Exploring Playback and Recording of Web-based Audio Media on Low-End Feature Phones

Akhil Mathur
Bell Labs Research
Alcatel-Lucent, Bangalore, India
akhil.mathur@alcatel-lucent.com

Shivam Agarwal
Indian Institute of Technology Bombay
Mumbai, India
shivamagarwal.iitb@gmail.com

Sharad Jaiswal
Bell Labs Research
Alcatel-Lucent, Bangalore, India
sharad.jaiswal@alcatel-lucent.com

ABSTRACT

Web-based audio information systems have the potential to bring the full promise of the Internet to the developing world. However, these systems run into a practical difficulty - insufficient support for playback and recording of web-based multimedia from feature phones. At the moment, feature phones constitute more than 90% of all phone shipments in emerging markets like India, and will continue to form a significant fraction (>50%) several years from now.

In this paper we explore the practical challenges associated with audio playback and audio recording on feature phones. We present a systematic evaluation of the options to play and record audio media in a range of feature phones, and highlight problems stemming from a lack of memory, slow processor speeds and no support for progressive streaming downloads. In summary, the players in these phones are usually designed to handle audio files of short durations (a few minutes). Anything longer results in frequent breaks in playback (of up to 300msecs), and a very poor experience for the end-users. We investigate various solution approaches that may overcome these issues, and present an implementation and evaluation that achieves a seamless user experience on any audio stream.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Design Studies, performance attributes.

General Terms

Measurement, Performance, Design, Experimentation, Human Factors

Keywords

J2ME, audio streaming, low memory, feature phones, developing regions, audio-based systems

1. INTRODUCTION

Developing countries have seen a massive explosion of mobile phones in the past decade. As per the Telecom Regulatory Authority of India (TRAI), there are nearly 800 million cellular subscriptions in India alone [24]. Mobile phones have been used for various purposes in developing regions such as facilitating education [11, 14], healthcare delivery [4.9], agriculture information dissemination [21], and crowdsourcing [22].

Due to the prevalent low-literacy levels in developing regions, researchers in the past have suggested moving away from text-based information content to audio content [21]. Clearly this has benefits from a usability perspective. Moreover, any individual with a microphone (embedded in their mobile phone) can potentially create audio content, and make it available for consumption.

Current audio-based information systems in developing regions are mostly built around IVRs or Community Radio (media) stations. IVRs suffer from poor *usability* [25] and *interactivity*, and the *reach* of Community Radio stations operating over FM is restricted by the power of their transmitter and government regulations. We believe that WWW can be an alternative for creating and disseminating audio content as it may solve many of these problems related to usability, reach, scalability and interactivity.

With recent trends showing that the usage of mobile internet has surpassed desktop internet in India [18], it is natural to expect that the adoption of web-based audio systems will happen via mobile phones. This belief is strengthened by the rapidly decreasing costs of mobile data plans in places like India [10], which has made it possible for a vast majority of the population to afford an internet data plan.

However, this vision runs into the practical constraint of suitable support from the handsets. Developing countries have a huge population of feature phone users. A recent study [17] brings out that as much as 94% of all phones shipped in the first quarter of 2012 in India were feature phones, and nearly 50% will remain so 5 years from now. The percentage number of feature phone users will be even higher in the underserved sections of the society who are the primary target audience for the ICT4D community.

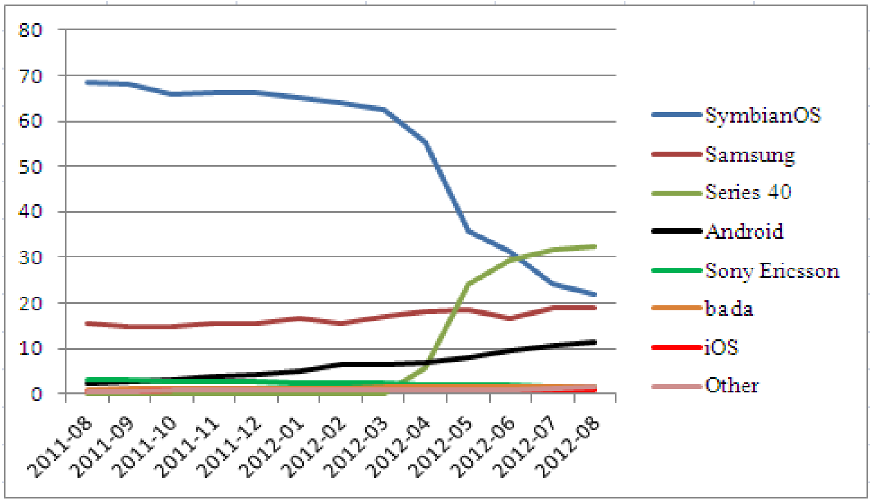

Figure 1: Percentage of market share for different mobile platforms in India (Source: statcounter.com)

Feature phones are typically characterized as constrained platforms, due to limitations in computing, storage and interface capabilities. Therefore, it becomes extremely important to ensure that web-based audio systems are supported on feature phones despite their computing limitations.

In this paper we explore the challenges associated with two of the primary features of any audio based information system – playing and recording audio files. We present a systematic evaluation of the options to play and record audio media in a range of feature phones, and highlight problems stemming from a lack of memory, slow processor speeds and lack of support for progressive streaming downloads. In summary, the players in these phones are designed to handle audio files of short durations (a few minutes). Anything longer results in breaks in playback, and a very poor experience for the end-users. We investigate various solution approaches that may overcome these issues, and present an implementation and evaluation that illustrates a much better user experience as compared to the existing options on feature phones.

