# Platform Support for Mobile Edge Computing

Jaehun Lee*, Hochul Lee*, Young Choon Lee†, Hyuck Han‡, Sooyong Kang*
*Dept. of Computer Science, Hanyang Univ., Korea, Email: {ljhokgo,lhochul2,sykang}@hanyang.ac.kr
†Dept. of Computing, Macquarie Univ., Australia, Email: young.lee@mq.edu.au
‡Dept. of Computer Science, Dongduk Women's Univ., Korea, Email: hhyuck96@dongduk.ac.kr

***Abstract*—Computing resources including mobile devices at the edge of a network are increasingly connected and capable of collaboratively processing what's believed to be too complex to them. Collaboration possibilities with today's feature-rich mobile devices go far beyond simple media content sharing, traditional video conferencing and cloud-based software as a services. The realization of these possibilities for mobile edge computing (MEC) requires non-trivial amounts of efforts in enabling multi-device resource sharing. The current practice of mobile collaborative application development remains largely at the application level. In this paper, we present *CollaboRoid*, a platform-level solution that provides a set of system services for mobile collaboration. CollaboRoid's platform-level design significantly eases the development of mobile collaborative applications promoting MEC. In particular, it abstracts the sharing of not only hardware resources, but also software resources and multimedia contents between multiple heterogeneous mobile devices. We implement CollaboRoid in the application framework layer of the Android stack and evaluate it with several collaboration scenarios on Nexus 5 and 7 devices. Our experimental results show the feasibility of the platform-level collaboration using CollaboRoid in terms of the latency and energy consumption.**

## I. Introduction

Mobile edge computing (MEC) is a recent distributed computing concept that promotes harnessing ever pervasive and capable mobile devices often for collaborative computing. For instance, collaborative software (e.g., messenger and video conferencing applications) has become a vital part of everyday life. Citizen science projects [1] are a larger-scale example of collaborative computing. The 'collective' access to these increasingly resource-rich mobile devices enables many capabilities once believed to be only available from traditional servers, for example, in clouds. In this paper, we study MEC from mobile collaboration perspective.

Today's mobile devices are smart gadgets having a variety of hardware sensors (e.g., gyroscope and accelerometer) with increasing computational power. Besides, network environments have improved dramatically in the past decade in particular owing to the emerging faster communication network (e.g., LTE) and widely deployed Wi-Fi networks. Applications collaboratively sharing a virtual pool of software/hardware resources from other devices via wireless networks open a new application development avenue promoting MEC. It is however not an average task for one mobile device to use resources from other devices.

Recently, several research efforts have been made to leverage collective or collaborative use of mobile devices in the context of MEC [2]–[5]. Cloudlet [2] and femtocloud [3] have sought to harness near-by mobile devices through using small-scale compute clusters (e.g., in a cafe or classroom) as servers or coordinators. In the meantime, Rio [5] and Flux [4] provide solutions, more specific to mobile collaboration, at kernel level and application context level, respectively. Nevertheless, none of previous solutions are designed for simultaneous sharing of resources in multiple heterogeneous mobile devices.

In this paper, we present *CollaboRoid* [6], a platform-level solution that enables multiple heterogeneous mobile devices to share their resources for effective mobile collaboration. It abstracts the sharing of hardware and software resources and multimedia contents among mobile devices. The novelty of CollaboRoid is in its platform-level design of core mobile collaboration functionalities that application developers currently have to develop application to application. In particular, our implementation of CollaboRoid incorporates its core functionalities in the application framework layer of the Android stack.[1] We summarize these core functionalities as follows:

- CollaboRoid detects nearby devices and identifies the status of these devices as well as their resources so that applications can exploit remote resources.
- CollaboRoid extends the existing system services of Android platform to leverage the sharing of hardware/software resources between heterogeneous mobile devices.
- CollaboRoid allows application developers to use the existing programming interface to communicate with remote devices through its platform-level services; this significantly facilitates the acquisition of events/data from remote devices for application development.

The advantages of CollaboRoid are four-fold: (1) it paves the way for the development of new types of collaborative application (i.e., multi-user applications that are executed in a single device exploiting multiple devices), (2) it makes existing collaboration-oblivious applications possible to be easily extended to support collaborative applications, (3) it facilitates convenient sharing of multimedia contents among multiple devices without file copies (i.e., streaming), and (4) it enables "power sharing" for low battery devices by offloading some tasks to remote hardware resources.

We have conducted experiments with multiple resource sharing scenarios on Nexus 5 and 7 smartphones. Our experimental results demonstrate the latency of remote resource

[1]The current implementation of CollaboRoid in Android is due to primarily the representativeness of Android as a mobile OS and the closed nature of iOS; however, it is not limited to a specific OS.

2159-6190/17 $31.00 © 2017 IEEE
DOI 10.1109/CLOUD.2017.84

access remains sufficiently low for mobile collaboration. More specifically, the latency of the remote hardware access is smaller than $100ms$ regardless of the hardware type and the number of remote devices. For example, average latencies of $46ms$ and $70ms$ for remote touch input, which is the most latency-sensitive remote resource access case, in a LAN environment and a WAN environment, respectively. The energy usage of extra communication for resource sharing is also insignificant considering the benefit of "power sharing" effect.

