Bluetooth audio and video streaming on the Java ME platform

Curtis Sahd and Hannah Thinyane
Department of Computer Science
Rhodes University, P. O. Box 94, Grahamstown 6140
Tel: +27 46 603 8111 , Fax: +27 46 622 5049
email: curtissahd@gmail.com, h.thinyane@ru.ac.za

Abstract- This paper investigates the feasibility of Bluetooth as a protocol for audio and video streaming between mobile phones using the Java ME platform. A comparison is made between Radio Frequency Communications (RFCOMM) protocol and the Logical Link Control and Adaptation Protocol (L2CAP) to determine which protocol can support the fastest transfer speed between two mobile devices. The L2CAP protocol is shown to be the most suitable, providing average transfer rates of 136.17 KBps. Using this protocol a second experiment is 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 emerged as the most suitable format. Another experiment is conducted which determines the optimum packet size for transfer between devices. A MTU size of 668 bytes is used, resulting in the highest transfer speed of 136.58 KBps. A final investigation is undertaken to determine the effects of distance on transfer speed, which shows that erratic transfer speeds occur between 7m and 15m. This research shows 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 are only playable once the entire media file has been transferred.

Index Terms—streaming, Bluetooth, audio and video, J2ME

I. INTRODUCTION

Across both the developed and developing world, mobile phones are growing in popularity and have overtaken computers in terms of number of handsets available [3]. Popularity has grown to such an extent that they are now the most popular Information and Communications Technology (ICT) devices [3]. 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 [3]. 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 paper 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 (Java ME) 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 [9]. It is used in cell phones, laptop computers, televisions, television remote controls and various computer peripherals [9].

At the same time there has been development on the software front, in the form of the Mobile Media API (MMAPI) [2]. 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 Hypertext Transfer Protocol (HTTP) and Realtime Streaming Protocol (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 [5]. 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 HTTP protocol and the RTSP protocol. 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 [5]. This paper describes an investigation into the feasibility of audio and video streaming using the Bluetooth protocol on the Java ME platform. It presents a testbed which was developed to determine the feasibility of audio and video streaming between two mobile devices.

II. RELATED WORK

Numerous researchers have proposed frameworks for streaming media using a variety of protocols.

Vazquez-Briseno and Vincent [11] 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. Their architecture takes the limitations of mobile devices into consideration and has thus been designed in an efficient manner. One of

their contributions to audio streaming is that they manage to stream partially downloaded Adaptive Multi-Rate (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 Realtime Transport Protocol (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.

Sillén and Nordlund [10] 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. Sillén and Nordlund [10] 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 also 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. They also encountered the problem of slight breaks in playback when switching from one buffer to another, which they minimized by ensuring that the player switched between buffers when there were breaks in the sentence. It can be seen that double buffering and multiple player instances enabled streaming on the Java ME platform, and it was thus decided to follow a similar double buffering approach for the mobile testbed used throughout experimentation.

III. Design and Implementation

This section 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 role of the receiving device being detailed. Finally, the significance of buffers and threads are highlighted, with particular attention being paid to the layout of buffers in the mobile testbed, along with media playback control mechanisms.

The mobile testbed was developed using the Ubuntu Linux operating system (Kernel 2.6.31-15-generic) with the Netbeans IDE (Version 6.7.1). The testbed was designed to conduct experiments on the Nokia N95 8GB and the Nokia N82 (Symbian version 9.2; S60 3rd Edition; Feature Pack 1) . The language chosen for development was the Java 2 Mobile Edition (Java ME) with CLDC (version 1.1)/MIDP (version 2.0). Development was made possible through the implementation of a number of optional APIs: the FileConnection API (version 1.0) [6] for media file access and serialization; the Bluetooth Wireless Technology API (version 1.1) [7] for transmission and receival of streams; and MMAPI (version 1.1) [2] for media playback and control.

