# An Efficient Implementation of Key Frame Extraction and Sharing in Android for Wireless Video Sensor Network

**Kang-Wook Kim**
Mobile Communication Division of Samsung Electronics Co., Ltd
129, Samsung-ro Yeongtong-gu, Suwon-si, Gyeonggi-do 443-742, Korea
[e-mail: ekans999@gmail.com]

*Received September 15, 2014; revised December 19, 2014; revised February 8, 2015; accepted March 4, 2015; published September 30, 2015*

## *Abstract*

Wireless sensor network is an important research topic that has attracted a lot of attention in recent years. However, most of the interest has focused on wireless sensor network to gather scalar data such as temperature, humidity and vibration. Scalar data are insufficient for diverse applications such as video surveillance, target recognition and traffic monitoring. However, if we use camera sensors in wireless sensor network to collect video data which are vast in information, they can provide important visual information. Video sensor networks continue to gain interest due to their ability to collect video information for a wide range of applications in the past few years. However, how to efficiently store the massive data that reflect environmental state of different times in video sensor network and how to quickly search interested information from them are challenging issues in current research, especially when the sensor network environment is complicated. Therefore, in this paper, we propose a fast algorithm for extracting key frames from video and describe the design and implementation of key frame extraction and sharing in Android for wireless video sensor network.

***Keywords:*** Wireless Video Sensor Network, Key frame, Android

**http://dx.doi.org/10.3837/tiis.2015.09.005** **ISSN : 1976-7277**

## 1. Introduction

With the rapid development of wireless communication technique, embedded technology and sensor technology, micro sensors with perception ability, computing ability as well as communication functionality begin to appear, which have attracted a lot of attention [1]. This kind of sensor network can harmoniously perceive, collect and process various environmental or monitoring objects information in network coverage area, which can be transmitted to people who need them. Sensor network has dramatically changed the way people interact with physical the world. They are deployed in a physical field to perform tasks from collecting information such as temperatures and real-time video data. Therefore, sensor networks can be widely applied to a large variety of fields such as the military operations, security surveillance, agriculture control, health care, and environmental monitoring, etc. The aspect of sensor network research is how to realize the collection, transfer and processing of simple environmental data such as temperature, vibration, humidity, and intensity etc. on micro senor nodes whose energy usages are very limited [2]. However, the monitoring environment is becoming more and more complicated. The simple scalar data which are acquired by traditional sensor network cannot satisfy people's whole demand to environmental monitoring. There is a strong need to introduce multimedia like images and videos of rich information to environmental monitoring activity based on sensor network. Therefore, there have been considerable works reported on the research of video sensor network. In [3], Obraczka et al. proposed the concept of video sensor networks and studied the management of information flow. In 2003, Rob Hofman et al. used video sensor networks to achieve coastal environmental monitoring [4].

Wireless video sensor network (WVSN) introduces multimedia information, including image and video into traditional sensor network, which provides rich information support to fine-grained, comprehensive and accurate environmental monitoring. It focuses on video, images and other environmental information collection, processing and transmission. The introduction of intuitive, rich visual information makes monitoring and sensing tasks more intelligent. Different from traditional wireless sensor network, WVSN utilizes image and video as front-end sensing signals. They can obtain more abundant information in the environment and provide a better basis for the backend sensing systems. But, a large amount of data will be processed and this requires time-consuming tasks. Therefore, how to efficiently store the massive data that reflect environment state of different times in a video sensor network and how to quickly search interested information from it are important parts of current studies. Therefore, it is one of the most challenging tasks to summarize and represent the content of a video, especially when the sensor network environment is complicated. There are a lot of difficulties in implementing an application based on Android such as target tracking and video surveillance. The effective extraction of key frames from a video sensor node is an essential task for summarizing and representing the content of a video [5]. Accordingly, we propose a fast algorithm for key frame extraction and describe the implementation of the algorithm in Android.

The remainder of this paper is organized as follows: In section 2, we briefly review the reference architecture and service scenario of WVSN using smartphones and a fast algorithm for key frame extraction that can be used for WVSN. Then, in section 3, we introduce the basic architecture and application framework of Android operating system and present detailed description of main structure of Android applications and the methods of developing

applications based on Android application framework. Section 4, 5 and 6 give details of software implementation process of the key frame extraction and sharing application in Android, demonstrating the class view and UI structure of the proposed method. Experimental results on various video sequences are presented in section 7, demonstrating the performance and validity of the proposed method. Finally, section 8 concludes the paper.

## 2. Related Work

### 2.1 Architecture of WVSN Based on Android Phone

The development in microelectronics, low-power multi-functional sensors, embedded OS technology has caused the progress in wireless video sensor network (WVSN) which is a network of wirelessly interconnected devices that are able to share and retrieve multimedia contents such as still images, video and audio data and scalar sensor data from the environment. For this reason, ubiquitous sensor network (USN) becomes more and more important especially for video data recorded by camera in a sensor node. Currently, the status of USN focuses on developing a technology for sensor node implementation and a protocol for efficient communication and inter-working with existing network environment. However, how to integrate effectively large amounts of data collected becomes an important part of study of WVSN because of the power and bandwidth requirements in video processing. To extract key information from video is one of the most challenging tasks in video sensor network, especially when the sensor network environment is complicated. There are a lot of difficulties in implementing a key frame extraction application based on Android such as target tracking, video surveillance, video retrieval and indexing.

In **Fig. 1**, we introduce the reference architecture for WVSN. Our WVSN system is composed of video processing nodes and a base station node. Here, video processing nodes can be Android smartphones which are sharing data with each other by using WIFI, NFC. Smartphone provides various sensor modules to get environmental information for USN applications, such as temperature sensor, humidity senor, light sensor, geomagnetic sensor, gesture sensor and infrared sensor, etc. Here, video sensor's data can be collected to a base node over wireless communication such as WiFi or 3G/4G mobile network. The mobile phones with camera have functionality to collect, transfer and control video and image data. Normally, it has an ARM processor and WiFi, Bluetooth, Near Field Communication (NFC) as external interface to communicate with sensor nodes. Mostly, Android smart phones support Near Field Communication. The Android SDK (Software Development Kit) provides an NFC API that can be used to develop NFC applications that conduct peer-to-peer (P2P) data exchange. Similarly, both WiFi direct and Bluetooth can be also available.