The paper is organized as follows: in Section 2 we discuss the related work in the space of audio-based systems and web-based solutions for developing regions. Next (Sec. 3.1), we systematically list down the challenges of playing audio files on feature phones, followed by (Sec. 3.2) a solution architecture that addresses these issues. We then carry out an extensive evaluation (Sec. 3.3) to bring out the benefits of our approach. Next (Section 4), we discuss the challenges of recording audio files on feature phones and explore various options to address these issues. Finally, we wrap up (Sec. 5) with summary and possible future directions.

2. RELATED WORK

Feature phones still remain the most popular phones in developing countries like India, and they are a platform to reach a broader segment of users in developing counties. As shown in Figure 1, feature phone platforms like Series 40, Samsung feature phone OS have a much higher percentage of market share when compared to Android or iOS.

To the best of our knowledge, we are not aware of any work in the HCI, ICT4D or Networking literature which looks at the challenges associated with playing and recording Web-based audio content on low-end feature phones. Some technical solutions to these issues have been discussed on software development forums, but there is no systematic evaluation of their pros and cons. Apart from playback and recording, there exist several *other* challenges for audio media support on feature phones – e.g. small screen real estate, limited navigation and interaction capabilities - but these are out of the scope of this paper.

There has been a recent burst of audio-based systems targeted towards the developing world. Mudliar et al. [19] developed CGNet Swara an IVR based system which enables callers to record messages of local interest, and listen to messages that others have recorded. Patel et al. [21] also developed an interactive voice forum to facilitate agriculture discussions among small-scale farmers in India. The Spoken Web project [1] aims to create a complimentary Web for people in developing regions through a voice based interface using an ordinary telephone. Chu et al. [8] worked on featherweight multimedia devices which combine audio with non-electronic visual displays such as paper and deployed them in agriculture and healthcare domains in India. Audio Green [3] uses low-cost MP3 players for disseminating audio content for agriculture extension. Finally, there are plenty of Community Radio (CR) stations across the world, which create hyper-local audio content for grassroot communities and broadcast them over FM.

However, none of these systems use the Web to create or deliver audio content to the end-users (and hence do not illustrate the issues we bring out in this paper). With the prices of internet plans going down in developing countries like India, we believe that mobile internet can be a cost-effective platform for creating and disseminating audio content, while also addressing issues around usability, reach and scalability encountered by the above systems. For instance, Web can be used by a Community Radio station to deliver audio content to users outside the reach of its transmitter. Crowdsourced tasks related to audio transcribing or audio translation (originating from anywhere in the world) can be easily distributed to crowd workers in developing regions using a web based system [22]. And a web-based system can be designed using a multimodal interface which augments the audio system with text or icons, thus solving several usability issues [25] with the unimodal IVR systems.

The complimentary problem of improving internet/web *access* in the developing world has been addressed by several works. Some examples include, TinyPC [2], accelerating web access using a caching and prefetching engine [6], RuralCafe [7] and mango [13].

Finally, there has been considerable activity in the wireless networking community, around scheduling algorithms, dynamic adaption of audio/video rates, and sizing of network buffers on end-user devices to enable low jitter/latency multimedia streaming [5, 16]. Our work is complementary in the sense it looks at the impact of device restrictions on the user experience.

3. CHALLENGES IN PLAYING AUDIO CONTENT ON FEATURE PHONES

We did a survey of the popular audio-based systems in the developing world, and found three major types of audio content being created:

a) Audio programs with discussions, debates or skits – these programs can be up to 30-40 minutes long and 15-20 MBs is size.

b) Short audios such as movie or folk songs which are typically 7-10 minutes long and 5-7 MBs in size.

c) Audio forums are also quite popular where users leave questions/comments for experts and other listeners. These questions/comments are relatively small (1-2 minutes long and ~500kBs in size).

One option to play such remote audio content on feature phones is to first download the entire file and then play it. However, downloading larger audio files (in case of debates or folk songs) is clearly not desirable because of the time involved in the process. For example, a 7MB audio file will take about 6 minutes to download on a 140kbps EDGE connection (or between 30secs – 1 min given the average 3G connection speeds), and this waiting time may lead to user dissatisfaction.

Although content like audio comments are smaller in size and take less time to download (a 500kB audio comment will take about 25 seconds to download on a 20kBps EDGE connection), they tend to be more interactive in nature. These comments are best heard in continuum, and even a small download time of 25 seconds for such interactive content may adversely affect the user experience.

Given the above reasons, we henceforth focus on support for ***streaming audio*** – instead of download and play.

3.1 Feasibility of audio streaming:

We explored possible device-independent ways to stream web-based audio content on the feature phones. Our methodology involved replicating or implementing these techniques on four different feature phones: Nokia C101 (Series 40, 6th edition), Nokia 3110c (Series 40, 3rd edition), Samsung Champ Deluxe Duos and Samsung Duos GTE2232. These models were selected as representative samples of low-mid range feature phones (based on their cost and features.). Following this we also carried out an exhaustive search and survey from various web forums to confirm or complement our findings (wherever possible) from actual experimentation.