The mobile testbed consists of two components: the sending device (media streaming source), and the receiving device (media streaming destination). For the purposes of brevity, only the functioning of the receiving device is discussed. Figure 1 shows the design of the testbed on the receiving device:

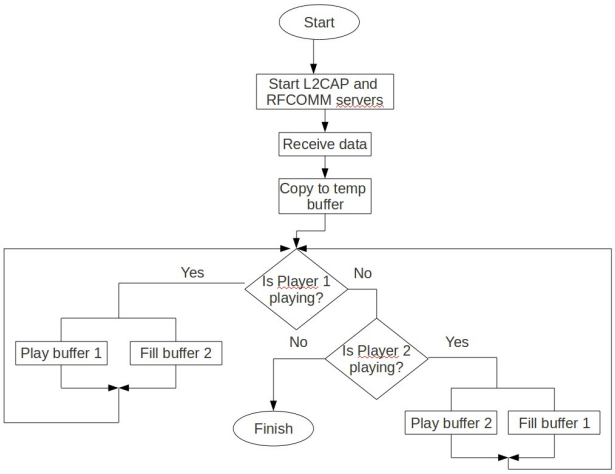

Figure 1: Flow chart diagram (receiving device side)

As can be seen in Figure 1, 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.

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. In the same way that Vazquez-Briseno and Vincent [11] and Sillén and Nordlund [10] implemented double buffering to overcome the limitations of the MMAPI, it was decided to use the same architecture for this testbed. As such, 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 Java ME 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 2 depicts the various buffers used and their relationships and functions.

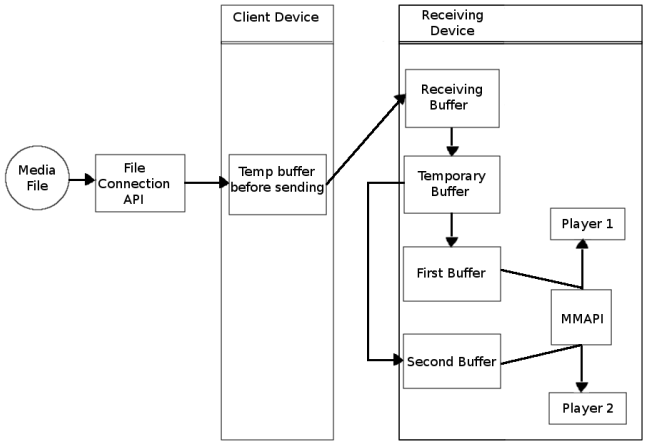

Figure 2: Testbed buffers and their relationships

From Figure 2, it can be seen that 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. 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 receival 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.

When receiving any of the media files, it was discovered that without a 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 in terms of playback quality and transfer speed, 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.

Since this testbed falls into the untrusted application category, the user has to grant access to the application when Bluetooth services are being accessed. Unlike the RFCOMM protocol, the L2CAP protocol requires the transmission unit size to be selected. Once the L2CAP server has been initialized and the transmission unit size selected, the media file to be streamed is selected; the file is serialized; receiving devices (with advertised services) are scanned; and the file is transmitted to the receiving device. The reason for the L2CAP server residing on both the sending and receiving devices, is to cater for a case where media files were streamed bi-directionally.

IV. Methodology

Six experiments were undertaken to assess the feasibility of Bluetooth audio and video streaming on the Java ME platform. Each experiment was conducted four times to minimize any inconsistencies encountered.

The first experiment conducts a comparison between the RFCOMM and L2CAP protocols which yields the superior protocol in terms of transfer speed and efficiency of media transmission. This experiment involved sending the following four files from the N95 to the N82 and vice versa: a 4.3MB MP3 file; a 6.2MB WMA file; a 6.6MB M4A/AAC file; and a 10.7MB MP4 file. The reason for the bi-directional transfer of the media files was to determine the differences (if any) when sending in opposite directions between the devices. These files, as well as the ones for Experiment 2 below, were transmitted with the MTU of 668 bytes at a distance of 30cm.

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 (Experiment 2) . The following media files were tested for their suitability for streaming: WAV; MP3; AAC; WMA; MIDI; AMR; 3GP; and MP4. Playback of the media files (excluding MIDI and AMR) was initiated once 600 KB of the media file had been transferred. Playback of the MIDI and AMR media streams was initiated once 50 KB and 100 KB had been transferred.

