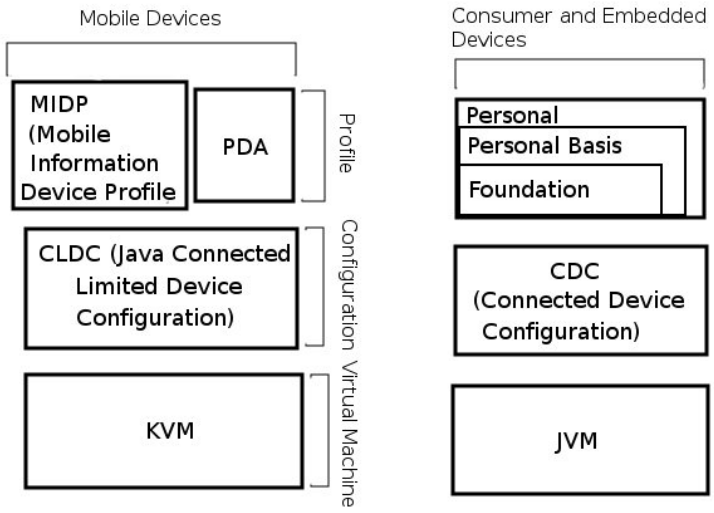


Figure 3.3: J2ME Configuration [57].

From Figure 3.3 it can be seen that the major advantage of the CDC configuration over the CLDC configuration is that it makes use of the JVM, thus enabling the execution of standard Java applications on devices implementing this configuration.

With the amazing evolution of mobile wireless technologies, there has been significant development in the area of mobile applications, especially in countries where mobile phone and PDA penetration is high. Service providers are also encouraging developers to develop applications that provide value added service and make full use of 3G networks and emerging technologies. Traditional mobile applications such as voice and data services are simply not enough anymore, since consumers are looking for mobile devices and applications that provide rich content entertainment, ubiquitous information access and agile business operations [57].

With an overview of the J2ME platform, it is necessary to outline some of the components which can be used for the streaming of audio and video, namely the File Connection API and the MMAPI.

3.4.5 The FileConnection API

Input/Output operations on a J2ME enabled device are made possible through the use of the Generic Connection Framework (GCF) Connection interface, which provides implementations specific to each type of connection. The Connection interface has access to many different protocols and specifies access to these files through the use of a URL syntax. The GCF has the

functionality to support connections to files, but this support is not mandatory and has never been part of J2ME or MIDP [40]. The FileConnection API provides support for accessing file systems such as local memory, flash memory or removable memory cards. Instead of replacing the Record Management System (RMS), the FileConnection API enables the seamless communication between MIDlets and the native applications on the device.

Due to the fact that certain directories and file systems are restricted to keep security mechanisms in tact, a security framework exists which controls the accessing of files on the device. This security framework throws a security exception when unauthorized directories are accessed. MIDlets can either be untrusted or trusted (signed MIDlet). In the case of an untrusted MIDlet, the user is prompted for authorization of each operation. With trusted MIDlets the device grants permission to the MIDlet based on the security domain settings which are contained within the MIDlet.

There are three modes which exist in the FileConnection API: READ mode; WRITE mode; and READ_WRITE mode. In the case of an untrusted MIDlet the user will be prompted for authorization for all of these modes attempting to access the file system. For trusted MIDlets, the authorization depends entirely on the permissions granted to the MIDlet upon compilation. The READ mode enables reading of files from the file system and allows for the opening of input streams to the file. The WRITE mode enables writing of data to the files and allows for the opening of output streams to the file. The READ_WRITE mode allows both reading and writing to the file, as well as permissions to open input and output streams.

The FileConnection API enables the reading of files into buffers which can then be used to transmit the audio data to another device, using any available transmission medium. It can be seen that the FileConnection API greatly simplifies the accessing and serialization of files necessary for multimedia streaming [40].

Since the FileConnection API and the MMAPI are closely linked with regard to multimedia streaming on the J2ME platform, Section 3.4.6 introduces the MMAPI, explains the associated architecture, and how the MMAPI integrates with the MIDP profile.

3.4.6 The Mobile Media API

The MMAPI has had a significant role to play in the introduction of media on mobile devices [21]. The MMAPI is a subset of the MIDP 2.0, and enables Java enabled mobile devices to discover and utilize media capabilities. These capabilities include media playback; advanced control; and capturing. Media can be played back from the local file system (whether this media is contained within a JAR file or simply stored on the device) or from the Internet.

Since most mobile devices which support J2ME are constrained in terms of memory and processing power, the MMAPI was designed in such a way so as to accommodate for these

constraints. This lead to the MMAPI being designed in accordance with the following set of rules: low footprint API; ability to support multiple media types; support for basic controls; support for device capabilities discovery; support for basic audio and tone generation. As mentioned above, MMAPI is usually present on devices with constrained memory, and thus it needs to be a low footprint API using as little memory as possible, since many Java enabled mobile phones run on the CLDC configuration. This memory is to be efficiently utilized by the virtual machine, core libraries, user MIDlets, as well as the MMAPI. Instead of the MMAPI specifying a list of supported protocols, it is rather defined as a set of interfaces which device manufacturers implement, thus truly making the MMAPI protocol and format agnostic, and allowing for the discovery of new protocols and formats. The support for basic controls guarantees uniform procedures for managing media (start, play, stop). Since each manufacturer implements the MMAPI based on the protocols and formats which their devices support, its is necessary for the MMAPI to be able to query the device for the supported protocols and formats. Not only does the MMAPI guarantee support for basic controls, but it also mandates the support for the playback of at least one media type as well as support for tone generation[21].

3.4.6.1 Compatibility of the MMAPI with MIDP 2.0

MMAPI is an optional package for the J2ME platform and is present in MIDP 2.0, which means that any device supporting MIDP 2.0 will be able to run applications which make use of the MMAPI.

Figure 3.4 shows the relationship between the MMAPI and the J2ME environment, its optional packages, profiles, and configurations:

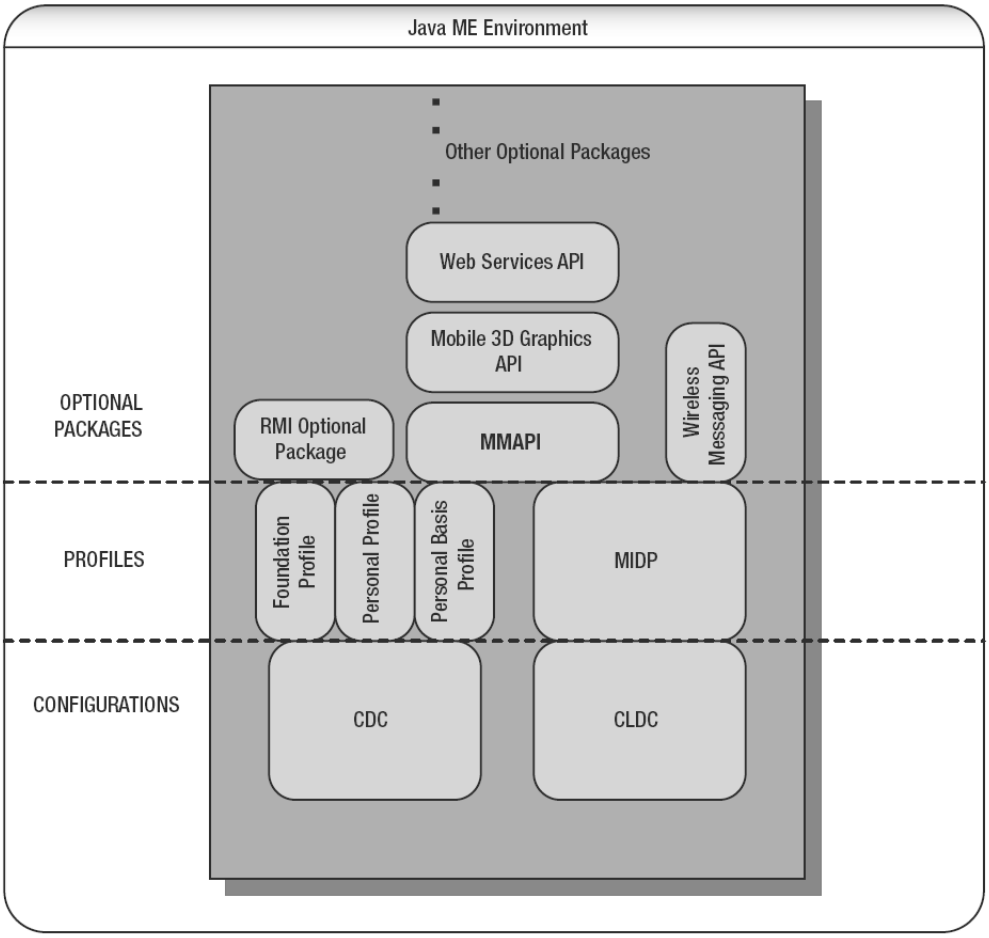


Figure 3.4: Relationship between the MMAPI and the J2ME Environment [21].

From this figure it can be seen that the CDC and CLDC configurations serve as a base for the J2ME environment. Optional packages are built on top of the profiles, which in turn are built on top of the configurations. The MIDP profile and the optional Wireless Messaging API are specific to the CLDC configuration, while the Foundation [12], Personal [17], and Personal Basis profiles [15] as well as the RMI optional package [28] are specific to the CDC configuration. The figure shows that the MMAPI is accessible from both configurations, meaning that multimedia capabilities can be implemented on higher and lower end devices.

3.4.6.2 MMAPI Architecture

The MMAPI is comprised of two main components: Player and DataSource. The Player is responsible for the processing and playback of media, whereas the DataSource is responsible for the location and retrieval of this media. The Player parses the data supplied to it from the DataSource and may then render this data on the device and also provide controls to manipulate it [21].

The Player has five states: unrealized, realized, prefetched, started, and closed, which can be

seen in Figure 3.5:

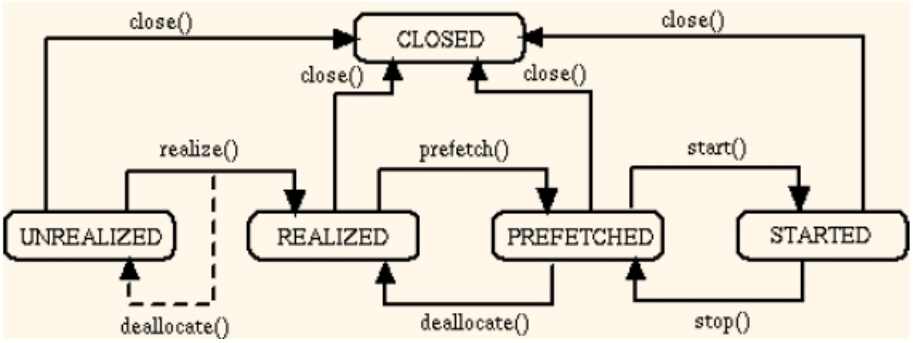


Figure 3.5: MMAPI States and Transitions [25].

Initially the Player is in the unrealized state and hasn't attempted to acquire the audio or any resources such as audio hardware. Once the Player has been initialized and the realize method has been called, the Player then enters the realized state at which point it has located the media intended for playback. The Player can go back to the unrealized state by calling the deallocate method. After the prefetch method is called the Player enters the prefetched state where it is ready to commence with audio playback, has populated any necessary buffers and acquired all necessary hardware. The Player is considered to be in the start state when it actually starts rendering the audio. The Player can enter the closed state from any of the other states, and is considered closed when all network connections have been closed and all resources have been released. Once a Player has been closed it cannot be reused [25].

The DataSource is comprised of SourceStreams. A SourceStream provides an abstraction for a single media stream, and one of its main advantages is that it is randomly seekable, meaning that the media can be accessed from a random position. Some media types do not support seekable operations, this is however taken care of by querying the SourceStream in accordance with its support for seekability. One of the three integer constants listed below indicates the seekability of the SourceStream:

1. NOT_SEEKABLE
2. SEEKABLE_TO_START
3. RANDOM_ACCESSIBLE.

The first constant indicates a non-seekable stream, while the second represents a stream which is seekable from the beginning. The third constant represents a truly seekable stream. which allows media to be randomly accessed from any position within the stream. Apart from being

randomly seekable, the SourceStream has the advantage of reading the logical transfer size of the data, which is ideal for creating in memory buffers of the correct size for multimedia receipt.

A Player is created through file extension evaluation and mapping the file to the appropriate Multipurpose Internet Mail Extensions (MIME) type. If the file extension is not present (when receiving data from an input stream) the user can create a Player by specifying the media MIME type.

The MMAPI was used extensively throughout the mobile testbed, and media was realized and played back by means of the Player within the MMAPI. The Player was used in experimentation with the audio and video file formats, and data sources were implemented in the form of input streams to provide a continuous stream of media for buffering and playback. Buffering was used throughout the implementation of the testbed to temporarily store partially downloaded media. The next section shows how buffering was implemented throughout the mobile testbed, and explains the importance of buffers in audio and video streaming on the J2ME platform.

3.4.7 Buffers

This testbed uses five buffers for temporary storage when serializing files, as well as for receiving and playback of transmitted media. The FileConnection API is responsible for populating the buffer used to store the serialized media file. Once the temporary buffer is filled with bytes representing the media file, it is then transmitted to the receiving device, where another buffer will receive the entire media file. The main receiving buffer then copies all of its received data into a temporary buffer which allows the receiving buffer to continue receiving the media file while the temporary buffer is utilized for media playback and manipulation. Two Players and two buffers are used for multimedia streaming on this testbed. While Player 1 commences with media playback from buffer 1, Player 2 remains in the unrealized state, whilst buffer 2 is populated with the next portion of the media stream. This technique is known as double buffering and overcomes the lack of support of J2ME and the MMAPI for media streaming across protocols other than HTTP. This lack of support is overcome by leading the Player into thinking that the buffer contains the entire media stream. Figure 3.6 depicts the various buffers used and their relationships and functions:

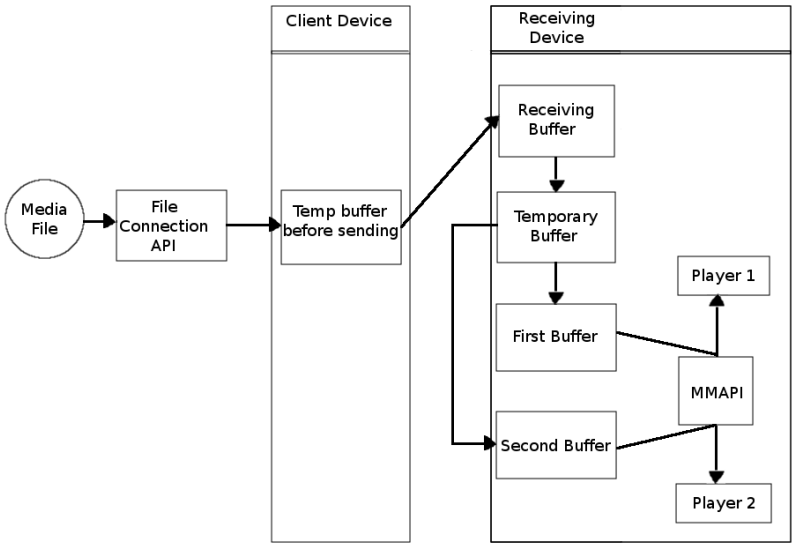


Figure 3.6: Testbed buffers and their relationships

The receiving buffer is responsible for receiving the entire media file, therefore the size of the data contained in this buffer is constantly expanding until the end of the media stream is reached. If either the first or second buffer had to be populated directly from the receiving buffer, problems regarding the monitoring of which bytes had been copied to the first and second buffers would be experienced. It was therefore decided to implement a temporary buffer which would remain constant in size until the media contained within had been copied to either the first or second buffers. Since the temporary buffer is constant in size before and during the transfer of media to the first and second buffers, this simplifies the process of determining exactly how many bytes have been copied between all the buffers, and ensures playback of the entire media file without trailing bytes (which would be the case if the first and second buffers were directly populated from the receiving buffer). The double buffering technique as well as the architecture used in this testbed are adapted from research conducted by Vazquez-Briseno and Vincent [54].