The rest of this paper is organized as follows. Section II provides background and related work. Section III describes the design overview of CollaboRoid. Sections IV and V detail two main components of CollaboRoid, i.e., Device Management Service and Resource Access Service. Section VI presents our evaluation results in various scenarios, followed by the conclusion in Section VII.

## II. Background and Related Work

In this section, we briefly describe Android focusing on its application framework layer and discuss related work.

### A. Application Framework of Android

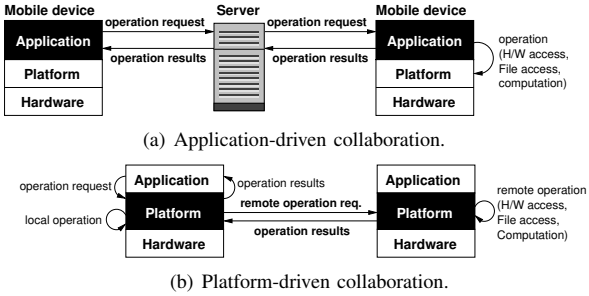
Android features a number of system services for applications to control hardware (e.g., vibrator) and to get data from hardware (e.g., input device, GPS and sensors). Applications can also use the software resources in Android, i.e., system library functions, via API calls.

To access a particular hardware resource, applications need to get the appropriate manager, for that hardware resource, allocated from Service Manager in the 'application framework'. The allocated manager coordinates with the corresponding System Service which controls or receives events from the dedicated hardware. There are two types of hardware resource in terms of communication direction. In particular, resources like vibrator are controlled by applications through corresponding service managers, e.g., vibrator manager. In the meantime, applications receive data from data acquisition hardware, such as input device, GPS device and sensors, via corresponding hardware managers. To receive data from a specific hardware resource, the application needs to register its listener to the corresponding hardware service manager. Then any data received from hardware is forwarded to the listener of the application via kernel, library, system service and service manager.

### B. Related Work

*1) Remote I/O sharing:* Sharing I/O resources in a remote device has been elaborated for a long time. Examples include IP Webcam [7], Wi-Fi Speaker [8] and Screen Sharing [9], which are 'application-level' solutions for remote I/O sharing. Their capacity is often limited to sharing/accessing a particular I/O resource in remote device since applications used are specifically designed for such sharing purpose.

A recent kernel-level approach (*Rio* in [5]) provides more generic solutions for remote I/O sharing. For a pair of local and remote devices, Rio maps the device file for an I/O resource in the remote device to a virtual device file for a corresponding I/O resource in the local device. Since Rio provides kernel-level solutions, the design and implementation of actual collaboration functionalities are largely left to application developers. CollaboRoid's platform-level approach significantly eases such design and implementation.




(a) Application-driven collaboration.

(b) Platform-driven collaboration.

Fig. 1. Mobile collaboration models.

*2) Remote Device Control:* Similarly to I/O resource sharing, controlling remote devices has been explored in several different ways, RemoteUI [10] and AirPlay [11]. RemoteUI controls remote devices using a web server and web sockets. It allows users to control buttons in a remote device and to share media files. AirPlay's intuitive design, of split user interface for shared panel and control panel, much improves user experience. These remote device control approaches primarily aims to share media files; and thus, hardware resources they can share are limited to, for example, speakers and display.

*3) Mobile Cloud Computing:* Offloading computation tasks in smart devices to servers in clouds or nearby mobile devices is another well studied approach as some form of collaboration. Works with the use of cloud servers for such offloading include MAUI [12], CloneCloud [13], ThinkAir [14], Cloudlet [2] and femtocloud [3]. In [15], they used a dedicated proxy system, instead of cloud servers, to offload computationally heavy JavaScript code for mobile client. With recent advances in mobile devices in terms of capabilities and capacities, these devices are directly used for computation offloading, e.g., Hyrax [16], Serendipity [17] and LWMR [18]. These types of mobile computing approaches essentially fall into cloud-dependent offloading or mobile distributed processing, rather than collaboration.

*4) Multimedia Contents Sharing:* There have been several efforts on multimedia contents sharing among mobile devices, and their approaches can be categorized into: 1) central or cloud server based contents sharing [19], and 2) device-to-device contents sharing based on opportunistic P2P network [20], [21]. While they use different approaches and technical schemes, they are common in that they are not directly supported by the mobile platform for their objectives.

## III. Model and Design of CollaboRoid

In this section, we first introduce a platform-driven collaboration model. Then, we describe the design of CollaboRoid.

We propose a *platform-driven* collaboration model, in which mobile platform takes the entire roles of the *physical* collab-

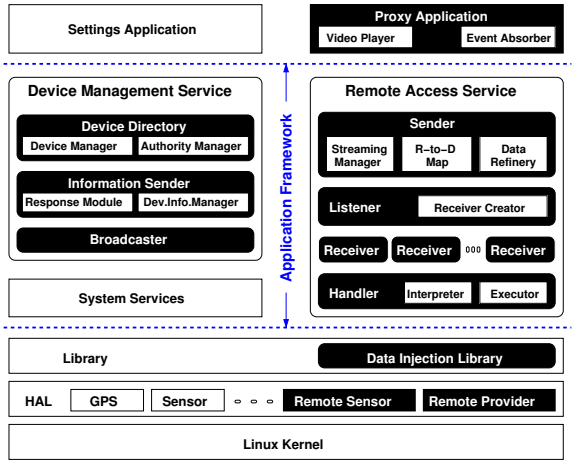



Fig. 2. Software architecture of CollaboRoid

oration among devices while application takes control of the platform in accordance with the semantic of the collaboration (Figure 1(b)). Traditionally, mobile collaboration has been driven by applications and their own servers (Figure 1(a)). There are several merits in the platform-driven model: 1) applications do not need to implement physical collaboration functionalities, 2) dedicated server is not needed for collaboration purpose, and 3) collaboration is possible without application being installed in all participating devices, among others. CollaboRoid is our solution to the realization of platform-driven collaboration model.