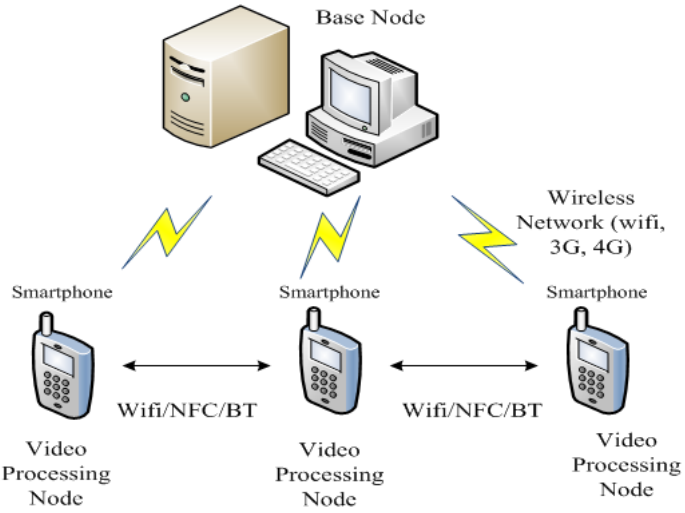



**Fig. 1.** Block diagram of WVSN using smartphone and example service scenario

In order to efficiently transmit and share the massive video data that reflect environmental state of different times in video sensor network, the key frame extraction technique should be integrated into the smart phone which takes a role of video sensor node. In section 2.2, the concept of optimization-based approach to key frame extraction for WVSN and the proposed method are presented in detail.

## 2.2 Key Frame Extraction Algorithm Based on Measuring DC Image Activity

DC images are spatially reduced versions of original images. Such spatially reduced images, once extracted, can be used for other applications beyond scene change detection, for example, the efficient comparison of video shots, automatic generation of compact documents, and nonlinear video browsing applications. Several fast algorithms to extract DC images from an MPEG compressed video using discrete cosine transform (DCT) DC coefficients in I type frame and motion compensated DCT DC coefficients in P or B type frames have already been proposed. It has been demonstrated that even at such a low resolution, global image features useful for specific classes for content-based operations on MPEG compressed video streams are well preserved [6]. After extracting the DC images from an MPEG compressed video, the next step is to detect the cuts, i.e., shot boundaries to segment the video into individual shots. To minimize the influence of non-relevant temporal variations, global frame visual features such as color and intensity histograms are used to detect a shot boundary. We adapt a sum of absolute difference between consecutive DC images and define a content variation function $CVF(k)$ for describing the relevant difference between frames $k$ and $k-1$ as:

$$CVF(k) = \sum_{i}^{M} \sum_{j}^{N} \left| I_{DC}^{k}(i,j) - I_{DC}^{k-1}(i,j) \right| \tag{1}$$

, where $k$ is the frame index, and $I_{DC}^{k}(i, j)$ means the pixel value at position $(i, j)$ in the $M \times N$ DC image. $CVF(k)$ measures the relative changes between two consecutive frames, thereby indicating the magnitude of any changes. A $CVF(k)$ curve and a sliding window are used to detect the cuts. The method of using a sliding window is to examine a few successive frame differences. Here, a scene change frame from frame $k-1$ to frame $k$ is declared if $CVF(k)$ is maximum within a sliding window. After the entire video sequence has been segmented into shots using the above mentioned method, the next step is to assign the appropriate number of key frames. A single key frame is very often unable to provide sufficient information about the video content of a given shot, especially for shots with a long duration. Moreover, important shots with a small duration may have no key frames, while shots with a longer duration can be represented by multiple frames with a similar content. We propose a simple intuitively appealing algorithm for allocating the number of key frames for each shot. This algorithm may not be optimal, but it allocates key frames to shots incrementally, one key frame at a time, in a way that yields a good assignment.

The basic idea is that in each of a total of $K_T$ key frames, one key frame is allocated where it will do the most good at this point. Let $M_i(K)$, called the content function, denote the content of the $i$ th shot for the key frame allocation of $K_i$ key frames. The content function of each shot is defined by

$$M_i(K_i) = g_i ACVF_i(L) 2^{-2(K_i-1)} \tag{2}$$

, where $ACVF_i(n)$ is the accumulated value of $CVF(k)$ from the beginning up to the final summation position $n$ and $g_i = g$ is a constant independent of $i$ for simplicity. $ACVF_i(n)$ can be calculated as follows:

$$ACVF_i(n) = \sum_{k=1}^{n} CVF(k) \tag{3}$$

, where $i$, $k$ are the shot and frame index, respectively. If the summation of eqn. (1) stretches through the entire frame within a shot, the total magnitude of temporal flow fluctuation in the shot is obtained which represents the content of the shot. In $ACVF_i(L)$ of eqn. (2), $L$ is the number of frames in the shot. Let $K_i(m)$ denote the total number of key frames allocated to the $i$ th shot after iteration $m$, i.e., after $m$ key frames have been allocated to the shots. Now the request $Q_i(m)$ associated with the $i$ th shot after the $m$ th iteration of the allocation algorithm can be defined according to:

$$Q_i(m) = M_i(K_i(m)) \tag{4}$$

That is, the request function $Q_i(m)$ after the $m$ th key frame has been assigned is simply the content of the $i$ th shot as regards its current key frames. The proposed algorithm assigns $K_i$ key frames to shot $i$ as below.

**Step 0**. Initialize the key frame allocation to one, so that $K_i(0)=1$ for each $i$ th shot and $m=0$. Set $Q_i(0)=M_i(K_i(0))$ as the initial values of request. (The reason for $K_i(0)=1$ is that at least one key frame must be allocated to each shot)

**Step 1**. Find the shot index $j$ with the maximum request.

**Step 2**. Set $K_j(m+1)=K_j(m)+1$, and set $K_i(m+1)=K_i(m)$ for each $i \neq j$, then set $Q_i(m+1)=M_i(K_i(m+1))$

**Step 3**. If $m<K_T-T-1$, increment $m$ by 1 and go to step 1. Otherwise stop.

$T$ is the number of shots in the entire sequence. This algorithm carries out a very simple and intuitive idea. That is, simply give away key frames to the neediest shot, one key frame at a time until you run out of key frames to give. The degree of neediness of each shot is measured based on the content it will yield if it were to operate with its current key frame assignment. By spreading the given maximal number of key frames $K_T$ along the entire video sequence, each shot of the sequence gets assigned a fraction of the given $K_T$ key frames according to its share of the content relative to the total content of the sequence. After assigning a certain number of key frames to each video shot, the next step is to find locations for these key frames within a shot so that they approximately contain the entire temporal information of a shot. Here, $l_u(u=1,\cdots,K_i)$ are the temporal locations of the key frames, while $n_{u-1}$ and $n_u$ are the breakpoints between the shot segments represented by key frame $l_u$. Notice that $n_0$ and $n_{K_i}$ are the known temporal beginning and end points of the $i$ th shot. The basic idea can be seen in **Fig. 2** with $K_i$ assigned key frame.

