CollaDroid: Automatic Augmentation of Android Application with Lightweight Interactive Collaboration

Jiahuan Zheng[1,2], **Xin Peng**[1,2], **Jiacheng Yang**[1,2], **Huaqian Cai**[3], **Gang Huang**[3], **Ying Zhang**[3] and **Wenyun Zhao**[1,2]

[1]School of Computer Science, Fudan University, China

[2]Shanghai Key Laboratory of Data Science, Fudan University, China

[3]School of Electronics Engineering and Computer Science, Peking University, China

{*jiahuanzheng13,pengxin,jiachengyang15,wyzhao*}*@fudan.edu.cn*

{*caihq,hg*}*@pku.edu.cn,*{*zhangying06*}*@sei.pku.edu.cn*

ABSTRACT

Collaborative work supported by mobile applications has become more and more popular. Mobile collaboration in some cases needs to be conducted in an interactive way to allow the sharing of the requester screen with the collaborator. Existing interactive screen sharing techniques, however, may cause heavy network traffic and high latency and lack fine-grained control of the scope of collaboration. In this paper, we propose CollaDroid, a lightweight and UI Description based technique for interactive collaboration of Android applications. CollaDroid can automatically transform an Android application to a collaboration augmented application with which a requester can interactively collaborate with a remote collaborator by synchronizing UI (User Interface) content and events. The results of our experimental study show that CollaDroid is applicable for a large part of applications in the Android Market and can provide an efficient collaboration mechanism with low network traffic and latency. And the results of our user study show that the collaboration mechanism implemented by CollaDroid is well accepted by users.

Author Keywords

Mobile applications; Android; interactive collaboration; program transformation

ACM Classification Keywords

K.4.3. Organizational Impacts: Computer-supported collaborative work

INTRODUCTION

With the widespread use of mobile devices (e.g., smartphones), collaborative work supported by mobile applications has become more and more popular. Users can use mobile applications to collaboratively create and edit various kinds of artifacts such as UML sketch [16], comic strip [10]. They can also use mobile applications to leverage mobile crowdsourcing and participatory sensing for various purposes such as estimating individual exposure to environmental pollution [15], predicting bus arrival time [18], and obtaining map-based information [7].

Mobile collaboration in some cases needs to be conducted in an interactive way, which requires that the screen of the requester's device can be interactively shared with the collaborator and the collaborator's inputs can be returned back. For example, a user who is using a mobile application to book a product for a friend may want her friend to remotely input the shipping information (e.g., address, delivery time, contact information) for her. The user hopes that her friend can view the product details on her screen and edit those fields related to the shipping information in an interactive way. This requires that the current UI (User Interface) and interaction logic be displayed and executed on a remote device. Moreover, the user hopes that the remote collaboration can be requested in a flexible way to allow her to specify the scope of collaboration on the user interface and choose the collaborator from her social network.

A traditional solution for computer-based interactive collaboration is interactive screen sharing (e.g., remote desktops) [11, 3], with which computer screen contents can be transmitted to and displayed on remote computers and remote user inputs can be returned back. Interactive screen sharing usually involves very frequent screen image capturing, encoding, transmission, decoding, and display, and thus causes heavy network traffic and high latency. Moreover, interactive screen sharing grants the client full control of the whole screen of the host and lacks fine-grained control of the scope of collaboration (i.e., the part of the screen that the client can control).

In this paper, we propose CollaDroid, a lightweight and UI Description based technique for interactive collaboration of Android applications. CollaDroid can automatically transform an Android application packaged in an APK (Android Application Package) file to a collaboration augmented application supporting view-based interactive collaboration. With a collaboration augmented Android application, a requester can collaborate with a remote collaborator using the CollaDroid client. The requester can specify a collaboration

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

CSCW '17, February 25-March 01, 2017, Portland, OR, USA
© 2017 ACM. ISBN 978-1-4503-4335-0/17/03...$15.00
DOI: http://dx.doi.org/10.1145/2998181.2998278

scope by selecting a rectangular area on the screen and choosing a collaborator from a list of online users. The CollaDroid client generates a mock UI on the collaborator device and updates it based on the UI content received from the requester. The collaborator accomplishes the requested work by interacting with the mock UI and the CollaDroid client captures the UI events and sends them back to the requester. The collaboration augmented application replays the received UI events and sends updated UI content to the collaborator.

We conducted an experimental study and a user study to evaluate the applicability, efficiency, and usability of CollaDroid. The results show that: 1) CollaDroid is applicable for a large part of applications in the Android Market; 2) It can provide an efficient collaboration mechanism with low network traffic and latency; 3) The collaboration mechanism implemented by CollaDroid is well accepted by users.

BACKGROUND

An Android application is usually written in Java language and packaged in an APK file for installation. To make an APK file, the source code of an Android application is first compiled into Java class files. These class files are then converted into .dex format, which is a kind of executable byte code that can run on Android's Dalvik Virtual Machine, using the dx tool in the Android SDK. Finally, the Dalvik byte code (.dex files), manifest file, and related resources (e.g., images, UI layout, strings) are packaged into an APK file and signed for publishing.

Android provides four types of building components for application development, i.e., *Activity*, *Service*, *ContentProvider*, and *BroadcastReceiver*. A developer can implement application-specific components by inheriting from these basic components. An *Activity* component provides a screen that users can interact with. A *Service* component runs in the background and is often used to perform long-running operations. A *ContentProvider* component encapsulates the data managed by an application and provides a standard interface for other applications to access the data. A *BroadcastReceiver* component registers for and responds to system or application events.

Android defines a series of lifecycle methods for *Activity*, which will be invoked by the Android system during different phases of the lifecycle of an *Activity* component. The lifecycle methods involved in our approach include *onCreate*, *onResume*, *onPause*, and *onDestroy*. The method *onCreate* is invoked when an *Activity* component is created. It is usually used to set the UI layout and bind various listeners to the UI components. The method *onResume* is invoked when the UI of an *Activity* component is fully displayed on the screen. The method *onPause* is invoked when a foreground *Activity* component is obstructed by other visual components and the invocation usually indicates that the user is leaving the *Activity* component. The method *onDestroy* is invoked when an *Activity* component is destroyed and it marks the end of the lifecycle of an *Activity* component.

An activity object is an instance of an activity class (i.e., descendant class of *Activity*). When an activity object is created, a *PhoneWindow* object representing a visual window will be created and attached to it. A *PhoneWindow* object has a field called *mCallback* which refers to the corresponding activity object. When a user touches the screen, the Android system will capture the events and encapsulate event data into event objects. The PhoneWindow object will dispatch the event objects to the activity object by invoking its *dispatchTouchEvent* method. We can thus add additional behaviors for collaboration lifecycle management (e.g., initializing collaboration request, specifying collaboration scope) to an activity object by adding code into its *dispatchTouchEvent* method.

