Shadow Internet

A censorship-free communication infrastructure

Mark van Beusekom
Nicolaas Herckenrath

Image Source: http://pdab.net/index.php?m=gallery&photo=sony_xperia_m_nfc_transfer.jpg

Shadow Internet

A censorship-free communication infrastructure

by

Mark van Beusekom
Nicolaas Herckenrath

in partial fulfillment of the requirements for the degree of

Bachelor of Science
in Computer Science

at the Delft University of Technology,

Coach:	Ir. Egbert Bouman	TU Delft
Client:	Dr. Ir. Johan Pouwelse	TU Delft
Bachelor Coordinator:	Dr. Martha A. Larson	TU Delft

Preface

This report details the development done during the ’Shadow Internet’ project for the course: TI3806 Bachelorproject. It was commissioned by the Tribler Team at the Parallel and Distributed Systems group (PDS) at the faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) at the Delft University of Technology.

The report documents the 9-10 weeks that were spent developing the Shadow Internet application, an Android application capable of recording videos and sharing them without the need of an internet connection. A large part of the project was to make Python code properly run on Android. The goal of the report is to show the reader the workings of the application, the process and choices that lead to the final product and to give recommendations for further developments.

We would like to thank the Tribler team for making the Tribler code available and for assisting us whenever we had questions. We would also like to thank the Anonymous HD Video Streaming for Android bachelor project group for compiling most of Tribler for Android and assistance with questions regarding this effort.

Special thanks goes out to our client Johan Pouwelse for his enthusiasm and vision for the project, to our coach Egbert Bouman for assisting us during the project process and to Jaap van Touw for his help in setting up Jenkins.

Mark van Beusekom
Nicolaas Herckenrath
Delft, June 2015

Abstract

Due to the rise of smartphone technology, numerous citizens carry Internet-enabled recording equipment with them at all times. Should atrocities be committed by the the government, they would have a hard time to cover-up the truth without blocking the Internet. Governments have demonstrated their ability to restrict access to the Internet during events like the Arab spring.

The Shadow Internet bachelor project aimed to create an application that thwarts government censorship by allowing the spread of incriminating footage without the use of the Internet. This way, a video could be passed around until an enterprising journalist can smuggle the footage out of a censored region. When in a safe area, he would be able to spread the footage to the world.

Our application implements this functionality by using Android Beam to transfer videos. Android Beam is Google's implementation of Bluetooth file transfer that is started through NFC. As the Google Play Store will not be available during an Internet blackout, it also supports installing itself on other phones through the same mechanism. This will allow the spread of both our application and the video without the use of the Internet. In order to facilitate the spread of evidence we have implemented Tribler support. This allows the anonymous uploading of videos into the Tribler network.

The project was divided into 3 Scrum cycles lasting two weeks each. Each cycle iterated upon the existing application, with a focus on adding additional functionality. The first cycle focused on creating the first prototype and getting used to the Kivy framework. The second cycle saw the addition of the core features such as NFC file transfer and the video recording. The final cycle focused on improving the user experience and adding Tribler support for uploading videos. This cycle was plagued with a number of problems that delayed progress, but were eventually resolved.

As of this writing, the application supports video recording and allows for file transfer through Bluetooth, along with the application itself, to another phone through the use of Android Beam. Sadly, Tribler support has not yet been fully implemented.

Contents

1 Introduction

The project is issued by Dr. Ir. Johan Pouwelse of the Parallel and Distributed Systems group (PDS), a section of the Department of Software and Computer Technology (SCT) at the Faculty Electrical Engineering, Mathematics, and Computer Science (EEMCS) of the Delft University of Technology. The Parallel and Distributed Systems group focuses on research into the fields of P2P systems and online social networks, massively multiplayer online games, grids and clouds, multi-core architectures and parallel programming. This project is part of the realization of the 'Shadow Internet', which is an attempt to make it impossible to censor the Internet, as it is an important place for free and innovative ideas. [1]

These days it has become harder than ever to cover up wartime atrocities. Due to the rise of smartphone technology, numerous citizens carry Internet-enabled recording equipment with them at all times. So if a government wants to keep crimes under wraps, monitoring or blocking the Internet is one of the options they might have to use. A number of governments, for instance, Egypt [2] and Syria [3], have used various methods to severely cripple communication and international involvement. For instance, during the Arab spring, Egypt forced the telecommunication companies to sever their broadband and mobile connections, effectively disabling the internet in Egypt [1]

Our Shadow Internet project focuses on making these kinds of blocks ineffective by creating an application that enables the spread of incriminating material without the use of the Internet. This way, the material could find it's way to a front-line reporter who can smuggle it to safety. When in a safer region, the application would allow the evidence to be spread anonymously via the Tribler network [4]. This way, the whistle-blower's identity will not be compromised.

