International Journal of Multimedia and Ubiquitous Engineering
Vol. 9, No. 10 (2014), pp. 197-208
http://dx.doi.org/10.14257/ijmue.2014.9.10.19

Development of Android App for Smooth Multimedia Streaming Service via Portable Media File Format

Sang-Min Seo[1] and Yoon-Ho Choi[2*]

[1] Department of Computer Science, Kyonggi University, 443-760, Suwon-Si, Gyeonggi-Do, Korea
[2] Department of Convergence Security, Kyonggi University, 443-760, Suwon-Si, Gyeonggi-Do, Korea
drone1256@naver.com, ychoi@kyonggi.ac.kr

Abstract

Existing Android applications for streaming video in real time are dependent on the codec, which composes the encoding function, and the version of Android operating system. Also, for streaming video in real time, most applications should be connected with a separate desktop PC. In this paper, we propose a new application, which overcomes these disadvantages and thus, streams video in real time. Specifically, to overcome these disadvantages, the proposed application uses the flash video file format, which is the common media file format supported by various versions of Android operating system. Through experiments, we show that it is possible for the proposed application to stream video in real time while using the existing video encoding methods.

Keywords: *mobile device, android app, jni, frame buffer, streaming service*

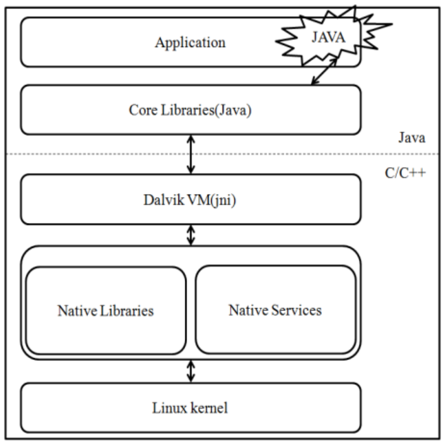
1. Introduction

Along with mobile networks, the performance improvement of mobile device is facilitating the development of mobile multimedia applications (hereafter, app), which deal with various forms of multimedia information such as sound and image data instead of general texts [1-5]. In recent years, various mobile operating systems have been developed, and various mobile devices including smart phones have also been equipped with the high computing capability, the large capacity of memory, and the high I/O speed that are similar to those of general PCs. Also, with the development of link layer technologies that enables the increase of the network bandwidth, mobile screen recording applications that enables video conferencing by using high bandwidths on the networks, video-on-demand (VOD), and recording and live streaming of video, have been commonly deployed.

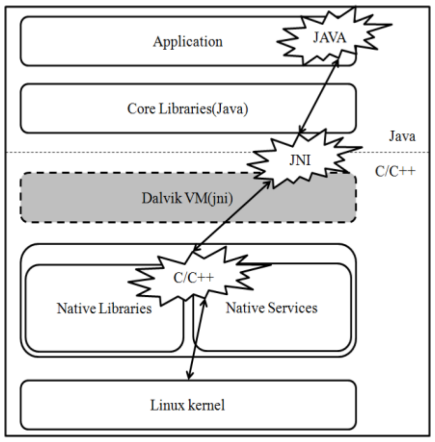
Apps for streaming video commonly consist of two functional modules: (1) one for collecting the screen information (image) and the sound information; and (2) another for transmitting the encoded image and sound via the wired and wireless networks. This is because to provide smooth multimedia streaming services, not only quick speeds of data transmission from the aspect of networks but also the high-speed process of video from the aspect of the mobile device [6-12]. However, most existing studies have focused on the encoding methods of multimedia sounds and images to improve the quality of streaming services during data transmission through the networks. As representative studies, a method

[*] Corresponding Author

that controls optimal bit ratios for encoding was suggested for static images [13] and another method that uses dependency among frames was proposed for moving images [14]. Also, the



(a) System architecture for collecting screen information by using the Android API



(b) System architecture for collecting screen information by using the jni and fb0

Figure 1. System Architectures for Collecting Screen Information from Android Operating System

other methods that classify macro blocks by considering human visual characteristics [15] and use the bit ratio-distortion model [16] were proposed.