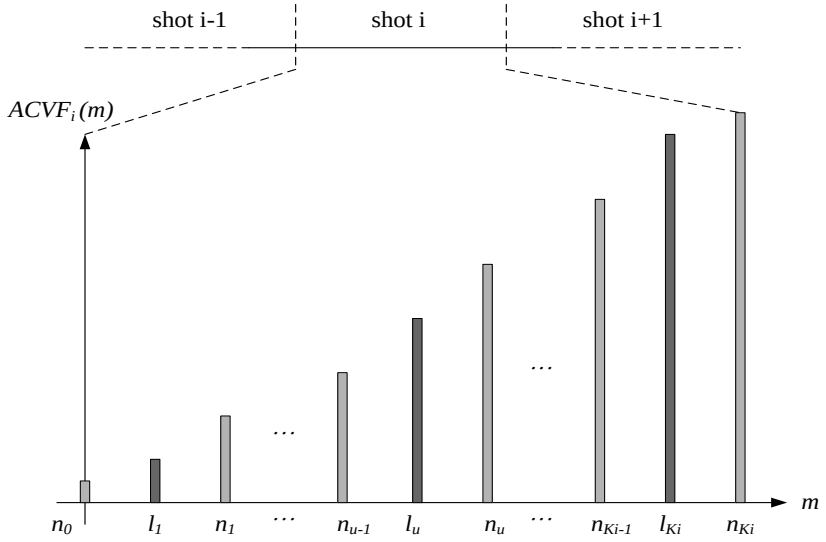



**Fig. 2.** Key frame distribution within $i$ th shot using assigned $K_i$ key frames

In order to locate the position of $l_u(u=1,\cdots,K_i)$, we propose a fast and effective which uses a probabilistic approach to locate the optimal position of the key frames. First, the normalized

$CAF_i(m)$ $(= NCAF_i(m))$ is calculated for the $i$ th shot, which is assumed to be composed of $n_{K_i} - n_0 + 1$ frames between frame $n_0$ and $n_{K_i}$. $NCAF_i(m)$ is computed as follows:

$$NCAF_i(m) = Int\left[ n_0 + (n_{K_i} - n_0) \cdot \frac{CAF_i(m) - CAF_i(n_0)}{CAF_i(n_{K_i}) - CAF_i(n_0)} + 0.5 \right], m = n_0, \cdots, n_{K_i} \quad (5)$$

, where $Int[x]$ represents the integer part of $x$. Using Eq. (3), the discrete $CAF_i(m)$ values that are not interpolated are normalized into integer values lying between the interval $[n_0, n_{K_i}]$. Next, the histogram $H(m)$ of $NCAF_i(m)$ is calculated, then the pmf (probability mass function) $P(m)$ and cdf (cumulative density function) $F(m)$ can be obtained from $H(m)$ using the following relations:

$$P(m) = \frac{H(m)}{n_{K_i} - n_0 + 1},$$

$$F(m) = \sum_{\alpha=n_0}^{m} P(\alpha), \; m = n_0, \cdots, n_{K_i} \quad (6)$$

The pmf $P(m)$ is referred to as the probability of change in the shot content. Consequently, only $H(m)$, $P(m)$, and $F(m)$ need to be calculated before distributing the key frames. The remaining key frame distribution procedure is performed by first computing the value $q_u$ such that $F(q_u) = u / K_i$ then finding $n_u = x$ such that $NCAF(x) = q_u$ for $u = 1, \cdots, K_i$. From the above computed $n_u$, the key frame positions can be easily decided sequentially as follows:

$$l_u = \frac{n_u + n_{u-1}}{2}, \; u = 1, \cdots, K_i \quad (7)$$

,where $n_0$ and $n_{K_i}$ are the known temporal beginning and end points of the $i$ th shot.

This procedure of distributing $K_i$ key frames over the $i$ th shot is very simple and fast. In addition, the proposed method does not require any recursive computations and is performed sequentially. It is intended that the given key frames are distributed over the shot according to the probability of a change in the shot content. **Fig. 3** illustrates a summary of the steps involved in the proposed algorithm.

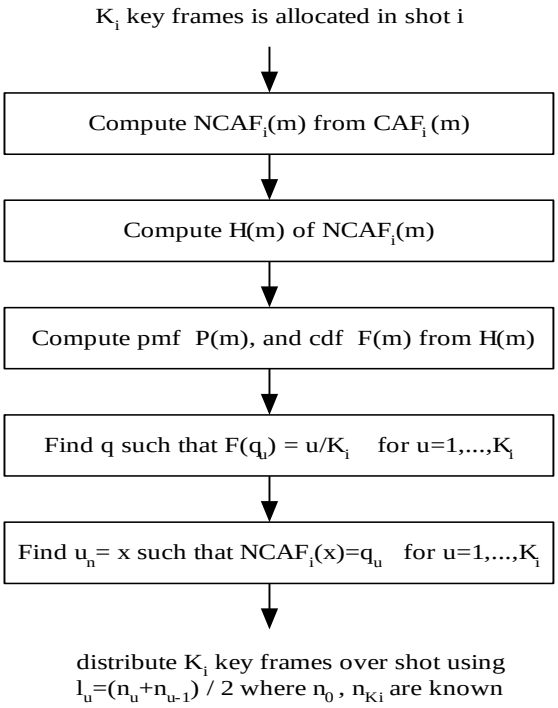



**Fig. 3.** Flow chart of proposed key frame distribution algorithm

## 3. Architecture of Android OS

Android is a kind of open source OS [7] launched by Google and OHA (Open Handset Alliance) for mobile devices on November 12, 2007. The architecture of Android consists of five main components from the top down is divided into four layers, including Application, Application Framework, Libraries, Android Runtime and Linux Kernel as shown in Fig. 4. Application programs are developed with Java programming language, thus, it is very easy to install the same application program in different hardware devices. The application framework released as the Android SDK provides high-level Java interfaces for accessing the hardware resources, such as camera, WiFi and Bluetooth. For instance, our video processing application uses the activity manager to detect and respond to events when it is triggered by other applications.