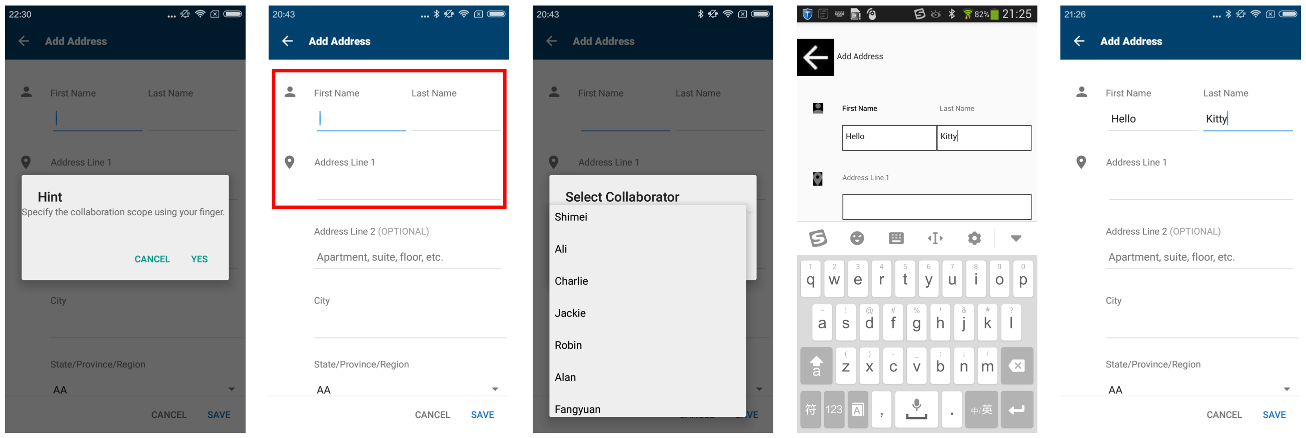
In Android, UI components such as *Button*, *EditText* are all subclasses of *View*. *ViewGroup* is a special view that can contain other views and *DecorView* a special view group that holds a windows background drawable. A *PhoneWindow* object has a field called *mDecor*, which refers to a *DecorView* object containing all the children views of the window. We can thus traverse the *DecorView* object of a *PhoneWindow* object to obtain all the children views and their layout.

OVERVIEW

CollaDroid provides a lightweight and view-based mechanism for interactive collaboration. It requires that the requester of collaboration uses a collaboration augmented Android application and the collaborator installs a CollaDroid client (an Android application) on her device as a daemon for collaboration request.

With a collaboration augmented Android application, a user can initiate a remote collaboration request when she is using the application by long pressing the screen. She can specify the scope of collaboration by selecting a rectangular area on the screen and then choose a collaborator from a list of online users obtained from a yellow page service. The CollaDroid client on the collaborator's device will receive the collaboration request and prompt the collaborator to accept or decline the request. If the request is accepted, the CollaDroid client will generate and show a mock UI based on the UI content extracted from the requester side. The views in the scope of collaboration can be manipulated by the collaborator while the other views on the UI are read only for the collaborator. The collaborator then can interactively manipulate the views in the scope and the CollaDroid client will capture the UI events and send them back to the requester side. The application at the requester side will receive the UI events and replay them on the current UI. After that the application will extract the updated UI and send the UI content to the collaborator side. The CollaDroid client will receive the UI content and update the collaboration UI accordingly. The collaboration process continues until the requester ends the process by clicking the Back Button. The runtime collaboration mechanism will be detailed in a later section (Collaboration Mechanism).

Figure 1 shows a collaboration augmented Android application for online shopping and the mock UI generated by the CollaDroid client on collaborator device. When a user long presses the UI for filling in shipping information, a dialogue will pop up (see Figure 1(a)) to prompt her to specify the collaboration scope. She can then specify the scope of collaboration by selecting a rectangular area as shown in Fig-



(a) Collaboration Prompt (b) Collaboration Scope (c) Collaborator Select (d) Mock UI (e) Shipping Info. UI

Figure 1. An Example of Collaboration Augmented Android Application and Mock UI on Collaborator Device

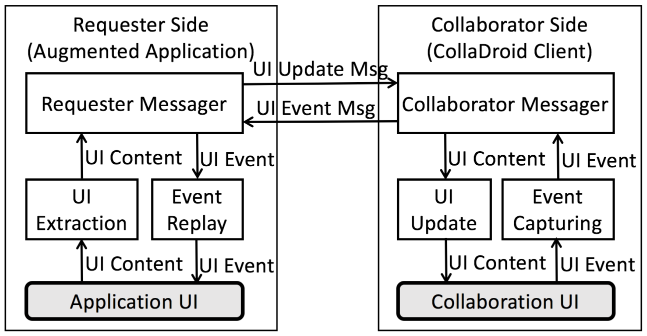


Figure 2. Runtime Collaboration Mechanism

ure 1(b). Then a dialogue will pop up (see Figure 1(c)) to prompt the user to select a collaborator from a list of online users. A mock UI generated by the CollaDroid client will be displayed on the collaborator device as shown in Figure 1(d). During the collaboration, the shipping information UI on the requester device will be updated by replaying the events received from the collaborator side (see Figure 1(e)).

To augment an Android application with interactive collaboration, CollaDroid implements a program transformation technique that can automatically convert an Android application into a collaboration augmented one. The technique can automatically analyze and rewrite the Dalvik byte code contained in an APK file. It augments an Android application from two aspects. First, it weaves additional behaviors into the lifecycle methods (e.g., *onCreate*, *onResume*), *dispatchTouchEvent* method, and associated view listener methods of each activity class to manage collaboration lifecycle, extract UI content, and replay collaborator side events. Second, it adds additional code for communicating with the yellow page service and the CollaDroid client. The program transformation technique will be detailed in a later section (Program Transformation).

COLLABORATION MECHANISM

CollaDroid involves a collaboration augmented application at the requester side and a CollaDroid client at the collaborator side. Both sides are supported by Android powered devices.

The CollaDroid server provides a yellow page service for mobile users to obtain candidate collaborator lists. To this end, the CollaDroid client periodically (e.g. once 1 minute) sends heartbeat messages to the server to indicate its availability with the user name and IP address. When a user initiates a collaboration with an augmented application, the application can request a candidate collaborator list from the server. The server will select and return a list from all the available users based on predefined conditions, for example all the friends or colleagues of the requester.

Figure 2 presents an overview of the runtime collaboration mechanism of CollaDroid, which shows the interactions within and between the requester side and the collaborator side. When a requesters collaboration request is accepted by a collaborator, the UI content of the current window of the application will be extracted and encoded into a UI update message object. This message object is sent to the CollaDroid client at the collaborator side by an asynchronous thread using a socket connection. The CollaDroid client then decodes the UI content in the message and generates a mock UI based on the received UI content. When the collaborator manipulates the mock UI, the CollaDroid client captures the UI events and encodes them into a UI event message object. This message object is sent back to the augmented application at the requester side by an asynchronous thread using a socket connection. The application then decodes the UI events in the message and replay the received events on the current window of the application. The updated UI content is then extracted and sent to the collaborator side and the collaboration process continues until the requester ends it.

Requester Side

Requester side collaboration mechanism includes collaboration lifecycle management, UI content extraction, and event replay.

Lifecycle Management

An augmented application manages the collaboration lifecycle of each activity object at the requester side as shown in Figure 3.

The initial state **NormalExecution** means that an activity object is in its normal execution state without collaboration. The collaboration state of an activity object is set to **NormalExecution** when the object is created.

The state **SpecifyingScope** means that the user is specifying the collaboration scope. The collaboration state of an activity object is set to **SpecifyingScope** right after a long pressing event is detected and the user confirms to initiate a collaboration request. During this state, the user selects a rectangular area on the screen as the collaboration scope.

The state **ChoosingCollaborator** means that the user is choosing the collaborator. The collaboration state of an activity object is set to **ChoosingCollaborator** right after the collaboration scope is specified. During this state, the user chooses a collaborator from a list of online users.