The effects of distance on transfer speeds of the L2CAP protocol are then determined (Experiment 3), which shows the range of distances in which the transfer speed is most suitable to multimedia streaming. A 6.6 MB M4A/AAC file was transmitted over a total distance of 15m at 1m intervals. An MTU of 668 bytes was used.

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 poor streaming quality. This experiment (Experiment 4) was conducted by sending a 6.6 MB M4A/AAC file from the Nokia N82 to the Nokia N95 at a distance of 30 cm. The MTUs were decreased from the maximum MTU size (668 bytes) to a minimum of 134 bytes in approximate 10% decrements of the MTU. The average time was compared to the expected average time to show the consistency (or lack thereof) of the Bluetooth protocol when the transmission unit size was altered. A tradeoff between the transmission unit size and the transfer speed is encountered throughout the variation of

the transmission unit size.

Experiment 5 deals determines the optimum transmission unit size for streaming each of the following three file types: MP3; WAV; and AMR. A user rating of the quality of playback determines the optimum transmission unit size for each file type. 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 Nokia N82 to the Nokia N95. 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.

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 (Experiment 6). 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.

V. Results and Analysis

This section presents the results of the six experiments mentioned above, which enable conclusions to be drawn about the suitability of audio and video streaming between mobile phones using the Bluetooth protocol, on the Java ME platform.

1) Experiment 1: RFCOMM vs L2CAP

The average time taken and average speed obtained when transferring four media files from the N95 to the N82 using the L2CAP protocol is highlighted in Table 1 below:

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 1: Average time and average speed obtained with the L2CAP protocol (N95 to N82)

The results from the Table 1 show that the average transfer speed obtained when transferring each media files between the N95 and the N82 is consistent with a standard deviation (STDV) of 1.031 (not shown). Transfer of the MP4 file yielded the highest transfer speed of 136.400 KBps followed closely by the WMA file with 135.803 Kbps.

The transfer speeds obtained with the RFCOMM protocol when sending from the N95 to the N82 yielded a STDV of 2.004 for the transfer speed, which shows that the RFCOMM protocol has a consistent transfer speed. The maximum transfer speed of the L2CAP protocol from Table 1 is 136.400 KBps, which is 32.341 KBps faster than the maximum transfer speed of the RFCOMM protocol (104.059 Kbps).

Paired t-tests with assumed equal variance, comparing the time taken to transmit from the N82 to the N95 and the time taken from the N95 to the N82 also revealed that there is no statistically significant difference between the transfer speeds when sending in opposite directions between the N95 and the N82 ($p = 0.21$ and $t = 0.824167$ for the L2CAP protocol; $p = 0.13$ and $t = 1.169407$ for the RFCOMM protocol).

Additional paired t-tests with assumed equal variance were conducted which showed that there is a statistically significant difference between the transfer speeds of the L2CAP and RFCOMM protocols ($p = 1.72 \times 10^{-44}$ and $t = -37.3121$). This led to the conclusion that the L2CAP protocol has much higher transfer speeds the RFCOMM protocol.

2) Experiment 2: Suitability of media types for streaming

Table 2 shows the suitability of the eight media types to streaming on the Java ME platform:

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

Table 2: Suitability of media types to streaming

This table shows the media type to be streamed and the associated minimum bitrate. 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 III above. 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.

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.

The WAV, MP3, MIDI, and AMR file formats were streamable with the MP3 file achieving the highest transfer rate of 141.988 Kbps during the population phase of streaming. Table 2 also shows the varying transfer speeds obtained during the buffer population and playback phases of media streaming. This experiment showed that even though a number of media file formats were supported, only

a small subset were streamable, none of which were video formats.