Our application builds upon the work of previous thesis projects such as Tribler Play and AT3 , but does not directly extend the existing applications. This was done in order to create a more direct implementation of Tribler instead of the current server-based version.

This report is structured as follows: First we cover the project description, along with it's goals. The subsequent chapter researches and compares the various technologies the application would need to use. During the implementation chapter the report details how each feature was implemented, along with it's pitfalls

2
Problem Description

The Shadow Internet is an envisioned alternative communication infrastructure. It's been actively developed to be resilient to sniffing, blocking, filtering and shutdown. This project focuses on creating a smartphone solution that uses the internet only when available and capable of using direct smartphone-to-smartphone communication when internet usage is blocked or dangerous.

2.1. Current Situation

Whilst a significant amounts of work has gone into several projects that are related to the Shadow Internet project, most of the former work has been focused around Tribler, a BitTorrent compatible social media sharing application, currently being developed at the PDS research group at the TU Delft. Tribler does offer various means to share media, it requires an Internet connection to do so. Another project done at the TU Delft, DroidStealth, is an application that enables its users to record videos, safely encrypt and store those files and to hide the application itself on your device. It also features a method to spread the application through the use of Android Beam, a NFC and Bluetooth connection native to Android. For a more detailed explanation of how these projects were used within this project, see 3.

2.2. Project Goals

The original assignment as found in appendix A has been redefined by the client. The project team is to create an application to be released on the Google play Store. The main focus of the application will be to facilitate the transfer of video files and the application itself to another device without needing an Internet connection. In order to provide a single integrated user-experience, the application should be able to use the device's video camera from within the application itself. This core functionality will allow for the recording and spreading of, for instance, war crimes in areas where the Internet is unsafe or impossible to use.

In order to increase the speed at which these videos will spread around the world, the application will support Tribler in order to upload videos in areas where the Internet is available and relatively safe to use. Tribler is a specialized Bittorrent client with a heavy focus on anonymity. The use of Tribler allows for anonymous seeding of the sensitive video material. By reducing the chance of identification, the application stimulates the creation and spreading of vital evidence.

Additionally, the project team will be working together with a team of bachelor students working on the project "Anonymous HD video streaming for Android". This project focuses on porting Tribler's streaming functionality to Android. This port will produce a number of libraries that are vital for the integration of Tribler in our Shadow Internet application. Their streaming functionality makes for an excellent addition to the user-experience, so one of the goals is to integrate both projects into a single application.

2.3. Use Cases

To better understand how the application would be used, several use cases were explored to understand the usage of the final product. The following use cases were explored: sharing the application, creating and sharing videos and downloading, viewing and sharing videos. We spoke to a potential end-user from a censored internet region, who confirmed that the second and third case currently occur in regions where the government monitors and censors the Internet.

- *Sharing the Application*
  In some areas, for instance Syria, the Google Play Store is blocked. To spread our application, users can bump phones while no video is selected. This will bring up the Android Beam prompt. Pressing this will transfer and install the application on the target phone.

- *Creating and Sharing Videos*
  A user can record a video by pressing the camera button. This will bring up the camera application. Pressing record will start the recording, and pressing it again will save the recording.

  After a user has recorded a video, he may want to share it. He could share all of his videos by pressing on the Tribler button. Alternatively, he could open up the triple dot menu of a video and select to beam this file. When he holds his phone against another, this will bring up the Android Beam prompt. Pressing the prompt will transfer the file to the other phone using Bluetooth.

- *Downloading, Viewing and Sharing Videos*
  A user can also download a video from the Tribler network. To accomplish this, he would have to press on the search button. This will bring up the search menu. After inputting a search term into the search bar, a number of files will show up. Pressing on the download button will start the download. After the download the user can press the video file and the media player will open.