In this paper, as shown in Figure 1, we note that two representative system structures can be used for collecting the screen information on Android operating system. When the Android application programming interface (API) in JAVA-written codes is called, the recording-related APIs are called in core libraries. The respective API is converted to .class (byte codes) and screen information is collected by using native libraries, which are provided by the Android operating systems through the Dalvik Virtual Machine (VM). Specifically, after core libraries call the Dalvik VM's java native interface (jni) to approach the frame buffer (fb), which stores screen information, the Android app collects the screen information from the path, /dev/graphics/fb0, which is the device driver. This is possible because the jni provides the interface that performs local system calls.

However, as shown in Figure 1(a), when screen information is collected by using Android's API, the image recording module undergoes various phases (JAVA→Core Libraries→Dalvik VM→Native Libraries→Native Services→Linux Kernel). Due to these multiple phases, screen information is collected at a speed of 7~8 frames per second (fps), where such a low speed causes the quality degradation of the collected images. For this reason, an efficient app implementation technique is required to encode an image for smooth streaming services.

In this paper, we propose a new streaming app, called FB-MSVR, where screen information is collected from the fb via the jni as shown in Figure 1(b). By accessing the fb via the jni, the proposed method overcomes the limitation that the quality of image when encoding it is not guaranteed in the process of app development. That is, when being compared with the case of using the Android API as shown in Figure 1(a), screen information is collected at an average speed of 23~24 fps by collecting images that are stored in the fb in the following order: JAVA→jni→C/C++ source code→frame buffer within the Linux Kernel.

To realize smooth streaming function, the FB-MSVR uses flash video (flv) file format, which is a media file format that has been provided by the lower versions of Android operating

Table 1. Summary of Codec, which is Provided by Android Operating System [17]

Type	Format / Codec	Encoder	Decoder
Audio	AAC LC	O	O
	HE-AACv1	O (ver. 4.1+)	O
	HE-AACv2	-	O
	AAC ELD	O (ver. 4.1+)	O (ver. 4.1+)
	AMR-NB	O	O
	AMR-WB	O	O
	FLAC	-	O (ver. 3.1+)
	MP3	-	O
	MIDI	-	O
	Vorbis	-	O
	PCM/WAVE	O (ver. 4.1+)	O
Image	JPEG	O	O
	GIF	-	O
	PNG	O	O
	BMP	-	O
	WebP	O (ver. 4.0+)	O (ver. 4.0+)
Video	H.263	O	O
	H.264 AVC	O (ver. 3.0+)	O
	MPEG-4 SP	-	O
	VP8	O (ver. 4.3+)	O (ver. 4.3+)
Flash		Support adobe flash 10.1 (ver. 2.2 +)	

system, and thus supports the streaming service with outstanding portability.

This paper is organized in the following way. In section 2, we analyze the characteristics of existing video streaming apps. After showing the functional architecture and operational sequences of the proposed app in section 3, we show the graphical user interface of the implemented app in section 4. In section 5, we show the evaluation results of the proposed app. Finally, we summarize this paper in section 6.

2. Related Works

2.1. Overview of Codecs Provided by Android

In Table 1, we summarize codecs provided basically by Android [17].Android supports Third-Party and Hardware Media Codecs, I/O devices, and interfaces for content policies. It supports the encoder or decoder for videos and images such as 3GPP, MPEG-4, AAC, MP3, MPEG-4, H.263, AVC, and JPEG. It also supports speech codecs including AMR-NB and AMR-WB, audio codecs including MP3, AAC, and AAC+, media recording codecs including 3GPP, MPEG-4, and JPEG, video calls based on 324-M and Adobe Flash. With the launch of Android, types of the supportable codecs are also being added. In addition, different smart

Table 2. Specifications of the Existing Apps and the Proposed FB-MSVR Higher-Version

Name	Streaming Service	
	quality (fps)	Codec
screen cast video recorder [1]	Not supported	
screen video recorder [2]	Not supported	
Z-screenrecorder [3]	Not supported	
afreecaTV [4]	23~24	H.264
mobizen [5]	Not supporting streaming	
FB-MSVR	23~24	flv

phone manufacturers provide their own additional codecs, thus different devices have different types of codecs supported. Thus, among codecs embedded in Android, the flv, which has been supported by the lower versions of Android, should be used for not being limited by smart phones and Android versions.