None of the buffering, and hence the streaming and playback of the media file would be possible without the aid of threading. This mobile testbed makes use of threading for all operations related to the transmission and receipt of the media stream, thus allowing user input to continue unhindered. Due to the L2CAP server constantly listening for incoming packets and utilizing all the resources provided by the main thread, user input would be intermittent and thus unacceptable without the use of threading.

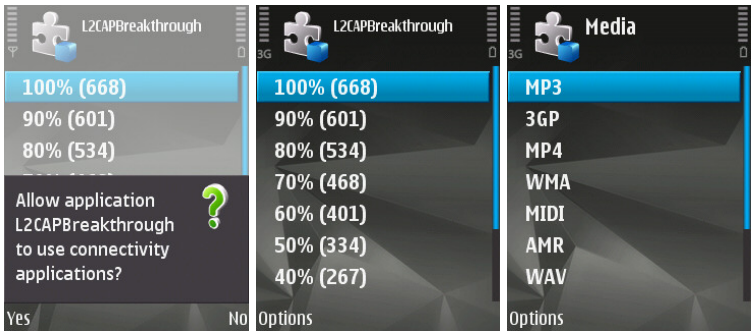
3.4.8 Media playback control mechanisms

When receiving any of the media files, it was discovered that without any mechanism for controlling the rate of transfer, bytes were lost at the end of the transmission, which resulted in the entire file not being downloaded. In addition to the loss of bytes, without any media playback controlling mechanism, the media would encounter considerable amounts of jitter during playback, or simply ceased playback altogether. Two media playback control mechanisms were experimented with. The first mechanism involved slowing down the transfer speed for each packet received, which resulted in less jitter and the complete transfer of the media file. It was found that this method of media playback control resulted in decreased transfer speeds, but still maintained an acceptable level of playback and response to user inputs.

The second media playback control mechanism was then implemented which involved slowing down the media stream once 95% of the file had been transferred. This resulted in the media file being streamed successfully, while simultaneously transferring the entire media file and maintaining an acceptable level of user input and response. The second method of media playback control resulted in increased amounts of jitter when compared to the first method. Since the amount of jitter essentially determines the efficiency of streaming, it was decided to adopt the first method of media playback control for all files other than the WAV audio file due to file size and strain being placed on the processing capabilities of the devices. The adoption of the first method resulted in decreased transfer speeds, but increased playback quality.

3.5 User interface and flow

The design of the user interface is minimalistic and enables the user to interact with the various components of the testbed with ease. Before any of the streaming can commence, the L2CAP server needs to be initialized, which is depicted in Figure 3.7(a):

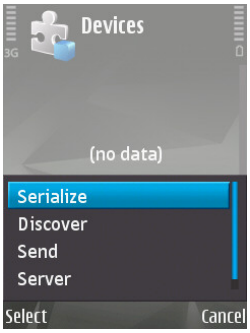


(a) L2CAP server initialization (b) MTU size choice (c) Media file choice

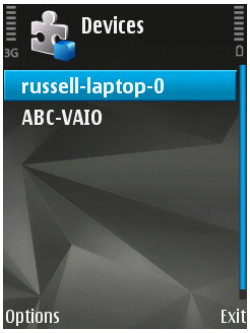
Figure 3.7

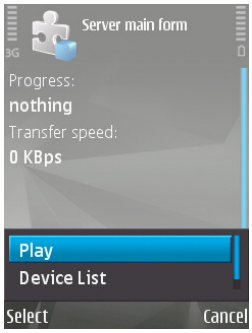
If the L2CAP protocol is chosen as the media delivery protocol, it is necessary to deal with the MTUs of transmitted media packets. The MTUs of both Device A and Device B are 668 bytes, which is the maximum transmission unit available (hence depicted as 100% in Figure 3.7(b)). A default MTU size of 668 bytes is provided for the RFCOMM protocol. In the case of the L2CAP protocol, once the MTU has been chosen, the type of media file to be streamed is chosen. The media file choice can be seen in Figure 3.7(c). Since this testbed is only a proof of concept, only one file of each type was stored on the device.

Once all the parameters necessary for streaming have been initialized, the device search and service discovery phases of the streaming process can commence. Figure 3.8 shows the device list screen, which lists all discovered devices, and serves as the main screen for controlling all events related to the transmission of the media stream:



(a) Device list screen





(b) Discovered devices (c) Media playback

Figure 3.8

From this screenshot it can be seen that the media file type chosen from Figure 3.7(c) can be serialized by means of the *Serialize* button. Devices (as well as services on the corresponding devices) can be discovered by the user pressing the *Discover* button. All discovered devices will be listed on the device list depicted in Figure 3.8(b), at which point the user can choose the device to which they want to stream media. Once the device has been chosen, service discovery needs to take place before media streaming can commence. Upon selecting the receiving device and initiating the streaming process, the services of the applicable device are discovered immediately prior to the commencement of streaming. The *Server* button navigates to the screen depicted in Figure 3.8(c), which deals with media receipt and playback.

3.6 Summary

This chapter has described the components necessary for streaming and detailed their role in the mobile testbed. The advantage of having sending and receiving capabilites on both the sending and receiving devices allowed for maximum flexibility and enabled bi-directional streaming. The FileConnection API and the MMAPI simplified streaming on the J2ME platform and served as vital components in file serialization and media playback and control respectively. The use of

threading allowed media receipt and thus buffer population while still maintaining a responsive user interface. The simple interface and minimalistic design enabled maximum efficiency and provided a suitable platform for addressing the research outcomes. The next chapter will detail the experiments conducted and results obtained in satisfying the research outcomes.

Chapter 4

Results and Analysis

4.1 Introduction

This chapter outlines the experiments conducted using the mobile testbed, and serves to address the research outcomes outlined in Section 1.3 by investigating the various streaming components and how they combine to achieve streaming on the J2ME platform. This chapter begins by illustrating the benchmark values associated with streaming, followed by an overview of the apparatus used, such as performance metrics and file compression. The various experiments relating to the research outcomes, as well as the results obtained are then described. These experiments include a comparison between the RFCOMM and L2CAP protocols which yields the superior protocol in terms of transfer speed and efficiency. Once the most efficient protocol is determined, it is necessary to optimize the process of multimedia streaming by determining the suitability of various media types for streaming. The effects of distance on transfer speeds of the L2CAP protocol are then determined, which shows the range of distances in which the transfer speed is most suitable to multimedia streaming. Since the transmission unit size of the L2CAP protocol can be altered, the effects of this alteration need to be determined so as not to implement multimedia streaming with transmission unit sizes which produce sub-standard streaming quality. With streaming being conducted at varying distances, it is reasonable to expect the presence of dropped packets, and it was deemed necessary to investigate the effects of dropped packets on multimedia streaming.

4.2 Streaming benchmarks

Before attempting to stream media, an understanding of bitrate, as well as a theoretical platform of streaming is necessary. Bitrate measures how much data is transmitted within a specified time

frame. As well as being used to describe rate at which bits are transferred, it can also be used to describe the quality of an audio or video file. For instance, if an audio file is compressed at 192 Kbps, the sound quality would be better than the same file compressed at 128 Kbps, since there are more bits for every second of playback [53].

In order to compare the theoretical bandwidth needed for streaming, a benchmark value of 136.39 Kilo**bytes** per second (KBps) is required, which is the result of the average transfer speed when transferring a 6.6 MB M4A/AAC file using the MTU of 668 bytes from Device B to Device A over the L2CAP protocol. The benchmark value was obtained by transferring the audio file twice, and then finding the average transfer speed. The reason for the audio file being transferred twice to calculate the benchmark value is simply for consistency purposes which minimizes the effects external factors (interference, reflection) have on the benchmark value. In order to determine if it is theoretically viable to stream a media file over the Bluetooth protocol, the bitrate needs to be calculated. Since we have the size of the file, and the length of the file, the required bitrate can be calculated as follows:

Algorithm 4.1 Calculating the required bitrate

x = Size of file (KB)
y = length of the media file (seconds)
z = bitrate of the media file

*z = (x * 8 * 1024 * 1024) / y*
*= (6727.51 * 8 * 1024) / 210*
= 262437.07 bps
= 32.04 KBps

Therefore it can be seen that the minimum bitrate needed to transfer the AAC file is 32.04 KBps, which is considerably less than the capable benchmark rate of 136.39 KBps.

The same formula can be applied to video files. In order to determine the bandwidth required to stream video files using the L2CAP protocol, a 6.8 MB MP4 video file of duration 22 seconds was used in the calculation below. Video files are comprised of two parts, namely the video data and the audio data. Both of these components can be encoded at the same bitrate or at different bitrates. Since we are concerned with streaming the video file as a whole, the only information which is needed according to the formula above is the size and duration of the media file. The required bitrate needed to stream the MP4 video file is 324.11 KBps, which

is considerably more than the capable benchmark rate of 136.39 KBps. Theoretically, the only possible way to stream this video file across Bluetooth would be to make use of EDR with 8DPSK modulation, which yields a transfer speed of 3 Mbps (384 KBps) [7]. Devices A and B have Bluetooth version 2.0 + EDR which means that they have a theoretical maximum useful transfer rate of 2.1Mbps (268.8 KBps) [31]. Table 4.1 shows the minimum bitrates required to stream MP3, MP4 and M4A/AAC files, as well as the benchmark transfer speed:

File Type	File size (MB)	Minimum transfer speed (KBps)	Benchmark transfer speed (KBps)
MP3	4.3	15.67	136.39
MP4	6.8	324.11	136.39
M4A/AAC	6.6	32.04	136.39

Table 4.1: Minimum and benchmark transfer speeds for multimedia streaming

Table 4.1 shows that the MP3 file requires the least amount of bandwidth for streaming using the L2CAP protocol, and it can be seen that the benchmark rate is more than capable of streaming the MP3 file, with an excess of 120.72 KBps.

4.2.1 Constant bitrate and Variable bitrate encodings

The efficiency of video steaming can be improved by adopting one of two bitrate encoding schemes: constant bitrate (CBR) and variable bitrate (VBR). CBR encodes the entire media file at a constant bitrate which results in quality of the encoded content being inconsistent. Since certain parts of a media file are more complicated to compress than others, the resultant quality of the more complicated parts of the media file stream are of lower quality when CBR encoding is used. VBR encoding fares well with media files which contain a mixture of simple and complex data. With VBR the encoder assigns less bits to the simple parts of the media file, thus allowing more bits and hence higher quality to be assigned to the more complex portion of the media file/stream [36]. As an example of the difference that encoding can make, consider the following scenario: for video with minimal change between frames (e.g a news reader's head and moving mouth), the benefits of VBR would not be reaped, since there is little difference between the simple and complex parts of the media file/stream. In such a case, the more efficient encoding option would be CBR. For video with constantly changing frames (e.g video of a 100m sprint), VBR would have a much better compression rate due to its assignment of varying bits to the simple and complex parts of the file. Sometimes CBR and VBR encoded files may be the same size, but the due to the superior compression of VBR, the resultant quality

of the VBR encoded file is considerably better than that of the CBR encoded file [36]. For the purposes of this research it was decided to implement streaming with CBR encoded files, as is the case with the MP3, M4A/AAC, MP4, 3GP, WAV, WMA, AMR, MIDI file formats. Since most of the file formats used in this research were audio files and the video files used have little to no frame variation, CBR was a more viable option.

4.3 Apparatus

This section describes the various performance metrics used throughout experimentation, as well as the way in which various files were compressed and converted to enable consistency of experiments across file types.

4.3.1 Performance metrics

In order to successfully compare the efficiency of the Bluetooth protocols and evaluate the effects of distance and MTU variation on transfer speed, metrics such as the transfer speed measured in Kbps and KBps are used. In collaboration with transfer speed, time was another metric used throughout experimentation. Time was recorded by making extensive use of the System.currentTimeMillis method. All timing was conducted on the receiving device, as the initial time taken to transfer the first byte between the devices is constant and as such was considered negligible. All experiments were conducted four times in order to minimize fluctuations encountered due to channel strength and quality. Since media file sizes vary considerably, the times taken to transfer each file will also differ.

The normalized time taken to transfer each file was used to enable a meaningful comparison to be made across files. Normalized time is calculated by dividing the average time by the file size of the file being experimented with. In order to assess the quality of audio playback, the subjective performance metric of clarity was included, which is assesed by users of the mobile testbed. Evaluation is conducted on a scale of one (poor quality) to ten (best quality) with the average of the four iterations for each transmission unit size being recorded. Standard deviation was included as metric in order to measure the variability of the results obtained.