### 3.1 Application

On the top level of the framework is the Application layer. The application layer accommodates many built-in applications packages, which contains most of Google's applications, for example, clock, calendar, SMS program, email client, maps, web browser, phone app, etc., as well as custom-developed applications downloaded from internet or installed via an SDcard device. All applications are written using the Java programming language.

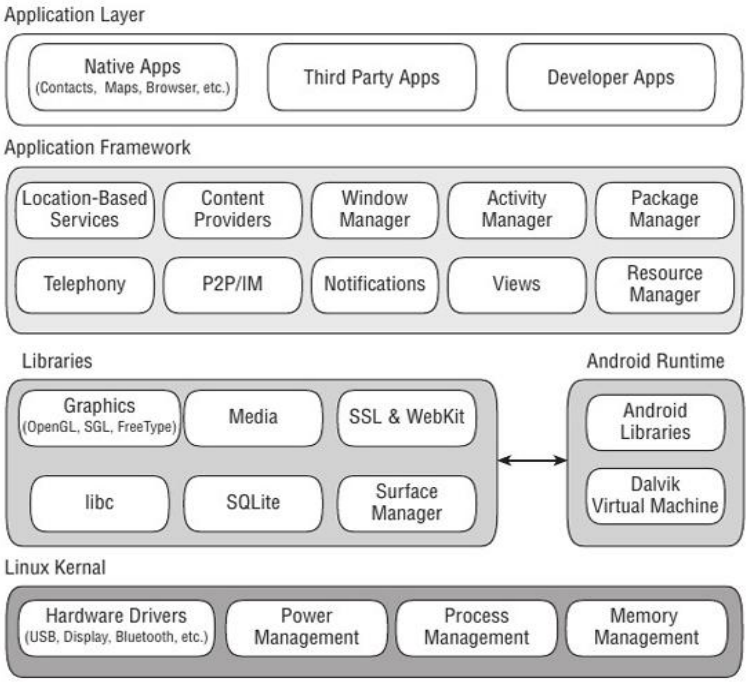



**Fig. 4.** Android architecture

### 3.2 Application Framework

The application framework provides many programming interfaces. By using the functions of these interfaces, programmers can easily design applications, thus simplifying processes and re-utilizing resources. The main components of application framework are as follows: the activity manager, the window manager, content providers, the view system, the notification manager, the package manager, the telephony manager, the resource manager, and the location manager. The application framework layer provides a development platform for developers, which facilitates the reuse and replacement of components and the building of all kinds of applications.

### 3.3 Libraries

Android includes many C and C++ libraries, connecting with the upper application framework, and with the lower operating system core, which are called by applications. The main core libraries are a System C library, a multimedia library based on OpenCore, WebKit library, a network library, a database library, an OpenGL ES 3D library, and a font library, etc.

### 3.4 Android Runtime

Android is running on the Linux kernel and its applications are written by Java programming language but Android doesn't provide J2ME to run Java programs, it uses its own Android runtime. Android runtime includes a set of core libraries and a Dalvik virtual machine (VM) that have been redesigned and optimized for the hardware features of mobile devices. Dalvik VM uses Linux kernel for underlying functionality such as threading and low-level memory

management. Every Android application runs in its own process, with its own instance of the Dalvik virtual machine so that a device can run multiple VMs efficiently. Dalvik VM is a register-based VM, executing files in the .dex (Dalvik Executable) format.

### 3.5 Linux Kernel

The Linux 2.6 version-based Android operating system provides the core system services such as security, memory management, thread management, network protocol stack, and driver model. The Linux core is an abstraction layer between the applications and the hardware, which manages all hardware resources. So, developers don't need to consider the hardware.

## 4. Design and Implementation of Key Frame Extraction Engine

The functions packaged in the form of library (DLL, SO) or executable file, such as Assembly, C and C++ can be called on Java layer through JNI (Java Native Interface) in Android. JNI comes from the following reasons: First, the application has to use the system-related functions, while Java does not support or is hard to implement. Second, there are many useful libraries written in other languages. Java programs can reuse them. Third, for higher performance issues, the developer has to use assembly or C/C++ code to implement some specific program modules [8][9]. For these requirements, Android platform supports the JNI method. In this paper, we use JNI because of the second reason, for reusing already implemented C codes for key frame extraction. JNI layer is exchanging the key frame data between Application UI and MPEG decoder library. Meanwhile, it provides the interface for controlling the DC images decoding.

Normally, native C code executes faster than Java code [10]. In view of the efficiency requirements of key frame extraction application, and the characteristics of Android hierarchy, the DC image extraction engine is located between Linux kernel layer and applications layer and realized by C/C++ programming language. In the DC image extraction engine, the function of Linux kernel and libraries are called to decode DC image from video stream and calculate the differences between DC images. Functions in Android application layer call the service provided by DC image extraction engine using JNI interface. The architecture of key frame extraction is designed into four layers as shown in **Fig. 5** In Android, applications are developed with Java programming language based on Android SDK, but key frame extraction engine is based on C programming language. In this paper, we develop dynamic linked library based C programming language (.so) by JNI, and then pack the ".so" file and the Java application as a ".apk" file by Android NDK. The advantage of this approach is we can upgrade and reuse each layer because only changing the common library allows us to develop new applications.

Combining hierarchical and modular design, the key frame extraction engine consists of mainly four layers, including the user interface, scene change detection, key frame allocation, and key frame distribution. This type of design approach can simplify video information processing. It is useful to develop and maintain video processing application using the key frame extraction engine. It is also easy to add a new functionality to our proposed design scheme. There is a mapping table between native functions and Android Java functions, which is registered to Dalvik VM.