The state **ExtractingUIContent** means that the application is extracting the UI content of the current screen. The collaboration state of an activity object is set to **ExtractingUIContent** right after the collaborator is chosen. During this state, the application extracts the UI content of the current screen and encodes it into a UI update message object for sending to the collaborator.

The state **WaitingForUpdate** means that the application is waiting for the collaborator to manipulate and update the current screen. The collaboration state of an activity object is set to **WaitingForUpdate** right after the UI content of the current screen is extracted and sent out or the current activity object is resumed.

The state **ReplayingEvents** means that the application is replaying the UI events received from the collaborator. The collaboration state of an activity object is set to **ReplayingEvents** right after a UI event message is received from the collaborator. During this state, the application is replaying the received UI events on the current screen.

The state **CollaborationPaused** means that the collaboration is currently being paused. The collaboration state of an activity object is set to **CollaborationPaused** right after the current activity object is paused (e.g., when the user presses the HOME button).

When an activity object is in the **WaitingForUpdate** state, the user can end the collaboration by clicking a Back Button and the collaboration state of the activity object will be restored to its initial state **NormalExecution**.

UI Extraction

The purpose of UI extraction is to extract the UI content of the current screen on the requester device for the CollaDroid client to generate or update the mock UI on the collaborator device. Note that although the current screen is captured and shown at the collaborator side, only the views within the collaboration scope are editable and the rest are read-only context for the collaborator. UI extraction is performed when a collaboration is started or the current screen is updated by event replaying (see Figure 3).

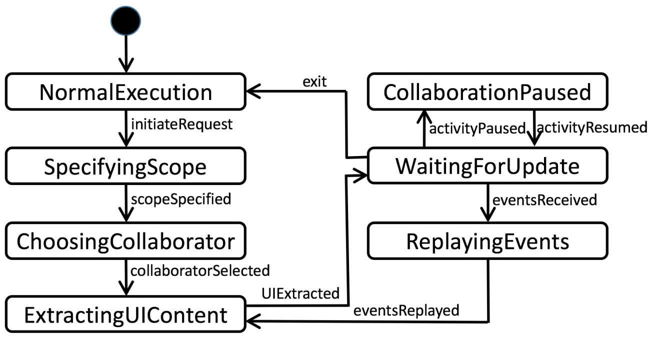


Figure 3. Requester Side Collaboration Lifecycle

As mentioned in the *Background* section, we can obtain a *DecorView* object from the *mDecor* field of the *PhoneWindow* object associated with the current activity object. This *DecorView* object contains all the children views of the current window organized in a tree structure. We can then extract the UI content by traversing all the views contained in the *DecorView* object. For each view, we can obtain a series of properties such as view type, width, height, visibility. To enable the retrieval of a view by ID in event replay, we assign a unique ID to each view within the collaboration scope if it does not have one. To adjust the views for different mobile devices, we also obtain screen properties such as its pixel density, width, and height. These screen properties are used to compute the scale the views on the collaborator device. The properties of all the views and the screen properties are extracted as the UI content and represented by a JSON object. An example of JSON object for UI content is shown as follows.

```
{
   "reqWidth":720,"reqHeight":1280,"reqxdpi":345.1,
   "reqydpi":342.2,"scope":"[213158, 3489761]",
   "viewInfo":{ [
                 {
                    "abstract type":"TextView",
                    "type":"android.widget.EditText",
                    "id":23487632,"x":136,"y":242,
                    "width":592,"height":242,
                    "isFocused":false,"visibility":0,
                    "text":"First Name","text size":20
                 },
                    ...
              ]}
}
```

This JSON object records the screen width (*reqWidth*), height (*reqHeight*), the dots per inch in horizontal direction (*reqxdpi*) and in vertical direction (*reqydpi*), and the IDs of the views within the collaboration scope (*scope*). After these screen information, the JSON object records the properties of each view including general properties (e.g., *type*, *id*, *x*, *y*, *width*, *height*) and properties specific to different types (e.g., *text*, *text size* for *TextView*).

A JSON object of UI content together with the collaborator's name and IP address are encapsulated into a message object, which will be put into a message pool and later sent to the collaborator.

Event Replay

The purpose of event replay is to replay the events received from the collaborator to obtain the inputs and respond to the operations of the collaborator. Event replay is performed when an event is received from the collaborator (see Figure 3).

The received event can be a touch event or a content change event. A touch event (i.e., an instance of *MotionEvent*) represents a touch operation on the screen. To replay a touch event, we need to dispatch the event to the current activity object. A content change event represents a change of the content of a view such as *EditText* and *Spinner*. To replay a content change event, we need to set the content of the corresponding view according to the received content.

UI event dispatch and view content update can only be executed by the UI thread in Android. As the UI thread is not thread safe, we need to pass a runnable object to the UI thread to replay the event. To this end, the application invokes the *runOnUiThread* method of the current activity object with a runnable object as the parameter. UI event dispatch and view content update are implemented in the *run* method of the runnable object. For a touch event, the *run* method extracts the required information from the received event message and restore it into a *MotionEvent* object. It then invokes the *dispatchTouchEvent* method of the current activity object with the *MotionEvent* object as the parameter. For a content change event, the *run* method uses the *DecorView* object associated with the current activity object to find the target view by its ID and then update it with the received content.

Collaborator Side

Collaborator side collaboration mechanism includes lifecycle management, UI update and event capturing.

Lifecycle Management

The CollaDroid client manages its collaboration lifecycle at the collaborator side as shown in Figure 4.

The initial state **Idle** means that the client is idle and waiting for collaboration request. When the collaborator receives and accepts a collaboration request, the client enters the **GeneratingUI** state, during which it generates a mock UI based on the received UI content. Right after the mock UI is generated, the client enters the **UserInteraction** state. In this state, the collaborator accomplishes the requested work by interacting with the mock UI and the client captures UI events and sends them back to the requester. When the client receives a UI update message from the requester, it enters the **UpdatingUI** state, during which it updates the mock UI based on the received UI content. Right after the mock UI is updated, the client restores to the **UserInteraction** state. When the client is paused (e.g., when the user presses the HOME button), it

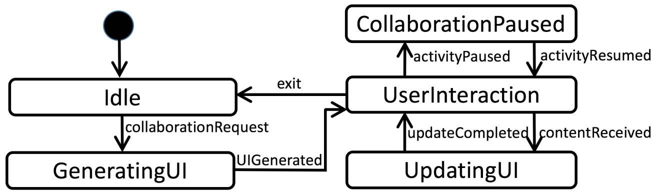


Figure 4. Collaborator Side Collaboration Lifecycle

enters the **CollaborationPaused** state. When the client is resumed, it restores to the **UserInteraction** state. When the client is in the **UserInteraction** state, the user can end the collaboration by clicking the Back Button and the client will restore to its initial state **Idle**.

UI Generation and Update

Once the collaborator accepts the collaboration request and the CollaDroid client receives a UI update message from the requester, the client generates a mock UI on a full-screen *RelativeLayout* (a subclass of *ViewGroup*) object. After that, the client updates the mock UI each time when a UI update message is received from the requester.

To generate a mock UI, the CollaDroid client restores a JSON object from the UI update message and extracts all the views contained in the JSON object. For each view the client generates a view object based on the received view properties such as type, width, height, and coordinates. Note that when the client generates a view the width, height, and coordinates of the view are scaled according to the different screen width, height, and pixel density of the requester device and the collaborator device. The client adds each generated view to the layout by invoking the *addView* method of the *RelativeLayout* object.

The update of a mock UI is done in a similar way to the generation of a mock UI. The difference is that the CollaDroid client first removes all the existing views from the current layout and then generates new view objects based on the received UI content and adds them to the current layout.