4.3.2 File Compression

The experimentation process was initially conducted with eight media formats. In order to maintain an acceptable level of consistency, the same media file was compressed and converted between the various media formats, which allowed a successful evaluation of the suitability of each file for multimedia streaming. As shown in Table 4.2 a 4.3 MB MP3 file was converted

using the WMA, M4A and AMR audio file formats, resulting in respective file sizes shown in Table 4.2. This MP3 file was not converted into the WAV and MIDI file formats. The WAV audio file was omitted from the conversion due to shear file size (36.5 MB), which resulted in device memory issues. The MIDI file was excluded from the converted audio formats due to incompatibility as a result of file structure. In the case of video compression, a 1 MB 3GP file was converted to a 1.2 MB MP4 file. These conversions are represented in Table 4.2:

Original multimedia format	Converted multimedia format	File Size
MP3 (4.3 MB)	WMA	6.2 MB
	M4A	6.6 MB
	AMR	290 KB
3GP (1 MB)	MP4	1.2 MB

Table 4.2: Multimedia conversions

It can be seen that the size of the AMR audio file is considerably less than that of the MP3, WMA, and M4A file types, which is due to AMR being designed for voice only audio. The reason why the 3GP and MP4 video files consist of smaller file sizes than the audio files, is due to short video clips being encoded. Due to the high bitrate encodings of video files and their resultant effects on device processing requirements, it was decided to utilize smaller video files across the experiments.

4.4 Experiment 1: RFCOMM vs L2CAP

Bluetooth is comprised of three data transfer protocols, two of which can be used to send serialized data between devices, namely: RFCOMM and L2CAP. As mentioned in Section 2.1.2 RFCOMM is situated higher up in the protocol stack than L2CAP, and RFCOMM takes care of minimum and maximum transmission units, which greatly simplifies the task of serialization and transfer of data when compared to L2CAP. Characteristics of both protocols are outlined in Table 4.3:

Table 4.3: Comparison between RFCOMM and L2CAP

	RFCOMM	L2CAP
MTU control	Yes	No
Required multiplexing	No	Yes
Bluetooth protocol stack layer	Higher	Lower
Complexity of implementation in J2ME based on layer level	Easier	More difficult

The RFCOMM protocol is believed to be simpler to implement due to the fact that it is situated higher up in the protocol stack than L2CAP. Figure 2.1 shows the layers of the Bluetooth protocol stack, and it can be seen that L2CAP has to multiplex between the RFCOMM and SDP protocols, which would reduce the bandwidth available for data transmission. RFCOMM however depends on the L2CAP protocol for transmission, thus it too is expected to have decreased transmission rates.

4.4.1 Hypothesis

This experiment aims to (1) determine if there is a difference in transfer speed when sending from Device A to Device B and vice versa and (2) investigate the effect that the different protocols (RFCOMM and L2CAP) have on the transfer speed of a file, thus determining the protocol which supports the fastest transfer speed.

Since Hypothesis 1 can only be entirely evaluated by analyzing the results of the L2CAP and RFCOMM protocols, it was deemed necessary to test this Hypothesis for 1(a) L2CAP, and 1(b) RFCOMM.

The null Hypothesis pertaining to 1(a) is therefore that there is no statistically significant difference between the transfer speeds when sending between devices in either direction (A to B or B to A). The alternative Hypothesis pertaining to 1(a) is that there is a statistically significant difference between the transfer speeds when sending between devices in either direction.

The null Hypothesis pertaining to 1(b) states that there is no statistically significant difference between the transfer speeds when sending between devices in opposite directions. The alternative Hypothesis pertaining to 1(b) is that there is a statistically significant difference between the transfer speeds when sending between devices in either direction.

Once the Hypotheses 1(a) and 1(b) have been tested, conclusions can be deduced as to whether or not there is a difference in transfer speed when sending between devices in either direction.

The null Hypothesis pertaining to 2 is that there is no statistically significant difference between the transfer speeds of the L2CAP and RFCOMM protocols. The alternative Hypothesis

for 2 is that there is a statistically significant difference between the transfer speeds of the L2CAP and RFCOMM protocols.

4.4.2 Methodology

Device A and Device B were placed 30 cm apart, and the following media files were sent from Device A to Device B: a 4.3MB MP3 file; a 6.2MB WMA file; a 6.6MB M4A/AAC file; and a 10.7MB MP4 file. Since the file size is known by the receiving device, the average transfer rate, as well as the total time taken to transfer the file could be calculated. In order to obtain an accurate representation of the transfer speeds, each media file was transferred four times to account for signal interference. To obtain a true representation of transfer speed, no buffering was used across any of the tests in the testing framework (refer to Section 3.4.7 for details on buffering). An MTU of 668 bytes was used when transferring the media files between Device A and Device B. To allow comparisons to be made across the different files and file sizes that were transferred, normalized transfer times were calculated. To determine the most efficient protocol for multimedia streaming, a number of paired t-tests were performed: using the L2CAP protocol, a t-test was conducted involving data from Device A to Device B and vice versa. Another t-test was conducted using the RFCOMM protocol to determine the significant difference (if any) from Device A to Device B and vice versa. A final t-test was then conducted to determine if there was a statistically significant difference, and if so, which protocol supported the fastest transfer rate for multimedia streaming.

4.4.3 Results and analysis

4.4.3.1 L2CAP - Device A to Device B

Table 4.4 highlights the average time taken and average speed when transferring four files using the L2CAP protocol from Device A to Device B:

Table 4.4: Average time and average speed obtained with the L2CAP protocol (Device A to B)

File Type	AVG time (Seconds)	AVG speed (KBps)
MP3	32.924	134.859
WMA	46.883	135.803
M4A/AAC	50.201	134.061
MP4	80.272	136.400

Table 4.4 shows that the average transfer speed obtained when transferring each media files between devices A and B is consistent with a standard deviation (STDV) of 1.031, and is close

to the benchmark speed of 136.39 KBps obtained in Section 4.2. Transfer of the MP4 file yielded the highest transfer speed of 136.400 KBps followed closely by the WMA file with 135.803 KBps. It can be seen that the times taken to transfer the files vary considerably (STDV 19.928) , and this is due to the variation in file size. Figure 4.1 shows the average time, average speed, and normalized time when sending four files from Device A to Device B using the L2CAP protocol:

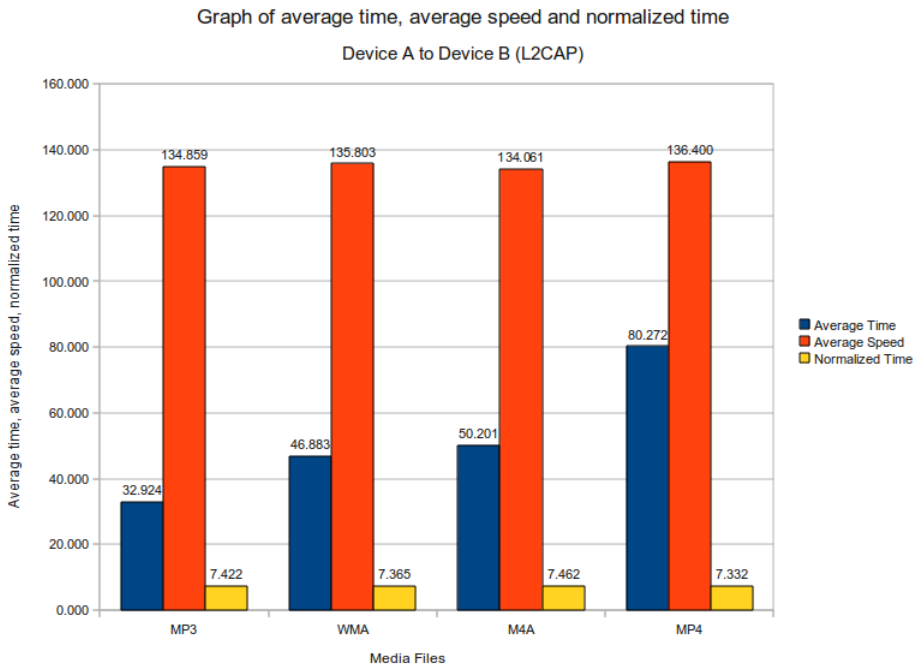


Figure 4.1: Graph of average time, average speed, and normalized time (Device A to B) [L2CAP]

This graph shows that for each of the media files, there was minimal fluctuation for the transfer speed. It is reasonable to expect that the time taken to transfer a larger file will exceed the time taken to transfer a smaller file, but in order to evaluate the consistency of the L2CAP protocol per byte transferred, the normalized times have to be observed which can be seen as the yellow bars in Figure 4.1. Since the normalized time of the MP4 file was 0.09 smaller than the normalized time of the MP3 file, it can be seen that the L2CAP protocol provides an equal level of efficiency for varying file sizes.

4.4.3.2 RFCOMM - Device A to Device B

Table 4.5 shows the average time taken and average speed when transferring four media files from Device A to Device B using the RFCOMM protocol:

Table 4.5: Average time and average speed obtained with the RFCOMM protocol (Device A to B)

File Type	AVG time (Seconds)	AVG speed (KBps)
MP3	42.792	104.059
WMA	63.667	100.015
M4A/AAC	68.671	100.183
MP4	109.629	99.967

The transfer speeds from Table 4.5 show that with a STDV of 2.004 for the transfer speed, the RFCOMM protocol is consistent with regard to transfer speed. The maximum transfer speed of the L2CAP protocol from Table 4.4 is 136.400 KBps, which is 32.341 KBps faster than the maximum transfer speed of the RFCOMM procotol from Table 4.5. As shown in Table 4.1, the minimum bandwidth required to stream a 6.6 MB M4A/AAC is 32.04 KBps, which is less than the difference in transfer speed between the L2CAP and RFCOMM protocols. Since this difference in transfer speed occurs when transferring multimedia files from Device A to Device B, the transfer speed when sending from Device B to Device A also needs to be determined in order to draw conclusions about the protocol with the faster transfer speed, indepedent of directional transfer. Figure 4.2 shows the average time, average speed, and normalized time when sending four files from Device A to Device B using the RFCOMM protocol:

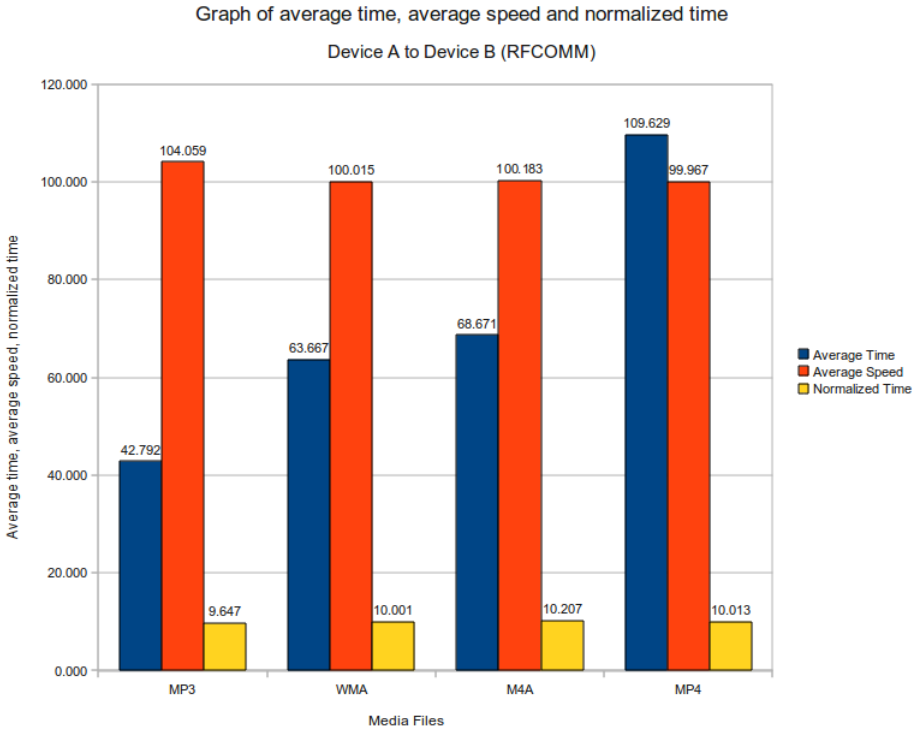


Figure 4.2: Graph of average time, average speed, and normalized time (Device A to B) [RFCOMM]

From the graph in Figure 4.2 the observed transfer speed has minimal fluctuation, but the time fluctuates considerably, with a STDV of 27.970. As can be seen in Figure 4.2, the average time taken (blue bar) to transfer the MP4 file exceeds the average transfer speed (orange bar) achieved while transferring that file. Since the Bluetooth protocol is limited in terms of bandwidth, a point will be reached where the time taken to transfer large files such as an MP4 file will exceed the average transfer speed. The duration of the MP4 file is 35 seconds, which is far less than the time taken to transfer the MP4 file. In such a case the MP4 file would be unstreamable due to the high bitrate used to encode the audio and video. Just as the normalized time is consistent for the graph in Figure 4.1, so too is the normalized time for the RFCOMM protocol, which illustrates that the Bluetooth protocol is consistent per byte transferred, regardless of the file compression scheme.

4.4.3.3 L2CAP - Device B to Device A

Table 4.6 highlights the average time taken and average speed when transferring four files using the L2CAP protocol from Device B to Device A:

Table 4.6: Average time and average speed obtained with the L2CAP protocol (Device B to A)

File Type	AVG time (Seconds)	AVG speed (KBps)
MP3	33.700	131.666
WMA	46.072	138.200
M4A/AAC	48.809	137.877
MP4	79.807	137.204

From Tables 4.4 and 4.6 it can be seen that the transfer speeds obtained when sending from Device B to Device A using the L2CAP protocol are marginally faster than those from Device A to Device B. The highest average transfer speed obtained from Device B to Device A was 138.200 KBps, while the fastest average transfer speed obtained from Device A to Device B was 136.400 KBps.