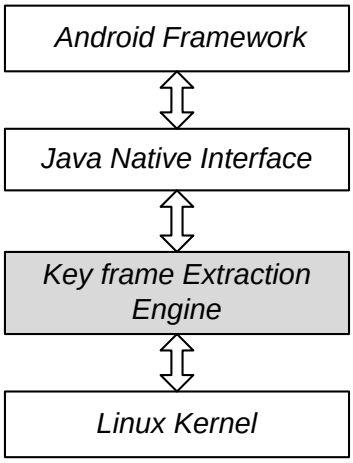



**Fig. 5**. Structure of Android key frame extraction application

The user interface layer is the interface of key frame extraction engine, and it is a JNI interface package of key frame extraction engine. Java applications can call the corresponding key frame extraction engine functions through JNI interface. The layer controls the flow of command from one layer to another. Related APIs implement the control of key frame number, widow size setting for scene change detection, and view options for display on screen. The main flow of the functional call is shown in **Fig. 6**. Every step's function is as follows:

- *mf_create_DCImg( )* : decode DC image from compressed video
- *mf_shot_detector( )* : segment video into shots using extracted DC images
- *mf_kframe_num_allocator( )* : allocate the number of key frames to a shot
- *mf_kframe_pos_locator( )* : **locate the position of key frames** over a shot

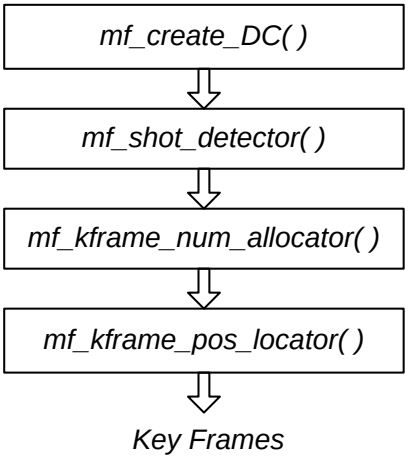



**Fig. 6**. The flow chart of key frame extraction function

## 5. Design and Implementation of Key Frame Sharing Between Node Devices

The Chord protocol [11] is one solution for connecting the peers of a P2P network. Chord can simplify the design of P2P and applications based on it by solving the tough issues such as load balancing, decentralization, scalability and flexibility. Chord will be an essential component for P2P and large-scale distributed applications. Excellent features of Chord include its simplicity, correctness, and good performance even in the case of concurrent node arrivals and departures. As a result, Chord is a very effective way to share content and user events in real-time between devices. The Samsung Chord [12] will extend its sharing presence beyond its own proprietary platform into the developer ecosystem. Samsung is fully committed to making Chord the useful sharing protocol for application developers. Chord is supported by Android 4.0 (Ice Cream Sandwich), API level 14 or above. Chord enables simple real-time sharing without the cloud. The basic features of the Chord SDK are as follows:

- Interaction between devices
- Synchronized content sharing
- Broadcasting of messages to nearby devices
- Data transferring between devices
- Multi-player games

Chord SDK helps you create a group with multi-devices in real-time, automatically, requiring no manual processing of devices which join or leave the group. All features are available through the functionality of the Chord SDK. We can play multi-user games or share newly-taken photos. Additionally, photos, documents and comments about it can be shared with group members in real-time. The Samsung Chord SDK allows application developers to develop local information-sharing applications without a detailed knowledge of networking. For this reason, we have concluded that the Samsung Chord SDK is the most efficient way to implement the prototype application for video sharing. We are able to implement a large variety of features with the Chord SDK. The Chord SDK is a network framework that makes it easy for users without professional networking skills to discover many devices connected to the same subnet. It helps them group the devices through Chord channels, and exchange files and data between the devices. There is the difference between Chord and other shot-distance network connections such as Wi-Fi direct and Bluetooth. In general, Wi-Fi Direct and Bluetooth refer to a physical network interface connection. Chord is a top layer messaging protocol that uses the TCP/IP network. Fig. 7 shows the Chord classes and interfaces that we can utilize in our application. The Chord classes and interfaces are described as follows:

- Sc*hordManager*: Used to create a node and manage the node's connection to channels.
- *SchordManager.StatusListener*: Listens to the connection status of the node.
- *NetworkListener*: Listens to the status of the network regardless of Chord connections.
- *SchordChannel*: Interface for acquiring node names and IP addresses as well as transferring data and files.
- Sc*hordChannelImpl*: Implementation of the *SchordChannel*.
- *SchordChannel.StatusListener*: Listens to joining and leaving channels, and file transfers.
- *InvalidInterfaceException*: Exception thrown when invalid interface type occurs.

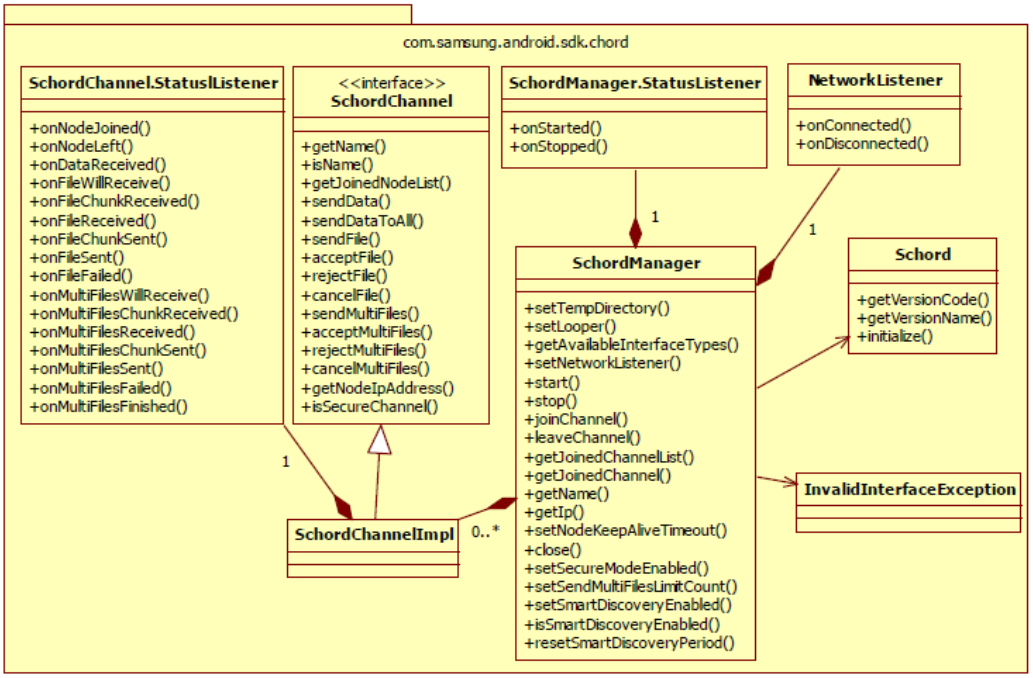



**Fig. 7.** Chord classes and interfaces

SChordChannel provides the following methods for sending and accepting data and files.