2.2. Characteristics of Existing Screen Recording Programs

In Table 2, we compare the streaming performance of recording apps, which are currently used at commercial Android smart phone, with that of the proposed app, FB-MSVR. On the following Android smart phone, we evaluated the performance of the existing apps and the proposed app:

- **Android device**: Galaxy Note 2
- **Development program**: eclipse juno 64bit
- **Android version**: Jelly Bean 4.1.2
- **Android NDK**: android-ndk-r9d-windows-x86_64bit
- **Tool for checking codec information (the length of images, the type of codec)**: mediainfo [20]

Among existing apps, only the screen video recorder [2] recorded the screen at an average speed of 24 fps by using the MPEG-4. However, it provided neither audio nor streaming services. The screen cast video recorder [1] and the Z-screenrecorder [3] did not properly record. Also, they had the disadvantage of not providing streaming services. In the case of AfreecaTV [4] that is most widely used for real-time streaming service, both recording and streaming functions work only in connection with a separate desktop. Also, the streaming function of AfreecaTV [4] was supported by using only H.264 codec. In the same way that

AfreecaTV provides both recording and streaming functions [4], the recording function of mobizen [5] was also provided after being connected with a separate desktop.

2.3. Characteristics of the Proposed App

The proposed app records the audio and the screen of Android smart phones by using of the codecs that are basically supported by Android. Thus, it is possible to share smart phone screens and audios with other users through the streaming function. In addition, because the