CollaboRoid's design (Figure 2) overarches two fundamental functionalities of device management and remote resource access with two other components of proxy application and data injection library. These collaboration functionalities are implemented on Android; hence, the name *CollaboRoid*. Devices in mobile collaboration are classified into the initiator and the participant(s). As we describe CollaboRoid from the initiator's perspective, we refer the initiator as the local device and the participant as the remote device.

CollaboRoid consists of four components: Device Management Service (DMS), Remote Access Service (RAS), Data Injection Library (DIL) and Proxy Application (PA). As the collaboration environment we concern is with mobile devices that can be connected without a server, DMS provides the information on mobile devices available for collaboration. In particular, DMS collects and maintains several pieces of information, including collaboration activation state and available resources, from mobile devices.

RAS is responsible for the device-to-device communication between mobile devices for the actual collaboration. This communication concerns control flow primarily from the local device to remote devices and data flow from remote devices to the local device. The former flow occurs by forwarding service calls, initiated from an application in the local device, to remote devices. The latter flow mostly takes place in response to the former. RAS feeds data sent by remote devices into the application in the local device in the same way data generated in the local device is fed. This data flow allows the development of collaborative applications to be almost similar to collaboration-oblivious application development.

DIL enables the transparent process of events received from remote devices by feeding them to Android's native library to be treated in exactly the same way as local events.

PA is a user-transparent system application designed to run for mobile collaboration. It performs a specific *role* depending on the type of mobile collaboration. The two roles in the current design of CollaboRoid are *Event Absorber* and *Video Player*, each of which is implemented as an activity in the PA. Event Absorber enables a remote device user to control his/her device even when the device is being used as a remote input device for an application in a local device. Video Player enables video streaming by decoding sample data chunks of a video file. We further elaborate activities of these two roles (or simply two activities) in the context of RAS in Sections V-A and V-C, respectively.

## IV. Device Management Service (DMS)

DMS consists of three sub-components: Broadcaster, Information Sender and Device Directory. Broadcaster sends a broadcast signal to devices connected to the same Wi-Fi access point (AP) with the local device requesting necessary information for collaboration. The broadcast signal is then handled by Information Sender of the remote device to respond to the local device with what is asked, such as availability for collaboration and the states of resources. Device Directory maintains the list of remote devices in the current collaboration.

DMS is capable of dealing with devices connected to the same Wi-Fi AP as well as those in the contact list. The latter set of devices may not share the same AP, but are connectable via mobile communication networks. This functionality of DMS allows the *Settings* application in a device to configure a number of settings, such as the set of remote devices to be participated in the collaboration and the type of collaboration. For instance, the collaboration initiator can configure the collaboration with two devices detected, one being shared its sensor resource and the other allowed for API calls. We assume a collaboration only takes place among devices whose owners accept the collaboration invitation.

## V. Remote Access Service (RAS)

RAS consists of four sub-components: Listener, Sender, Receiver and Handler. Listener is the entity for establishing the communication channel between devices, either for control (from local to remote devices) or for data transmission (from remote to local devices). Once Listener gets a request to establish a session, it creates an instance of Receiver and associates the communication channel (socket) to the Receiver instance. Sender and Receiver are paired up for control and data communication. When the local device issues a request to access resources of a remote device, Sender in the local device forwards that request to Receiver in the remote device. The result of that request (e.g., sensor data) is then returned back by the remote device's Sender to the Receiver instance

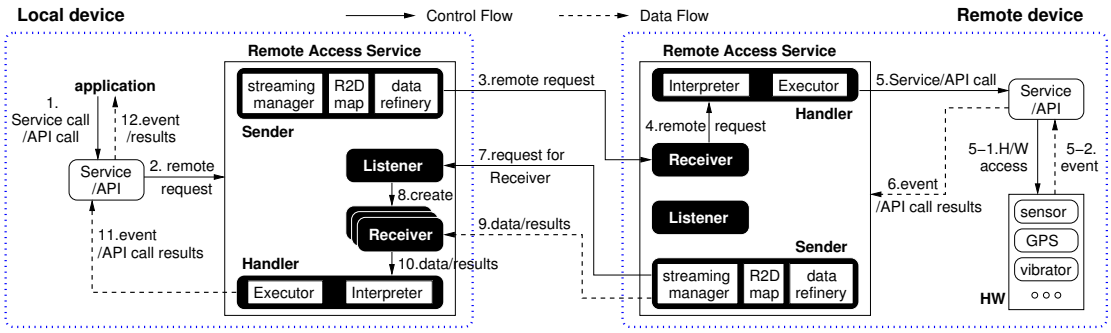



Fig. 3. Procedure of remote hardware/software resource access.

specifically created, for such communication, by the local device's Listener. The R-to-D (resource-to-devices) map in the Sender contains resource sharing information that describe which resource is being shared with which devices. This information is used by the Sender either when sending remote resource access requests to remote devices or when sending the results of remote requests. Handler processes the request/data Receiver gets according to the type of resource. Figure 3 shows the procedure of the remote resource access in CollaboRoid.