1) **Provide a URL to the native players**: All phones come with a native media player which is capable of playing local audio files. We investigated if it is possible to pass an HTTP or RTSP URL to the player for playing audio streams. We found that native players in all the devices considered could only play a local file, and did not support playing a remote file from a URL.

2) **Streaming audio using a web browser:** A web browser can theoretically stream data using RTSP or HTTP Progressive Download, and pass the data stream to the native player for playback. Our trials with the aforementioned four feature phones showed that none of them support RTSP or HTTP progressive downloads, therefore streaming data is not possible. We also tried some popular audio/video apps like YouTube and VideoStreamer from GetJar.com, a marketplace for J2ME apps. These apps also seem to launch the browser to access videos, and as expected – they could not support media streaming on these phones. Moreover, majority of feature phones do not support multitasking, so even if a phone was capable of streaming using RTSP, it wouldn't be able to invoke the media player (a new process) at the same time.

The above two directions explore native support for audio streaming in feature phones. The next set will explore supporting these features via applications written for feature phones. As things stand now, the J2ME framework is the only widely supported programming environment on feature phones (other options like BREW or Samsung's BADA are limited to particular handset manufacturers). For the remaining part of the paper – as we explore an application level framework for audio streaming, we will focus on J2ME.

3) Launching the native player through J2ME: A J2ME application is allowed to launch the native player of the phone using the "platformRequest" API. Therefore, a possible solution to the streaming problem is to have a two-threaded J2ME application, in which one thread keeps downloading and saving the remote audio in a local buffer on the filesystem, while the other thread invokes the native player to play from the downloaded buffer. In this scheme, the 'download' thread will run in the background and keep writing data to the local buffer, while the native player will be in the foreground playing the available data from the buffer.

However, there are various factors that prevent a universal adoption of this approach. Firstly, it is entirely up to the device manufacturer if it wants to support invoking of native player from J2ME on a particular device, thus making this solution device and manufacturer dependent. Secondly, manufacturers such as Nokia also impose a restriction that only the apps signed by the manufacturer or a mobile operator can invoke the native player [15] – this considerably increases the complexity of widespread deployment of such an approach.

4) **Using the J2ME Player class:**

The J2ME Player[1] is capable of downloading and playing content from a given HTTP URL. However, there are several challenges involved in playing a web-based audio file using the J2ME Player:

No inherent support for streaming: It is known that most low-end feature phones do not support RTSP and HTTP streaming - we verified it on four different phones (Nokia C101, Nokia 3110c, Samsung Champ Deluxe Duos and Samsung Duos GTE2232). Therefore, in order to play an audio file, the J2ME player requires the entire file to be first downloaded into the runtime memory. As we discussed earlier, this 'download and play' approach is not desirable for the kinds of audio content being created in existing ICT4D initiatives.

Moreover, the size of runtime memory available to an J2ME application on low-end features phones is typically between 1-2 MB; therefore loading any audio file larger than 1-2 MB into the memory would result in an **Out of Memory** error. E.g. the audio programs created in a Community Radio station are typically 30 minutes long, and even with a small audio bit rate of 32kbps; the size of a program will be close to 7 MB, thus causing the J2ME player to run out of memory.

Chunking and inter-chunk delays: A possible way to implement streaming and avoid the memory issues is to download the audio files in small chunks less than 1-2MB and pass them to the audio player sequentially. From the player's perspective, these chunks are separate audio files, each of size 1-2 MB. It will load the first chunk into the memory, play it, remove it from the memory, and then go on to load the next chunk as shown in Figure 2.

This scheme indeed solves the streaming and low-runtime memory issues, but it degrades the user experience because of the delay (e.g. t3 – t2 in Figure 2) caused in playing of two successive chunks. We now explain the source of this inter-chunk delay.

A J2ME player has to go through two stages before it can play a media file:

Realize: In this stage, the player communicates with the remote server, downloads the chunk required for playing the audio and loads it into the runtime memory. The time taken to complete the Realize state (D_R) is a function of the chunk size.

$$D_R = T_{download} + c.\ D_{1kB}$$

where $T_{download}$ is the time taken to download a chunk, D_{1kB} is the time taken by the player to load 1kB of data in the memory and c is the size of the chunk in kB

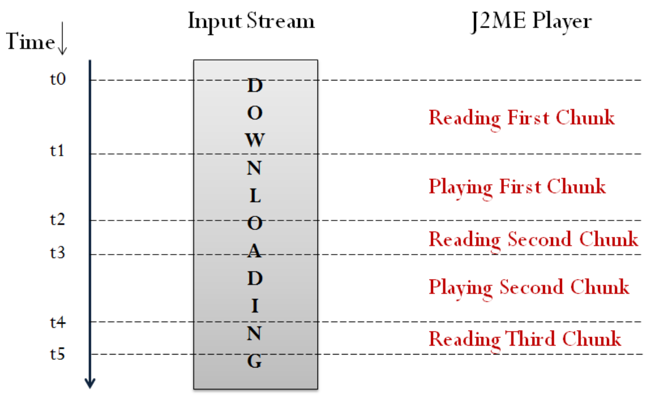

Figure 2: One-player streaming scheme.

Prefetch: Here, the player performs various constant-time operations like acquiring scarce or exclusive resources to prepare itself for playing the audio. The time taken to complete this stage (D_P) is independent of the chunk size.

