Link

Hans Krutzer

June 28, 2013

1 Preface

In this document, we will describe the design and structure of the "Link" Android application we built for the course Research and Development 1. First we will describe what problem we solve, what the solution looks like (visually), and then how the underlying program works, because this is also the order in which the program was designed and built. Finally, we will reflect on the result of the project, as well as the process.

2 Description

Link is currently able to send and retrieve text from a computer[1], send and retrieve image files from a computer, and retrieve files of any other type, such as PDF or MP3 files. Link can retrieve[2] or send one file at a time, and strictly speaking it is up to the software on the computer to provide a way of selecting which file Link will receive when a user opts to receive it. Selecting this file can only be done from the computer. In OS X and Windows 7 implementations, Link will receive whatever is on the computer's clipboard. If Link receives text, it is placed on Android's clipboard, if it receive an image, it is saved in Android's download folder and opened in the Android Gallery, and if it receives a file of any other type, it will save it in the download folder, but not open it.

In both implementations, when Link is used to send text, it will be placed on the computer's clipboard, while images will be opened in the OS's default image viewer. The user can then copy that image, save it, or edit it if the default viewer allows editing.

Link does not require or even allow any configuration. Discovery of other devices on the network is done automatically. Implementations on non-mobile platforms are expected to listen for Link's requests for identification. This means that the only thing the user needs to do is install and run the software. It also means Link can only be used across devices on the same network.

2.1 Justification

We built this application for people that use their smartphone in conjunction with their phone. As we don't believe people turn off their phone when they use their computer, so we expect there to be many potential users, but we think the app will be most useful for computer and smartphone "power-users".

Similar applications that employ the clipboard do not allow file transfers, and require configuration. Services that do allow file transfers require an account, and are transferred via their servers over the internet, not the local network (e.g. Dropbox, Google Drive). This creates a lot of overhead and slows down the process.

[1] When we use the term "computer" throughout this document, we mean desktop and laptop computers, not smartphone computers

[2] We deliberately use the word "retrieve", and not receive, because the action is always initiated by the phone

2.2 Specifications

2.2.1 Functional requirements

use case ID	use case name
1	device discovery
2	action selection
3	send text
4	retrieve text
5	send image
6	retrieve image
7	retrieve file

Use case ID	5
Use case name	send image
Description	The user sends an image to a computer
Basic course of events	1. The user selects "Send image" from the menu in the action selection use case. 2. The user is presented with the Android Gallery, from which they will pick an image. 3. The application displays a loading indicator while the image is being transferred. When completed, the application returns to the menu.
Alternate paths	1. The user selected an image from the Android Gallery and presses the "Share" button and selects the application. 2. The discovery use case is launched. 3. The use case continues from BCoE step 3, but returns to the gallery instead.
Preconditions	The software works correctly on the computer The discovery was successful.
Postconditions	The computer received an image file.

2.2.2 Non-functional requirements

- The only method of transferring data will be the (wireless) local network.
- The application can not have a significant battery impact (e.g. more than 1% battery usage after a day of normal use of the phone and the application).
- Transferring files from the computer to the mobile device must be as fast, or at least not noticeably slower, than downloading the file from a local web server running on the computer.