3 Research Phase

This chapter describes the research performed such as a review of previous work and similar programs. This will give further insight into the proper approach of certain elements of our final product and into solutions that could be used to create all requested functionalities.

In section 3.1, we will review earlier projects and released products that could be used in our project. In section 3.2, we will discuss the solutions that are available for the various goals we have set for our project. Finally, we will discuss our chosen solutions in section 3.3.

3.1. Related Work

This section reviews earlier work that has been done and is both related and can be used to further pursue the goals of 'the Shadow Internet'. These works are:

- **Tribler Play & Android Tor Tribler Tunneling**
  In the previous year, two groups of students have created two applications for the Parallel and Distributed Systems (PDS) research group at the TU Delft, Tribler Play [5, 6] and AT3 [7, 8]. Tribler Play is an Android application that is able to use Tribler to search for and play video's through torrents. AT3 is an Android application that is able to set up an anonymous tunnel to the internet in order to download files anonymously. Parallel to our project, another group of students will combine these two applications and we will provide a hook for the resulting application in order to gain the anonymous tunnel functionality to upload media files anonymously[9].

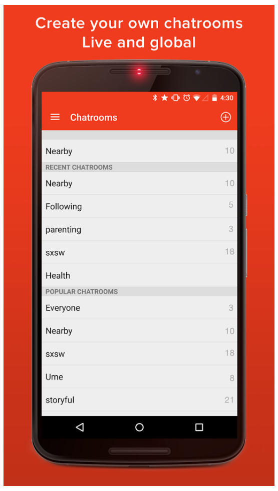
- **DroidStealth**
  DroidStealth [10, 11] is an application created by the Hacking Lab course IN4253ET, which enables users to save files on their devices and hide them. The application currently uses a flawed method of 'morphing' in order to hide the application from device inspections by transforming its assets to mimic another application, but it can still be found in the list of installed applications. A master student at the TU Delft is currently working on integrating a better form of morphing to remove this weakness. This project should be able to use the upgraded Droidstealth functionality to increase the application's safety by allowing the user to successfully hide the application from device inspections.

- **Python-for-Android**
  Python-for-Android [12] is a framework that allows for developers to run Python code on Android, whose development environment is based on both Java and XML. It is based on the Scripting Layer for Android project. The original Python-for-Android project is not being supported as of August 2012, but a forked version that is part of the Kivy GUI framework is still being supported by its developers. This forked version also includes support for Kivy and a program called Buildozer, that can package Python application for Android.

- **SuperBeam**
  SuperBeam [13] is a freemium Android application that extends the core Android Beam functionality of Android for file transfer. SuperBeam is capable of transferring files by using Wi-Fi Direct after starting a

link between two devices through either Android Beam or a QR-code scan and can fall back to a normal Wi-Fi connection if a Wi-Fi Direct connection fails. Due to its freemium nature, the application itself is a closed source project.

- **FireChat**
  FireChat [14] is a communication application for Android that uses Bluetooth to message people around you in a similar fashion as current SMS applications. Messaging works with user-created groups, which are broadcast and can be joined by others (shown in 3.1. FireChat also supports multi-hopping to allow chats to reach farther than the standard Bluetooth range. Due to using Bluetooth, FireChat is still available even if the internet or telephone connections are unavailable. It is because of this feature that the application gained widespread fame during the 2014 Hong Kong protests (also known as the Umbrella Revolution/Movement). The application was used by the protesters, as the telephone connection during the protests became strained due to the amount of people present, but protesters did realize that it could be used to organize the protests, should the government have decided to cut both the internet and telephone lines [15].



Nearby	10
RECENT CHATROOMS	
Nearby	10
Following	5
parenting	3
sxsw	18
Health	
POPULAR CHATROOMS	
Everyone	3
Nearby	10
sxsw	18
Ume	8
storyful	21

Figure 3.1: FireChat's message group screen.

Source: https://play.google.com/store/apps/details?id=com.opengarden.firechat

- **eyewitness & eyeWitness to Atrocities**
  Both eyewitness [16] and eyeWitness to Atrocities [17] are Android applications that enable the user to document events near them as either a picture, a sound recording or a video and to share these files. Whilst the applications both have the same concept, the resulting approach differs. eyewitness focuses on daily crime and enables users to create and file reports to the proper authorities (including some resources to determine if something is or isn't a criminal act), whereas eyeWitness to Atrocities focuses on (war) crimes occurring in more troubled regions in the world. To do this, it provides features to increase the user's safety and the ability to encrypt and anonymously report witnessed crimes.