Figure 4.3 shows the average time, average speed, and normalized time when sending four files from Device B to Device A using the L2CAP protocol:

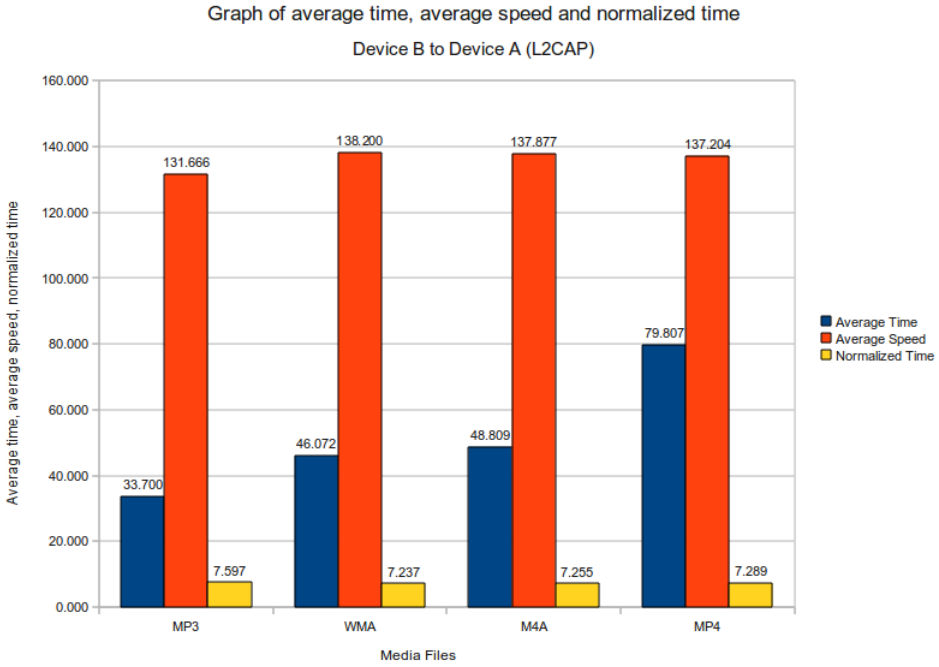


Figure 4.3: Graph of average time, average speed, and normalized time (Device B to A) [L2CAP]

The graph in Figure 4.3 shows an upward progression for average time, which is to be expected from the increasing file sizes. The average normalized time for the L2CAP protocol from Device A to Device B is 7.395, while the average normalized time from Device B to Device

A is 7.345, which confirms the marginally faster transfer speed from Device B to Device A. Even though the transfer speed of L2CAP is 32.341 KBps faster than RFCOMM, the MP4 file still remains unstreamable due to insufficient bandwidth. According to Table 4.1 the minimum bandwidth required to stream the MP4 file is 324.11 KBps, which is considerably more than the benchmark rate of 136.39 KBps, as well as the transfer speed of 137.204 KBps achieved while transferring the MP4 file seen in Figure 4.3. So as not to discount all video streaming, Experiment 2 in Section 4.5 shows the ability of the Bluetooth protocol to stream smaller MP4 files.

4.4.3.4 RFCOMM - Device B to Device A

In order to completely evaluate the differences in transfer speed (if any) between the RFCOMM and L2CAP protocols, results pertaining to the average transfer speed and average time when transferring from Device B to Device A are highlighted in Table 4.7:

Table 4.7: Average time and average speed obtained with the RFCOMM protocol (Device B to A)

File Type	AVG time (Seconds)	AVG speed (KBps)
MP3	44.453	99.904
WMA	62.819	101.364
M4A/AAC	68.676	98.034
MP4	111.046	98.598

The RFCOMM protocol yields a STDV of 1.483 when transferring from Device B to Device A, which once again shows the consistency of RFCOMM and the Bluetooth protocol as whole. The average transfer speed obtained when transferring the MP4 file is 98.598 KBps which is 0.564 KBps faster than the average transfer speed obtained when sending the M4A/AAC file, which shows the capability of the RFCOMM protocol to maintain a consistent transfer speed regardless of file size. The average transfer speed of RFCOMM from Device A to Device B is 101.056 KBps, while the average transfer speed from Device B to Device A is 99.475 KBps. In the same way that the L2CAP protocol produced marginally faster transfer speeds when transferring from Device A to Device B, so to does the RFCOMM protocol (with a difference of 1.581 KBps). Figure 4.4 shows the average time, average speed, and normalized time when sending four files from Device B to Device A using the RFCOMM protocol:

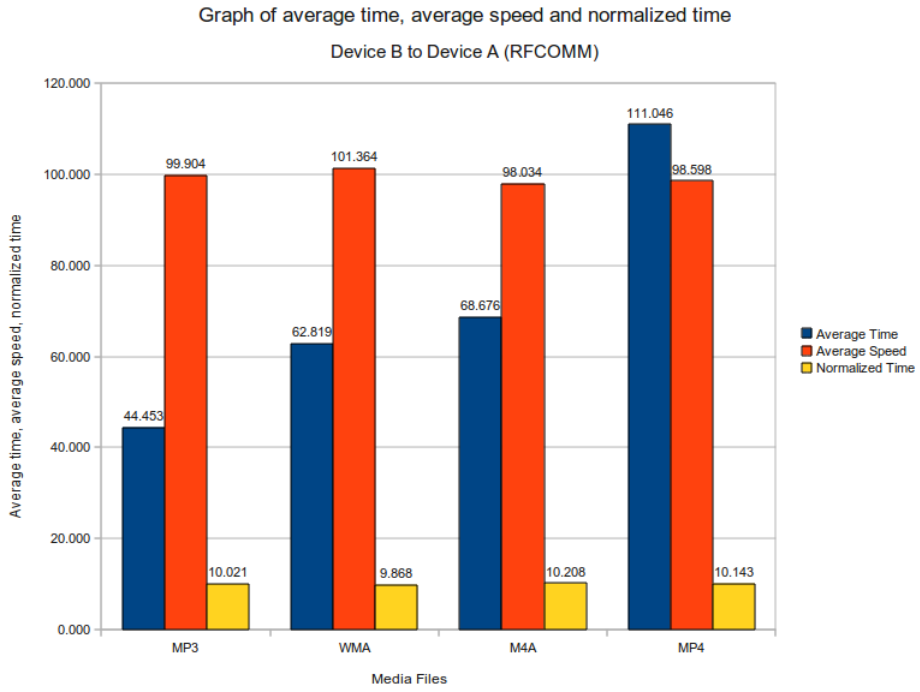


Figure 4.4: Graph of average time, average speed, and normalized time (Device B to A) [RFCOMM]

The graph in Figure 4.4 shows the same upward progression of time in accordance with increasing file size. Transfer speeds have minimal fluctuation and this shows that for the Bluetooth protocol as a whole (L2CAP and RFCOMM) the transfer speed remains constant independent of directional transfer between devices. Figure 4.4 shows the normalized time with a STDV of 0.150, which is close to the STDV of 0.233 for the normalized time from Device A to Device B. This negligible difference between the STDVs is an indication that for the RFCOMM protocol, directional transfer has no effect on the normalized time and thus on transfer speed. Appendix B contains information relating to the average time, average transfer speed, and normalized times for the MP3, WMA, M4A, and MP4 media files.

In order to conclude with a high degree of certainty that transfer speed is independent of directional transfer, a two sample t-test with assumed equal variance was performed for the L2CAP and RFCOMM protocols. Table 4.8 shows the t-test relating to the acceptance or rejection of Hypothesis 1(a):

Table 4.8: T-test of L2CAP protocol - Device A to Device B **vs.** Device B to Device A

L2CAP A to B vs. B to A	
T-Test: Two-Sample Assuming Equal Variances	
df	30
t Stat	0.824167
P(T<=t) one-tail	0.21

From Table 4.8, it can be seen that P > 0.05, therefore we fail to reject null Hypothesis 1(a) and we can conclude that there is no statistically significant difference between the transfer speeds of the L2CAP protocol when sending from Device A to Device B than from Device B to Device A. In order to accept or reject Hypothesis 1(b) with highest degree of certainty, it is necessary to perform the t-test on the normalized times between Device A and Device B as well as vice versa for the RFCOMM protocol. Table 4.9 shows the t-test for the RFCOMM protocol between Devices A and B:

Table 4.9: T-test of RFCOMM protocol - Device A to Device B **vs.** Device B to Device A

RFCOMM A to B vs. B to A	
T-Test: Two-Sample Assuming Equal Variances	
df	30
t Stat	1.169407
P(T<=t) one-tail	0.13

Table 4.9 shows that P > 0.05, therefore we fail to reject null Hypothesis 1(b) and we can conclude that there is no statistically significant difference between the transfer speeds of the RFCOMM protocol when sending from Device A to Device B than from Device B to Device A.

Since there is no statistically significant difference between the transfer speeds of Device A and Device B and vice versa for both protocols, it can be concluded that the transfer speed obtained is independent of the direction of transfer between Devices A and B for both the L2CAP and RFCOMM protocols. Therefore for the remainder of this thesis, experimentation was conducted in either direction between devices.

In order to determine the protocol that provided the highest transfer speed for multimedia

streaming, it was first necessary to elliminate the possibility of directional transfer speed fluctuation, between devices A and B for both the RFCOMM and L2CAP protocols. Table 4.10 shows the t-test for the comparison of the L2CAP and RFCOMM protocols with the results from Device A to Device B and vice versa combined for both protocols:

Table 4.10: T-test of L2CAP **vs.** RFCOMM

L2CAP vs. RFCOMM	
T-Test: Two-Sample Assuming Equal Variances	
df	62
t Stat	-37.3121
P(T<=t) one-tail	1.72×10^{-44}

From Table 4.10 P <= 0.01, which leads to the rejection of null Hypothesis 2 and the conclusion that there is a statistically significant difference between the transfer speeds of the L2CAP and RFCOMM protocols. The t Stat of -37.3121 shows that the direction of the statistical difference is in favour of the L2CAP protocol with significantly higher normalized times for the RFCOMM protocol, meaning that the normalized times, and thus the transfer speed of L2CAP is faster than the RFCOMM protocol.

4.4.4 Summary

This experiment showed that although there is no statistically significant difference within the protocols, L2CAP performed significantly faster than RFCOMM. A possible reason for such significant differences in transfer speeds is due to the L2CAP protocol being lower in the Bluetooth protocol stack than the RFCOMM protocol, and it thus has less protocols/levels to forward the data on to. Even though the L2CAP protocol multiplexes between SDP and RFCOMM protocols as can be seen in Figure 2.1, it is still significantly faster than the RFCOMM protocol despite the multiplexing.

It can be concluded that there is no difference in transfer speed when transmitting in both directions between Devices A and B. It can also be concluded that the L2CAP protocol is significantly faster than the RFCOMM protocol in terms of transfer speed. Now that the fastest protocol has been determined, the remainder of this thesis continues experimentation with the L2CAP protocol alone.

4.5 Experiment 2: Suitability of media types for streaming

Since it has been established that the L2CAP protocol is more suited to multimedia streaming than the RFCOMM protocol, all experimentation is pertinent to the L2CAP protocol. Even though the MMAPI is protocol and format agnostic, device manufacturers only provide support for certain media formats, and of these supported formats, only a small subset are streamable.

4.5.1 Hypothesis

This experiment aims to investigate the suitability of various media types for streaming by determining whether playback is possible before the entire media has been transferred from one device to the other. Media file formats with smaller file sizes are expected to outperform larger file formats encoded with the same file format. Since two methods of media playback control were used throughout this experiment, it was considered necessary to determine which method yielded the faster transfer speed for the population and playback phases of media streaming.

The null Hypothesis for the t-test relating to the population of buffers states that there is no statistically significant difference between the transfer speeds of method 1 and method 2 of the media playback control mechanisms for the population phase of streaming. The alternative Hypothesis for the t-test relating to the population of buffers states that there is a statistically significant difference between the transfer speeds of method 1 and method 2, which leads to the conclusion that either method 1 has a faster transfer speed than method 2 or vice versa.

In order to determine which method yields the faster transfer speed between methods 1 and 2 for the playback phase of streaming, it is necessary to conduct a t-test to establish whether there is a statistically significant difference.

The null Hypothesis for the t-test relating to the transfer speeds of methods 1 and 2 for the playback phase of media streaming states that there is no statistically significant difference between the transfer speeds of methods 1 and 2. The alternative Hypothesis states that there is a statistically significant difference between the transfer speeds of methods 1 and 2 for the playback phase of media streaming.

4.5.2 Methodology

In order to test the suitability of media streaming over the Bluetooth protocol, eight media files were transferred between the sending and receiving devices: WAV (1410.76 KB); MP3 (4436 KB); AAC (6727.51 KB); WMA (6366.06 KB); MIDI (93.06 KB); AMR (290.97 KB); 3GP (1041.48 KB); and MP4 (1257.82 KB). The files were streamed with an MTU of 668 bytes at a distance of 30 cm, and were transferred from Device B to Device A. Each file was transmitted four times to account for possible inconsistencies. All results were rounded to the

third decimal place. The streaming process was initiated by the user from Device B. Device A displayed the progress of the file transfer, and thus to test whether streaming for that particular file was possible, the user initiated playback of the media file once 600KB of the transfer had completed. In the cases of the AMR and MIDI files, playback was initiated as close as possible to 100 KB and 50 KB respectively. If the media file began playback before the entire file had been transferred, then streaming was considered possible. In order to stream the entire contents of the media file, a technique known as double buffering was used (refer to Section 2.2.3) .

4.5.3 Results and analysis

Table 4.11 shows the various media files, their suitability to multimedia streaming, and the effects of the media playback control mechanisms on their transfer speeds:

Table 4.11: Suitability of media types to streaming

Media Type	MIN Bitrate (KBps)	Device Support	Streamable (J2ME)	AVG Speed (method 1 population)	AVG Speed (method 2 population)	AVG Speed (method 1 playback)	AVG Speed (method 2 playback)
WAV	88.173	Yes	Yes	N/A	134.265	N/A	95.254
MP3	15.675	Yes	Yes	141.988	138.212	93.487	95.493
AAC	32.036	No	No	138.627	139.361	95.661	97.134
WMA	22.495	Yes [1]	No	131.258	139.485	95.144	108.729
MIDI	0.308	Yes	Yes	132.545	138.487	95.678	103.778
AMR	1.056	Yes	Yes	134.749	137.228	99.153	100.001
3GP	24.220	Yes	No	134.261	137.454	98.346	101.496
MP4	29.252	Yes	No	133.114	135.263	94.111	94.755

This table has eight columns, with the first column showing the media type to be streamed and the second column showing the minimum bitrate for each media type (as described in Section 4.2). In order for streaming to be possible, the minimum bitrate has to be achieved. The lowest bandwidth needed is for the MIDI file with a rate of 0.308 KBps, and the highest being the WAV file of 88.173 KBps. It should be noted that the transfer speeds achieved when attempting to stream these media files varied from 141.998 KBps when simply receiving the media file and populating the buffers, to 93.487 KBps when actually playing back the partially downloaded media file while using the first method of media playback control mentioned in Section 3.4.8. When using the second method of media playback control, the transfer speeds achieved when receiving the media file and populating the buffers varied from 139.485 KBps to 94.775 KBps when playing back the media file.

The third column shows the device support for each of the media types. Yes [1] signifies support for playback of the WMA file type on Devices A and B, without such support being stated in the specifications. Even though the 3GP and MP4 video formats are supported by

Device A, these files can only be played back once the entire file has been downloaded, and thus it can be said that no progressive download capabilities exist for these common video formats on the J2ME platform. As such, for the remainder of this thesis, experiments were only conducted with audio formats. The fourth column specifies whether streaming was possible with the associated media format.