3 Design

3.1 Visual design

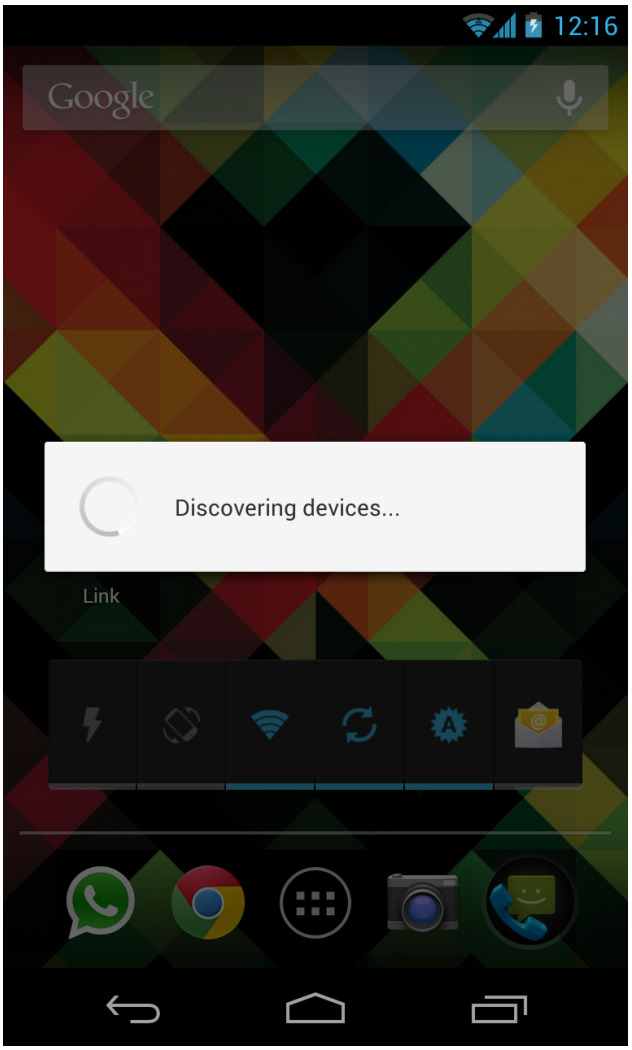
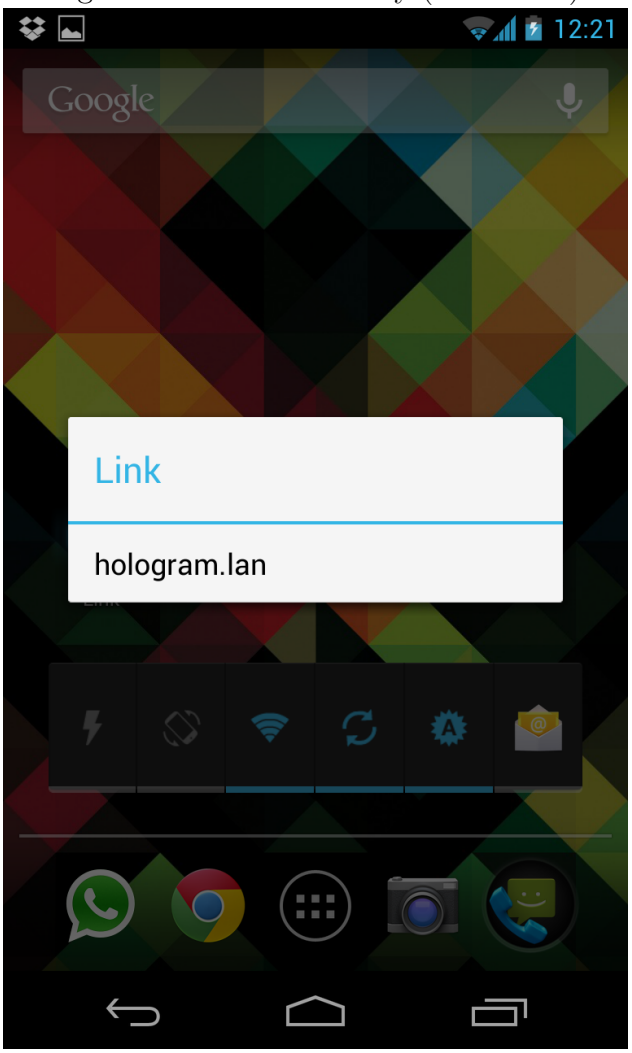


Figure 1: Device discovery (use case 1)

In the initial plans the application was described as being a widget, so that a user can quickly launch it without feeling like they are launching a complete (and possibly slow and large) application.

Figure 2: Device discovery (use case 1)



This turned out to be unnecessary and a waste of effort. Adding a widget to an Android application takes configuration and requires additional classes.

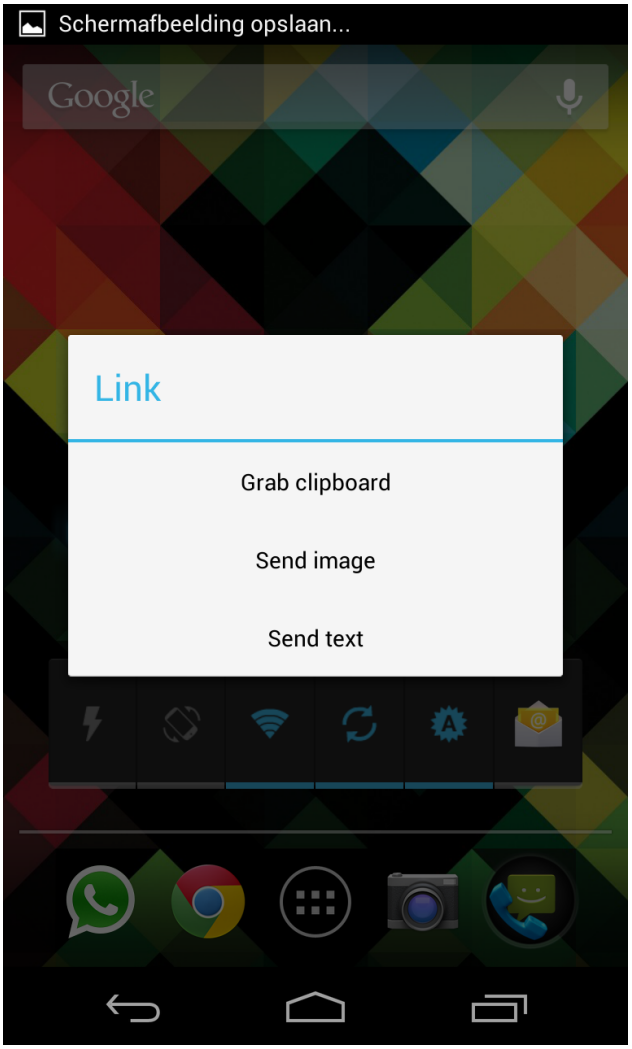


Figure 3: Selection an action (use case 2)