An exception is that the *EditText* view that the collaborator is editing is not removed but all its properties except the text are updated. As an *EditText* view that is under editing usually involves soft keyboard on the screen, regenerating the view may cause the change of the soft keyboard and thus influence the experience of the collaborator.

Event Capturing

In the **UserInteraction** state, the CollaDroid client captures UI interaction events.

Touch events (i.e., instances of MotionEvent) are captured in the *dispatchTouchEvent* method of the current activity object. For a captured touch event, the client determines its coordinates. If the touch is on a view within the collaboration scope, the client records its properties such as coordinates and event type. Otherwise, the touch is ignored. Note that before recording the coordinates of a touch event are scaled according to the different screen width, height, and pixel density of the requester device and the collaborator device. CollaDroid does not support multi-touch event currently. Extending CollaDroid to support more event types is our future work.

To capture events related to the change of content of views such as *EditText* and *Spinner*, the CollaDroid client adds a listener to each of this kind of views when it is generated. When the content of a view is changed, the listener captures the event and records its properties such as text.

The properties of a captured UI event are represented by a JSON object. Two examples of JSON object for UI event are shown as follows.

```
{
    "x":413, "y":1095, "downTime":8352678,
    "eventTime":8352703, "action":1,
    "metaState":1, "edgeFlags":0,
    "xPrecision":1, "yPrecision":1
}
{
    "id":28356714, "text":"Zheng"
}
```

The first JSON object records a touch event with a series of properties such as scaled coordinates (*x*, *y*), the time (in ms) when the user presses down (*downTime*), the time (in ms) when the event is generated (*eventTime*). The second JSON object records a content change event specifying that the text of a *EditText* view is changed to "Zheng".

A JSON object of UI event together with the requester's name and IP address are encapsulated into a message object, which will be put into a message pool and later sent to the requester.

PROGRAM TRANSFORMATION

The application transformer of CollaDroid takes as input an Android APK file and produces a collaboration augmented Android application. The Dalvik byte code contained in an APK file preserves the object-oriented program structure of the application, including classes, methods, and relationships between classes. The transformer automatically analyzes and rewrites the Dalvik byte code to weave collaboration related behaviors (e.g., collaboration lifecycle management, UI extraction, event replay) into the application.

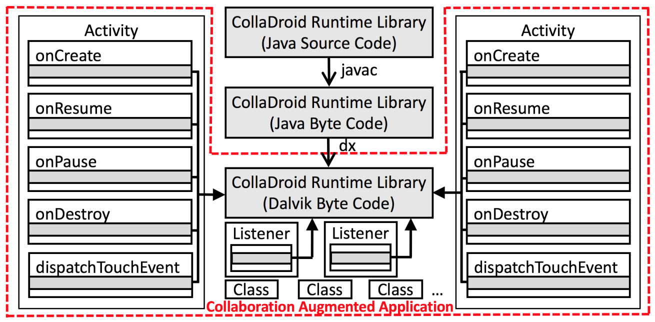

Activity
onCreate
onResume
onPause
onDestroy
dispatchTouchEvent
CollaDroid Runtime Library (Java Source Code)
javac
CollaDroid Runtime Library (Java Byte Code)
dx
CollaDroid Runtime Library (Dalvik Byte Code)
Listener
Listener
Class
Class
Class
...
Collaboration Augmented Application
Activity
onCreate
onResume
onPause
onDestroy
dispatchTouchEvent

Figure 5. Overview of CollaDroid Program Transformation

Figure 5 presents an overview of CollaDroid Program transformation. In the figure, the white parts mean the code in the original Android application, while the gray parts mean the code introduced by the transformer. The transformer uses dex2jar[1], a tool to work with Android .dex and Java .class files, to parse the Dalvik byte code in an APK file and manipulate the activity classes in it. For each activity class, the transformer introduces additional code for collaboration lifecycle management into the lifecycle methods (e.g., *onCreate*, *onResume*), *dispatchTouchEvent* method, and associated view listener methods of it. These code invoke the CollaDroid runtime library to implement collaboration related functionalities such as gesture detection, collaboration confirmation, UI extraction, event replay, and communication with client. The CollaDroid runtime library is first implemented in Java and then converted into Java byte code using java and Dalvik byte code using dx. All the revised activity classes, the CollaDroid runtime library (Dalvik byte code), and all the other classes in the original Android application are packaged into an augmented Android application (also an APK file) together with the original manifest file and resources. In the rest of this section, we will detail the transformation of activity classes and listener classes.

Table 1. Transformation of Activity Classes and Listener Classes

Method	Additional Code
Activity .onCreate	set collaboration state; create and associate gesture detector; bind UI layout update listener
Activity .onResume	set collaboration state; set the current activity object as the collaboration activity
Activity .onPause	set collaboration state
Activity .onDestroy	remove the current activity object from the collaboration management
Activity .dispatchTouchEvent	set collaboration state; detect gesture operations to initiate collaboration request and select collaboration scope; pop up collaborator choosing dialog; discard touch events when the collaboration state is **SpecifyingScope**, **ExtractingUIContent**, or **WaitingForUpdate**
view listener methods	extract and send UI content to the collaborator

The methods of activity classes and listener classes that are involved in program transformation are listed in Table 1. For the lifecycle methods of each activity class, the introduced code sets the collaboration state, binds UI layout update listener, and manages the collaboration activity object. For the *dispatchTouchEvent* method, the introduced code detects user gesture, manages the initiation of a collaboration process, and sets the collaboration state accordingly. Moreover, it discards touch events when the collaboration state of the current activity object is **SpecifyingScope**, **ExtractingUIContent**, or **WaitingForUpdate** to disable the UI operations of the requester (the UI will be updated by event replay when a UI event is received from the collaborator). View listener methods are the event response methods defined in the listener classes bound to the content change events of the views within the collaboration scope. These methods will be invoked when the content of corresponding view is changed. For example, for an *EditText* view, the transformation involves the *afterTextChanged* method of the listener class that implements the

[1]https://code.google.com/p/dex2jar

TextWatcher interface and bound to the view. For a *CheckBox* view, the transformation involves the *onCheckedChanged* of the listener class that implements the *OnCheckedChangeListener* interface and bound to the view. The introduced code to these methods extracts the updated UI content and sends a UI update message to the collaborator. To ensure that all the content changes of the views in the collaboration scope are captured, the transformer binds a UI layout update listener to the DecorView object associated to the current activity object in its onCreate method. This listener will be invoked when the layout of the current window changes (e.g., when the window is redrawn or new views are added). It checks each of the new views and if it has no content change listener binds a view listener to it to capture its content change.

```
  public class ExampleActivity extends Activity {
      public boolean onCreate() {
+         ViewRepository.invokedInOnCreate(this);
          // Original Codes …
      }
  // If the class doesn't have onResume Method,
  // CollaDroid will add it automatically.
+     public boolean onResume() {
+        ViewRepository.invokedInOnResume(this);
+     }
      // …
  }
```

Figure 6. An Example of Program Transformation of Activity Lifecycle Methods

Figure 6 shows an example of program transformation of activity lifecycle methods. Note that the transformation is actually performed on Dalvik byte code and here we show the decompiled Java source code for convenience and clarity. In the figure, the lines starting with "+ " are the code introduced in program transformation. In the *onCreate* method, the program transformer adds a method call to *ViewRepository.invokedInOnCreate* with the current activity object as the argument. The invoked method is defined in the CollaDroid runtime library, which implements collaboration functionalities such as setting collaboration state, binding UI layout update listener as shown in Table 1. Similarly, the transformer adds a method call in the *onResume* method. As this method is not defined in the original activity class, the transformer adds a complete *onResume* method to the current activity class.

```
  public class ExampleActivity extends Activity {
      public boolean dispatchTouchEvent(MotionEvent e) {
+         ViewRepository.interceptDispatchTouchEvent(this, e);
+         if (ViewRepository.isUIOperationEnabled(this)){
              // Original Codes
+         }else{
+             return true;
+         }
      }
  }
```

Figure 7. An Example of Program Transformation of *dispatchTouchEvent* Method

Figure 7 shows an example of program transformation of *dispatchTouchEvent* method. The transformer adds a method call to *ViewRepository.interceptDispatchTouchEvent* with the current activity object and the event object as the arguments. The invoked method is defined in the CollaDroid runtime library, which captures user gesture, initiates collaboration request, and sets the collaboration state accordingly (see Table 1).