- **Anonymous Streaming**
  During the course of this project, another team of three bachelor students will be working on the Anonymous HD video streaming for Android project, in which they will combine two earlier projects, Tribler Play for Android (or formerly known as Tribler Streaming Android Project (TSAP)) and Android Tor Tribler Tunneling (AT3), into a single unified project that is capable of using tunneling to remain anonymous whilst streaming videos from Tribler. Should both projects progress without issue, we will attempt to merge parts of these projects to create a video sharing service that can spread files without the use of the internet.

3.2. A COMPARISON OF POSSIBLE TECHNOLOGIES

This chapter discusses the different kinds of technologies that are currently available and could be used to develop our application. Considering our problem definition, we identified the following key features that we cover in this chapter: the file transfer, the application's Graphical User Interface (GUI), the upload of media and the version of Android itself. We will discuss several technologies per feature and give an insight on both the positive and negative aspects of these technologies, after which we will discuss our final choices on the used technologies, also taking into account the preferences of our client.

3.2.1. File Transfer

The key part of the application will be its ability to share both the application itself and video files wirelessly without relying on the internet for connection purposes. For our application we reviewed the following methods of wireless file transfer: Near Field Communication (NFC), Quick Response (QR) codes, Bluetooth and Wi-Fi Direct.

- **Near Field Communication**
  Near Field Communication (NFC) is a specification for contact-less communication between two devices based on technology used for Radio-frequency Identification (RFID) [18]. NFC communication is limited to 10 centimeters and is currently used primarily to make small transactions, exchange digital content and connect devices with each other. NFC in smart-phones functions via NFC chips which can both send and receive data from other NFC chips. NFC supports two methods of communication, namely active communication, which requires both chips to generate their own electric fields to communicate, and passive communication, in which only the sender creates an electric field which is then imitated by the receiver. In order to activate any kind of transfer between two NFC enabled devices, it is therefore only necessary to hold both chips near each other as shown in figure 3.2.

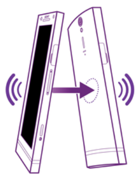
  

  Figure 3.2: A schematic example of an NFC transfer

  Image Source: http://thecorda.com/what-is-nfc/

  Due to the short range of NFC communication, it is not as easy to eavesdrop or intercept the communication itself. Whilst it it certainly possible to tap the communication using an antenna, due to the signal strength of NFC, the effective range for this is either up to 10 meters when the NFC chip is broadcasting in its active state or up to 1 meter when it sends data whilst in its passive state. So even though NFC communication is not fully safe from third parties attempting to spy on the file transfer, the short intercept range makes it nearly impossible to be wiretapped without the users knowledge.

  According to a 2014 analysis by IHS Inc. 18.2 percent of all shipped smart-phones in 2013 where equipped with NFC capabilities [19]. Whilst this is a low percentage the analysis also concluded that this number would grow to 64 percent by 2018. This increase it attributed towards a global push to use smart-phones to make small monetary transactions. Due to this, it is very likely that as the amount of NFC enabled smart-phones increases, that so too will NFC become more familiar to the Android user base.

  The transfer speed of NFC is lacking in comparison to its competitors. NFC supports multiple data rates, but none higher than 848 Kbit/s [18]. Seeing as most video files will typically reach sizes of several tens or hundreds of megabytes, this would result in file transfers that can take up to two or three hours to complete.

  Another problem is that the communication that takes place during an NFC transfer is not encrypted. Considering this, if the file transfer was successfully tapped then the party engaged in the wiretapping would be able to access all the sent data, which would undermine the safety of the application.

- **QR-Code**
  A QR-Code is a 2 dimensional bar-code capable of storing small amounts of data [20]. QR codes are

used by applications to send small files by parsing the transferred file into a QR code, which is then scanned by the receiver using a camera, which allows the receiver's device to recreate the original file.

As the method of communication requires a line of sight which a camera enabled device, it is impossible to eavesdrop on this form of communication without being physically present during the file transfer. QR-codes are also widely accessible hardware wise, as the only requirement for a smart-phone is that it has a camera to record the QR-code with.