As discussed earlier, a player treats each chunk as a separate media file, so it has to go through the Realize and Prefetch stages after playing each chunk. Therefore, an inter-chunk (IC) delay of $D_R + D_P$ is introduced after each chunk.

$$D_{IC} = D_P + D_R$$

$$D_{IC} = D_P + c\ .\ D_{1kB} + T_{download}$$

The download time for a chunk ($T_{download}$) is dependent on the network speed, and can be reduced by buffering appropriate amount of data at the beginning of the audio. Such techniques have been studied extensively in the

[1] Player is the name of a class in J2ME which handles audio playback and recording. It does not have any UI associated with it.

computer networking literature and can be applied to reduce the download time between successive chunks.

We are here mainly interested in the interchunk delay which is independent of the network conditions and is caused because of loading the chunk into a player's memory.

$$D'_{IC} = D_P + c \cdot D_{1kB}$$

To measure the extent of this interchunk delay D'_{IC}, we did an initial experiment with a 64kbps bit-rate audio file. We played the audio file for 2 minutes using the scheme described in Figure 2, and we observed the following inter-chunk delays D'_{IC} (averaged across 5 trials):

Table 1: Inter-chunk delays with three different chunk sizes

Chunk size	D'_{IC}
128kB (Playing time: 16 seconds)	514ms
48kB (Playing time: 8 seconds)	346ms
32kB (Playing time: 4 seconds)	293ms

We see that a 32kB chunk from a 64kbps audio will play for 4 seconds (32 * 8 / 64) and then there will be an IC delay of ~300 milliseconds before the next chunk is played. This delay is easily discernible by the human ear, and such high frequency of IC delays can degrade the user experience.

On the other hand, if we choose a larger chunk size (c) like 128kB with playing time of 16 seconds (128 * 8 / 64), the frequency of IC delays will go down, but the magnitude of the delay will increase which may also degrade the user experience.

3.2 Solution architecture

An alternate strategy for streaming audio on low-end phones is using two interleaved audio players operating in sync with each other. When the first player (P1) is playing

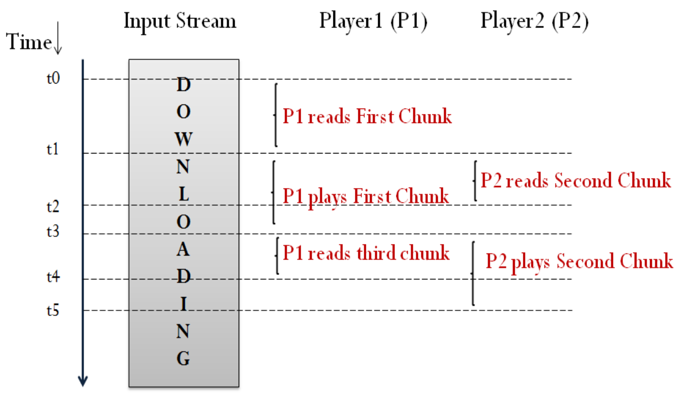

Figure 3: Two-player streaming scheme

an audio chunk, the second player (P2) downloads the next chunk into its memory and prepares itself for playing it. When P1 finishes playing the chunk, P2 immediately starts playing – thus theoretically resulting in no inter-chunk delays. Both players continue this cycle of playing and buffering chunks until the entire audio file is played. Figure 3 illustrates the scheme.

We will now describe the technique in detail. Let us say that an HTTP input stream is opened with the remote server which starts downloading the data at some time t before t0.

- At t = t0, Player1 starts reading the first chunk from the input stream into the runtime memory, i.e. the Realize and Prefetch stages of P1 start.
- At t = t1, Player1 completes reading the data, i.e. it finishes both its Realize and Prefetch stages, and is ready to play the first chunk.
- At t = t1, Player1 starts playing the data. At the same time, Player2 is also initialized and asked to read the next chunk from the input stream in the background, while Player1 is still playing the first chunk.
- At t = t2, Player2 has finished reading the second chunk into the memory.
- For t2 < t < t3, Player2 has already downloaded the second chunk and is waiting for Player 1 to finish playing the first chunk.
- At t = t3, Player 1 finishes playing the first chunk and Player 2 starts playing the second chunk.
- Now, it's the turn of Player1 to act as the buffer. For t3 < t < t4, Player1 starts reading the third chunk from the input stream. While this is happening in the background, Player 2 is playing the second chunk.
- At t = t4, Player 1 finishes reading the third chunk.
- For t4 < t < t5, Player 1 is waiting for Player 2 to finish playing the second chunk.
- At t = t5, Player 2 finishes playing the second chunk and Player 1 starts playing the third chunk.

The above cycle continues till the entire audio is streamed. As evident, there will be theoretically no inter-chunk delay in this two-player setup because the second player already finishes the Prefetch and Realize stages for the $(n+1)^{th}$ chunk while the first player is still playing the n^{th} chunk.

3.3 Evaluation

We now evaluate the performance of a 1-player streaming application with a 2-player streaming application.