The transformer also adds a branch statement to determine whether the UI operation is enabled currently. If not (e.g., when the current collaboration state is **WaitingForUpdate**), the current event will be discarded (see Table 1).

EVALUATION

To evaluate the applicability, efficiency, and usability of CollaDroid, we conducted a series of experiments and a user study to answer the following research questions.

- RQ1 (applicability): Can CollaDroid automatically transform off-the-shelf Android applications into reliable collaboration augmented applications? Is it widely applicable for the applications in the Android Market?
- RQ2 (efficiency): Can CollaDroid ensure an acceptable response time and network traffic for collaboration augmented applications under various network conditions? Is CollaDroid more efficient than existing screen sharing techniques?
- RQ3 (usability): Can the users accept the collaboration mechanism implemented by CollaDroid? How is the user experience of the collaboration augmented Android applications and the CollaDroid client?

RQ1: Applicability

To answer RQ1, we collected eight Android applications (see Table 2) and tested CollaDroid on them. For each application, we tried to use the CollaDroid transformer to transform it into a collaboration augmented application. To validate the collaboration augmented applications, we recruited two master students to test all the transformed applications to confirm whether they can be reliably used for view-based collaboration together with the CollaDroid client. The Android devices used in our testing included a Mi Mi2S (Snapdragon 600 CPU, Adreno320 GPU, 2GB RAM, 1280 × 720 pixel IPS LCD) as the requester device and a Samsung Galaxy S4 (Exynos 5410 CPU, Imagination PowerVR SGX544 MP3 GPU, 2GB RAM, 1920 × 1080 pixel Super AMOLED HD) as the collaborator device.

The results of applicability evaluation are shown in Table 2. All these eight applications are well known and widely used. Among them WPS is an office suite, and the others are online shopping applications. The transformation of each application was completed in about 30 seconds. The collaboration augmented versions of two applications (TaoBao 5.7.2 and JingDong 5.1.0) failed in the testing. They threw an exception or refused to execute when they started up. The failures were due to the self-preservation mechanisms of the original applications. The other six collaboration augmented applications can be used normally. The size of these applications (by .dex files) increased by 0.8% to 13.13% after transformation. There were two applications (Amazon 5.51.6710 and SFBest 3.1.4) whose size increased by more than 10%. The cause is

Table 2. Results of Applicability Evaluation

Application	Test Results					Size Change
Amazon 5.51.6710	Main Window	Login Window	Commodity Classification Window	Product Display Window	Shopping Cart Window	8,445KB ⇒ 9,342KB (+10.6%)
	✓	×	✓	✓	×	
YiHaoDian 4.1.7	Main Window	Search Window	Single Class Commodity Display Window	Product Display Window	Commodity Classification Window	9,464KB ⇒ 9,629KB (+1.74%)
	✓	✓	✓	✓	✓	
DangDang 6.1.3	Main Window	Discovery Window	Search Window	Personal Info Window	Commodity Instruction Window	9,750KB ⇒ 10,109KB (+3.68%)
	✓	✓	✓	✓	✓	
SFBest 3.1.4	Main Window	Personal Info Window	Login Window	Search Window	Commodity Classification Window	10,387KB ⇒ 11,751KB (+13.13%)
	✓	✓	✓	✓	✓	
WPS 9.7.1	Main Window	Personal Info Window	Login Window	My Task Window	Commodity Classification Window	29,337KB ⇒ 29,573KB (+0.8%)
	✓	✓	✓	×	×	
NewEgg 4.0.2	Shipping Address Window	Main Window	My Account Window	Wish list Window	Order History Window	11,210KB ⇒ 11,462KB (+2.25%)
	✓	✓	✓	✓	✓	
TaoBao 5.7.2	×					×
JingDong 5.1.0	×					×

that their original versions did not completely include the Android Support V4 Library, which is required by CollaDroid for gesture detection, so the CollaDroid transformer added missing classes of this library to the collaboration augmented applications.

For each of the six collaboration augmented applications, we chose five different UI windows and tested the collaboration mechanism on these windows. From Table 2 we can see that the collaboration mechanism worked well on all the UI windows except four (e.g., the Login Window and Shopping Card Window of Amazon 5.51.6710). The failures of the collaboration mechanism were caused by Android *WebView* (a view that displays web pages), which is currently not supported by CollaDroid. When a requester initiates a collaboration request on a *WebView* area, the CollaDroid client can not generate corresponding views on the collaborator device.

These results show that CollaDroid can automatically transform off-the-shelf Android applications into reliable collaboration augmented applications and it is applicable for a large part of applications in the Android Market. CollaDroid works well with Android application windows that are not implemented by *WebView*. For the failures caused by self-preservation mechanisms, we think they can be solved if the developers perform the CollaDroid transformation before they add the self-preservation mechanisms into applications.

RQ2: Efficiency

To answer RQ2, we tested the network traffic and response time of the collaboration supported by CollaDroid. We used the Mi Mi2S phone and the Samsung Galaxy S4 phone mentioned in the previous subsection as the requester device and collaborator device respectively, and a Dell server (PowerEdge R210 II, Xeon E3-1220 V2, 8GB RAM) as the yellow page and messaging server.

The testing of network traffic was conducted on YiHaoDian 4.1.7. We tested 10 different windows of it and recorded the network traffic during the collaboration process. This was done by obtaining collaboration related packets and measuring their size. For each window, we ran the collaboration process 20 times and calculated the average data size of UI event messages and UI update messages.

The testing results show that the data size of a touch event message is around 300 bytes and the data size of a content change event is about 155 bytes, which are very small. The data size of a UI update message is much bigger than that of a UI event message. The data size of UI update messages for different windows are shown in Table 3. It can be seen that most of the data size is smaller than 20 KB and there are only three windows whose UI update data size is bigger than 20 KB. The data size highly depends on the number and size of images contained in a window. Note that a UI update message will only be sent when a collaboration is initiated or right after the requester UI is updated by event replaying. Moreover, an image contained in a window can be sent to the collaborator only when it is updated

The above results and analysis show that the response time and network traffic of the collaboration process implemented by CollaDroid are acceptable.