The fifth, sixth, seventh and eighth columns relate to the effects of media playback control on transfer speed. The fifth and sixth columns refer to the transfer speeds achieved by methods 1 and 2 respectively when the media files were simply being transferred between the devices and the buffers were being populated. Columns seven and eight refer to the transfer speed achieved by methods 1 and 2 respectively when the media files were being transferred and the media being played back during the transfer. Since media streaming involves playback of the media file and not only the population of buffers, the more important transfer speeds observed above are from columns seven and eight (the playback of media). Method 1 of media playback control involves limiting the transfer speed before the receipt of each packet, whereas method 2 involves limiting the transfer speed once 95% of the media file has been transferred (refer to Section 3.4.8 for more details). In order to determine the media playback control method which yields the highest transfer speed for the population and playback phases of streaming, two t-tests with assumed equal variance were conducted.

Table 4.12 shows the two sample t-test with assumed equal variance which determines if there is a statistically significant difference between the transfer speeds of methods 1 and 2 for the population phase of streaming:

Table 4.12: t-test of Method 1 vs. Method 2 - Population

Method 1 vs. Method 2 – Population	
t-Test: Two-Sample Assuming Equal Variances	
df	12
t Stat	-1.7660
P(T<t) one-tail	0.05

From Table 4.12 , it can be seen that P<=0.05, therefore we reject the null Hypothesis and we can conclude that there is a statistically significant difference between the transfer speeds of method 1 and method 2 for the population phase of streaming. The next t-test determines whether there is a statistically significant difference between the transfer speeds of methods 1 and 2 for the playback phase of media streaming.

Table 4.13 shows the two sample t-test with assumed equal variance which determines if there is a statistically significant difference between the transfer speeds of methods 1 and 2 for the playback phase of streaming:

Table 4.13: t-test of Method 1 vs. Method 2 - Playback

Method 1 vs. Method 2 – Playback	
t-Test: Two-Sample Assuming Equal Variances	
df	12
t Stat	-2.0877
P(T<t) one-tail	0.06

From Table 4.13 it can be seen that P > 0.05, therefore we reject the alternative Hypothesis and we can conclude that there is no statistically significant difference between the transfer speeds of methods 1 and 2 for the playback phase of media streaming. Even though there is no statistically significant difference between the transfer speeds of methods 1 and 2 for the playback phase, the t Stat has a value of -2.0877 which shows that method 1 yields a marginally faster transfer speed than method 2 for the playback phase of media streaming. With the method 1 being faster than method 2 for the population phase (statistically significant), and method 1 being marginally faster than method 2 for the playback phase (not statistically significant), it was decided to implement the remainder of the experiments using method 1 for all media file formats except the WAV format, for reasons mentioned in Section 3.4.8.

The WAV, MP3, MIDI and AMR audio files are streamable on the J2ME platform, with the MP3 audio format achieving the best balance between clarity of playback and minimal file size. Even though the WAV format is streamable, the necessary bitrate for successful playback of the WAV file according to Table 4.11 is 88.173 KBps, and the transfer speed achieved during the playback phase of media streaming was 95.254 KBps, which leaves little room for poor Bluetooth link quality. AMR audio streaming is also possible, but due to the intended purpose of these files (playback of voice) [26], the quality of audio playback is poor. The AAC audio format is not supported by Device A and B, and is thus considered unstreamable.

4.5.4 Summary

The J2ME platform provides little support for audio streaming, and even less support is provided for video streaming. From this experiment it can be seen that MP3, WAV, MIDI and AMR audio formats are streamable over the Bluetooth protocol on the J2ME platform. This experiment

also showed that with the currently available hardware, audio formats were more suited to multimedia streaming than video files, with MP3 emerging as the the superior file type in terms of audio clarity and minimal file size. The t-tests conducted in this experiment showed that method 1 of the media playback control mechanisms yielded faster transfer speeds than that of method 2.

4.6 Experiment 3: Effects of distance on transfer speeds of L2CAP

Due to the fact that this research is implemented on mobile devices, the effects of distance on the transfer speed of the L2CAP protocol are important to obtain a complete understanding of streaming in the mobile context.

4.6.1 Hypothesis

This experiment aims to investigate the effects of distance on transfer speed for the L2CAP protocol. Class 2 Bluetooth devices have a maximum transmission distance of 10m, and thus it can reasonably be assumed that the further the distance, the slower the transfer speed, and thus the lower the bitrate of the audio files. The null Hypothesis therefore is that the distance has no effect on the transfer speeds of the L2CAP protocol and the alternative Hypothesis is that distance has an effect on the transfer speeds of the L2CAP protocol.

4.6.2 Methodology

A total distance of 15m was measured, with markings at 1m intervals. This experiment was conducted in an open space so as to avoid adverse effects from reflection and absorption of signal. The mobile devices were placed apart from one another at 1m intervals, and at a height of 1m above the ground to allow the signal to travel unhindered between the devices for each trial. The size of the transmitted file was 6.6 MB and the MTU of 668 bytes was used. Across all iterations, the media file was sent from Device A to Device B. In order to obtain the most accurate results possible and account for inconsistencies such as signal interference, each distance experiment was performed four times to accommodate for any variations in transfer speed. The time taken to transfer the 6.6 MB file from Device A to Device B was recorded and averaged for each 1m distance interval.

4.6.3 Results and analysis

Table 4.14 contains the average results obtained when transferring the 6.6 MB file from Device A to B over the four iterations for each 1m distance interval:

Table 4.14: Average time taken and average transfer speed at distance intervals

Distance (Meters)	Time taken (seconds)	Transfer speed (KBps)
1	33.416	201.326
2	34.815	193.236
3	34.571	194.600
4	36.501	184.310
5	39.370	170.879
6	40.570	165.825
7	95.805	70.221
8	124.150	54.189
9	81.705	82.339
10	102.289	65.770
11	66.113	101.758
12	132.356	50.829
13	117.141	57.431
14	Error -6305	N/A
15	128.620	52.306

In this table, the error refers to link disconnection, which can be interpreted as two devices being too far from one another for the transfer to commence. At distances above 15m the error was increasingly prevalent, and it was deemed unnecessary to include all these results in Table 4.14. Since the maximum theoretical distance of class 2 Bluetooth devices is 10m, any distance exceeding 10m was expected to yield erratic results, which was clearly the case at the 14m and 15m marks. While the link disconnection error occured at the 14m mark, the error was absent at the 15m mark. This result was consistent over the four iterations that this experiment was performed.

An unusual finding was that the average transfer speed at 9m was 82.339 KBps, which was faster than the average transfer speed at 8m (54.190 KBps) and 7m (70.220 KBps). The average transfer speed at 11m (101.760 KBps) was faster than the transfer speeds at 10m, 9m, 8m, and 7m, which is an indication that the L2CAP protocol is inconsistent when transferring files between devices at distances, which was to be expected. From the data in Table 4.14, the maximum transfer speed was achieved at 1m (201.326 KBps), while the lowest transfer

speed was at 12m (50.830 KBps). It is logical to assume that the fastest transfer speed would be achieved at the shortest distance and the slowest transfer speed would be achieved at the furthest distance. This logical assumption holds true for the fastest transfer speed at 1m, while the slowest transfer speed was encountered at 12m, which is not the furthest distance. Figure 4.5 is a visual representation of the timing and distance data obtained above:

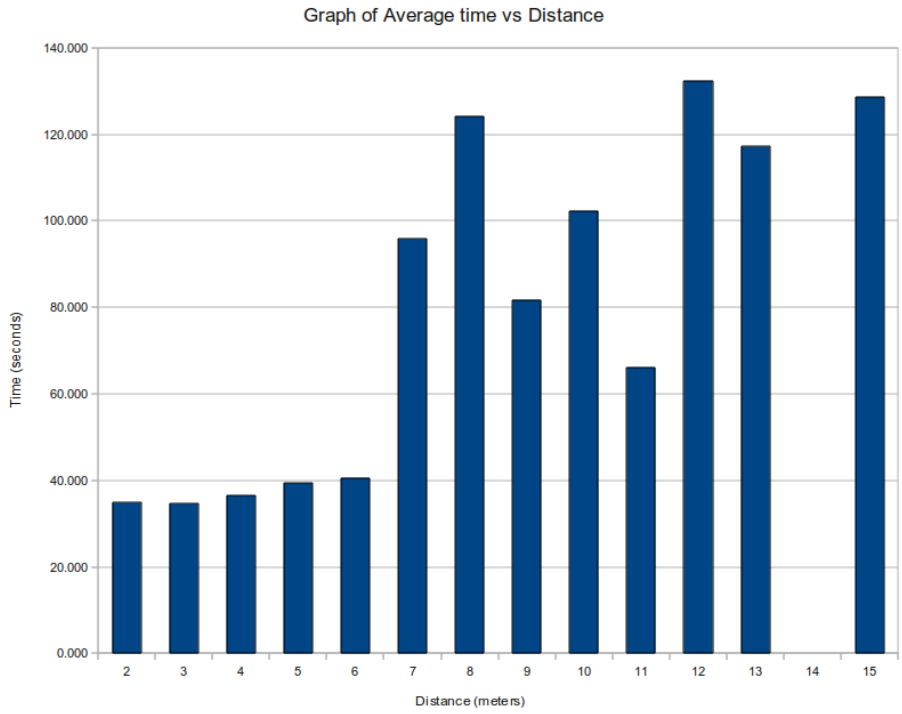


Figure 4.5: Graph of Average time vs Distance

The graph in Figure 4.5 shows that the average time taken to transfer the 6.6 MB file was relatively consistent for distances up to and including 6m. For distances exceeding 6m the time taken fluctuated considerably, with abnormal spikes at 8m, 10m, 12m, and 15m. There is no representation for timing at 14m, since the link disconneciton error was encountered as described above.

One point which should be noted about the transfer times obtained for distances from 7m upwards, is that when the transfer was initiated from distances 7m through to 13m, the link disconnection error was prevalent. When the transfer was initiated from a closer distance (5m) and gradually moved further to distances 7m through to 13m, then no error occurred and the transfer was successful. Refer to Appendix C for the number of times and at which distances the link disconnection error was encountered.

As can be seen from the times obtained in Table 4.14, the transfer times taken for each distance measurement were variable, and it can be concluded that the greater the distance be-

tween the sending and receiving devices, the more variable the time taken to transfer the data between the two devices. The L2CAP protocol was reasonably stable for distances up to and including 6m, but at distances in excess of 6m, the timing results were erratic, varying from 66.113 seconds at 11m to 132.356 seconds at 12m.

It can thus be concluded that Bluetooth is affected by distance. Section 4.2 outlines the theoretical bandwidth needed to stream a 6.6MB M4A/AAC file, which is 32.036 KBps. From Table 4.14 it can be seen that the lowest transfer speed was 50.829 KBps at 12m, which is 18.793 KBps faster than the minimum theoretical transfer speed needed to stream the AAC file. In order to ensure a constant transfer speed, and to be able to stream audio files of higher quality, distances above 6m between the sending and receiving devices should be avoided.

4.6.4 Summary

This experiment showed that Bluetooth was affected by transmission at distance and that erractic results were encountered at distances in excess of 7m. This experiment also showed that for distances from 1m to 6m the Bluetooth protocol yielded consistent transfer speeds, and that any streaming above 6m should be avoided so as to achieve the highest possible transfer speed and therefore be able to stream audio files of higher bitrates.

4.7 Experiment 4: Variation of transmission unit size

Since bandwidth, memory, and processing power of the Bluetooth protocol are limited, it is imperative to achieve the highest possible transfer rate with minimal resources. By lowering the transmission unit size, the processing requirements of both the transmitting and receiving devices are lowered. There is a tradeoff between the transmission unit size and the transfer speed achieved with the L2CAP protocol.

4.7.1 Hypothesis

This experiment aims to investigate the effects of transmission unit size variation on transfer speed and time. The balance of fastest transfer rate with minimal resources could possibly be achieved through alteration of the transmission unit size. Transmission unit size variation is expected to have drastic effects on the transfer speed and hence the time taken for the transfer.

Algorithm 4.2 Calculating the expected time

x = time taken to transfer file using the maximum MTU
n = percentage of the maximum MTU
*Expected time = (x * (100% - n%)) + x*

4.7.2 Methodology

This experiment was conducted by sending a 6.6 MB M4A/AAC file from Device B to Device A at a distance of 30 cm. The MTUs were decreased from the maximum MTU size to a minimum of 134 bytes in approximate 10% decrements of the MTU (668 bytes). In order to eliminate inconsistencies with regard to the fluctuating transmission strength and possible interference, the audio file was transmitted four times for each transmission unit size decrement. The average time was compared with the expected average time in order to reveal the advantages and disadvantages of the Bluetooth protocol. The expected time was calculated by making use of the following formula:

4.7.3 Results and analysis

Table 4.15 presents the average time; average expected time; average expected time - average time; average speed; and the average expected speed for each 10% decrement of the maximum MTU:

Table 4.15: Transfer times and average speed with MTU variation

MTU Size (Bytes)	% of maximum MTU	AVG time (sec)	AVG Expected time (sec)	AVG expected time – AVG time	AVG speed (KBps)	AVG expected speed (KBps)
668	100	49.143	-	-	136.913	-
601	90	54.422	54.058	-0.364	123.620	124.466
534	80	52.005	58.972	6.967	129.366	114.094
468	70	50.004	63.886	13.882	134.560	105.317
401	60	58.660	68.800	10.140	114.696	97.795
334	50	66.646	73.715	7.069	100.948	91.275
267	40	79.772	78.629	-1.143	84.341	85.570
200	30	105.237	83.541	-21.696	63.928	80.539
134	20	154.327	88.458	-65.870	43.593	76.062
67	10	303.591	93.372	-210.219	22.160	72.059

Table 4.15 consists of seven columns, which show the average and expected times and speeds as well as the difference between the average expected time and the average time. The average times were obtained by timing the transfer of the 6.6 MB audio file from Device B to Device A. The expected average time was calculated using the fomula in Algorithm 4.2, and is thus dependent on the actual times obtained during the transfer. The average expected times for each of the transmission unit sizes are dependent on the average time obtained with the transmission unit size of 668 bytes as the baseline. The column containing the difference between the average expected times and the average times is the basis for the row shading observed in Table 4.15. Data highlighted in green represents a faster transfer speed than was expected, thus resulting in faster average times than expected average times. Differences greater than or equal to -2 seconds and less than or equal to 0 seconds are considered negligible and are highlighted in orange. When the average time was slower than the expected average time, the data was highlighted in red. The relationship between the expected time taken and the actual time taken per transmission unit is highlighted in Figure 4.6:

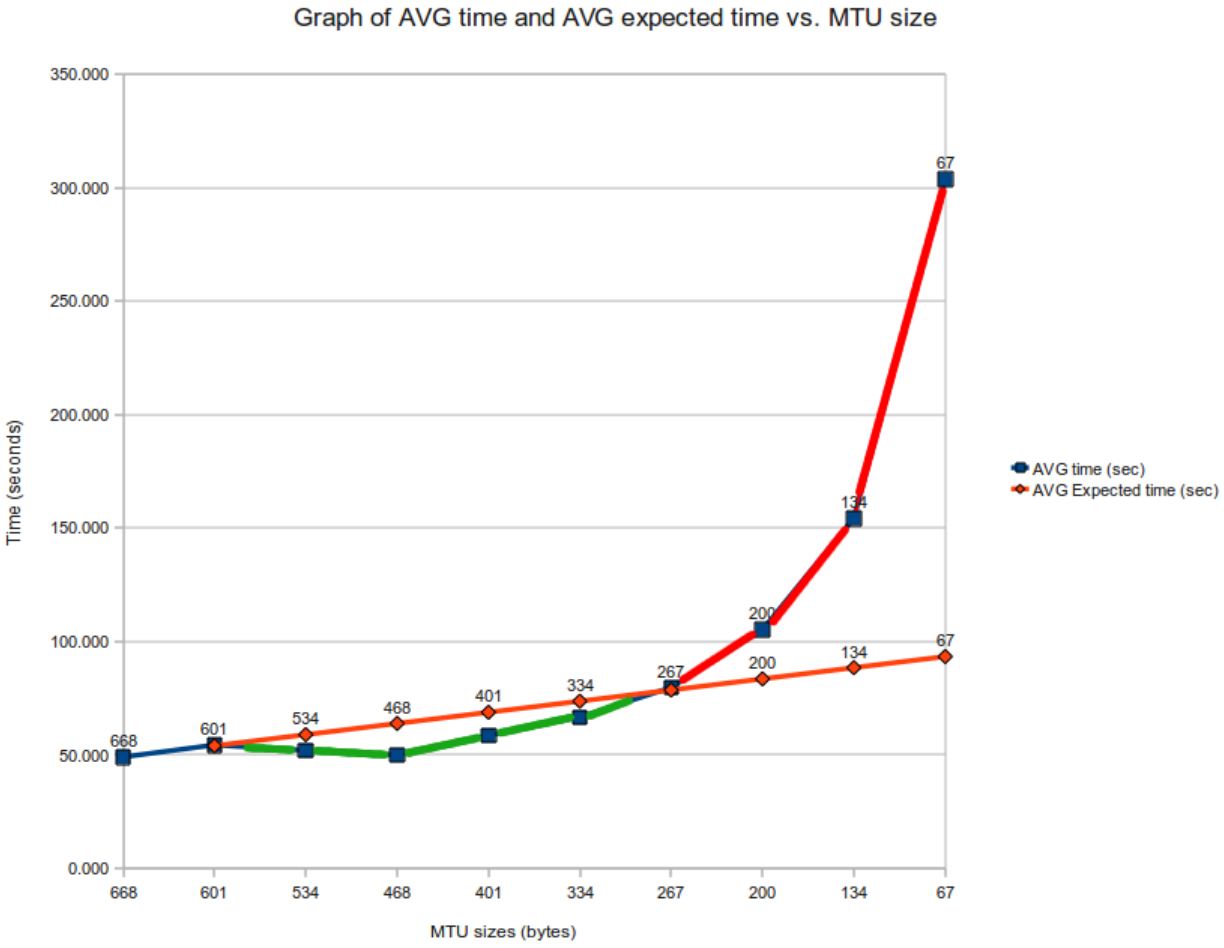


Figure 4.6: Average time vs Transmission unit size

Figure 4.6 plots the average time and the average expected time on a single graph, and each point a decrease in size of the transmission unit by a factor 67 bytes (10% of the MTU). The most obvious points on the graph are the intersection points of the expected time and the actual time taken. The two lines intersect at transmission unit sizes 601 and 267. It can be seen that from transmission unit size 534 to 324, the actual time taken for the transfer is faster than the expected time. The data highlighted in green from Table 4.15 represents the portion of data where the actual time taken is faster than the expected time taken. The green portion of the data from Table 4.15 is seen as the line arching under the orange line in Figure 4.6. The data highlighted in orange from Table 4.15 represents the portion of data where the actual time and the expected time coincided, and this can be seen as the intersection of the two lines at points 601 and 267 in Figure 4.6. The data highlighted in red from Table 4.15 is where the expected time was in fact faster than the actual time, and this can be seen in Figure 4.6 where the blue line follows an exponential pattern above the orange line from transmission unit size 267 through to 67.

Even though an analysis of the results of the transmission unit size variation can be conducted based on Figure 4.6 alone, Figure 4.7 aids in understanding transmission unit size variation. Figure 4.7 shows the relationship between the average speed and the average expected speed for each transmission unit size variation:

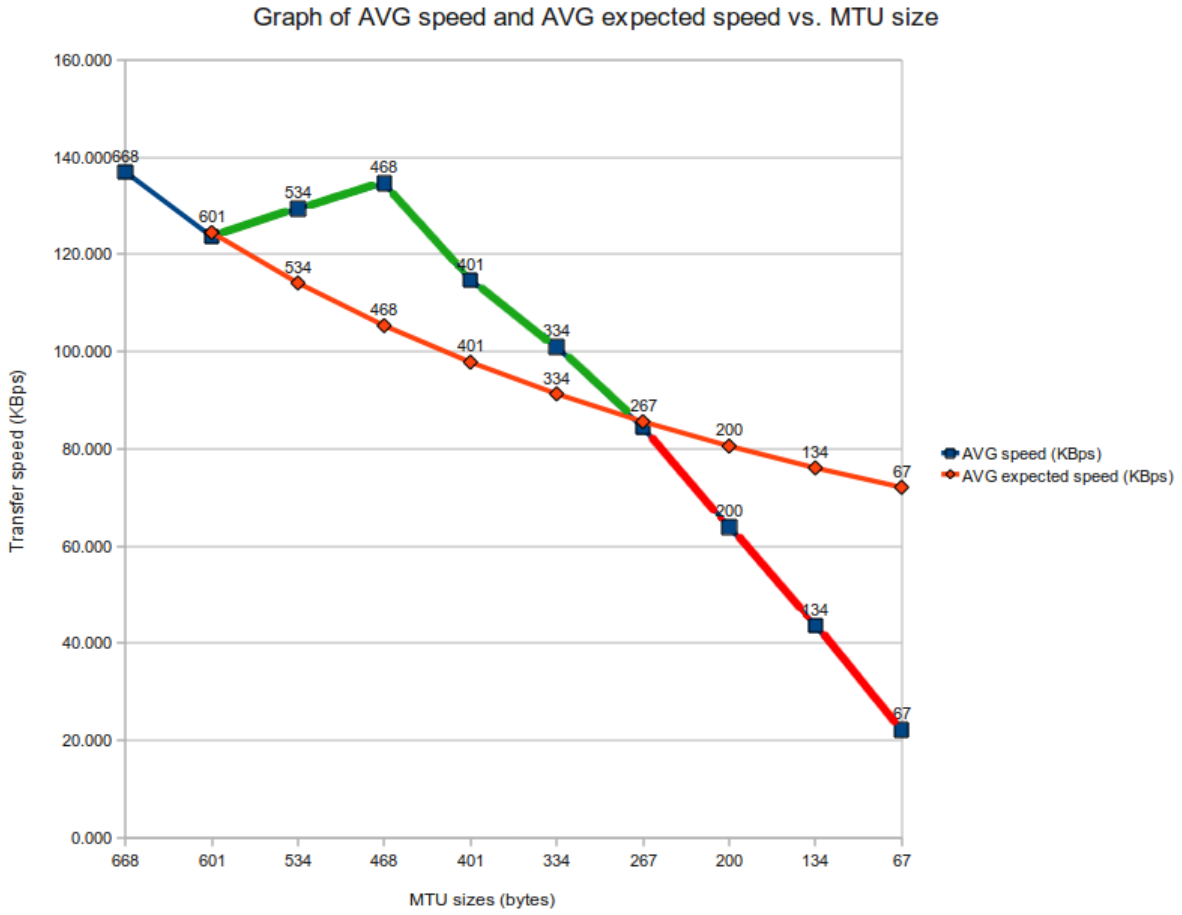


Figure 4.7: Average speed vs Transmission unit size

The time taken and thus the transfer speed is stable for transmission units equating to more than 40% of the MTU (267 bytes), thereafter there is a drastic increase in time and a rapid decrease in transfer speed. For the purposes of audio streaming, it is unlikely that packet sizes would be less than 267 bytes, except in cases where devices are constrained in terms of memory and buffer sizes. For packet sizes from 267 bytes through to 601 bytes, the L2CAP protocol fares well by performing faster than expected. With smaller transmission units, a given file would use more packets and therefore it would take longer to transfer that file. This however is not the case with transmission units 534 bytes and 468 bytes. Transmission unit 534 has a faster transfer speed than transmission unit 601 and transmission unit 468 has a faster transfer speed than both transmission units of sizes 534 and 601 bytes. It took 50.004 seconds to transfer the 6.6 MB file with a transmission unit of size 468 bytes, which is impressive considering that this transmission unit is 200 bytes smaller than the MTU, and the time taken to transfer the file is only 0.861 seconds slower than if the file were transmitted using the MTU of 668 bytes. From the graph in Figure 4.7 it can be seen that for packet sizes smaller than 267 bytes, the transfer speed decreases at what seems to be an exponential rate. This indicates that the L2CAP protocol

was intended for larger packet sizes, and the transfer speeds obtained with lower packet sizes are insufficient for audio streaming reminiscent of local audio playback.

It can thus be concluded that the variation of the transmission unit size has drastic effects on the transfer speed and thus the time taken for the transfer. It can also be concluded that larger transmission unit sizes are more suited to achieving higher transfer rates.

4.7.4 Summary

This experiment showed that transmission unit size variation effects the transfer speed, and that larger transmission unit sizes are suited to files of higher bitrates and should be used if higher transfer speeds are desired. This experiment also showed that larger transmission unit sizes don't necessarily achieve the highest transfer speeds, even though this is often the case. This experiment showed that for the majority of the transmission unit sizes the L2CAP protocol performed better than expected.

4.8 Experiment 5: Optimum transmission unit size for audio streaming

From the previous experiment detailed in Section 4.7, it can be seen that the transmission unit size greatly affects the transfer speed, with lower transfer speeds placing less demand on the processing requirements of the device, thus enabling the device to better deal with rendering the audio. As highlighted in Section 4.2, available bandwidth and processing power both affect the clarity of audio playback.

4.8.1 Hypothesis

This experiment aims to investigate the optimum transmission unit size for multimedia streaming. The purpose of determining the optimum transmission unit size is to enable audio streaming without any jitter (or as little as possible). The quality of the audio or video stream should be comparable to local playback of the media file.

4.8.2 Methodology

The evaluation of the quality of media playback will ultimately determine the optimum transmission unit size. The quality evaluation was conducted four times for each transmission unit size, for each of the three files. Three users evaluated the quality of the media streaming, by rating the playback quality on a scale of 1 (worst) to 10 (best). Each file type pertaining to the transmission size was played back in a random order and upon playback completion, the users then ranked the playback quality. The users quality evaluations were conducted seperately in order to elliminate the rankings of users influencing each other.

To reduce the number of dependent variables in this experiment and to achieve the highest possible transfer speed, the distance was kept constant at 1m. Transmission unit sizes of 668, 601, 534, 468, 401, 334, and 267 bytes were used. In order to cater for the possibility of various media types favouring different transmission unit sizes, it was decided to conduct this experiment by sending the following media types: MP3; WAV; and AMR. These files were sent from the Device B to Device A. The MIDI file format was excluded from this experiment, based on the assumption that due to such a small file size, any transmission unit size would suffice in streaming this file type. For reasons mentioned in Section 3.4.8 the MP3 and AMR files were streamed with method 1 of the media playback control mechanisms, and the WAV file was streamed using method 2.

4.8.3 Results and analysis

Table 4.16 shows the average rating of all the users for the user study per transmission unit size:

Table 4.16: Optimum transmission unit size for audio streaming

File Type	Transmission unit size	Average Rating
MP3 (bitrate 15.675 KBps)	668	5.333
	601	5
	534	5.666
	468	5.666
	401	7.666
	334	7
	267	5.666
WAV (bitrate 90.29 KBps)	668	6.666
	601	6
	534	6.333
	468	5.666
	401	5.666
	334	4.666
	267	3.333
AMR (bitrate 1.05 KBps)	668	5.666
	601	5.666
	534	5.666
	468	5.333
	401	6.333
	334	5.666
	267	5.666

From Table 4.16 it can be seen that for the MP3 and AMR files higher user ratings were encountered for smaller transmission unit sizes, whereas the opposite is true for the WAV file. A possible reason why users preferred playback of the WAV file with higher transmission unit sizes is because playback commenced without jitter due to the higher transfer speed. At lower transmission unit sizes, the transfer speed for the WAV file was lower than the required transfer speed of 88.173 KBps. The higher the bitrate of the file, the faster the required transfer speed. For larger media files with higher bitrate encodings, larger transmission unit sizes are favourable as illustrated by the rankings for the WAV file. For the MP3 and AMR formats, the superior quality according to the users is achieved with a transmission unit size of 401 bytes. Smaller transmission unit sizes improve the efficiency of media playback control mechanisms, since they inherently provide decreased rates of incoming data, hence automatically catering for

limited resources of the mobile devices for media receipt and playback. The individual users ratings of media playback quality can be seen in Appendix D.

4.8.4 Summary

This experiment showed that files with lower bitrate encodings and hence smaller file sizes are better suited to streaming with lower transmission unit sizes, with the opposite applying to files with higher bitrate encodings. This experiment showed that the MP3 and AMR audio formats are more suited to streaming than the WAV file, since they were streamable using larger variations in packet size. This experiment also showed that users preferred the audio quality when smaller transmission unit sizes were used for the MP3 and AMR formats, while for the WAV audio format they preferred the quality of playback when larger transmission unit sizes were used.

4.9 Experiment 6: Effects of dropped packets on audio playback

The quality of an audio or video stream is ultimately assessed by the clarity of the audio and video playback. The clarity of the playback is influenced by a number of factors: bitrate (quality) of the audio or video file; available bandwidth; the processing power of both the transmitting and receiving devices; the capability of the media player itself; and the number of lost packets.