Folk songs, folk music, and mainstream music are some of the popular content types on local media platforms like Community Radios. A minimum 128 kbps bit rate is

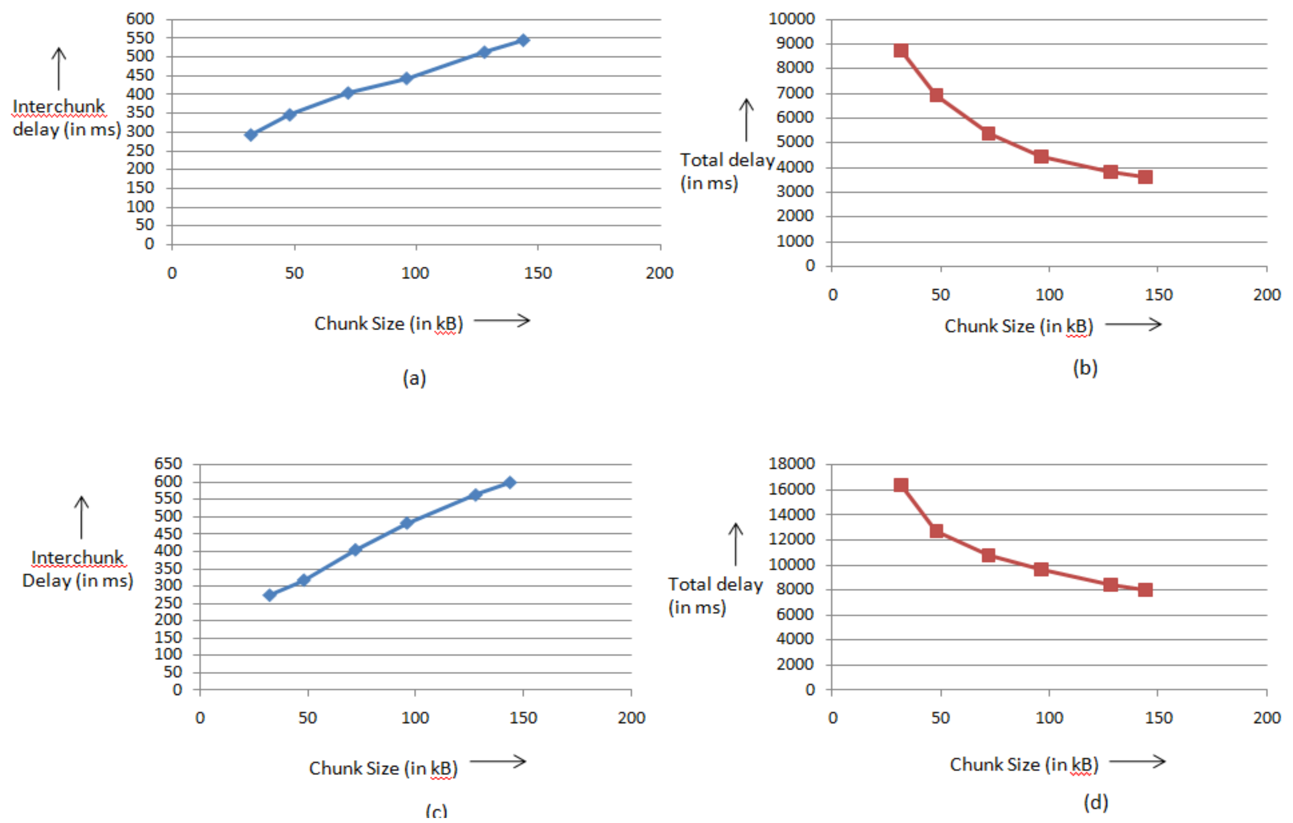

Figure 4: One-player performance (a) Interchunk delay with 64kbps audio (b) Total delay with 64kbps audio (c) Interchunk delay with 128kbps audio (d) Total delay with 128kbps audio. The duration of each audio was 2 minutes.

required for a satisfactory playback of the musical audios [20]. However, audio content which only involves human voice may be encoded at a lower bit rate of 64 kbps. Therefore, we ran our experiments with audio files of two different bit rates: 64kbps and 128 kbps. The experiments were conducted on Nokia C1-01 and Samsung Duos GTE2232 phones running on a 2G connection. We installed both 1-player and 2-player audio streaming J2ME applications on each phone. A logging mechanism was built into each application which calculated the time taken to download a chunk, time taken to play a chunk, interchunk delays (D'_{IC}), and total delays.

One-player performance:

The experiments were conducted with five different chunk sizes: 32kB, 48kB, 96kB, 128kB, and 144kB. As mentioned earlier, audios files for two different bit rates (64kbps and 128 kbps) were used for measurements.

For each combination of (chunk size, bit rate), we ran 5 trials with a 2-minute audio file. Our goal was to understand which chunk size gives the best performance in a one-player setup, i.e. for which chunk size does inter-chunk delay and overall delay is the least.

Results:

Results show that the inter-chunk delay (D'_{IC}) increased with the chunk size (c) for both audio bit-rates. For smaller chunk sizes like 32kB, although D'_{IC} was small (273ms for 128kbps audio, 292 ms for 64kbps audio), the frequency of its occurrence was high (a 32kB chunk will experience a delay after 4 seconds while a 64kB chunk will have it after 8 seconds). On the contrary, large chunk size like 144kB would reduce the frequency of delay, but increase the amount of D'_{IC} (573ms for 64kbps, 601ms for 128kbps) Figure 4a and 4c illustrate our findings.

Therefore, we concluded that both very small and very large chunk sizes degrade the user experience, and a mid-range chunk size like 96kB provides better performance with a 1-player scheme.

We also observed that the total delay ($D_{Total} = N.\ D'_{IC}$, where N is the number of chunks) decreases as the chunk size (c) increase. For a larger chunk size (c), we will need less number of chunks (N) to play the complete file, therefore the total delay decreases with increasing c. Figures 4b and 4d illustrate our results.