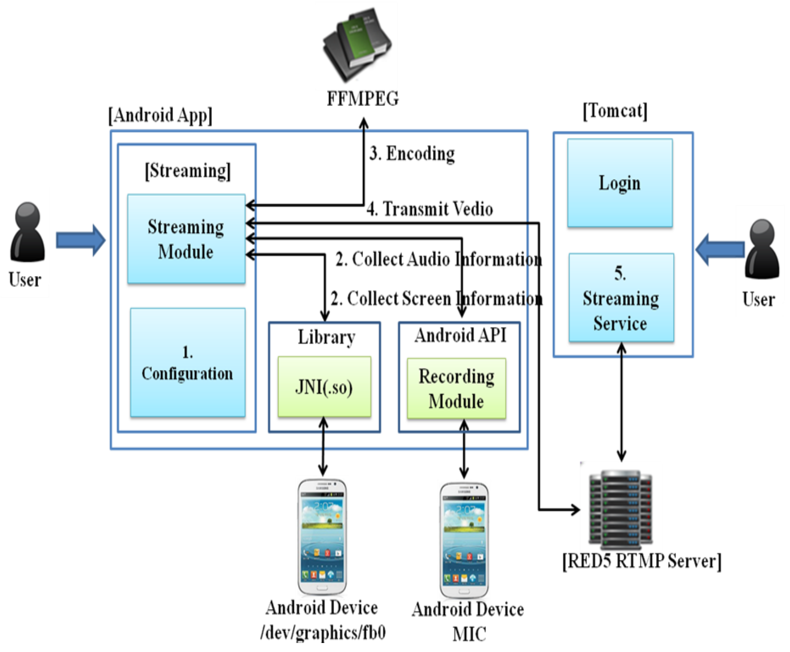


Figure 2. Architecture of the Streaming Function

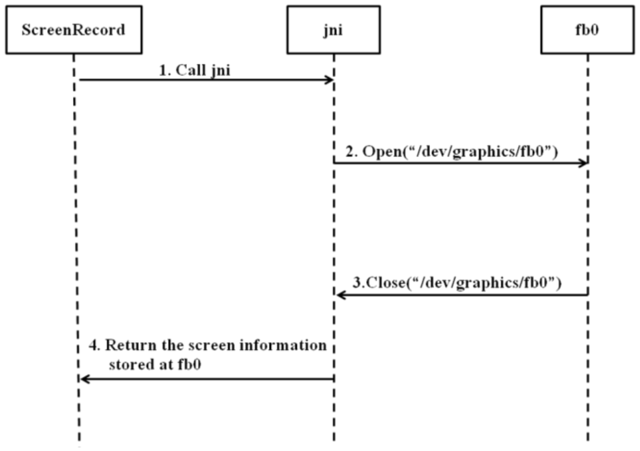

ScreenRecord
jni
fb0
1. Call jni
2. Open("/dev/graphics/fb0")
3.Close("/dev/graphics/fb0")
4. Return the screen information stored at fb0

Figure 3. Sequence Diagram for Collecting Screen Information from fb0

proposed app uses the flv file format for streaming service, it is possible to view the recorded images and the audios on various devices.

3. Architecture and Operation of the Proposed App

It is worth noting that FFMPEG open source libraries [19] support various forms of encoding and decoding for images. Thus, we implement the proposed app by using FFMPEG open source libraries when encoding images. Specifically, the proposed app encodes an image by using the MPEG-4 and ACC codecs, and transmits the encoded image by transforming it into the flv file format for streaming service. Here, we use the flv file format for keeping portability because the flv file format is supported by various versions of Android operating system. Also, because the flv file format can maintain the quality of images while reducing the size of the recorded image, the proposed app can take into account the amount of bandwidth consumption while streaming the encoded image via the wireless network, and can