In the following, we give more detailed description of Remote Access Service's design specifically discussing how it addresses the access to different types of resource, i.e., hardware, software and media content.

### *A. Hardware Resource Access*

CollaboRoid provides applications a set of remote hardware types, such as TYPE_REMOTE_ORIENTATION for remote orientation sensor and REMOTE_GPS_PROVIDER for remote GPS. If an application issues a service call for a remote device, i.e., when a remote hardware type is used, the corresponding service sends a remote request to RAS, which eventually will be forwarded to Receivers in all collaborating devices listed in the R-to-D map. Otherwise, it accesses local hardware. In this way, CollaboRoid provides uniform access interface for both local and remote resources.

Upon receiving the remote request, Receiver in a remote device forwards the request to Handler. Interpreter and Executor in Handler identifies the type of service and makes an appropriate service call for the request, respectively. The service called for the request gets and sends the event from the hardware resource to RAS (Steps 5-2 and 6 in Figure 3). Sender in the remote device sends a message to Listener in the local device; this message is for the creation of a Receiver instance to establish a session that is used for returning the event. Once Receiver is ready for communication, Sender in the remote device sends the event data to the Receiver. At this time, instead of sending the event itself, Sender extracts only key attributes from the event using Data Refinery and sends them to the local device, not only to minimize the communication overhead but also to remove device-dependent data in the event. Data Refinery also transforms the extracted data, if necessary, to a normalized form so that the data receiving device can adjust them to its hardware specification. For example, the key attribute of touch events is the set of $x$ and $y$ coordinates. In case of two collaborating devices with different display resolutions (a form of resource heterogeneity), coordinates conversion must be done to resolve any discrepancy. Data Refinery converts absolute coordinates into relative ones before sending them.

Handler in the local device rebuilds the remote event using the received data and local device-dependent data. Interpreter of Handler, if necessary, adjusts the received data to overcome the hardware heterogeneity, e.g., converting relative coordinates to absolute ones. During rebuilding the event, Handler embeds the source device ID to the event so that applications can identify the source device of each event. For this purpose, when a remote device connects to the local device, CollaboRoid in the local device assigns a new ID to the remote device. It is worth note that by assigning one Receiver instance to each remote device, applications in the local device can use hardware resources in multiple remote devices, simultaneously. In this way, CollaboRoid permits more than one remote devices to participate in collaboration, which enables developers to implement multi-user applications exploiting multiple devices.

Finally, Handler passes the rebuilt remote event to the local service using DIL. In Android, system services use native library to obtain local hardware-generated events. DIL enables uniform data acquisition by 'injecting' remote events to the native library. System services do not need to differentiate how to obtain local and remote hardware events. Also, by registering remote hardware in the Hardware Abstraction Layer (HAL) and using DIL, applications can use those hardware which are not available in the local device.

Note that after the creation of Receiver in the local device (Step 8), only data flow steps (Steps 5-2, 6, 9, 10, 11 and 12) are repeated upon subsequent hardware events until remote device user disables the remote sharing capability. However, such disabling (by 'touching' display to launch the Settings app and disable the capability) from remote device is not direct due to the forwarding of hardware events (i.e., motion event, in this case) to local device.

To resolve this problem, CollaboRoid essentially uses bidirectional event forwarding meaning that every hardware event in the remote device is forwarded to both the local and remote devices, so that a remote device user can control his/her device

even when the device is being used as a remote input device. However, in that case, a motion (touch) event in a remote device intended to be forwarded to the local device as a remote input can unintentionally control the remote device itself. Proxy application (PA) solves this problem. When an application in the local device wants to exploit a remote device as an input device, it can first request to the remote device to launch a PA with *Event Absorber* activity which merely displays a black screen and ignores any motion event forwarded to itself. Then any motion event generated from the remote device does not control the device but is forwarded, as a remote input, to the application in the local device. Remote device user can terminate the PA at any time by pressing (or touching) 'back button', and then can control the device again.

### B. Software Resource Access

Access to software resources we concern in this work is assumed to be made possible via API calls. Therefore, applications should be able to identify the destination device the API call is for. CollaboRoid adopts function overloading to make a remote API call. In particular, we add a flag to the original API that can be set to either local call, remote call or both. The process of remote API call is similar to that of accessing hardware resources shown in Figure 3. We envision that this function overloading for remote API calls would be meaningful for MEC with computationally intensive tasks. In particular, custom-designed compute-intensive library functions can be installed to multiple devices for MEC or more broadly distributed computing. This use case scenario illustrates the possible use of CollaboRoid as a base layer of distributed computing across mobile devices.

### C. Multimedia Contents Sharing

Multimedia content sharing is probably one of most common use cases of mobile collaboration. However, the current mobile platforms do not provide content sharing functions by default. The common practice for content sharing largely remains the use of simple file copy, which may have copyright violation issues; instead, we adopt streaming.