Two-player performance:

We compared the performance of a 2-player system vs. the 1-player system with the same experimental conditions as above.

Interchunk delay: Results show very small interchunk delay (10-20ms) for a 2-player system because both Realize and Prefetch stages of a player are completed while the other player is doing audio playback. Figures 5 and 6 illustrate this finding.

We also found that the chunk size has no significant effect over the interchunk delay in a 2-player scheme, e.g. for 64kbps audio, the IC delay was 16ms for 32kB chunk and 21ms for 144kB chunk. Therefore, unlike the 1-player scheme where a higher chunk size is bad for user experience, the chunk-size in a 2-player scheme can be increased as long as it is less than the Java runtime memory restrictions. At any time, the two-player scheme will have two chunks loaded into the runtime memory – hence, the combined size of the two chunks should fit within the restrictions of Java runtime memory.

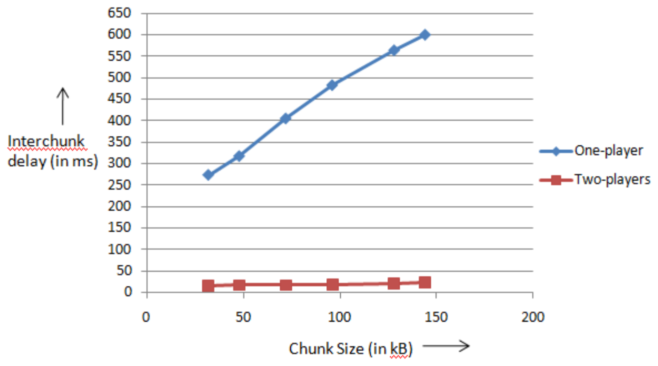

Figure 5: Comparison of interchunk delay (D'_{IC}) in 1-player and 2-player schemes for a 128kbps audio

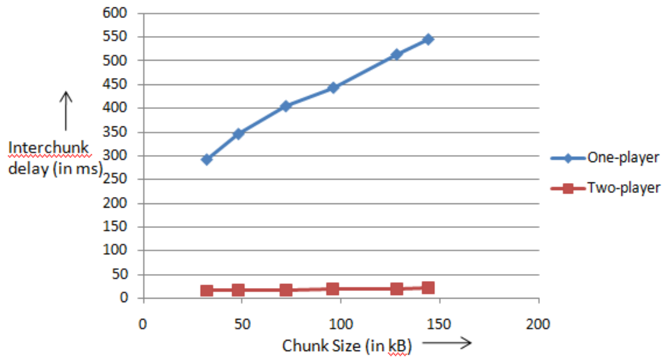

Figure 6: Comparison of interchunk delay (D'_{IC}) in 1-player and 2-player schemes for a 64kbps audio

Total delay: The total delay ($D_{Total} = N.\ D'_{IC}$, where N is the number of chunks) with a 2-player scheme is shown in Figures 7 and 8. For a 32kB chunk and a 2-minute long 128 kbps audio, the 1-player scheme resulted in a total delay of 16380 ms as compared to 900 ms (18x less) delay in the 2-player scheme. Similarly, for a larger chunk size like 144kB, there was a 21x decrease in the total delay of 2-player (306.66 ms) when compared to 1-player (8013.33 ms).

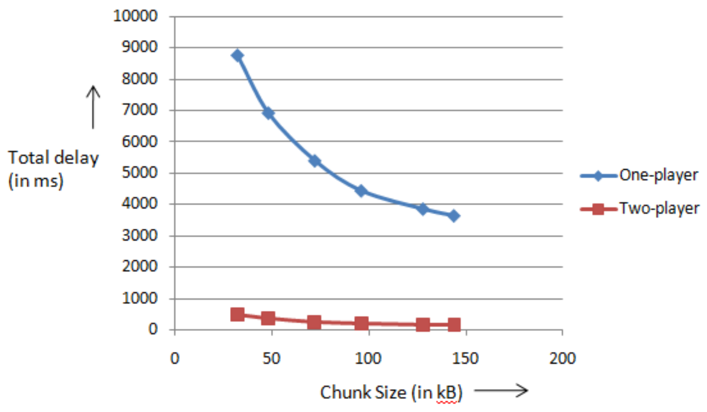

Figure 7: Comparison of total delay in 1-player and 2-player schemes for a 128kbps audio

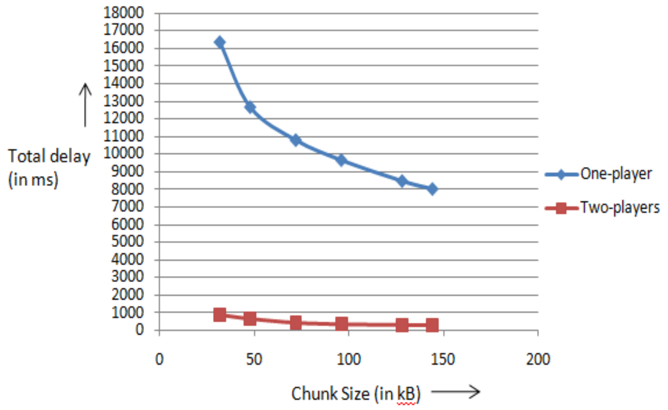

Figure 8: Comparison of total delay in 1-player and 2-player schemes for a 64kbps audio

The increase in the total delay does not affect the user experience, but it will cause the application to run for a longer duration, which may result in more battery consumption. The exact magnitude of battery savings, however, still needs to be experimentally calculated.