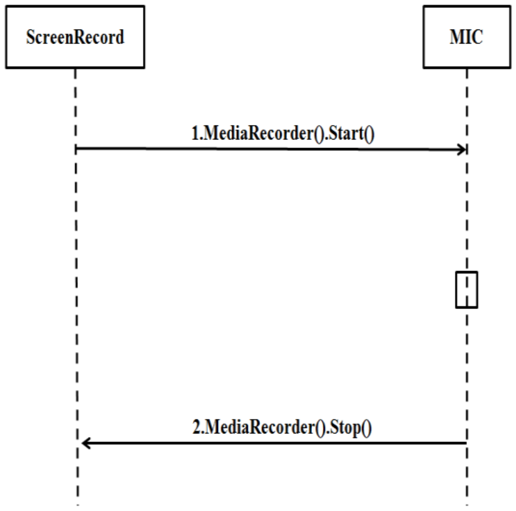


Figure 4. Sequence Diagram for Collecting Sound Information

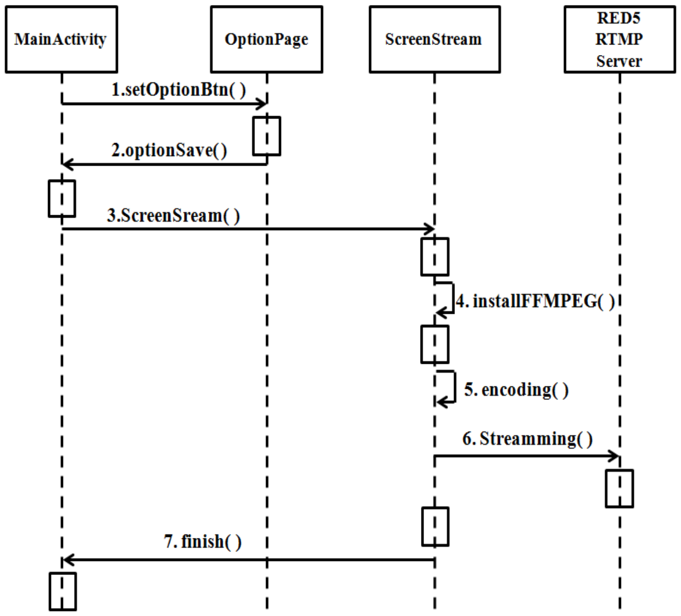


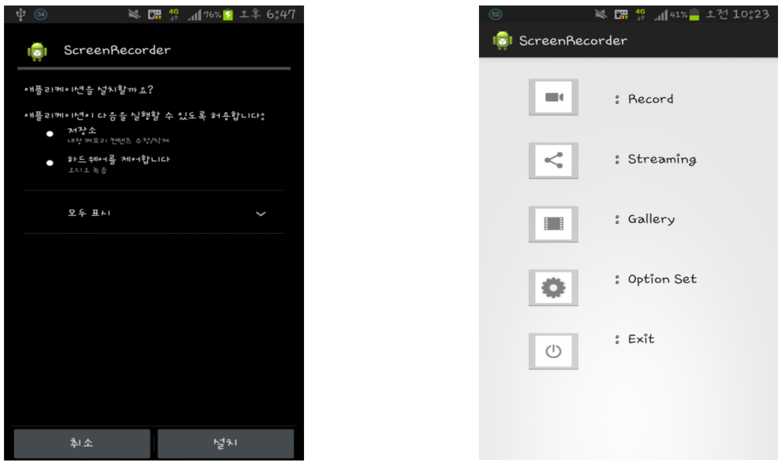
Figure 5. Sequence Diagram for the Streaming Function

be used effectively not only in the Android environment but also in the Web. In the following subsections, we show the overall architecture and operation of the streaming function.

3.1. Overall Structure of the Streaming Function

To broadcast the current screen image and audio on the smart phone properly, the proposed app consists of the following modules. After streaming module collects screen information and audio information by using the jni and recording module, it encodes the screen image in the form of video in the fb0 and transmits the video file to the Red5 RTMP server [19], and then streaming service can be provided to users via the Tomcat web server in real time. As shown in Figure 2, the streaming function operates based on collaboration among these modules in the following way.

3.2. Sequence Diagrams for Supporting the Streaming Function



(a) Screen when install App **(b) Main screen**

Figure 6. The Developed App's Installation and Environment Setting Screens

3.2.1. Sequence Diagram for Collecting Screen Information: To collect screen information on Android devices, the frame buffer should be accessed from the application layer. Types of the frame buffer include fb0 and fb1: fb0 for storing the current screen information and fb1 for storing the future screen information. However, it is impossible to access the frame buffers from the application layer directly. Thus, by following the sequence in Figure 3, the proposed app collects screen information from fb0.

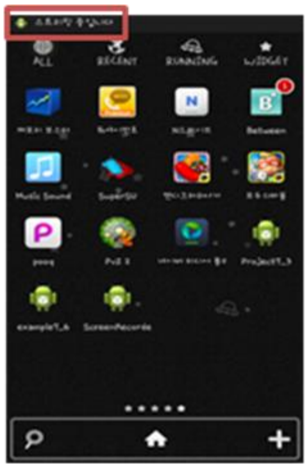
1. From the Android source code, which is written in JAVA language, the jni is called for collecting the screen information stored at the fb0.
2. Because the jni can access to /dev/graphics/fb0, the screen information at the fb0 is opened via Open("dev/graphics/fb0").
3. Close the fb0 by using Close("/dev/graphics/fb0").
4. Screen information, which is brought from the fb0, is collected at the recording module.

3.2.2. Sequence Diagram for Collecting Sound Information: As shown in Figure 4, sound information is collected in the following way.

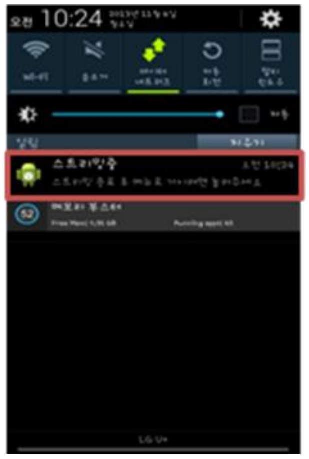
1. Sound information from the device's microphone (MIC) is collected by calling MediaRecorder().Start().
2. Stop collecting the sound information by calling MediaRecorder().Stop().

3.2.3. Sequence Diagram for the Streaming Function: As shown in Figure 5, the streaming function operates in the following way.