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

Figure 4 shows the relationship between the average transfer speed and distance:

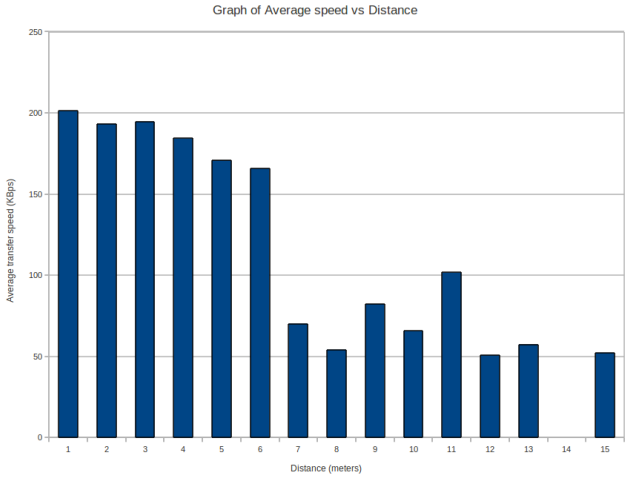

Figure 4: Graph of average transfer speed vs distance

From Figure 4 it can be seen that the average transfer speed is consistent for distances up to and including 6m, after which distance the speed decreased and fluctuated considerably. No results were obtained at 14m due to link disconnection errors encountered. This experiment thus showed that the optimum streaming performance occurred between 1m and 6m (inclusive) which resulted in the ability to stream files of higher bitrates.

4) Experiment 4: Variation of transmission unit size

Figure 5 shows the average speed and average expected speed vs. the transmission unit size:

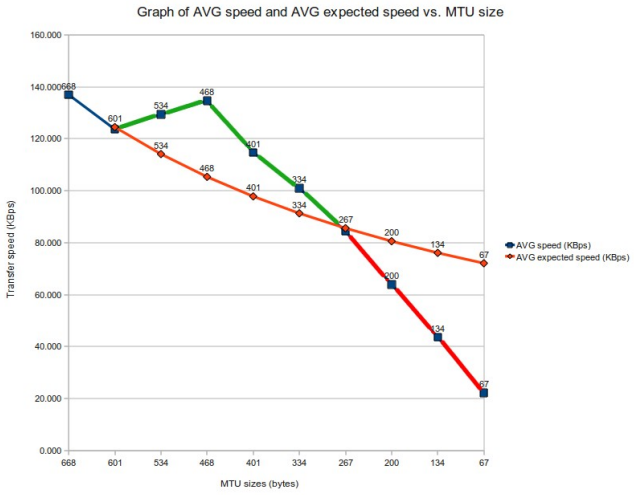

Figure 5: Graph of average and expected speed vs. transmission unit size

The transfer speed is stable for transmission units equating to more than 40% of the MTU (267 bytes), thereafter there is a drastic 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 extremely constrained in terms of memory and buffer size. For packet sizes from 267 bytes through to 601 bytes, the L2CAP protocol fares well by performing faster than expected (actual average speed depicted as the green line with the blue dots, and expected average speed depicted as orange line). 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.

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.

5) Experiment 5: Optimum transmission unit size for audio streaming

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.

6) Experiment 6: Effects of dropped packets on audio playback

Table 3 shows the transfer speeds for increasing and decreasing distances with associated playback quality:

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 3: Transfer speeds and playback quality ratings

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).

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 the N95 and the N82 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.

VI. Conclusion

It can be concluded that the Java ME 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.

VII. Acknowledgements

The work reported in this paper was undertaken in the Telkom Centre of Excellence in Distributed Multimedia at Rhodes University, with financial support from Telkom, Comverse, Tellabs, Stortech, Amatole Telecom Services, Bright Ideas 39 and THRIP.

References

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

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

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

[4] 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.

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

[6] JARVINEN, B. JSR 75: PDA Profile for the J2ME Platform, 2004.

[7] KUMAR, B. Jsr-82: Java APIs for bluetooth, 2004.

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

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

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

[11] VAZQUEZ -B RISENO , M., AND V INCENT, P. An Adaptable Architecture for Mobile Streaming Applications. IJCSNS 7, 9 (2007), 79.

Curtis Sahd received his undergraduate degree in 2007 from Rhodes University and is presently studying towards his Master of Science degree at the same institution. His research interests include: Bluetooth; mobile; human computer interaction; mobile device development; instant messaging and voip.