QR-codes do have a rather large downsize to them. The largest version of QR code can store up to 2,953 bytes [21]. Due to this size limitation, it is impractical to use a QR-code to transfer any size-able file. Additionally, QR-codes are a one-way method of communication, which makes setting up a secure communication environment challenging. As Android does not have native QR-code support for our use cases, this would need to be implemented by our application

- **Bluetooth**
  Bluetooth is a wireless technology standard that allows for the exchange of data over short distances, using short wavelength Ultra High Frequency (UHF) radio waves. Bluetooth operates with a master-slave structure, which allows a single 'master' unit to connect to up to seven different 'slave' units. The process in which two or more Bluetooth devices connect to each other is called 'pairing'. Pairing is usually done manually by turning on the Bluetooth adapter, scanning for other Bluetooth devices and then selecting the device(s) one wishes to connect with.

  Bluetooth has three classes that indicate the range, which is around 10 meters for most mobile devices [22]. The available transfer speed of Bluetooth is dependent on the version and is 1.0 Mbit/s or 0.125 MB/s for version 1.2, 3.0 Mbit/s or 0.375 MB/s for version 2.0 and 24.0 Mbit/s or 4.0 MB/s for versions 3.0 and 4.0. According to predictions from ABI [23], around 75% of all Bluetooth shipped devices run version 4.0, as seen in figure 3.3 Security for Bluetooth connection consists of a modified version of the SAFER+ algorithm [24]. This algorithm is used by Bluetooth to generate the keys for encryption for both the pairing and the transfer process. It can be successfully attacked through a brute force approach, but this approach does rely on the Bluetooth connection being instantiated long enough for it to deduce the current keys in use.

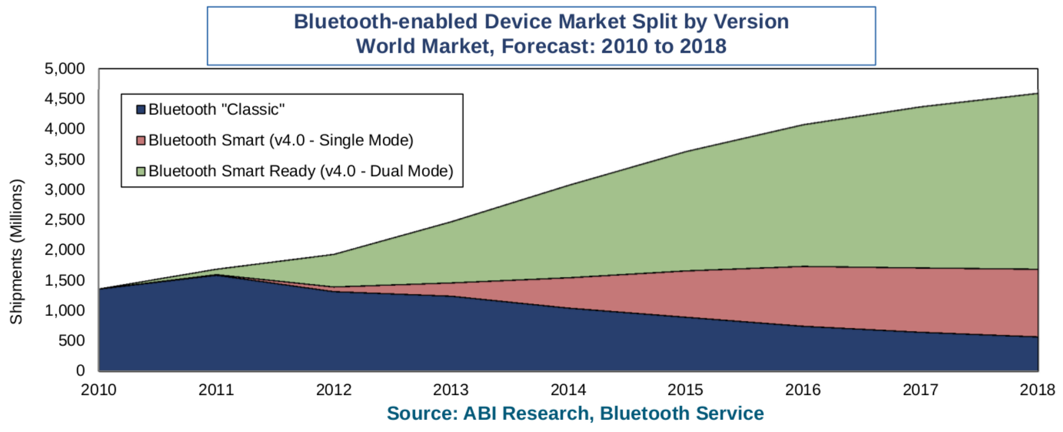


Figure 3.3: Market Split estimation

Image Source: [23]

- **Wi-Fi**
  Wi-Fi (or WiFi) is a local area wireless computer networking technology, which enables devices to connect to a network, most notably using the 2.4 and 5 GHz bands. Because the most used Wi-Fi standard, IEEE 802.11, establishes a connection by using an access point (AP) to relay information to and from a network, it is unsuitable for our application, as we intend to transfer files directly from one phone to another. Wi-Fi does offer an additional Wi-Fi standard called Wi-Fi Direct (initially called Wi-Fi P2P). Wi-Fi Direct allows for a direct connection between two Wi-Fi enabled devices without the need for an AP [25].

The transfer speed whilst using Wi-Fi Direct is dependent on the hardware (e.g. the wireless card and the protocols it supports), but Wi-Fi Direct can handle speeds up to 250 Mbps [26]. SuperBeam, an application that uses Wi-Fi Direct which was touched upon in 3.1, claims that the average transfer speed lies around 20-40 Mbps in practice [13].

Wi-Fi Direct can reach up to 200 meters [27] which is much further than other wireless technologies. Whilst Wi-Fi Direct employs Wi-Fi Protected Setup (WPS) to ensure security with Wi-Fi Protected Access II (WPA2) security protocol, the long range makes it unable to properly verify if a third party is attempting to tap into the communication.