1. After moving to the OptionPage by calling setOptionBtn(), set the environment for recording.
2. Save values for environment setting by calling optionSave().

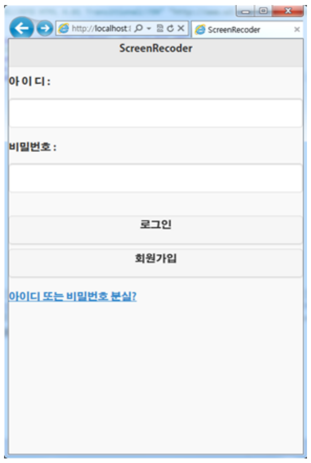


(a) Screen while streaming video

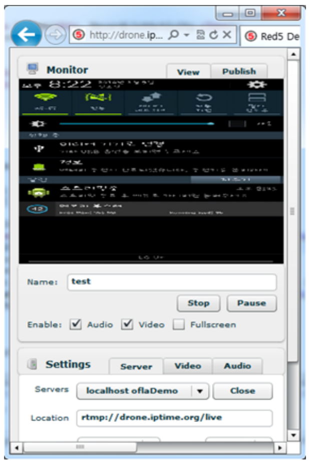


(b) Screen for showing in-progress service status

Figure 7. Screens when Streaming Video



(a) Screen for Web server login



Name: test
Enable: ☑ Audio ☑ Video ☐ Fullscreen
Servers: localhost oflaDemo
Location: rtmp://drone.iptime.org/live

(b) Screen while watching the video

Figure 8. Screens for Watching Streaming Video at the Web Server

3. By calling ScreenRecord(), the proposed app operates in the background mode.
4. FFMPEG open source libraries are installed by calling installFFMPEG().
5. By calling encoding() without calling Recording() as shown in Fig.7, screen information in the fb0 is collected, and sound information is also collected from the device's microphone by using the basic Android audio API. Here, the collected information is encoded in the format of flv through FFMPEG open source libraries.
6. The video stream is transmitted to the Red 5 RTMP server via the wireless network by calling Streaming().

7. After finish() is called, the streaming service that has been operating in the background is terminated and completed.

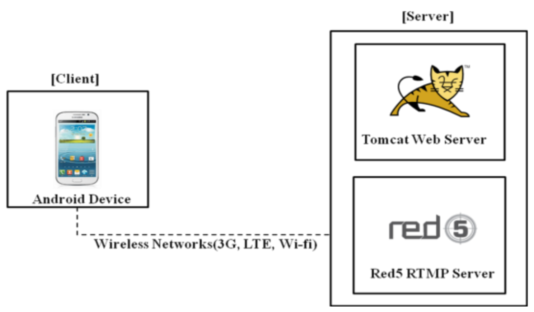


Figure 9. Organization of the Experimental Environment

4. Graphic User Interfaces

Figure 6 shows the installation (Figure 6(a)) and main (Figure 6(b)) screens of the proposed app. While installing the proposed app, it is required to allow the proposed app access the fb0 and the microphone. Thus, while installing the app, the root user's right should be accepted by the Android operating system.

Figure 7 shows the screens during streaming (Figure 7(a)) and displaying the progress of streaming (Figure 7(b)). While the app operates in the background, the status bar on the upper screen in Figure 7(a) shows alarm messages such as "Streaming is on (red box, "스트리밍 중입니다" in Korean, in Figure 7(a))". For terminating streaming service, if you drag down the status bar and then, click the notification "Streaming (red box, "스트리밍중" in Korean, in Figure 7(b))", the streaming service is terminated, and the control of the program returns to the main screen.

In Figure 8, we show the screens for the Web server log-in (Figure 8(a)) and watching streaming video (Figure 8(b)). If you access to the Web page for watching streaming service, the login screen is shown. After a user successes to login to the Web server, the login screen changes into the streaming service page, which is provided by the Red 5 Server, and the user can watch the streaming video.