Instead, the application uses a regular "main activity" to launch it. To get the light widget look-and-feel, the activities are styled using a theme that overlays on top of the homescreen or another application. Not using a widget reduces complexity, but still allows the user to feel like they are not launching a full application. The theme also allows the same activity to be reused in the Android share menu.

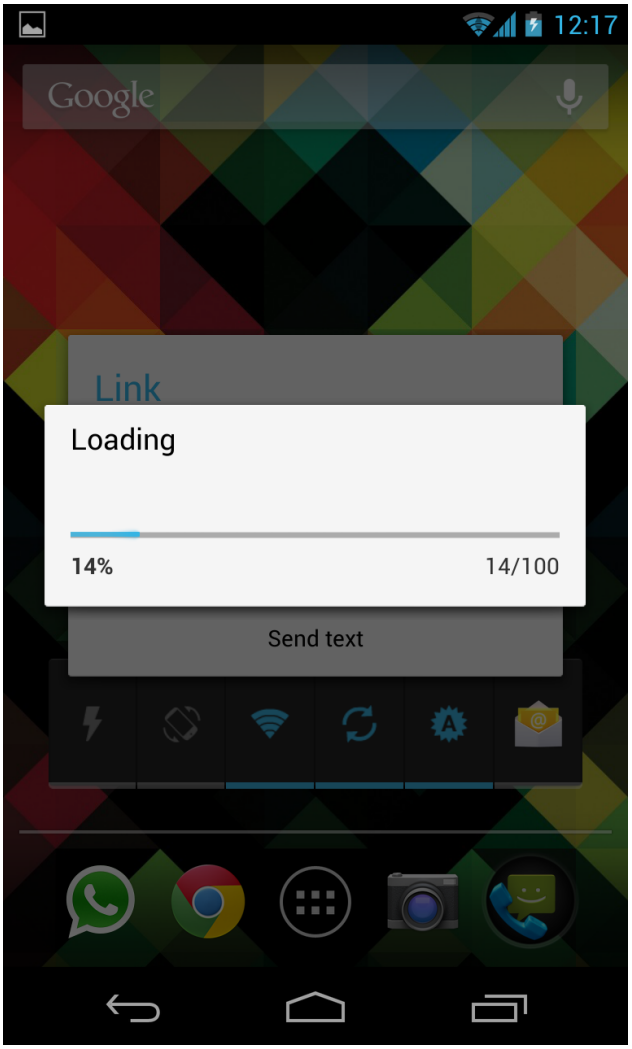


Figure 4: Retrieving a file (use cases 6, 7)