Table 3. Data Size of UI Update Messages

Window	Data Size (kB)
Login Window	11.6
Sign Up Window	19.1
Profile Window	44.0
Comment Window	23.2
Addresses Window	10.1
Comment Window	43.6
Add Address Window	17.8
Credit Window	16.0
Feedback Window	8.7
Recharge Window	15.0

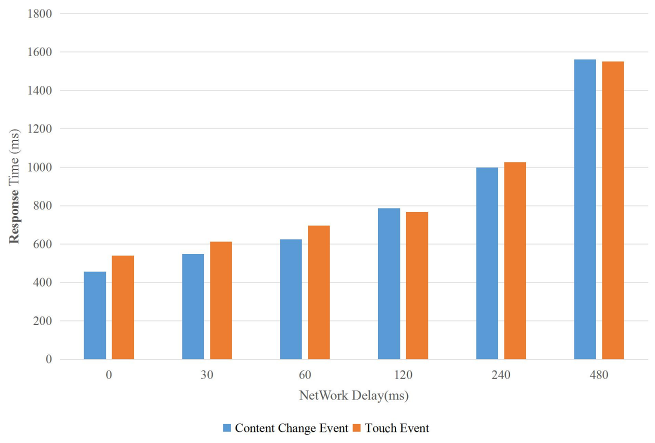


Figure 8. Response Time under Different Network Conditions

The testing of response time used the login window of YiHaoDian 4.1.7 as the subject and followed a three-step collaboration process: the CollaDroid client captures a UI event at the collaborator side and sends a UI event message back to the requestor side going through the message server; the collaboration augmented application receives the message and replays the event at the requester side and sends a UI update message to the collaborator side; the CollaDroid client receives the message and updates the mock UI at the collaborator side. Note that the response time for the collaborator side is more important, as the collaborator needs to interact with the mock UI while the requester can just wait for the content update during the collaboration process. Therefore, we collected and evaluated the end-to-end response time of the collaboration process from the collaborator side, i.e., the time between UI event capturing and mock UI update, for touch event (e.g., the click of checkbox) and content change event (e.g., the change of text input) respectively.

To simulate different network conditions, we used the Linux TC (Traffic Control) tool on the server to control the traffic and simulate different network delays (from 0 ms to 480 ms). Under each network condition, we ran the collaboration process 20 times and calculated the average response time for touch event and content change event respectively. The results of response time testing are shown in Figure 8. It can be seen that the response time was around or lower than one second under most of the network conditions for both touch event and content change event. We think this latency is acceptable for our collaboration scenario.

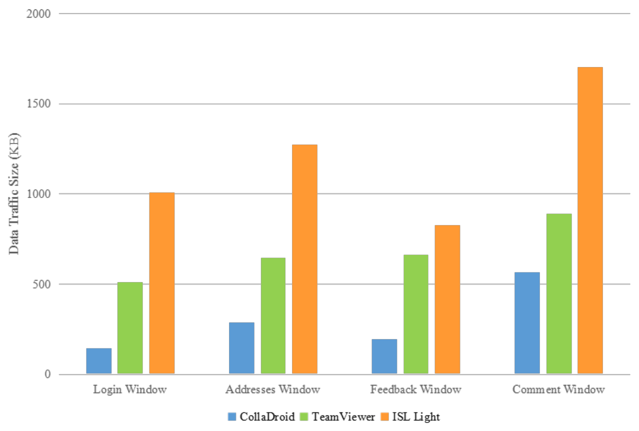


Figure 9. Comparison of Data Traffic Consumption

To compare CollaDroid with screen sharing techniques, we chose two popular remote collaboration tools that have Android clients: TeamViewer[2] and ISL Light[3]. Currently TeamViewer and ISL Light do not support collaboration between two Android devices but only support collaboration between Android device and computer, so we tested them with a mobile phone as the requester device and a computer as the collaborator device. For TeamViewer, we installed TeamViewer QuickSupport 11.0.4368, a TeamViewer client for Android, on the Mi Mi2S phone. For ISL Light, we installed its 3.0.0 version of Android client on the Mi Mi2S phone. For both of them, we used a computer equipped with Windows 8.1 and an Intel i5 3230M processor (2.6 GHz), 8 GB RAM, and a 256 GB SSD as the collaborator device. We installed TeamViewer 11 and ISL Light 4.1.1 on the computer. For CollaDroid, we used the Mi Mi2S phone and the Samsung Galaxy S4 phone as the requester device and collaborator device respectively. We chose four windows of YiHaoDian 4.1.7 (i.e., login, address, feedback, comment) to compare the efficiency of CollaDroid, TeamViewer, and ISL Light. For each window, the same collaborator was requested to complete exactly the same collaboration task 10 times with the same speed with CollaDroid, TeamViewer, and ISL Light, respectively. A traffic monitor tool named MyTraffic was installed on the requester device (i.e., the Mi Mi2S phone) to calculate the data traffic of TeamViewer and ISL Light during the collaboration process. Figure 9 shows the average data traffic of CollaDroid, TeamViewer, and ISL Light for each window. From the result, it can be seen that LSI Light costs the highest data traffic and CollaDroid is much more efficient than the two screen sharing techniques by data traffic.

RQ3: Usability

To answer RQ3, we recruited 15 master students and conducted a user study with them to evaluate and compare the usability of CollaDroid, TeamViewer, and ISL Light. The subject application is the collaboration augmented version of YiHaoDian 4.1.7.

[2]https://www.teamviewer.com/en/use-cases/remote-support/
[3]http://www.islonline.com/remote-support/

The participants were trained to use the collaboration augmented application and the CollaDroid client for collaboration. The training session lasted 10 minutes, during which two of the authors gave a short tutorial and answered the questions raised by the participants. After that, the participants were asked to use the collaboration mechanism implemented by CollaDroid in a five-minute session. They could try the collaboration mechanism on any window of the application. For each try, they first used the collaboration augmented application on the Mi Mi2S phone to initiate a collaboration request and then used the CollaDroid client on the Samsung Galaxy S4 phone to accomplish the collaboration.

Similar training sessions were conducted for TeamViewer and ISL Light. The participants were asked to use TeamViewer QuickSupport 11.0.4368 on the Mi Mi2S phone as requesters and use TeamViewer 11 on the computer mentioned in the previous subsection as collaborators to try the same collaboration tasks in a five-minute session. They were also asked to use ISL Light Android client 3.0.0 on the Mi Mi2S phone as requesters and use ISL Light 4.1.1 on the computer as collaborators to try the same collaboration tasks in a five-minute session.

After the participants finished all the sessions, they were asked to complete a questionnaire to provide feedback on the usability of the three collaboration mechanisms by rating the following six statements.

S1 (requester): The whole process of initiating a collaboration request and waiting for the inputs from the collaborator is smooth and the UI is responsive.

S2 (requester): I feel the whole collaboration process is under my control and I do not think the collaboration mechanism has adverse impact on my privacy and other aspects.

S3 (requester): I am willing to request the collaboration of others using this technique when I need others' help in this way in real life.

S4 (collaborator): When receiving a collaboration request I can quickly understand the request and start my work.

S5 (collaborator): The UI of the collaboration client is clear and responsive, and I can finish a collaboration task in a similar way to a local application.

S6 (collaborator): I am willing to help if someone else requests collaboration in this way in real life.

The first three of the statements are from the perspective of the requester and the last three are from the perspective of the collaborator. The participants were asked to rate each of the statements on a scale of 1 (strongly disagree) to 5 (strongly agree). We also conducted a group interview to get more feedback and clarify some problems.