- sendData() sends data to a specific node on a channel.
- sendDataToAll() sendsdata to all nodes on a channel.
- sendFile() sends a file to a specific node on a channel.
- cancelFile() cancels a file transfer.
- acceptFile() accepts a file transfer.
- rejectFile() declines a file transfer.
- sendMultiFiles() sends files to a specific node on a channel.
- cancelMultiFiles() cancels transfer of multiple files.
- acceptMultiFiles() accepts transfer of multiple files.
- rejectMultiFiles() declines transfer of multiple files.

**Fig. 8** shows the flow of sending and accepting a key frame file between two video processing nodes. When a node sends files to another node, there is an acknowledgement to verify that the information was successfully received. The information is broken down into chunks and the successful receipt of each chunk is acknowledged. When the file is complete, the sender receives a message saying so. For file transfers, Node A begins with sendFile(). This is passed to Node B as a onFileWillReceive(), and the user needs to either accept (acceptFile()) or refuse (rejectFile()). If Node B accepts the file, the Node B application calls acceptFile() and Node A begins sending chunks of the file to Node B.

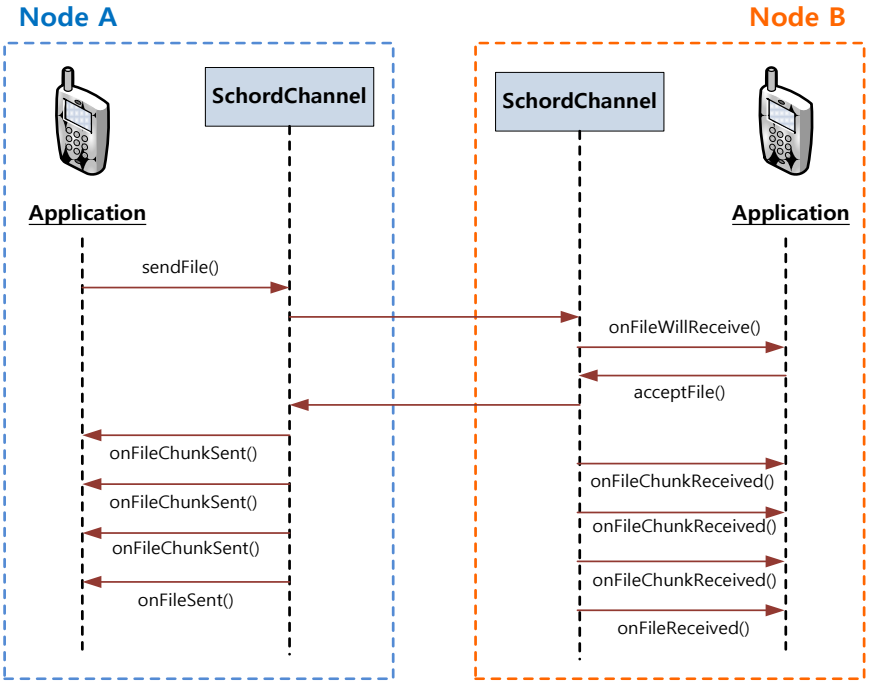



**Fig. 8.** Flow of sending and accepting a key frame file

As each chunk is sent and verified, Node A receives onFileChunkSent() and Node B receives onFileChunkReceived(). When the file transfer is complete, Node A receives onFileSent()and Node B receives onFileReceived(). The sequence of interaction for sending and accepting a file is as follows:

1. Node A's application begins by calling sendFile(). This signals that it would like to send a file to Node B.
2. onFileWillReceive() is called on Node B's application. This is to ask if the Node B user wants to receive the file.
3. If the user accepts, then Node B's application calls acceptFile().
4. The file transfer begins, and Node A starts sending chunks of data to Node B. Each time a chunk is successfully received, Node A's application receives an onFileChunkSent().
5. Node B's application receives an onFileChunkReceived() for each chunk of data.
6. When the last chunk is successfully received, the sender gets an onFileSent(), while the receiver gets an onFileReceived() from the SchordChannel.

A master device provides information on its states to slaves. In case slave joins the channel, functions in SchordChannel is called from both master's and slave's side. Information for a slave to handle is server IP address, title, video list, subtitle, and the number of participants. Besides the Chord classes for implementation of the video sharing module, we are required to make use of the Media Libraries provided in the Android SDK. It involves extending the SurfaceView class, creating a SurfaceHolder class and a MediaRecorder class, and implementing a SurfaceHolder callback function. The SurfaceView class provides a dedicated drawing surface for the picture captured, which can be displayed on the LCD screen. SurfaceHolder provides the user with the interface to control the surface size and format, to

edit the pixels in the surface, and to monitor changes to the surface. SurfaceHolder callback will be called if there is any change to the surface.

## 6. Design and Implementation of User Interface

We need to use a fragment structure for application UI which is provided by Android 3.0 APIs as shown in **Fig. 9** because we can easily compose different functionality for each layout in activity. An activity in Android OS represents a single screen with a user interface. In a multiple activities application, generally, an activity is defined as the "main" activity, which is presented to the user when user first executes the application program. A main activity of key frame list view is an object of activity type and it provides interface to users and communicates with the common library. View pager contains content providers provided by system to get key frame information from the common library. These components need to cooperate with each other in order to extract and show key frames on Android platform.

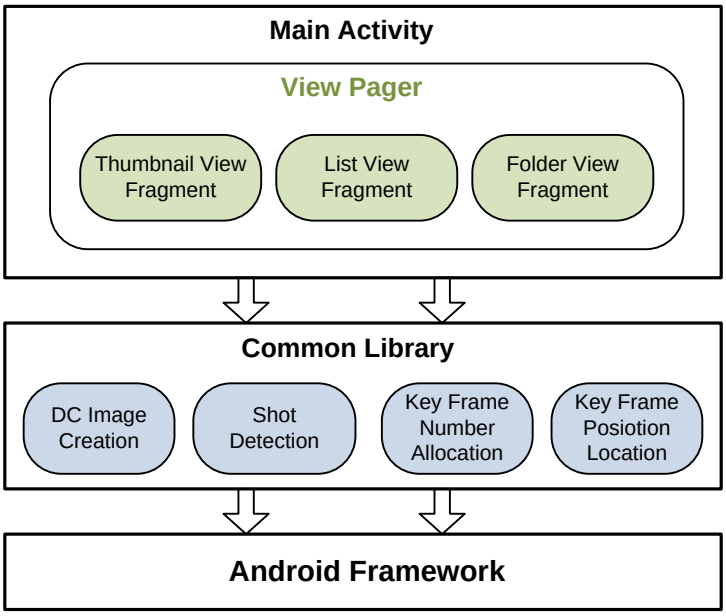



**Fig. 9.** Fragment structure for key frame extraction application UI

To extract key frames, the video common library should collect key frames then display them after decoding DC image, scene change detection, key frame allocation and distribution. According to the four steps, this paper designs the key frame extraction application based on this hierarchy. In the Java layer of key frame extraction application, the relationship of function classes and context view structure is shown in **Fig. 10**.