3.2.2. GRAPHICAL USER INTERFACE

An essential part of any Android application is its Graphical user interface (GUI). This provides the user's control over the application, so an intuitive design is key to understanding and using the application's functionality. A number of frameworks can be used to design a GUI, so choosing one was a key decision early within the project. can For this project, we considered the following GUI frameworks: the Android framework, the Kivy framework and the Qt framework.

- **Native Android**
  The first option for our GUI is to create one in the Android environment itself. Android supports the creation of a GUI through XML. In these XML files it is possible to define GUI elements and to attach variables and actions to these parts. It is then possible to make the GUI interact with the application's code by calling it through its identifier. The application itself is build with a program called Gradle, which compiles the code into an APK that can be installed on Android.

  As this method of creating a GUI is supported through the Android API itself, it is simple to create a slick looking GUI for an application that conforms to Play Store prerequisites and to user expectations.

  Using this approach would force us to use Java as our programming language for the application itself. This could prove problematic, as certain features we would like to implement are not written in Java (i.e. Tribler, which is written in Python). This would make implementing these features a harder and most likely require workarounds.

- **Kivy**
  Kivy [28] is an open source Python Library for the development of multi-platform applications that use user interfaces. Kivy specializes in creating applications that use touch based input and uses OpenGL to handle the graphical aspects of the GUI. Kivy comes bundled with the Python-for-Android project [12], which makes it possible to port the resulting application from Python to Java, to enable it to be run on Android. The Python-for-Android project also includes PyJnius, which is a library that is capable of calling upon the Java API of Android, which allows Kivy to access Android specific functions. It also comes with a (alpha) program called Buildozer, which is Kivy's equivalent to Gradle.

  As Kivy is a Python Library it is able to call upon the many Python libraries and use their functionalities. This combined with the ability to call upon Java libraries through PyJnius gives Kivy a larger set of libraries to work with to create the requested functionality.

  Kivy is, however, unable to access the standard GUI functions that the native Android approach has and this means that many standard GUI features must be recreated within Kivy. This will mean that it will be harder to create the standard Android look and feel, as we need some workarounds to implement standard functions (i.e. we will need to manually insert system icons).

- **Qt**
  Qt [29] is a cross platform application and UI framework that uses C++, HTML5 and Qt Quick, a CSS and Javascript like language, to create applications. It also features the PySide open source project [30], which is a project that allows Python to create bindings to Qt, resulting in the ability to create a Qt application in Python. Qt has platform specific frameworks and the Qt for Android framework features the ability to create a native Android look and feel since Qt 5.4.

  Whilst Qt is usually run with C++, it should be possible to use PySide in combination with Python-for-Android in order to directly access Android, should the built-in access through Qt for Android's own QuickQt not be sufficient enough. The GUI not be difficult to create and adjust to be like typical Android GUI's. The Qt documentation for Python-for-Android does lead to the inactive version from

September 2012 [31]. This would mean that possible problems with the framework will have to be solved by ourselves.

3.2.3. UPLOAD

After passing along footage from phones to other phones using file transfer method that doesn't rely on the internet and reaching a safe haven, it is essential that users have a method to upload their recorded footage onto the internet in order for it to spread globally. For this uploading functionality, we considered the following methods: uploading via Tribler and leaving the uploading to the user (e.g. via Dropbox or Youtube).