The feedback of the participants on the six statements is summarized in Table 4. For each statement, we list the median (M), mean ($\overline{x}$), and variance (σ^2) of the scores of the participants. From the table, it can be seen that Colladroid was rated much higher than ISL Light on all the statements. For TeamViewer, it had comparable rating with that of CollaDroid on S1 and S5. In other words, the participants agreed that both TeamViewer and CollaDroid provide clear and responsive UI for remote collaboration. Colladroid was rated much higher than TeamViewer on all the other statements. The participants's concerns about TeamViewer mainly lie in its security and privacy. When using TeamViewer, the collaborator takes full control of the requester's device, so the participants worried that the requester's private information may be observed or modified by the collaborator. For example, if the requester's device receives a message during the collaboration process, the collaborator may see the content of the message. Moreover, the participants worried that installing and using TeamViewer on the requester's device may require very high Android permission (e.g., root privilege).

They were also asked the question: what approaches will you use to obtain the collaboration of others without these collaboration mechanisms. All the participants said that they would use instant messaging tools such as WeChat and WhatsApp. But they also mentioned that these tools could provide neither the required application context (e.g., the state of the application at the requester side) nor the interactive user interface for the collaborator. Four participants said that they would also make a telephone call and two said that they would also send short messages to obtain the collaboration of others.

In the questionnaire and the group interview the participants were also asked to summarize their experience on the collaboration mechanism and suggest possible improvement of it. Seven participants mentioned that the collaboration mechanism implemented by CollaDroid was convenient and easy to use. Two of them emphasized that users with limited knowledge of mobile applications such as the elderly can easily ask others to remotely help them to complete complex form using CollaDroid. Four participants said that it was very easy to learn how to use the CollaDroid collaboration mechanism. Two participants mentioned that the UI update speed is fast and they felt no noticeable delay during the interactive collaboration process.

Regarding the improvement of the collaboration mechanism, eight participants mentioned that the way of specifying collaboration scope can be further improved. They said in some cases they need to try several times to fully include all the desired views in the rectangular area. One participant suggested to support the selection of multiple areas on the screen as the collaboration scope. One participant suggested to provide more options for privacy setting, for example, to allow to make the views out of the collaboration scope invisible to the collaborator.

The above results show that the collaboration mechanism implemented by CollaDroid can be well accepted by users. The expected improvements of CollaDroid are mainly on more user-friendly approach for specifying collaboration scope and more options of privacy settings.

RELATED WORK

Our work is related to mobile collaboration and program transformation of Android applications.

Table 4. Usability Comparison of CollaDroid, TeamViewer, and ISL Light

	S1			S2			S3			S4			S5			S6		
	M	$\bar{x}$	σ^2	M	$\bar{x}$	σ^2	M	$\bar{x}$	σ^2	M	$\bar{x}$	σ^2	M	$\bar{x}$	σ^2	M	$\bar{x}$	σ^2
CollaDroid	5	4.57	0.26	5	4.86	0.13	5	4.86	0.13	4	4.36	0.4	4	4.29	0.37	5	4.64	0.25
TeamViewer	4	4.14	0.44	2	2.14	0.44	3.5	3.29	0.99	4	3.64	0.25	4	4.43	0.26	4	4	0.62
ISL Light	2	2.36	0.55	2.5	2.64	1.17	2	2.14	0.75	3	3.21	0.49	2	2.5	1.04	3	2.57	0.88

JAMM [2] is a Java runtime environment that supports the shared use of existing Java applets for synchronous collaboration. It is based on a replicated architecture, where each user maintains a copy of the Java applet, and the users input events are broadcast to each applet copy. JAMM is implemented by modifying the Java Development Kit (JDK). CollaDroid does not modify the runtime environment but automatically converts an Android application into a collaboration augmented one. Moreover, CollaDroid does not require the collaborator to maintain a copy of the app but generates a mock UI for the collaborator.

There have been researches on collaborative work on mobile devices. Ah Kun and Marsden [1] developed a mobile application that allows users to share photos with other co-present users by synchronizing the display on multiple mobile devices. BeThere [14] supports collaborative interactions involving 3D input. The 3D representation of the local environment at the requester side is captured by integrated depth sensors and displayed on the mobile screen of both sides. The collaborators 3D gesture thus can be captured and a virtual hand is formed on the screen of the requester device to instruct her to move the physical blocks in her environment. Gauglitz et al. [4] proposed a framework and implemented a prototype for mobile collaboration in the physical environment. Based on the framework, a requester and a collaborator can share real-time environment video captured by the filming equipment at the requester side and visual annotations labelled by the collaborator. MobiSurf [13] integrates an interactive surface into the interaction with people's own personal and mobile devices using existing interaction technologies and techniques It supports co-located collaboration through integrating mobile devices and interactive surfaces. Different from these works, CollaDroid supports interactive and remote collaboration on mobile applications.

Van't Hof et al. [6] implemented a prototype called Flux which enables unmodified Android applications running on a device to be migrated to another device without losing its status. Different from the application migration supported by Flux, CollaDroid supports interactive remote collaboration between two mobile devices.

Most of the existing researches on program transformation of mobile applications focus on improving the performance and saving the energy of applications by refactoring. Zhang et al. [17] developed an automatic refactoring tool for Android applications which enables the applications to perform on-demand computation offloading at runtime to improve the performance and save energy. Lin et al. [9] developed an automatic refactoring tool which transfers incorrect Android async constructs into correct ones. The tool can extract long-running operations and encapsulate them in *AsyncTask* for better performance. The application refactoring tool ASYNCDROID [8] helps Android developers to correct misused asynchronous programming constructs (*AsyncTask*) to *IntentService*, which uses a distributed communication instead of shared memory communication to prevent memory leak. Hao et al. [5] proposed a binary instrumentation framework for Android applications which can be used to specify instrumentation location compactly and precisely at different granularities. Lenin Ravindranath et al. [12] implemented an automatic fault detecting system for mobile applications by instrumenting the application binary to emulate various user, network and sensor behaviors to detect failures. Although these works also involve the analysis and transformation of Android applications, our work is targeted at a different specific goal, i.e., augmenting applications with lightweight interactive collaboration.

LIMITATIONS AND FUTURE WORK

The limitations of the current implementation of CollaDroid lie in the following aspects.

Limited Event and View Types

Currently, CollaDroid does not support multi-touch events. These events will be discarded by the CollaDroid client during a collaboration process. CollaDroid supports most of the commonly used view types such as *TextView*, *EditText*, *ImageView*, *Button*, *Spinner*. These view types can cover a large part of the UI windows of commonly used Android applications. Currently, CollaDroid does not support *WebView*, so UI elements embedded in web pages cannot be supported by the collaboration mechanism of CollaDroid. In the future, we will further improve and extend the implementation of CollaDroid to support more event and view types.

Collaboration Scope Specifying

Currently, CollaDroid only supports a single collaboration scope on a UI window. However, in some cases the requester may need the collaborator to manipulate views in several different rectangular areas. Moreover, CollaDroid only supports one privacy policy which allows the collaborator to see all the views on the current window but only edit the views within the collaboration scope. In the future, we will extend CollaDroid to support more flexible ways of collaboration scope specifying (e.g., specifying multiple rectangular areas) and provide more options of privacy policy (e.g., setting the views out of the collaboration scope invisible to the collaborator).

Collaboration Performance

Although the current performance (including response time and network traffic) of collaboration is acceptable, it can be further improved. Currently the messages between the requester and the collaborator are forwarded by the server. This can be improved by providing a direct messaging mechanism to allow UI event and UI update messages to be directly sent between the requester and the collaborators. On the other hand, the current UI update message includes a complete representation of the whole window, which can be improved by an incremental representation of the updated views.