To summarize, we showed that in a 2-player approach:

(a) The interchunk delays are very small (~10-20ms), which improves the user experience,

(b) The amount of IC delays is independent of the chunk size. Hence, the chunk size can be increased so as to reduce the frequency of IC delays.

4. CHALLENGES IN RECORDING AUDIO CONTENT ON FEATURE PHONES

We explored the following options to record audio content on a phone and send it to a remote Web server.

Launch the native recorder from a J2ME app: Similar to the native audio player, J2ME apps are also able to launch native audio recorders on the phones. However, this approach has the exact same problems as with launching native players, i.e. need for operator or manufactured signed apps, and lack of universal implementations of 'platformrequest' API.

Use the J2ME audio recorder: J2ME's Player Class can also be used to record microphone's audio inputs. While recording, these audios can either be temporarily stored in a byte array in the memory or they can be stored in the app's RecordStore which is a persistent storage structure on the phone, specific to each J2ME app. As soon as the audio recording finishes, the data can be sent to a remote server.

a) If the recording is **stored in the memory**, it can soon cause Out of Memory errors as the runtime memory is very low on feature phones. We recorded audios and stored them in a byte array on two different phones (Nokia C101 and Samsung Champ Deluxe Duos) and observed that both phones run out of memory soon after the recording begins. For example, if the available memory at the beginning of audio recording is 1000kB and the bit-rate of audio is 128kbps, the application will run out of memory in 1000 * 8 / 128 = 62.5 seconds. Therefore, this method is unsuitable for large audio recordings.

b) By storing the recording in a **RecordStore**, the runtime memory problem can be addressed. However, the maximum size of the RecordStore that an app can create is fixed, and this causes a bound on the length of the audio recording. We observed that the maximum permitted size for a RecordStore on both the phones (Nokia C101 and Samsung Champ Deluxe Duos) was 512kB, thus making this technique also inappropriate for large audios.

c) **A two-recorder approach**: Similar to the two-player scheme, we propose using two audio recorders operating in sync, to get around the practical constraints on the maximum size of audio recording.

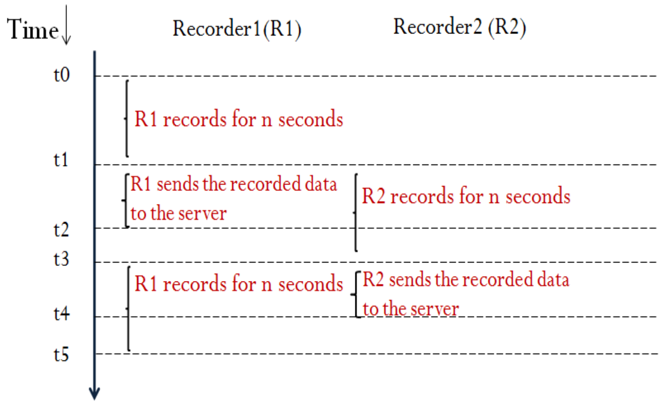

Figure 9: Recording audio using 2 recorders

As shown in the figure 9, the recorder R1 starts recording at time t0. From t0 to t1, R1 records the audio input. At t = t1, R2 takes control of the audio recording. From t1 to t3, R2 records the audio input. In the meantime, R1 uploads all its recorded data to a remote server and frees all its memory resources. After releasing all its resources, R1 is scheduled for the next round of audio recording beginning at t = t3. While R1 is doing the recording from t3 to t4, R2 releases all the recorded data by sending it to a remote server, and gets scheduled for the next round of recording. This cycle keeps going on, thus avoiding any out of memory errors.

We tested this approach on Nokia C101 and Samsung Champ Deluxe Duos with very long audio recordings (30-40 minutes), and we observed no out of memory errors.

5. CONCLUSIONS AND FUTURE WORK

Audio-based information systems like IVRs and Community Radios are gaining popularity in developing regions. However, these systems face challenges related to interactivity, scale, reach and usability. Many of these challenges can be addressed by developing a web-based audio system. The falling costs of mobile data plans in developing regions also make the deployment of web-based systems economically feasible.

Designing web-based audio systems for low-end feature phone pose several challenges. The main focus of this paper was to understand the issues related to the recording and playback of web-based audio content on feature phones.

We discussed the limitations of the features phones with respect to runtime memory and how it affects the playing of large audios. We evaluated various approaches of streaming audio content on feature phones and showed that by using two audio players running in sync with each other, we can support seamless audio streaming with no interchunk delays. We also explored various options to record audios on a feature phone and send it to a remote server. We found that the standard approaches of audio recording are not suitable for recording large audios. Instead, we propose a scheme of using 2 interleaved audio recorders and show that it can record large audio files. We have implemented our proposed solution using standard J2ME libraries and it does not have any dependencies on the phone manufacturer or the phone model. We have verified that the solution works on various J2ME phones such as Nokia C1-01, Nokia C2-01, Nokia X2, Nokia 5580, and Samsung Duos.

We would like to emphasize that the problems discussed in this paper are not hard engineering challenges, but very practical problems that emerge because of limited computation capabilities of the feature phones. As smartphones become popular in developing regions, these problems will automatically disappear. However, given the very high current penetration of feature phones in developing regions, we believe that it is important to address these issues for the benefit of those ICT4D practitioners who may want to deploy web-based audio systems in the near to mid–term.