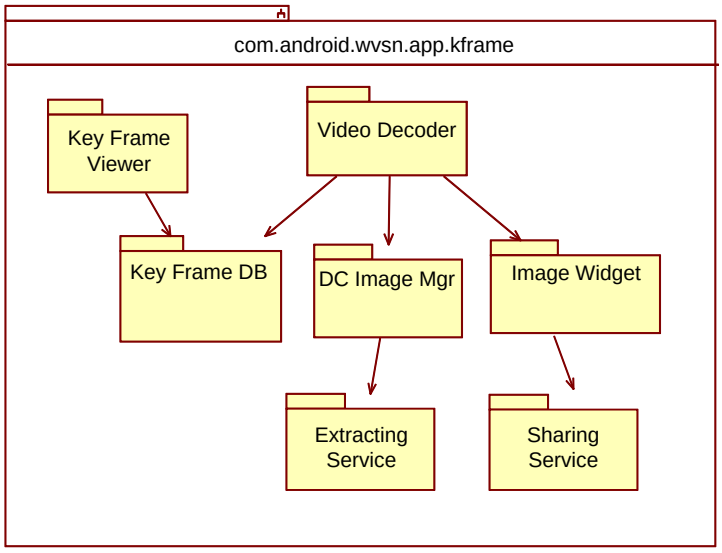
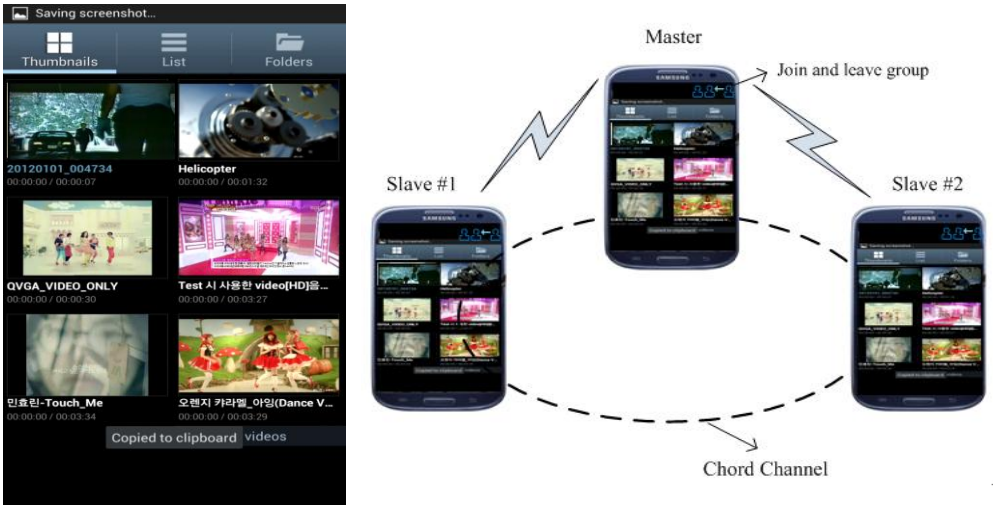



**Fig. 10.** Relationship of the classes and context view structure

We installed the application on each of these phones. User interface is shown in **Fig. 11**. It shows a screen shot of the user interface of the application and its menu tabs. It has a simple and clear user interface with three tap buttons in the view layout. When we touch the key frame, video is played from the position of corresponding key frame. Due to the use of the standard Android development kits, the application can be easily built on all of these mobile phones without modifications to the engine source code.




(a) (b)

**Fig. 11.** Implemented Android key frame extracting and sharing application.
(a) Screen shot of key frame display on Android phone (b) Example service scenario of key frame sharing application

Key frames are delivered from a master device to a slave via HTTP server-client mechanism. Considering the bandwidth of Wi-Fi, a maximum 10 devices are connected at the same time through the wireless AP and are sharing key frames in real-time. Our key frame sharing application is based on P2P communication between mobile phones, i.e. without the use of video processing servers or network infrastructure. Thus, it shows the good feasibility of such a P2P-based content sharing application for Android phones.

## 7. Experimental Results

Since there are various formats of video files, our system need to parse most of universal video formats. The formats we support are as list in **Table** 1. So, we modify the original media player in Android to support various kinds of video files listed in table 1 and then use Android API to run media scanner service, which reads metadata from the file and adds the file to the media content provider.

**Table 1.** Supported video file formats

| **supported video files** | | | |
|---|---|---|---|
| video/mp4 | video/3gp | video/3gpp | video/3gpp2 |
| video/x-ms-asf | video/x-ms-wmv | video/x-ms-wma | video/divx |
| video/avi | video/flv | video/mkv | - |

Android is a standardized platform for mobile devices, therefore, we may expect that a development process is quite simple. On the contrary, due to the various and heterogeneous hardware base of Android compatible devices, we found that we need to test our application on as many different devices as possible to ensure its functionality. After we implemented the application, compatibility testing was conducted on the application to evaluate the application's compatibility with the contents, device environment and Android OS version. Therefore, we tested various kinds of video sources, which have different video formats, audio formats and resolution. There are hundreds of devices with Android system. It is not easy to test application compatibility for all of the devices. So we choose several phones of major Android mobile phone manufacturers such as Samsung, HTC, and Google, which have different systems and hardware. We verified our application for various video formats and resolution to see if there are performance or compatibility issues using several Android phones. Test results showed that our proposed application is fully compatible for android 3.0, android 4.0 and android 4.1 and later version.

The performance assessments were compared in terms of processing time to extract and display key frames on Android phone according to percentage of selected frames. The proposed key frame extraction method was validated by experiment using several long video sequences, as listed in **Table 2**. The test data were digitized at a $1280 \times 720$ spatial resolution from consumer-grade video recordings of HDTV broadcasts and then compressed in HEVC format at 30 frame/s. The sequences were also available as DC sequences, obtained from HEVC streams with frame sizes of $160 \times 90$.