The first step to ensuring quality audio playback is the choice of audio files with a good balance between audio quality (bitrate) and file size. From Section 4.8 it can be seen that for the transmission of larger files, a larger transmission unit is favourable (e.g. WAV), while for smaller files, smaller to medium transmission unit sizes are favourable (e.g. MP3 and AMR). Larger files also require more resources (memory and buffer sizes) than smaller files to commence with playback. Quality of the audio playback is largely dependent on the type of media file which is being streamed, since certain media types are better suited to streaming than others (MP3 is better suited to streaming than WAV). The suitability of certain media types for streaming was highlighted in Section 4.5. Since dropped packets are either ignored or dealt with by some retransmission scheme, it is necessary to understand the retransmission scheme of the L2CAP protocol which was highlighted in Section 2.1.4.

In order to understand why the quality of audio varies, the concept of jitter should first be understood. Jitter can be defined as the variation in time between packets arriving, caused by network congestion (referred to as available bandwidth), timing drift, or route changes [49].

Since Bluetooth is a wireless transportation medium, it is susceptible to interference, and

is constrained in terms of distance. Interference can significantly affect the quality of the audio stream resulting in lost packets which in turn leads to jitter. If the source and the destination move out of range of one another, packets are lost, resulting in interrupted playback. Ideally, playback of an interrupted stream should be resumable, with minimal loss to the quality or continuity of the audio or video stream.

4.9.1 Hypothesis

This experiment aims to investigate the effects of dropped packets on multimedia playback by transmitting a media file at varying distances and assessing the playback quality. It is expected that the greater the distance between Device A and Device B, the lower the transfer speed and hence the lower the playback quality of the media file.

4.9.2 Methodology

A 6.6 MB M4A was sent from Device B to Device A at varying distances. Since Section 4.6 showed that the MTU of 668 bytes yielded the highest transfer speed, the MTU of 668 bytes was used. In order to simulate the effects of dropped packets the two devices were initially placed at a distance of 4m apart, and the distance was gradually increased to 15m at which point the two devices are out of range of one another and retransmission commenced. The devices were kept at a distance of 15m for 6 seconds or less, and then the distances were decreased to the starting distance of 4m. When more than half of the MP3 file had been transferred the media playback sequence was initiated on Device A, and the user then began to increase the distance between Device A and Device B. This experiment was performed four times to eliminate any inconsistencies. The quality of playback was assessed at each 1m interval from 4m through to 15m and Packets were considered dropped when increased amounts of jitter occurred. The transfer speeds at each 1m interval were recorded and averaged over the four iterations of this experiment. One user evaluated the quality of playback for each of the iterations, which determined whether playback quality was: good; average; or poor. Since packets are dropped at varying distances throughout multiple iterations, it was considered sufficient to grade the playback quality based on the preferences of only one user.

Distance (meters)	Transfer speed increasing (KBps)	Transfer speed decreasing (KBps)	Playback quality
4	92.421	90.553	Good
5	91.444	89.682	Good
6	91.623	89.394	Good
7	72.147	71.798	Good
8	69.467	66.137	Good
9	71.871	65.741	Average
10	65.222	63.227	Average
11	52.465	48.765	Average
12	48.993	43.119	Average
13	41.717	40.899	Poor
14	32.189	30.285	Poor
15	28.772	26.388	Poor

Table 4.17: Transfer speeds and playback quality ratings for increasing and decreasing distances

4.9.3 Results and analysis

Table 4.17 shows the transfer speeds obtained between distances 4m and 15m (inclusive) and shows the playback quality for each distance interval:

This table consists of four columns, with the first column showing the distance at which the transfer speeds were recorded and the quality assessments conducted. The second column shows the transfer speed obtained when the distance was increased from 4m to 15m, and the third column shows the transfer speed obtain when the distance was gradually decreased from 15m to 4m. The fourth column describes the quality of audio playback at each of the associated distances.

As the distance between the two devices approached 15m, the transfer speed decreased from 92.421 KBps at 4m to 28.772 KBps at 15m and Device A was less responsive to user actions than normal. The playback quality was good between distances of 4m and 8m, but as the distance between the two devices approached 15m, an increased amount of jitter resulted in audio playback. Playback quality was average between 9m and 11m and poor between distances of 13m and 15m. Once the devices were at a distance of 15m, they remained at that distance for 3 seconds, after which the distance between the devices was gradually decreased to 4m. During this gradual decrease in distance, playback continued, and the transfer speed increased and varied between 26.338 KBps at 15m to 90.553 KBps at 4m. When the devices were kept at a distance of 15m apart for more than 5 seconds, exceptions were thrown on Device B indicating no power for the device (Symbian error code -18), and link disconnection (Symbian error code

-6305).

From Section 4.6 it can be seen that as distance increases, the transfer speed decreases, and from Section 4.5.2 it is observed that the transfer speed decreased to 60 KBps when playback of the media file commenced during transmission. From these decreased speeds and Device A being less responsive to user actions, it can be concluded that the device's processing functions are under pressure due to the simultaneous tasks of user input; media receipt and playback; and load requirements as a result of increased distance between the two devices.

When the devices were between 4m and 8m apart, the effects of distance on the processing requirements of the device were not significant enough to cause sub-standard media playback quality. During the five seconds the devices were at 15m apart, the L2CAP protocol was initiating retransmission procedures and ensuring successful delivery of the media.

It can be concluded that dropped packets have little to no effect on the L2CAP protocol, since retransmission ensures that any packets which may go missing are retransmitted, and if the maximum number of retransmissions is exceeded, or the timeout value with no communication during an active session is reached, the link is considered disconnected (Symbian error code -6305) or the devices are considered to have no power (Symbian error code -18). Once the devices are out of reach of one another for more than five seconds, either the timeout value is reached, or the maximum number of retransmissions is reached. Media transmission and playback is resumable if the devices are out of reach of one another for less than or equal to five seconds.

4.9.4 Summary

This experiment showed that dropped packets have little effect on the L2CAP protocol and thus the quality of audio playback. This experiment also showed that if Devices A and B were kept at a distance of 15m apart for more than 5 seconds, power and link disconnection errors were prevalent, and media transmission and playback is thus resumable if the devices are out of range of one another for leass than or equal to 5 seconds.

4.10 Discussion

All of the above experiments and their results contribute to determining the viability of streaming on the J2ME platform. The experiment comparing the RFCOMM and L2CAP protocols was necessary in order to determine factors such as ease of implementation, availability of bandwidth, and suitability for media streaming. The results of this experiment showed that the

L2CAP protocol yields faster transfer speeds than the RFCOMM protocol, thus making it the preferred protocol for audio streaming on the J2ME platform.

Since different media types are suited to different purposes, it was necessary to determine the most suitable media type for streaming on the J2ME platform. This experiment dertermined the suitability of the following media types for streaming: WAV; MP3; M4A/AAC; WMA; MIDI; AMR; 3GP; and MP4. This experiment showed that the MP3, AMR, MIDI, and WAV audio formats are streamable, with the MP3 audio format emerging as the superior file format in terms of audio clarity and minimal file size. This experiment also showed that the MP4 and 3GP video file formats were unstreamable, and they could only be played back once the entire media file had been transferred from one device to the other. Due to the nature of mobile devices, streaming is often conducted at distances, which affect the transfer speed and thus the quality of the media playback throughout streaming. Experiments throughout this chapter showed that streaming is affected by distance with increasing levels of influence for distances of 7m and above for the L2CAP protocol. This same experiment (Experiment 3) showed that in order to maintain a consistent transfer speed, audio streaming beyond 6m should be avoided.

In the same way as distance had an effect on audio streaming, so too does the manipulation of the transmission unit size, which was covered in Experiment 4. This experiment showed that transmission unit size variation has drastic effects on the transfer speed of the L2CAP protocol, and that overall, the transmission unit size variation yielded better transfer speeds and times than expected. Since media files have varying internal structures, some will be suited to different transmission unit sizes than others, and Experiment 5 that smaller transmission unit sizes were favourable for the MP3 and AMR audio file formats, while larger transmission unit sizes were favourable for the WAV audio file type.

Since media files are transmitted at varying distances and varying transmission unit sizes, the likelihood of dropped packets is increased, which lead to an investigation into the effects of dropped packets on media streaming for a low powered wireless protocol such as Bluetooth. This experiment (Experiment 6) showed that dropped packets don't have much effect on the playback of the media stream, except in the case where transmission was conducted for a period longer than 5 seconds at the 15m mark.

4.10.1 Summary

As can be seen, many of the experiments deal with obtaining optimum values, and this is the case due to the limited bandwidth of the Bluetooth protocol and the constrained devices between which data is streamed. The next chapter concludes this thesis and outlines possible future work.

Chapter 5

Conclusion and Future Work

This thesis aimed to determine the feasibility of audio and video streaming with the Bluetooth protocol on the J2ME platform. In the introduction, this problem was divided into four sub-problems. This chapter concludes this dissertation by showing how each of the problems were solved and then offers suggestions for future work.

This thesis determined the current state of Bluetooth audio and video streaming on mobile phones, by showing that solutions involving a technique known as double buffering were prevalent among other researchers in the field. From related work, a mobile testbed was developed for the J2ME platform, which enabled testing of both the L2CAP and RFCOMM protocols. The efficiency of audio streaming was evaluated through a series of experiments conducted on the mobile testbed (Chapter 4), relating to the effects of distance on transfer speed; the effects of transmission unit size variation; and the effects of dropped packets on streaming. Media playback control mechanisms were implemented to increase the efficiency of audio streaming, and minimize jitter which is caused as a result of the limited processing capabilities of the mobile devices. These experiments found that the L2CAP protocol yields faster transfer speeds than the RFCOMM protocol, and that transfer speed is unaffected by the bi-directional transfer of media files. These experiments also showed that the MP3, WAV, MIDI, and AMR media files were streamable, with the MP3 file format emerging as the as the superior file format in terms of audio clarity and minimal file size. These experiments showed that transfer speed was adversly affected by distances greater than 6m and that higher transmission unit sizes were suited to media files of higher bitrates (which was not always the case). The MP3 and AMR file formats proved to be more suited to streaming than the WAV file format due to these formats being streamable with a larger range of transmission unit sizes. User ratings showed that the MP3 file format produced the highest quality of audio playback when compared to the AMR and WAV file formats. The final experiment showed that dropped packets have little effect on the L2CAP protocol and link disconnection errors were encountered when devices were out of range of one

another (greater than 15m) for more than 5 seconds.

It can be concluded that the J2ME platform is feasible for audio streaming over the Bluetooth protocol between two mobile devices. Video streaming is however, infeasible due to insuffient bandwidth provided by the Bluetooth protocol. Although the quality of audio playback from media streams does not always match the quality of local playback of media files, this research provided a valuable insight into the advantages and disadvantages of using this platform in conjunction with the Bluetooth protocol for audio and video streaming. Factors such as the optimum transmission unit size for the L2CAP protocol and the effects of distance on transfer speeds enable future undertakers of streaming with the Bluetooth protocol to progress more rapidly towards an optimized solution for the streaming of audio and video in such a constrained environment. It can be concluded that the L2CAP protocol is superior to the RFCOMM protocol in terms of transfer speed, and the L2CAP protocol is thus the recommended protocol for media streaming over the Bluetooth protocol.

This research can be extended in a number of ways:

- The streaming of live video feeds over Bluetooth - By streaming live video, this research could contribute towards the implementation of a surveillance system. Such a system could have multiple video feeds coming from various sources (cell phones, web cams connected up to computers, and Bluetooth enabled cameras) being displayed on a central device. Motion detection could be an addition to such a system, thus analyzing the video capabilities of the MMAPI. This research would involve maximizing the available bandwidth due to multiple incoming video feeds.
- The streaming of a live analog radio feed over Bluetooth - Since bandwidth in South Africa is still expensive, and the streaming of radio stations requires substantial bandwidth, analog radio could be streamed from a cell phone to the computer at which point it would be digitized and possibly re-streamed (across LAN) to multiple networked computers.
- The altering of video streams to enable live streaming of video files instead of having to wait for the entire media file to be downloaded before the commencement of playback.
- The influence of the RTSP on the viability and efficiency of streaming over J2ME.
- A comparison between the streaming capabilities of the J2ME platform and the Symbian OS.

References

[1] APPLE. Bluetooth device access guide. Available at: http://developer.apple.com/mac/library/documentation/DeviceDrivers/Conceptual/Bluetooth/Bluetooth.pdf, 2007. [Accessed 05-07-2009].

[2] ARSOSA, M. What is 3gp? Available at: http://ezinearticles.com/?What-is-3GP?&id=1552818, 2009. [Accessed 07-11-2009].

[3] AU-SYSTEM. Bluetooth white paper. Available at: http://www.ausystem.com, 2000. [Accessed 13-05-2009].

[4] BIALOGLOWY, M. Bluetooth security review. Available at: http://www.securityfocus.com/infocus/1836, 2005. [Acessed 05-02-2009].

[5] BILAN, A. Streaming audio over Bluetooth ACL links. In *Information Technology: Coding and Computing [Computers and Communications], 2003. Proceedings. ITCC 2003. International Conference on* (2003), pp. 287–291.

[6] BLUETOOTH. Fast facts - bluetooth technology 101. Available at: http://www.bluetooth.com/Bluetooth/FastFacts.htm, 2009. [Accessed 08-06-2009].

[7] BLUETOOTH. Bluetooth radio. Available at: http://www.bluetooth.com/Bluetooth/Technology/Works/ArchitectureRadio.htm, 2009. [Accessed 19-11-2009].

[8] BLUETOOTH, S. Bluetooth specification version 1.1. *Available HTTP: http://www. bluetooth. com* (2003).

[9] BLUETOOTH DOC, . ADVANCED AUDIO DISTRIBUTION PROFILE SPECIFICATION, 2007.

[10] BLUETOOTH SPECIFICATION, . v2. 0+ EDR, 2004.

[11] CANALYS. Canalys quarterly research highlights. Available at: http://www.canalys.com/pr/2010/r2010021.html, February 2010. [Accessed 29-04-2010].

[12] CONFIGURATION, C. and the Foundation Profile. *Sun Microsystems* (2001).

[13] COSTELLO, S. What is streaming? Available at: http://ipod.about.com/od/glossary/g/streamingdef.htm, 2010. [Accessed 28-04-2010].

[14] COULTON, P., AND EDWARDS, R. *S60 programming: a tutorial guide*. Wiley, 2007.

[15] COURTNEY, J. JSR129: Personal Basis Profile Specification. Available at: http://jcp.org/en/jsr/detail?id=129, December 2005. [Accessed 29-04-2010].

[16] COURTNEY, J. JSR135: Mobile Media API. Available at: http://jcp.org/en/jsr/detail?id=135, 2006. [Accessed 29-04-2010].

[17] COURTNEY, J. JSR62: Personal Profile Specification. Available at: http://jcp.org/en/jsr/detail?id=62, March 2006. [Accessed 29-04-2010].

[18] CURTIS, S. Global telecoms insights 2010 focus report. Available at: http://www.tnsglobal.com, 2010. [Accessed 29-04-2010].