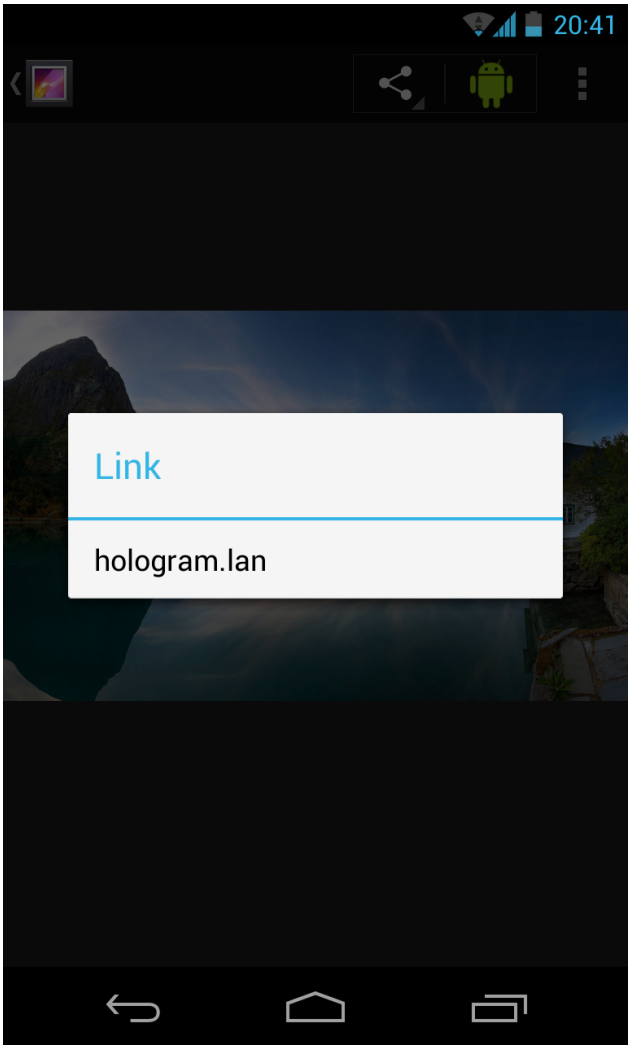


Figure 5: Transferring an image from the Android Gallery (use cases 3)

3.2 Global program design

The program is split into three components: the discovery component, the file transfer component, and the interface component. The interface will first invoke the discovery components, retrieve its result, and launch the appropriate file transferring component.

The file transfer component has three sub components: one that sends text, one that sends image files, and one that retrieves files or text. The text and file sending operations are split into different classes, because sending files sufficiently different from sending just text with the protocol use, to merit this.

The user interface is driven by only two activities, one that launches device discovery, and offers the user a list of discovered devices, and then launches the second activity, where the user can choose what to do with the discovered device.

3.3 Program design in detail

In this section we are going to look at the Java classes involved in use case 5. This use case is started after the user has gone through the discovery use case (use case 1), and chose "Send image" in the action selection use case (use case 2). Three classes are relevant:

- FriendActionActivity
- Download
- DownloadResult

When use case 5 starts, FriendActionActivity's *grab*() method will be invoked. This method will construct a Download object, and pass itself (the object) to its constructor. It will then call the *execute* method on the Download object, passing in a URL to download.

The Download object extends the Android AsyncTask class, and overrides the onPreExecute, onProgressUpdate, onPostExecute, and doInBackground methods. The onPreExecute, onProgressUpdate, onPostExecute respectively create, update, and dismiss a progressbar to indicate download progress. The activity passed into the constructor is used to retrieve the Android Context, which allows the dialog to be displayed. The actual downloading happens in the doInBackground method. In this method, a stream is opened, the content type is determined, and because in this use case an image is being retrieved, the image will be downloaded and written to the Android download folder. The method then indicates to the Android media scanner that a new file exists, again using the activity passed into the constructor as the Context.

Finally, the method constructs a DownloadResult object, using the overloaded constructor that accepts a URI as a parameter, and passes it back to the activity that launched the Download task. The FriendActionActivity then retrieves this URI, and uses it to open the image in the Android Gallery.

All methods and fields of the DownloadResult and Download classes are listed below.

Download
public Download(FriendActionActivity)
protected DownloadResult doInBackground(String... URLs)
private String getFileName(String header)
protected void onPreExecute()
protected void onProgressUpdate(Integer... progress)
protected void onPostExecute(DownloadResult s)
private ProgressDialog mProgressDialog
private final FriendActionActivity mContext

DownloadResult
public DownloadResult(String) public DownloadResult(Uri) public String getResult() public Uri getURI() public ResultType getResultType()
public enum ResultType image, text private final ResultType resultType; private String result; private Uri uri

3.4 Device discovery method in detail

During and before development, we considered three ways of finding the address of computers to load from and send data to.

The easiest way would be to have the user enter the local IP address of a computer to connect to. This however, is only easy for the developer, and not the end user, as it takes effort and some technical knowledge from them. We quickly disregarded this option and explored options that would allow other devices running our software to be discovered automatically.

This led us to two other options. One being aggressive TCP connection attempts: take the entire subnet and attempt to connect to each IP in it. This is very far from efficient, and we were unsure whether this would be fast enough.

We therefore went with the option of broadcasting UDP messages, and having computers send a broadcast in response when they receive a UDP broadcast. This required a bit of tuning, as UDP packets can easily get lost, and the same Java thread cannot be sending messages and listening for them at the same time. We worked out a pragmatic semi-reliable way to use it, but there is still testing to be done on unreliable networks.

4 Reflection

Overall, we are satisfied with both the process and the result of the project. The core functionality is working and stable. Due to time constraints and perhaps inefficient use of time, we feel the usability of the application is not yet of ready-to-ship quality. Most importantly, notifications are completely missing, resulting in users not being sure the application did anything. We think better short-term planning would have lead to more efficient use of development time.

We are very satisfied with the visual style of the end product. It is looks and works almost exactly as planned, and the plan turned out to be very good. We think the Android platform does not offer any better options than how it looks and operates than it does now.

In conclusion, we think Link is a great proof of concept, and with some additional effort could quickly be made into a polished application ready to be used by real world users.