**Table 2.** Video sequences used in experiments

| Video sequences | No. of frames | Bit rate | min:sec |
|---|---|---|---|
| TV news | 20,000 | 1.300 Mbps | 6:33 |
| Animation | 27,630 | 1,054 Kbps | 15:22 |
| Music video | 21,337 | 1,200 Kbps | 17:49 |
| Documentary | 36,560 | 1,354 Kbps | 20:22 |

The results are shown in Table 3. In order to measure the processing time, we used four kinds of HEVC contents which have a resolution of $1280 \times 720$ compressed at 30 frame/s as MP@L9.3. The proposed method is compared with Bede Liu's method [6], which is conventional scheme for extracting key frame. As shown in Table 3, proposed scheme has better performance than Bede Liu's method in processing time. When 5% of sequence is selected as key frames, the proposed method takes 13.5 seconds, while the Bede Liu's method takes about 5 times more time.

**Table 3.** Comparison of performance

| processing time / % selected frames | Proposed method | Bede Liu's method |
|---|---|---|
| 1 % (200 K-frames) | 2.5 s | 11.0 s |
| 2 % (400 K-frames) | 5.6 s | 20.2 s |
| 3 % (600 K-frames) | 7.8 s | 31.0 s |
| 4 % (800 K-frames) | 11.0 s | 44.3 s |
| 5 % (1000 K-frames) | 13.5 s | 67.9 s |

So far, section from 4 to 7 has illustrated all the design, implementation, and testing. The main advantage of our proposed algorithm is that we can support various video file formats and time-exhaustive computations are not needed in distributing the key frames over the shot. And also, the procedure of key frame extraction is performed fully automatically. In addition, the set of key frames is not dependent on subjective thresholds or any manually given parameters. In a video sensor network environment, the speed performance of key frame extraction is an indication of the feasibility of the application. Application test results on target devices confirm the validity, availability and usefulness of the proposed method. In addition, the proposed key frame allocation framework demonstrates can provide a sufficient platform for many WVSN applications.

## 8. Conclusion

In this paper, we designed and implemented a powerful key frame extraction application in Android applicable to video content summarization and visualization in wireless video sensor network. We proposed a fast algorithm for key frame extraction and described the implementation of the Android-based application by using JNI. The proposed algorithm and implemented application can apply effectually to the WVSN application service development. This application enables users to share key frames extracted from recorded video data by using wireless network established between mobile phones. In addition, one of the main advantages of our system is to use DC images, thus, it runs faster than the existing approaches that use fully decoded images. By researching on key frame extraction application in Android, it will help us to develop various applications on target monitoring, scene reconstruction and video surveillance, etc.

## References

[1] Akyildiz IF, Su w, Sankarasubramaniam Y, Cayirci E., "Wireless sensor networks: A survey," *Journal of Computer Networks*, vol. 38, no 4, pp. 393-422, 2002. Article (CrossRef Link)

[2] Li JZ, Li JB, Shi SF, "Concepts, issues and advance of sensor networks and data management of sensor networks," *Journal of Software*, vol. 14, no 10, pp. 1717-1727, 2003. Article (CrossRef Link)

[3] K. Obraczka, R. Manduchi and J.J. Garcia–Luna–Aveces, "Managing the Information Flow in Visual Sensor Networks," in *Proc. of the 5th Int. Symp. on Wireless Personal Multimedia Communications*, vol. 3, pp. 1177-1181, 2002. Article (CrossRef Link)

[4] Rob Holman, John Stanley, and Tuba Ozkan-Haller, "Applying Video Sensor Networks to Nearshore Environment Monitoring," *IEEE Trans. on Pervasive Computing*, vol. 2, no. 4, pp. 14-21, 2003. Article (CrossRef Link)

[5] Huang-Chia Shih, "A Novel Attention-Based Key-Frame Determination Method," *IEEE Trans. on Broadcasting*, vol. 59, no. 3, pp. 556-562, Sep. 2013. Article (CrossRef Link)

[6] B.L. Yeo and B. Liu, "Fast Extraction of Spatially Reduced Image Sequence from MPEG-2 Compressed Video," *IEEE Trans. on CSVT*, vol. 9, no.7, pp. 1100-1114, 1999. Article (CrossRef Link)

[7] Developer resources for Google Android, http://developer.Android.com.

[8] Cheng-Min Lin, Jyh-Horng Lin, Chyi-Ren Dow, Chang-Ming Wen, "Benchmark Dalvik and Native Code for Android System," in *Proc. of the 2nd Int. Conf. on Innovations in Bio-inspired Computing and Applications*, pp. 320-323, December 2011. Article (CrossRef Link)

[9] Damianos Gavalas and Daphne Economou, "Development Platforms for Mobile Applications," *IEEE Software*, vol. 28, no.1, pp. 77-86, Feb. 2011. Article (CrossRef Link)

[10] SJ Cho, KJ Kim, EH Hwang, SH Yoon and JW Jeon, "Benchmarking Java Application Using JNI and Native C Application on Android," in *Proc. of ICC*, pp. 284-288, 2012. Article (CrossRef Link)

[11] Ion. Stoica et al., "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Trans. on Networking*, vol. 11, no. 1, February, 2003. Article (CrossRef Link)

[12] Samsung mobile SDK, http://developer.samsung.com/chord.

[13] Scott Pudlewski and Tommaso Melodia, "A Tutorial on Encoding and Wireless Transmission of Compressively Sampled Video," *IEEE Commun. Surveys & Tutorials*, vol. 15, no. 2, pp. 754-767, 2013. Article (CrossRef Link)

[14] Xiang Sheng, Jian Tang, Xuejie Xiao, and Guoliang Xue, "Sensing as a Service: Challenges, Solutions and Future Directions," *IEEE Sensors*, vol. 13, no. 10, pp. 3733-3741, 2013.

Article (CrossRef Link)
[15] A. C. Begen, T. Akgul, and M. Baugher, "Watching video over the web, part I: streaming protocols," *IEEE Trans. on Internet Computing*, vol. 15, no. 2, pp. 54-63, March 2011.
Article (CrossRef Link)

**Kang-Wook Kim** received the B.S., M.S., and Ph. D. degrees in Electronics Engineering from Kyungpook National University, Korea in 1996, 1998 and 2002 respectively. He is currently a principal engineer in R&D Group, Mobile Communication Division, Samsung Electronics Co., Ltd. His research interests include visual communication, mobile embedded system, and Android application.