5. Performance Evaluation

5.1. Experimental Environment

As shown in Figure 9, we organized the experimental environment, and the experiment was performed to compare the speed of obtaining video information by using the existing API as shown in Figure 1(a) and the jni as shown in Figure 1(b), respectively. For client side, we used the Android smart phone, whose specification was shown in section 2.2, and for the server side, we configure it as follows.

- **Server OS**: Windows 7 64bit Ultimate
- **Web Server**: Tomcat 7.0
- **Streaming Server**: Red5 1.0.1

Table 3. Average fps for Streaming Video Files by Using the API and the Proposed App

# of trials	1	2	3	4	5	Average (fps)
API	8.1	7.4	7.3	8.9	8.2	7.8
FB-MSVR	24.3	23.4	24.2	22.9	24.8	23.6

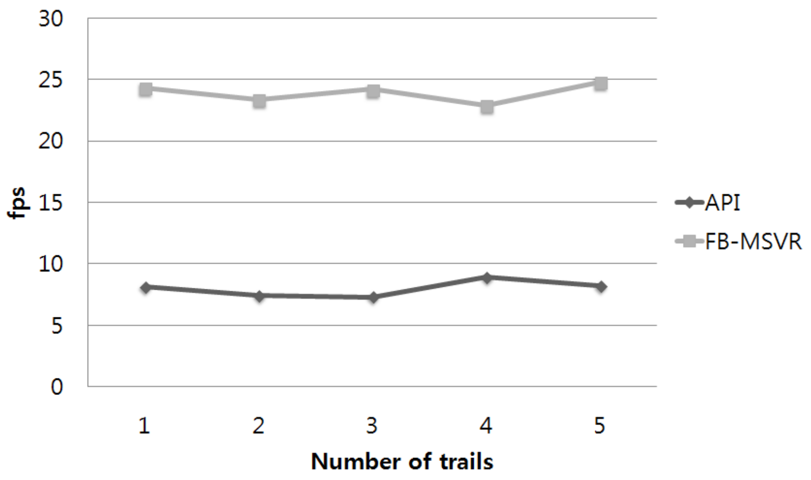


Figure 10. Average fps for Streaming Video Files by Using the API and the Proposed App

5.2. Evaluation Results

To compare the streaming speed between the existing API and the jni, we measured fps for 10 seconds at each trial in an average. In Figure 10, we show the evaluation results after measuring the speed of obtaining screen information by using the Android API and accessing fb0 in the native domain via the jni together with the NDK. In Table 3, we also summarize the results in a numerical manner. The evaluation results show that the average fps is 7.8 fps when using the Android API and 23.6 fps when using the jni. That is, it is shown that while streaming service provided by the Android API is not fully met the ream time streaming service, and thus the results of encoding showed video with less smoothness, streaming service provided by the jni met over 20 frames per second, and thus provides smooth video.

6. Conclusion

The existing Android screen recording app is dependent on the system specifications of the recording function and should provide services by connecting with a separate desktop PC in a wired manner. Thus, this study suggested an efficient method of proving the real-time streaming service of video file by using flash video files and then, its performance was verified in terms of fps during stream service. From the experimental evaluation results, it was shown that the proposed app can encode the images smooth by as much as 20 fps without changing the encoding method. After the further improvement of the streaming service, we expect that many users can watch the streaming video via Android devices.

Acknowledgments

This work was supported by the Industrial Strategic Technology Development Program (10047541, Development of Self-Defending and Auto-Scaling SDN Smart Security Networking System) funded by the Ministry of Knowledge Economy(MKE, Korea)".