An important merit of the platform-level media streaming service in our CollaboRoid design is that users can share media content in a 'push' manner. By push we mean the owner of contents controls the entire sharing process from browsing and selecting contents and transferring the contents to the remote devices. Such push approach ensures secure and legitimate sharing of media content. In the following, we show how CollaboRoid processes video and audio contents.

For video content sharing, the video player application first sends the URI of media file to the media framework (Step 1 in Figure 4). The media framework creates a socket for transferring the media file and forwards the remote request containing the socket to Receiver in the remote device via Sender in the local device (Steps 2-3). Receiver forwards the request to Handler (Step 4) for taking an appropriate action. CollaboRoid streams a video file in a sample data chunk, which is the unit of media playback, at a time. However,

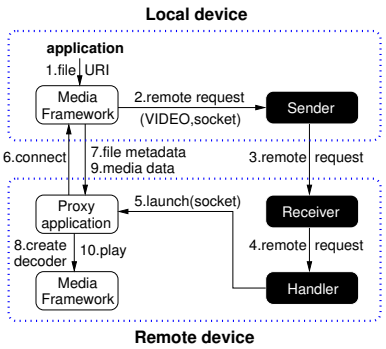



Fig. 4. Video contents sharing in CollaboRoid

MediaPlayer in Android is unable to play such sample data directly. To this end, CollaboRoid directly uses MediaCodec in Media Framework to decode sample data chunks, and uses PA to take control of the entire playback process in the remote device. Upon receiving the remote request, Handler in the remote device launches PA with the *Video Player* activity, and it passes the socket to PA (Step 5). PA connects to the media framework in the local device using the received socket (Step 6). Once the connection is established, the media framework extracts the metadata from the media file and sends it to PA (Step 7). PA creates a decoder using the metadata (Step 8). The media framework then streams down media data to PA (Step 9). The media data is played by PA using the decoder associated with it (Step 10). Steps 9 and 10 are repeated until all video data is streamed and played.

Since playing a video using MediaCodec requires separate processes for audio and a video, CollaboRoid uses two different threads. Because MediaCodec is unable to process synchronization between audio and a video, CollaboRoid adopts a simple time-delay method to remedy this synchronization issue. In particular, for a given pair of audio and a video of sample data, it puts the thread, whose data (audio or video) arrived earlier, into sleep for a short period of time, i.e., $10ms$. After the sleep, the thread checks if the other type (audio or video) of sample data has arrived. If so, the current sample data gets played and otherwise, the thread sleeps again for another $10ms$. This way, the synchronization error bound is up to $10ms$ maximum in CollaboRoid. We adopt $10ms$ for this synchronization purpose as the work in [22] found that a synchronization error range between $\pm 80ms$ is acceptable.

As is noticed, the media framework in the local device directly streams video data to PA in the remote device, bypassing RAS. This is because video sharing with long and heavy data transfer imposes significant overhead on RAS. Our design of the direct connection using a socket associated with PA much eliminates such overhead.

Audio sharing is handled differently from video sharing in that it does not require the control of the display nor PA. The major difference between audio and video content sharing is the use of media framework directly by Handler. In our previous work [23], we implemented an application framework service which enables remote music play in Android. We refer readers interested in the implementation details of the audio sharing in CollaboRoid to our prior work.

TABLE I
ACTIONS FOR GENERATING EVENTS AND REQUESTS.

| Hardware | Actions |
|---|---|
| Touch | Touch and drag (10 times) |
| Key | Random keyboard input (100 times) |
| Orientation | Repeatedly change orientation for 5 seconds |
| Proximity | Move a finger closer to the sensor and further away (30 times) |
| Location | Generate random coordinates (100 times) |
| Vibrator | Issue random vibration (100 times) |

## VI. EVALUATION

To verify the feasibility of the platform-level collaboration using CollaboRoid, we measured both the latency and energy consumption for various types of mobile collaboration. We used one Nexus 7 and eight Nexus 5 devices for experiments.

We define the latency of remote resource access as the elapsed time from the occurrence of an event in the remote device to its arrival at an application in the local device. In case of multimedia content sharing, the latency is defined as the elapsed time from when the local device user issues the content playback command (by touching the 'Play' button) to the time playback starts in the remote device. For accurate clock synchronization in all devices, we used Network Time Protocol (NTP) [24] with millisecond-level synchronization error correction. We measured latencies in both LAN and WAN environments.

As energy consumption is another important metric for mobile devices, we measured power consumption for remote hardware access and multimedia content sharing and compared it to the vanilla Android. The power consumption was measured using the Trepn Power Profiler [25] developed by Qualcomm. For the fair comparison, we only used Nexus 5 devices in experiments when measuring energy consumption.

### A. Latency in LAN Environment

For the latency measurement, we used Nexus 7 as a local device and up to eight Nexus 5 devices as remote devices, all of which are connected to the same Wi-Fi AP that supports 802.11 a/b/g/n/ac. We implemented two simple test applications for latency measurement: one for vibrator services and another for other services. Both applications are executed in the *local* device. The vibrator application issues remote vibration requests to the vibrator service. It logs timestamp the very moment after calling vibrator service. We modified the vibrator *service* in the *remote* device so that it logs timestamp the very after issuing a command to the vibrator hardware. The difference between timestamps in both local and remote devices is the latency of the remote vibrator access.