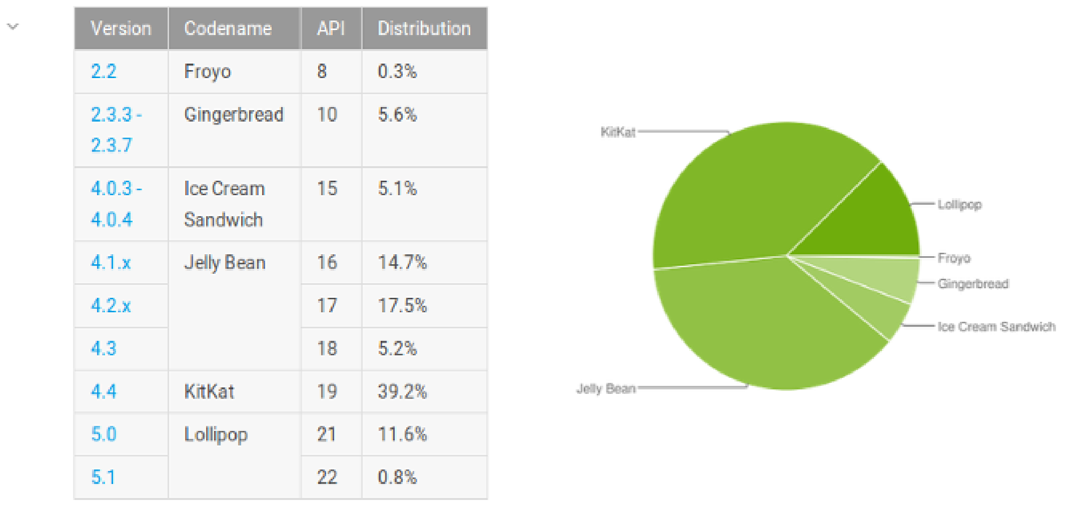
- **Tribler**
  Tribler [4] is an open source resource project of the TU Delft that allows its users to search for, download and upload files anonymously to the internet. Tribler achieves this anonymity by using an onion routing network, which routs all the Tribler traffic through several proxies that encrypt the data and ensures that only the intended receiver can correctly decrypt the received data. Though experimental, currently Tribler uses three layers of proxies to protect the user whilst downloading and has support for the same three layer proxy protection for users that upload files.

  While Tribler supports the ability to anonymously upload files, these files are not yet encrypted, which makes it possible for a third party to read the content and possibly retrace the content to the user through the content itself. It does make it harder to do so, as the anonymity prevents a third party from simply checking the users IP address. As Tribler is a BitTorrent client with around 4000 monthly users and can interact with non-Tribler Bittorrent swarms (giving access to more than 150 million people [32]), so uploaded content can have a potential global reach.

- **User uploading**
  Another option for uploading would be to make the files accessible to the user in a way that the files can be easily retrieved from the application, so that the user can use an uploading method they are comfortable using. This does place a heavy responsibility on the user, as the entire process of safely and anonymously spreading the content globally will lie in his choice of uploading.

3.2.4. ANDROID VERSION

Choosing an appropriate target platform is an important part of creating a potential user base. In the case of Android, this means targeting an API version that has a high degree of penetration. Luckily, Google provides such information to it's developers.



Version	Codename	API	Distribution
2.2	Froyo	8	0.3%
2.3.3 - 2.3.7	Gingerbread	10	5.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.1%
4.1.x	Jelly Bean	16	14.7%
4.2.x		17	17.5%
4.3		18	5.2%
4.4	KitKat	19	39.2%
5.0	Lollipop	21	11.6%
5.1		22	0.8%

Figure 3.4: Data collected during a 7-day period ending on May 4, 2015. Any versions with less than 0.1% distribution are not shown.

Image Source: https://developer.android.com/about/dashboards/index.html?utm_source=suzunone

According to figure 3.4, a significant portion of telephones run Android version 4 and higher as of May 4th

2015. As Android is built to be backwards compatible; applications that target a lower API level than that a device is running does not create serious problems. Running an application that targets a higher API level than the device is not possible, as higher API levels usually implement new features which cause incompatibility in this scenario. Thus whilst targeting a lower API level increases the amount of supported devices, targeting a higher API level gives access to more and better Android functions.

3.3. CHOSEN TECHNOLOGIES

This chapter reviews our choices for the application with a detailed explanation of our final choices out of the considered options presented in 3.2.

3.3.1. FILE TRANSFER

For the file transfer functionality, our client pushed for the usage of Android Beam, which means that we combine both NFC and Bluetooth in our solution. Android Beam is a native Android function for Android devices that support NFC. By holding the backs of two Android devices against each other while this function is enabled, the two devices will perform a NFC handshake. If one of the users then taps his device's screen, it will start a Bluetooth transfer between the two devices. What kind of file is sent through this exchange is dependent on this context of the sending device and the file can encompass a simple file such as a single contact number to a large video file.

3.3.2. UPLOAD

For the upload functionality of the application, our client requested that we integrate parts of Tribler. With Tribler the application will be able to preserve the user's identity when the file is uploaded to the internet. In order to implement Tribler, we will use code that will be created by the Anonymous HD video streaming for Android bachelor project group later in the project's time line. To prepare our application for this integration we set up our code base so that the GUI already has the proper functions to call upon Tribler code.