References

[1] Screen Cast video recorder. https://play.google.com/store/apps/details?id=com.ms.screencastfree
[2] Screen video recorder v3.0. https://play.google.com/store/apps/details?id=com.cocoapps.screenrecorder
[3] Z-screenrecorder. https://play.google.com/store/apps/details?id=com.zausan.zscreenrecorder
[4] AfreecaTV. https://play.google.com/store/apps/details?id=kr.co.nowcom.mobile.afreeca
[5] Mobizen. http://www.mobizen.com/
[6] H. Kim, J. Koo, and K. Chung, "Equation-based Quality Control Scheme for Improving QoE of Multimedia Streaming Service", Proceeding of the KIISE Fall Conference, (**2010**) November Korea (in Korean).
[7] J. Koo and K. Chung, "A Novel Rate Control for Improving the QoE of Multimedia Streaming Service in the Internet Congestion", Journal of KIISE: Information Networking, vol. 6, no. 36, (**2009**) (in Korean).
[8] B. Wang, W. Wei, Z. Guo, and D. Towsley, "Multipath Live Streaming via TCP: Scheme, Performance and Benefits", Proceeding of the ACM CoNEXT, (**2007**) December.
[9] D. Nguyen and J. Ostermann, "Congestion Control for Scalable Video Streaming Using the Scalability Extension of H.264/AVC", IEEE Journal of Selected Topics in Signal Processing, vol. 2, no. 2, (**2007**).
[10] B. Libaek and O. Kure, "Generic Application Level Rate Control for Scalable Video Using Shadow Probing", Proceeding of the ICSNC, (**2009**).
[11] R. Rejaie, M. Handley, and D. Estrin, "Layered Quality Adaptation for Internet Video Streaming", IEEE Journal of Selected Areas in Communications, vol. 12, no. 18, (**2000**).
[12] M. Zink, J. Schmitte, and R. Steinmetz, "Layer-Encoded Video in Scalable Adaptive Streaming", IEEE Transaction on Multimedia, vol. 1, no. 7, (**2005**).
[13] A. Ortega, K. Ramchandran, and M. Vetterli, "Optimal trellis-based buffered compression and fast approximations", IEEE Transaction on Image Processing, vol. 1, no. 3, (**1994**).
[14] K. Ramchandran, A. Ortega, and M. Vetterli, "Bit allocation for dependent quantization with applications to multi-resolution and MPEG video coder", IEEE Transaction on Image Processing, vol. 5, no. 3, (**1994**).
[15] ISO-IEC/JTC1/SC29/WG11, Test Model 5, (**1993**).
[16] J. Zdepsky, D.Raychaudhuri, and K. Joseph, "Statistically based buffer control policies for constant rate transmission of compressed digital video. IEEE Transaction on Communications", vol. 6, no. 39, (**1991**).
[17] Embedded Android Codec. http://developer.android.com/guide/appendix/media-formats.html
[18] FFMPEG. http://ffmpeg.mplayerhq.hu/
[19] Red 5. http://www.red5.org/
[20] Mediainfo. http://mediaarea.net/ko/MediaInfo
[21] S. Choi, S. Lee, Y. Lee, C. Kim and T. Jeong, "Secure Video Transmission on Smart Phones for Mobile Intelligent Network", IJSIA, vol. 1, no. 1, (**2013**).
[22] Y.-S. Park, K.-. Kim, J. Ahn, C.-S. Kim and J.-C. Ryou, "A Real Time Voice Transmission Method for Voice Privacy between CDMA Mobile and PSTN Terminal", IJSIA, vol. 2, no. 1, (**2007**).
[23] J. Veijalainen, "Autonomy, Heterogeneity, Trust, Security, and Privacy in Mobile P2P Environments", IJSIA vol. 1, no.1, (**2007**).

Authors

Sang-min Seo graduated from Yong-In high school at 2008. He is currently in the department of computer science, Kyonggi University, Suwon, Korea. He is specialized at JAVA programming, Android programming, Embedded System and so on.

Yoon-Ho Choi is an assistant professor at department of convergence security in Kyonggi University, Suwon, Korea. He received the M.S. and Ph.D. degrees from school of electrical and computer engineering, Seoul National University, S. Korea, in Aug. 2004 and Aug. 2008, respectively. He was a postdoctoral scholar in Seoul National University from Sep. 2008 to Dec. 2008 and in Pennsylvania State University, University Park, PA, USA, from Jan. 2009 to Dec. 2009. He has served as TPC members in various international conferences and journals. His research interests include Deep Packet Inspection (DPI) for high-speed intrusion prevention, mobile computing security, vehicular network security, big data analysis and so on.