The other application is for other hardware peripherals (i.e., touch, key, sensor and GPS) from which received data is delivered to the application in the local device. It logs timestamp whenever receiving an event from the remote device. In the *remote* device, each system service for hardware access is modified to log timestamp whenever it receives an event from the hardware. Hence, for each event, two timestamps (local and remote) are generated, whose difference is the latency of the event transfer from remote system service to local application.

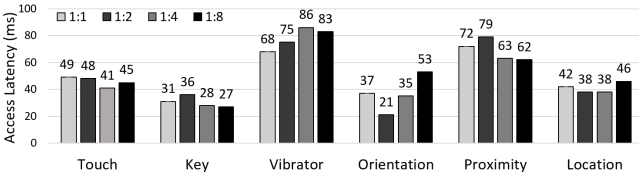
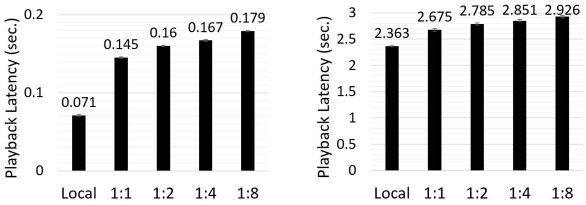



Fig. 5. Remote hardware access latency in LAN environment.




(a) Audio (b) Video

Fig. 6. Latency in multimedia content sharing.

Table I shows actions to generate remote hardware events and remote vibration requests. Actions for all hardware sensors except for vibrator were performed in the remote device.

*1) Remote Hardware Resource Access:* Figure 5 shows the average latency of remote hardware access. In the figure, '1:*N*' denotes the average latency when a local device accesses *N* remote devices, simultaneously. In all cases, the latency is smaller than $100ms$ (in particular, smaller than $50ms$ in latency-sensitive touch hardware. Besides, the correlation between remote hardware access latency and the number of remote devices increases is weak—at least up to eight remote devices. Together, the latency of remote hardware access has little effect when using CollaboRoid.

*2) Multimedia Content Sharing:* To measure the latency in multimedia content sharing, we used native media playback applications in Nexus devices: 'Gallery' and 'Music' for video and audio playback, respectively. Since both applications call MediaPlayer service as soon as the 'play' button is touched by user, we modified the 'start()' function in the MediaPlayer service to log timestamp at the beginning of the function, in the *local* device. In the *remote* device, we modified the MediaCodec class so that it logs timestamp as soon as decoding the first media sample data. The difference between local and remote timestamps becomes the playback delay in the remote device. We used two media files whose sizes are 5.4 MiB (audio) and 1.2 GiB (video), respectively. The Nexus 7 device was used as a local device which is the only one where media content reside and up to eight Nexus 5 devices were used as remote devices for simultaneous content sharing.

Figure 6 shows the average latency in multimedia content sharing, varying the number of remote devices. 'Local', in the figure, denotes the local media file playback latency in Nexus 7 with the vanilla Android, which was measured in exactly the same way as the sharing case. As expected, the latencies when content is shared with remote devices are larger than the local playback latency because of the communication overheads between local and remote devices. Nevertheless, the latency increases are smaller than $110ms$ and $600ms$ in

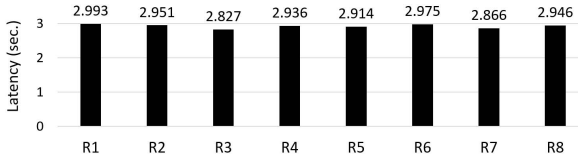



Fig. 7. Video sharing latency in each remote device.

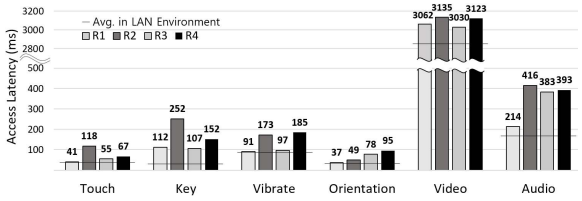



Fig. 8. Latencies in WAN environment with 1:4 case.

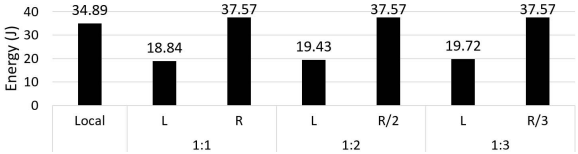



Fig. 9. Energy consumption when exploiting remote GPS for one minute.

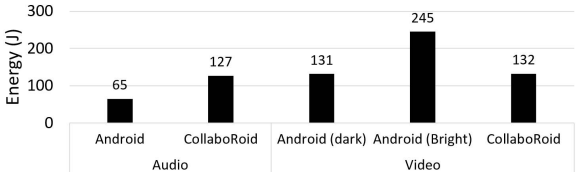



Fig. 10. Local device energy consumption for local and remote playback.

audio and video sharing even with eight remote devices. Based on the results, we can say that CollaboRoid provides a very feasible and convenient way for multimedia content sharing among multiple mobile device users, provided that all devices are connected to the same AP.