CollaDroid Client and Heartbeat Messages

Currently, a collaborator needs to install the CollaDroid client on her mobile device and continuously run it in the background to respond to collaboration requests. Moreover, the CollaDroid client needs to periodically send heartbeat messages to the server, which causes additional energy consumption. This problem can be alleviated by integrating the CollaDroid client with the clients of mobile social networking platforms such as WeChat and WhatsApp. By doing this, the CollaDroid client can be run as a plugin of a mobile social network client and reuse the heartbeat messages of the social network client. Moreover, the CollaDroid client can use the social network service to obtain the list of online friends.

CONCLUSION

In this paper, we propose CollaDroid, a lightweight and UI Description based technique for interactive collaboration of Android applications. It captures the UI content of the views within the collaboration scope at the requester side and generates a mock UI at the collaborator side. The collaborator can then interact with the mock UI and the UI events are sent back to the requester side to replay them on the requester device. CollaDroid provides a program transformer that can automatically transform an Android APK to a collaboration augmented application. The results of our experimental study show that CollaDroid is applicable for a large part of applications and can provide an efficient collaboration mechanism with low network traffic and latency. The results of our user study show that the collaboration mechanism implemented by CollaDroid is well accepted by users.

ACKNOWLEDGMENTS

This work is supported by National High Technology Development 863 Program of China under Grant No. 2015AA01A203.

REFERENCES

1. Leonard M. Ah Kun and Gary Marsden. 2007. Co-present Photo Sharing on Mobile Devices. In *Proceedings of the 9th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI '07)*. ACM, New York, NY, USA, 277–284. DOI: http://dx.doi.org/10.1145/1377999.1378019

2. James Begole, Craig A. Struble, Clifford A. Shaffer, and Randall B. Smith. 1997. Transparent Sharing of Java Applets: A Replicated Approach. In *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology, UIST 1997, Banff, Alberta, Canada, October 14-17, 1997*. 55–64. DOI: http://dx.doi.org/10.1145/263407.263509

3. Surendar Chandra, John Boreczky, and Lawrence A. Rowe. 2014. High Performance Many-to-many Intranet Screen Sharing with DisplayCast. *ACM Trans. Multimedia Comput. Commun. Appl.* 10, 2, Article 19 (Feb. 2014), 22 pages. DOI: http://dx.doi.org/10.1145/2534328

4. Steffen Gauglitz, Cha Lee, Matthew Turk, and Tobias Höllerer. 2012. Integrating the physical environment into mobile remote collaboration. In *Mobile HCI '12, Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services, San Francsico, CA, USA, September 21-24, 2012*. 241–250. DOI: http://dx.doi.org/10.1145/2371574.2371610

5. Shuai Hao, Ding Li, William G. J. Halfond, and Ramesh Govindan. 2013. SIF: a selective instrumentation framework for mobile applications. In *The 11th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys'13, Taipei, Taiwan, June 25-28, 2013*. 167–180. DOI: http://dx.doi.org/10.1145/2462456.2465430

6. Alexander Van't Hof, Hani Jamjoom, Jason Nieh, and Dan Williams. 2015. Flux: multi-surface computing in Android. In *Proceedings of the Tenth European Conference on Computer Systems, EuroSys 2015, Bordeaux, France, April 21-24, 2015*. 24:1–24:17. DOI: http://dx.doi.org/10.1145/2741948.2741955

7. Susanne Hupfer, Michael Muller, Stephen Levy, Daniel Gruen, Andrew Sempere, Steven Ross, and Reid Priedhorsky. 2012. MoCoMapps: Mobile Collaborative Map-based Applications. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work Companion (CSCW '12)*. ACM, New York, NY, USA, 43–44. DOI: http://dx.doi.org/10.1145/2141512.2141534

8. Yu Lin, Semih Okur, and Danny Dig. 2015. Study and Refactoring of Android Asynchronous Programming (T). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*. 224–235. DOI: http://dx.doi.org/10.1109/ASE.2015.50

9. Yu Lin, Cosmin Radoi, and Danny Dig. 2014. Retrofitting concurrency for Android applications through refactoring. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*. 341–352. DOI: http://dx.doi.org/10.1145/2635868.2635903

10. Andrés Lucero, Jussi Holopainen, and Tero Jokela. 2012. MobiComics: Collaborative Use of Mobile Phones and Large Displays for Public Expression. In *Proceedings of the 14th International Conference on*

Human-computer Interaction with Mobile Devices and Services (MobileHCI '12). ACM, New York, NY, USA, 383–392. DOI: http://dx.doi.org/10.1145/2371574.2371634

11. Zhaotai Pan, Huifeng Shen, Yan Lu, Shipeng Li, and Nenghai Yu. 2013. A Low-Complexity Screen Compression Scheme for Interactive Screen Sharing. *IEEE Trans. Cir. and Sys. for Video Technol.* 23, 6 (June 2013), 949–960. DOI: http://dx.doi.org/10.1109/TCSVT.2013.2243056

12. Lenin Ravindranath, Suman Nath, Jitendra Padhye, and Hari Balakrishnan. 2014. Automatic and scalable fault detection for mobile applications. In *The 12th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys'14, Bretton Woods, NH, USA, June 16-19, 2014*. 190–203. DOI: http://dx.doi.org/10.1145/2594368.2594377

13. Julian Seifert, Adalberto Lafcadio Simeone, Dominik Schmidt, Paul Holleis, Christian Reinartz, Matthias Wagner, Hans Gellersen, and Enrico Rukzio. 2012. MobiSurf: improving co-located collaboration through integrating mobile devices and interactive surfaces. In *Interactive Tabletops and Surfaces, ITS'12, Cambridge/Boston, MA, USA, November 11-14, 2012*. 51–60. DOI: http://dx.doi.org/10.1145/2396636.2396644

14. Rajinder Sodhi, Brett R. Jones, David A. Forsyth, Brian P. Bailey, and Giuliano Maciocci. 2013. BeThere: 3D mobile collaboration with spatial input. In *2013 ACM SIGCHI Conference on Human Factors in Computing Systems, CHI '13, Paris, France, April 27 - May 2, 2013*. 179–188. DOI: http://dx.doi.org/10.1145/2470654.2470679

15. Agustinus Borgy Waluyo, David Taniar, Bala Srinivasan, and Wenny Rahayu. 2013. Mobile Query Services in a Participatory Embedded Sensing Environment. *ACM Transactions on Embedded Computing Systems* 12, 2 (2013), 31:1–31:24.

16. Dustin Wüest, Norbert Seyff, and Martin Glinz. 2015. Sketching and notation creation with FlexiSketch Team: Evaluating a new means for collaborative requirements elicitation. In *23rd IEEE International Requirements Engineering Conference, RE 2015, Ottawa, ON, Canada, August 24-28, 2015*. 186–195.

17. Ying Zhang, Gang Huang, Xuanzhe Liu, Wei Zhang, Hong Mei, and Shunxiang Yang. 2012. Refactoring android Java code for on-demand computation offloading. In *Proceedings of the 27th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2012, part of SPLASH 2012, Tucson, AZ, USA, October 21-25, 2012*. 233–248. DOI: http://dx.doi.org/10.1145/2384616.2384634

18. Pengfei Zhou, Yuanqing Zheng, and Mo Li. 2012. How long to wait?: predicting bus arrival time with mobile phone based participatory sensing. In *The 10th International Conference on Mobile Systems, Applications, and Services, MobiSys'12, Ambleside, United Kingdom - June 25 - 29, 2012*. 379–392. DOI: http://dx.doi.org/10.1145/2307636.2307671