There are several opportunities for future work on this topic. Firstly, it can be evaluated whether the proposed 2-player approach works with video content. Does switching between video players create any significant 'visual jitter' which will adversely impact the user experience? Secondly, the impact of the 2-player approach on battery life of the phone needs to be carefully evaluated. Our experiments showed that the 2-player approach resulted in significantly less total delay than the 1-player approach; therefore the 2-player application will complete the audio playback in lesser time. It may result in battery gains, but this claim requires an experimental evaluation.

As a next step, we plan to deploy our solution with a Community Radio (CR) station in India. The CR station intends to use mobile web to make its radio programs available to a broader set of listeners (many of whom use feature phones). We believe that our technique of playing and recording audio content can prove beneficial for this initiative.

6. REFERENCES

[1] Agarwal S., Jain A., Kumar A., and Rajput N. 2010. The World Wide Telecom Web browser. In *Proc.* ACM DEV '10.

[2] Ali M. and Langendoen K. 2007. TinyPC: enabling low-cost internet access in developing regions. In *Proc.* of the 2007 workshop on Networked systems for developing regions *(NSDR '07).*

[3] Audio Green. http://research.microsoft.com/en-us/um/india/projects/digitalgreen/audio.htm

[4] Black J. et al. 2009. Mobile solutions for front-line health workers in developing countries. In *Proc.* Healthcom'09. IEEE Press. 89-93.

[5] Chesterfield J. et al. Experiences with multimedia streaming over 2.5G and 3G Networks. In Proc. IEEE BroadWiM 2004.

[6] Chen J., Hutchful D., Thies W., and Subramanian L. 2011. Analyzing and accelerating web access in a school in peri-urban India. In *Proc.* WWW '11, 443-452

[7] Chen J., Subramanian L., and Li J. 2009. RuralCafe: web search in the rural developing world. In *Proc.* WWW '09. 411-420.

[8] Chu G., Satpathy S., Toyama K., Gandhi R., Balakrishnan R., and Menon S.R. 2009. Featherweight multimedia for information dissemination. In *Proc.* ICTD'09. IEEE Press. 337-347.

[9] Danis C. et al. 2010. Mobile phones for health education in the developing world: SMS as a user interface. In *Proc.* ACM DEV '10.

[10] Falling costs of data plans in India http://articles.economictimes.indiatimes.com/2012-05-23/news/31826308_1_3g-data-plan-rates-gsm-subscriber-base

[11] Ford, M. and Leinonen, T. MobilED - A Mobile Tools and Services Platform for Formal and Informal Learning. In Proc. mLearn 2006, 1-23.

[12] Gitau S., Marsden G., and Donner J. 2010. After access: challenges facing mobile-only internet users in the developing world. In *Proc.* CHI '10. 2603-2606.

[13] Jain A., Jaiswal S., Majumder A., Naidu K.V.M., Narlikar G., Shrivastava N., and Poosala V. 2009. mango: low-cost, scalable delivery of rich content on mobiles. In *Proc. 1st ACM workshop on Networking, systems, and applications for mobile handhelds* (MobiHeld '09). 73-74.

[14] Kam, M., Mathur A., Kumar A., Canny J. Designing Digital Games for Rural Children: A Study of Traditional Village Games in India. In Proc. CHI 2009, ACM Press (2009), 31-40.

[15] Launching native apps in J2ME. Nokia developer resources. http://www.developer.nokia.com/Resources/Library/Java/developers-guides/invoking-applications/invoking-applications-in-java-me.html

[16] Kang K. et al. Dynamic scheduling for scalable media transmission over CDMA2000 1xEV-DO broadcast and multicast networks. In IFIP Networking, 2005.

[17] Mobile handset shipments in India http://cmrindia.biz/india-mobile-phone-sales-cross-50-million-mark-in-jan-mar-2012-up-9-1-yoy/

[18] Mobile internet trends. KPCB. http://kpcb.com/insights/2012-internet-trends

[19] Mudliar P., Donner J., and Thies W. 2012. Emergent practices around CGNet Swara, voice forum for citizen journalism in rural India. In *Proc.* (ICTD '12), 159-168

[20] MP3 bit rates. http://computer.howstuffworks.com/mp32.htm

[21] Patel N., Chittamuru D., Jain A., Dave P., and Parikh T. 2010. Avaaj Otalo: a field study of an interactive voice forum for small farmers in rural India. In *Proc.* CHI 2010, 733-742

[22] Samdaria N., Mathur A., Balakrishnan R.: Paying in Kind for Crowdsourced Work in Developing Regions. 2012. In Proc. Pervasive 2012: 10th International Conference on Pervasive Computing.

[23] Sherwani J., Palijo S., Mirza S., Ahmed T., Ali N., and Rosenfeld R. 2009. Speech vs. touch-tone: telephony interfaces for information access by low literate users. In *Proc.* ICTD '09. IEEE Press, 447-457.

[24] Telecom stats India. Telecom Regulatory Authority of India.http://trak.in/tags/business/2011/03/08/indian-telecom-subscriber-growth-january-2011/

[25] Yin M. and Zhai S. 2006. The benefits of augmenting telephone voice menu navigation with visual browsing and search. In *Proc.* ACM CHI '06. 319-328.