In real time streaming-based multimedia content sharing, it is also important to synchronize multiple content streams to each device, that enables participating users to watch or listen to the same scene/sound of the content. Then users are able to have offline talk on a particular scene while everyone is watching the same scene on his/her own device. We can determine whether or not CollaboRoid provides synchronized content streams by comparing the latencies in all devices.

Figure 7 shows the individual latency in each remote device in the 1:8 video sharing case. The maximum latency difference among eight devices is 166$ms$, which means that CollaboRoid offers almost synchronized content sharing, provided that all devices are connected to the same AP.

### B. Latency in WAN Environment

To measure the latencies in WAN environment, we used five Nexus 5 devices where one (L) is for a local device and the others (R1, R2, R3, R4) for remote devices (i.e., 1:4 sharing case). Each device is connected to the subscriber line through Wi-Fi AP at home. The remote devices are about 18.5 Km, 22.6 Km, 10.3 Km and 9.5 Km away from local device, as the crow flies. L, R1, R2, R3 and R4 are connected to the 100 Mbps, 320 Mbps, 100 Mbps, 100 Mbps and 100 Mbps subscriber lines, respectively. For this experiment, we modified DMS in CollaboRoid so that devices can communicate each other through manual setting of IP address.

Figure 8 shows the latencies in WAN environment. The most notable difference between results in LAN and WAN environments is that the latency differencies among remote devices in WAN environment are far larger than those in LAN environment. However, interestingly, the latency itself was not as large as we expected. The largest hardware access latency in this experiment was 252*ms* for remote Key input from R2, and the latencies for remote touch input, which is the most latency-sensitive case, from all remote devices were less than 120*ms*. In video sharing case, while the average latency in WAN environment (3,088*ms*) increased by about 240*ms* than in LAN environment (2,851*ms*), the largest latency difference among all devices was 560*ms* (local: 2,575*ms*, R2: 3,135*ms*) which was similar to that in the LAN environment (570*ms*). These results suggest that CollaboRoid can be effectively used for resource sharing among mobile devices in the WAN environment, too.

### C. Energy Consumption

Remote resource access and content sharing incur additional energy consumption due to the extra communication associated with them. Meanwhile, by exploiting remote resources, a local device with low battery can save its energy, which we call the *power sharing effect*. In this section, we measure both the additional energy consumption and power sharing effect of CollaboRoid with remote GPS access and multimedia sharing. In this experiment, we use only Nexus 5 devices.

*1) Remote GPS Access:* Figure 9 shows the consumed energy in each device when location data is received from up to three remote devices simultaneously. 'L' and 'R' denote the energy consumption in local and remote devices, respectively. 'Local' denotes the energy consumption when local GPS is used on the vanilla Android.

Since each remote device performs exactly the same operations, they consume the same amount of energy regardless of the number of remote devices. The energy consumption in the local device slightly increases as the number of remote devices increases, since it receives location data from all remote devices. As expected, not only the total energy consumption (L+R), but also the remote device energy consumption (R) are higher than the 'Local' case, due to the additional energy consumption by communication. However, it is notable that the local energy consumption (L) in the 1:1 case is far smaller than the 'Local' case. It means that, by exploiting remote GPS instead of local one, we can effectively save the local device energy, which demonstrates the energy sharing effect. Hence, for a device with low battery, we can effectively use the remote hardware access functionality in CollaboRoid for energy saving.

*2) Multimedia Content Sharing:* We used audio and video files whose sizes and playback times are 5.4 MiB / 215 sec. and 80 MiB / 139 sec., respectively. We measured the total energy consumption during the playback.

Figure 10 shows the energy consumption in the local device when the vanilla Android and CollaboRoid are used for local and remote content playback, respectively. When CollaboRoid was used, only remote device plays the content (i.e., 1:1 content sharing without local playback). If the local energy consumption in CollaboRoid is smaller than that in the Android, we can save the local device energy by exploiting the remote software (i.e., decoding and rendering) and hardware (i.e., speaker and display).

When an audio content is played in the remote device instead of the local device, the local device cannot save its energy because data streaming via Wi-Fi network consumes more energy than the software decoder and speaker. When a video content is played in the remote device instead of the local device, the local device can save its energy since the display consumes large amount of energy. While CollaboRoid consumes similar amount of energy with Android when the display brightness is set to the minimum level ('Dark'), it consumes far less energy than Android with the maximum brightness level ('Bright'). Hence, the multimedia content sharing functionality of CollaboRoid can be effectively exploited when a user wants to enjoy a video content in his/her smartphone using a large screen tablet as a remote playback device, while obtaining the benefit of energy sharing effect.

## VII. Conclusion

Mobile platform support for inter-device collaboration provides a new possibility for applications to evolve into collaborative ones. In this paper, we have presented a novel mobile platform-level solution, CollaboRoid, which provides inter-device collaboration functionalities including remote hardware/software resource access and multimedia content sharing. Using only a few additional or modified programming interfaces, application developers can easily develop collaborative applications on CollaboRoid, in comparison with the case for the development with legacy mobile platforms. Our experimental results show that CollaboRoid brings the collaboration functionalities to applications with acceptable overhead in terms of both latency and power consumption. CollaboRoid also enables users to save energy in their low battery devices by exploiting remote hardware instead of local ones. Together, CollaboRoid would open a new avenue of mobile collaborative application development expecting it to be an enabling solution for mobile edge computing.

## Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2014R1A2A2A01004187). Sooyong Kang is the corresponding author of this paper.

## References

[1] E. Hand, "Citizen Science: People power," *Nature*, vol. 466, no. 7307, pp. 685–687, 2010.
[2] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Adaptive Deployment and Configuration for Mobile Augmented Reality in the Cloudlet," *J. Network and Computer Apps*, vol. 41, pp. 206–216, 2014.
[3] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto Clouds: Leveraging Mobile Devices to Provide Cloud Service at the Edge," in *Proc. IEEE Int'l Conf. Cloud Computing (CLOUD '15)*, 2015, pp. 9–16.
[4] A. Van't Hof, H. Jamjoom, J. Nieh, and D. Williams, "Flux: Multi-surface Computing in Android," in *Proc. ACM European Conf. Computer Systems (EuroSys '15)*, 2015.
[5] A. Amiri Sani, K. Boos, M. H. Yun, and L. Zhong, "Rio: A System Solution for Sharing I/O Between Mobile Systems," in *Proc. Int'l Conf. Mobile Systems, Applications, and Services (MobiSys '14)*, 2014.
[6] J. Lee, H. Lee, Y. C. Lee, H. Han, and S. Kang, "Demo: CollaboRoid for Mobile Collaborative Applications," in *Proc. 14th Annual Int'l Conf. Mobile Systems, Applications, and Services (MobiSys '16)*, 2016, (Extended abstract, http://dcslab.hanyang.ac.kr/collaboroid/).
[7] P. Khlebovich, "IP Webcam," 2015, https://play.google.com/store/apps/details?id=com.pas.webcam\&hl=en.
[8] W. Morrison, "WiFi Speaker," 2013, https://play.google.com/store/apps/details?id=pixelface.android.audio\&hl=en.
[9] "Wifi Display (Miracast)," 2015, https://play.google.com/store/apps/details?id=com.wikimediacom.wifidisplayhelper\&hl=en.
[10] D. Thommes, Q. Wang, A. Gerlicher, and C. Grecos, "RemoteUI: A high-performance remote user interface system for mobile consumer electronic devices," in *Proc. IEEE Int'l Conf. Consumer Electronics (ICCE '15)*, 2012.
[11] Apple, "AirPlay," 2015, https://developer.apple.com/airplay.
[12] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making Smartphones Last Longer with Code Offload," in *Proc. Int'l Conf. Mobile Systems, Applications, and Services (MobiSys '10)*, 2010.
[13] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic Execution Between Mobile Device and Cloud," in *Proc. ACM European Conf. Computer Systems (EuroSys '11)*, 2011.
[14] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. the 31st IEEE Int'l Conf. Computer Communications (INFOCOM '12)*, 2012.
[15] S. Park, Q. Chen, H. Han, and H. Y. Yeom, "Design and Evaluation of Mobile Offloading System for Web-Centric Devices," *Journal of Network and Computer Applications*, vol. 40, pp. 105–115, 2014.
[16] E. E. Marinelli, *Hyrax: Cloud Computing on Mobile Devices using MapReduce*. Masters Thesis, Carnegie Mellon University, 2009.
[17] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling Remote Computing Among Intermittently Connected Mobile Devices," in *Proc. ACM Int'l Symp. Mobile Ad Hoc Networking and Computing (MobiHoc '12)*, 2012.
[18] D. Diaz-Sanchez, A. Lopez, F. Almenares, R. Sanchez, and P. Arias, "Flexible Computing for personal electronic devices," in *Proc. IEEE Int'l Conf. Consumer Electronics (ICCE '13)*, 2013.
[19] X. Wang, M. Chen, T. Kwon, L. Yang, and V. Leung, "AMES-Cloud: A Framework of Adaptive Mobile Video Streaming and Efficient Social Video Sharing in the Clouds," *IEEE Transactions on Multimedia*, vol. 15, no. 4, pp. 811–820, 2013.
[20] E. Toledano, D. Sawada, A. Lippman, H. Holtzman, and F. Casalegno, "CoCam: A collaborative content sharing framework based on opportunistic P2P networking," in *Proceedings of the 2013 IEEE Consumer Communications and Networking Conference*, ser. CCNC '13, 2013.
[21] A. Qureshi, D. Megías, and H. Rifà-Pous, "PSUM: Peer-to-Peer Multimedia Content Distribution Using Collusion-Resistant Fingerprinting," *Journal of Network and Computer Apps*, vol. 66, pp. 180–197, 2016.
[22] R. Steinmetz, "Human perception of jitter and media synchronization," *IEEE J. Selected Areas in Communications*, vol. 14, no. 1, 1996.
[23] H. Lee, J. Lee, H. Han, and S. Kang, "Mobile Platform Support for Remote Music Play," in *Proc. the 30th Annual ACM Symp. Applied Computing (SAC '16)*, 2016.
[24] D. Mills, J. Martin, J. Burbank, and W. Kasch, "Network time protocol version 4: Protocol and algorithms specification," Tech. Rep., 2010.
[25] Qualcomm Technologies, "Trepn Power Profiler," 2015, https://developer.qualcomm.com/software/trepn-power-profiler.