[19] DE JODE, M., ALLIN, J., HOLLAND, D., NEWMAN, A., TURFUS, C., LITOVSKI, I., HAYUN, R., SEWELL, G., LEWIS, S., AUBERT, M., ET AL. Programming Java 2 Micro Edition on Symbian OS. *John Wiley & Sons 7* (2004), 264–269.

[20] GADD, S., AND LENART, T. A hardware accelerated mp3 decoder with bluetooth streaming capabilities. *Master of science Thesis. Lund University.* (2001).

[21] GOYAL, V. *Pro Java ME MMAPI: Mobile Media API for Java Micro Edition*. Apress, 2006.

[22] GOYAL, V. Experiments in streaming content in java me. Available at: http://today.java.net/article/2006/08/18/experiments-streaming-content-java-me, 2006. [Accessed 28-04-2010].

[23] GRATTON, D. *Bluetooth profiles: the definitive guide*. Prentice Hall PTR, 2003.

[24] HAARTSEN, J., NAGHSHINEH, M., INOUYE, J., JOERESSEN, O., AND ALLEN, W. Bluetooth: Vision, goals, and architecture. *ACM SIGMOBILE Mobile Computing and Communications Review 2*, 4 (1998), 45.

[25] HO, T. Sound implementation on v300/v500/v600, 2009.

[26] IETF, R. 3267: RTP payload format and file storage format for the Adaptive Multi-Rate (AMR) Adaptive Multi-Rate Wideband (AMR-WB) audio codecs, 2002.

[27] INC, Q. Binary Runtime Environment for Wireless (BREW). *URL< http://brew. qualcomm. com*.

[28] INC, S. RMI optional package specification version 1.0, 2003.

[29] JARVINEN, B. JSR 75: PDA Profile for the J2ME Platform. Available at: http://jcp.org/en/jsr/detail?id=75, 2004. [Accessed 13-02-2010].

[30] KAPOOR, R., CHEN, L., LEE, Y., AND GERLA, M. Bluetooth: carrying voice over ACL links. In *Proceedings of Mobile and Wireless Communications Networks* (2002), pp. 379–383.

[31] KEWNEY, G. High speed bluetooth comes a step closer: enhanced data rate approved. Available at: http://www.newswireless.net/index.cfm/article/629.

[32] KOENEN, R. Overview of the mpeg-4 standard. Available at: http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm, 2002. [Accessed 23-09-2009].

[33] KUMAR, B. Jsr-82: Java APIs for bluetooth. Available at: http://jcp.org/en/jsr/detail?id=82, 2004. [Accessed 28-04-2010].

[34] LONGORIA, R. *Designing software for the mobile context: a practitioner's guide*. Springer-Verlag New York Inc, 2004.

[35] MACKER, J., AND CORSON, S. Mobile ad hoc networks (manet):routing protocol performance issues and evaluation considerations. Available at: http://tools.ietf.org/rfc/rfc2501.txt, January 1999. [Accessed 06-09-2010].

[36] MICROSOFT. Constant, variable, and multiple bit rate encoding. Available at: http://msdn.microsoft.com/en-us/library/cc294530.aspx, 2009. [Accessed 15-11-2009].

[37] MONTOYA, T. Performance analysis of jxta/jxme applications in hybrid fixed/mobile environments. Available at: http://caribdis.unab.edu.co/pls/portal/url/ITEM/3F735BFA822E629FE0440003BA3D5405, 2006. [Accessed 25-07-2010].

[38] MORROW, R. *Bluetooth operation and use*. McGraw-Hill Professional, 2002.

[39] NOKIA. Bluetooth technology overview. Available at: www.forum.nokia.com, April 2003. [Accessed 19-06-2010].

[40] NOKIA. Midp: Fileconnection api developer's guide. Available at: http://www.forum.nokia.com, 2006. [Accessed 19-06-2009].

[41] OAK, M. How does bluetooth work? Available at: http://www.buzzle.com/articles/how-does-bluetooth-work.html, 8 2008. [Accessed 20-04-2009].

[42] OBJECTS, C. Delivery Multimedia Integration Framework, ISO/IEC 14496-6 Final Draft International Standard, ISO. Tech. rep., IEC JTC1/SC29/WG11.

[43] PCSTATS. Philips implements wireless streaming audio. Available at: http://www.pcstats.com/releaseview.cfm?ReleaseID=763, 2001. [Accessed 28-08-2009].

[44] PEURSEM, J. JSR 118 Mobile Information Device Profile 2.0, 2002.

[45] PINKUMPHI, S., AND PHONPHOEM, A. Real-Time Audio Multicasting on Bluetooth Network. *Group 1*, P1.

[46] POSTEL, J., ET AL. Internet protocol-DARPA Internet program protocol specification. Tech. rep., STD 5, RFC 791, DARPA, 1981.

[47] RATHI, S., AND ARCHITECT, M. Blue tooth protocol architecture. *Dedicated Systems Magazine* (2000), 28–33.

[48] RHODES, C. Bluetooth Security. *East Carolina University* (2006).

[49] SANTKUYL, M. Unified communications definitions - jitter. Available at: http://searchunifiedcommunications.techtarget.com/sDefinition/0,,sid186gci213534,00.html, 2008. [Accessed 05-10-2009].

[50] SILLEN, M., AND NORDLUND, J. Real-time audio streaming in a mobile environment using j2me. Master's thesis, Umea University, 2005.

[51] SONICSPOT. Wave file format. Available at: http://www.sonicspot.com/guide/wavefiles.html, 2007. [Accessed 10-12-2009].

[52] SUN. Java ME at a Glance. Available at: http://java.sun.com/javame/index.jsp, 2009. [Accessed 05-03-2009].

[53] TECHTERMS. Bitrate. Available at: http://www.techterms.com/definition/bitrate, 2009. [Accessed 27-10-2009].

[54] VAZQUEZ-BRISENO, M., AND VINCENT, P. An Adaptable Architecture for Mobile Streaming Applications. *IJCSNS 7*, 9 (2007), 79.

[55] WEBB, W. FEATURES-Bluetooth vendors bite the bullet-After a year of indecision over interoperability issues, manufacturers bravely line up to test the market with a flood of new Bluetooth products. *EDN 46*, 7 (2001), 119–124.

[56] XIAOHANG, W. Video Streaming over Bluetooth: A Survey.

[57] ZHENG, P., AND NI, L. *Smart phone and next generation mobile computing*. Morgan Kaufmann, 2005.

Appendix A

Glossary of Acronyms

J2ME - Java 2 Mobile Edition

HTTP - Hypertext Transfer Protocol

RTSP - Realtime Streaming Protocol

RFCOMM - Radio Frequency Communication

L2CAP - Logical link Control and Adaptation Protocol

MPEG - Moving Picture Experts Group

MP3 - MPEG-1 Audio Layer 3

MTU - Maximum transmission unit

MMAPI - Mobile Media API MNP

JAR - Java Archive

WAV - Waveform audio file

AAC - Advanced Audio Coding

WMA - Windows Media Audio

MIDI - Musical Instrument Digital Interface

AMR - Adaptive Multi-Rate

3GP - 3rd Generation Partnership Project container

MP4 - MPEG 4

A2DP - Advanced Audio Distribution Profile

PAN - Personal Area Network

ISM - Industrial, Scientific, and Medical

HCI - Host Control Interface

RF - Radio Frequency

SCO - Synchronous Connection Oriented

ACL - Asynchronous Connectionless Link

OBEX - Object exchange

SDP - Service Discovery protocol

UUID - Universally Unique Identifier

GAP - Generic Access Profile

HCI - Human interface device

PIM - Personal Information Management

MANET - Mobile ad-hoc network

PSK - Phase-shift keyring

pi/4-DQPSK - Quadrature phase shift keyring

8DPSK - Differential phase shift keyring

NIST - National Institute of Standards and Technology

DoS - Denial of Service

SMS - Short Message Service

PIN - Personal Identification Number

QOS - Quality of Service

PDA - Personal Digital Assistant

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

VDP - Video Distribution Profile

GAVDP - Generic Audio/Video Distribution Profile

LMP - Link Management Protocol

AVDTP - Audio/Video Distribution Transport Protocol

SRC - Source

SNK - Sink

RTP - Real-time Protocol

GPRS - General Packet Radio Service

3G - Third generation

SIG - Special Interest Group

MNP - Multicast Notification Packet

BNEP - Bluetooth Network Encapsulation Protocol

SBC - Sub-band codec

PPP - Point-to-point protocol

DT - Client which accesses services

LAP - Client which hosts services

3GPP - Third Generation Partnership Project

ETSI - European Telecommunications Standards Institute

AMR-WB - Adaptive Multi-Rate Wideband

CMR - Codec Mode Request

3GPP2 - Third Generation Partnership Project 2

RIFF - Resource Interchange File Format

API - Application Programming Interface

OS - Operating system

BREW - Binary Runtime Environment of Wireless

CLDC - Connected Limited Device Configuration

CDC - Connected Device Configuration

MIDP - Mobile Information Device Profile

RFID - Radio Frequency Identification

STDV - Standard Deviation

JVM - Java Virtual Machine

KVM - Kilobyte Virtual Machine

RAM - Random Access Memory

GCF - General Connection Framework

RMS - Record Management System

MIME - Multipurpose Internet Mail Extensions

CBR - Constant Bitrate

VBR - Variable Bitrate

Appendix B

L2CAP and RFCOMM speeds and times

This appendix details the average time, average speed and normalized times when transferring between Devices A and B using both the L2CAP and RFCOMM protocols.

Table B.1: Average time, average speed and nomalized times obtained with the L2CAP protocol (Device A to B)

File Type	AVG time (Seconds)	AVG speed (KBps)	Normalized Time
MP3	32.924	134.859	7.422
WMA	46.883	135.803	7.365
M4A/AAC	50.201	134.061	7.462
MP4	80.272	136.400	7.332

Table B.2 highlights the average time, average speed and normalized times obtained with the L2CAP protocol when sending from Device B Device A:

Table B.2: Average and normalized speeds and times obtained with the L2CAP protocol (Device B to A)

File Type	AVG time (Seconds)	AVG speed (KBps)	Normalized Time
MP3	33.700	131.666	7.597
WMA	46.072	138.200	7.237
M4A/AAC	48.809	137.877	7.255
MP4	79.807	137.204	7.289

It can be seen that the normalized times between devices are consistent for the L2CAP protocol.

Table B.3: Average time, average speed and nomalized times obtained with the RFCOMM protocol (Device A to B)

File Type	AVG time (Seconds)	AVG speed (KBps)	Normalized Time
MP3	42.792	104.059	9.647
WMA	63.667	100.015	10.001
M4A/AAC	68.671	100.183	10.207
MP4	109.629	99.967	10.013

Table B.3 highlights the average time, average speed and normalized times obtained with the RFCOMM protocol when sending from Device A to Device B:

Table B.4: Average and normalized speeds and times obtained with the RFCOMM protocol (Device B to A)

File Type	AVG time (Seconds)	AVG speed (KBps)	Normalized Time
MP3	44.453	99.904	10.021
WMA	62.819	101.364	9.868
M4A/AAC	68.676	98.034	10.208
MP4	111.046	98.598	10.143

The normalized times for the RFCOMM protocol are also consistent, which shows the consistency of the Bluetooth protocol per byte transferred for varying file sizes.

Appendix C

Link disconnection error prevalence at distance

As the distance of the transmitted 6.6 MB file increased the link disconnection error became more prevalent, and was especially noticeable from distances of 7m upwards. Table C.1 shows the distances where the link disconnection error occured:

Table C.1: Distances of link disconnection occurence

Distance	Attempt 1	Attempt 2	Attempt 3	Attempt 4
1				
2				
3				
4				
5				
6		Error -6305		
7		Error -6305		
8				Error -6305
9	Error -6305	Error -6305		
10				
11	Error -6305		Error -6305	
12	Error -6305	Error -6305		
13	Error -6305	Error -6305		Error -6305
14	Error -6305	Error -6305	Error -6305	Error -6305
15				

Table C.1 contains five columns, with the first column indicating the distance at which the 6.6

MB file was transmitted, and the remaining columns showing the iterations of each transmission at the 1m distance intervals. From Table C.1 it can be seen that link disconnection errors first occured at the 6m mark, and gradually became more prevalent as the distance increased (approaching the theoretical maximum class 2 Bluetooth distance of 10 and beyond). The error occured three times at the 13m mark and four times at the 14m mark, resulting in no results for the transmission at 14m. Apart from the possibility of transfer speeds being influenced by interference in the 2.4 GHz frequency range mentioned in Section 4.6.3, the increasing amount of link disconnection errors could also account for such erratic behaviour at distances 12m, 14m and 15m.

Appendix D

Audio quality ratings for transmission units

This appendix shows the ratings of audio quality for each user, each audio file type, and each transmission unit size.

Table D.1: Optimum transmission unit size for audio streaming

Transmission unit size (Bytes)	File Type	Bitrate (KBps)	User 1	User 2	User 3
668	MP3	16	6	5	5
601	MP3	16	5	5	5
534	MP3	16	5	6	6
468	MP3	16	5	6	6
401	MP3	16	8	7	8
334	MP3	16	7	7	7
267	MP3	16	4	7	6
668	WAV	90.29	7	7	6
601	WAV	90.29	6	6	6
534	WAV	90.29	7	6	6
468	WAV	90.29	6	6	5
401	WAV	90.29	6	5	6
334	WAV	90.29	4	5	5
267	WAV	90.29	3	4	2
668	AMR	1.05	5	6	6
601	AMR	1.05	5	6	6
534	AMR	1.05	5	6	6
468	AMR	1.05	6	5	5
401	AMR	1.05	6	7	6
334	AMR	1.05	6	6	5
267	AMR	1.05	6	5	6

For most of the transmission unit sizes, users had similar ratings for the quality of audio playback, except in the cases of transmission unit 267 for MP3 and WAV.

Table D.2: Inconsistent ratings for audio playback

Transmission unit size (Bytes)	File Type	User 1	User 2	User 3
267	MP3	4	7	6
267	WAV	3	4	2

A possible explanation for users not agreeing on the audio quality for transmission unit size 267, is because a very low transfer rate is achieved due to the smaller packet size and with MP3

and WAV formats being considerably larger than the AMR format, they require more bandwidth to improve the clarity of audio playback. With limited bandwidth due to the decreased packet size, jitter results in playback and is the cause of sub-standard audio playback. User ratings for the transmission unit size of 267 for AMR are relatively consistent due to AMR audio files requiring a fraction of the bandwidth of MP3 and WAV files, thus still maintaining an acceptable level of audio quality.