3.3.3. GUI

For the application's GUI, we decided to use Kivy. While we lose the ability to easily define a proper layout to the application, we decided that the ability to use Python outweighed this aspect due to the requested Tribler integration. In an earlier bachelor project, Tribler Play [5], the team went with the native Java GUI which caused some issues when they had to access Tribler which they solved by implementing Tribler as a service which was accessible through a XMLRPCServer. We intend to avoid similar problems by using Kivy, as it will be possible to call upon Tribler functionality by accessing it as a Python library.

3.3.4. ANDROID VERSION

For the application's Android version we considered the following criteria: Android function requirements and the size of the user base. As we determined that we will be supporting Android Beam for the file transfer, this meant that the lowest API level that we could target was level 16, or Android 4.1.x Jelly Bean, as this was the first version of Android to support file transfer for Android Beam (which was introduced in API level 15, or Android 4.0.3 Ice Cream Sandwich). Luckily, as shown in 3.4, this API level can support 88.7% of the total amount of Android enabled devices. Thus by targeting Android API level 16, we fulfill both our criteria of being able to implement our application's functionality and reaching a large user base.

4 Design Phase

This chapter will highlight the design process of the project as well as the final choices made about the design of the product application. In section 4.1 the requirements of the final product will be discussed.

4.1. Requirements (MoSCoW)

This section lists all identified requirements for the final product. As the final project has several requirements the MoSCoW method was used to add priority to all these requirements. This will help in keeping track of the importance of requirements, which will help in the planning of the development.

4.1.1. Must have(s)

- **Shadow Internet File Transfer**
  The user should be able to send and receive files through the application without the need for an internet connection. The files types that must be supported must at least include the application's own APK and video files.

- **File List**
  The user should be able to easily browse through their recorded videos in order to find the videos that are going to be sent through the application. The user should also be able to rename the files in the file list and have the option to delete them.

- **Production Ready User Interface**
  The application should have an user interface that is easy to understand and is similar to other major Android applications in look and feel. This will make is easier for users to properly navigate through the application.

- **Video Recording**
  The user should be able to record videos with the application.

4.1.2. Should have(s)

- **Tribler File Upload and Seeding**
  In order to make the process of spreading the video to the internet easier for the user, the application should be able to create a torrent file of the video and be able to upload this torrent to the internet through Tribler.

- **Tribler Torrent Search**
  The application should support the Tribler torrent search in order for users to easily find the videos that are uploaded by other users.

- **Private Storage**
  For a better user experience it is useful if the application has its own storage folder and can redirect received files from Android Beam to this folder.

- **Google Play Release**
  The application should be made with a possible release onto the Google Play Store and this should play a role during development choices.

- **Single Button Recording**
  For a smoother user experience, it's important that the camera is fully integrated into the application and can be called upon with a single button press.

- **Viewing videos**
  The user should be able to view their recording and downloaded videos.

4.1.3. COULD HAVE(S)

- **Advanced Tribler Support**
  Tribler also features support for Channels that refer to all files uploaded by a certain user. The application could integrate these features to extend searching and uploading of videos.

- **File Encryption**
  File encryption would give plausible deniability, stopping third parties from viewing any files the user has collected.

- **Wi-Fi Direct Support**
  Wi-Fi Direct support is considerably faster than Android Beam, which uses Bluetooth. File transfer over Bluetooth takes a long time, as video files are generally large.

4.1.4. WOULD HAVE(S)

- **Detailed Video Information**
  Once Tribler torrent search has been implemented, we could retrieve additional data about downloaded video files such as the video length. We could also allow the user to add such data to their own recorded videos.

- **DroidStealth Morphing**
  DroidStealth's morphing capability allows the application to hide itself from casual inspection. This capability would be useful in highly censored regions.

- **Full Tribler support**
  Besides the features mentioned in both 4.1.2 and 4.1.3, the application could integrate the entirety of Tribler to have access to all the features that Tribler offers for creating, downloading and uploading torrents.

5
Implementation Phase

This chapter describes the project's progression throughout it's duration. In 5.1 the implementation process of the project will be discussed. In sections 5.2, 5.3 and 5.4 the actual development is discussed. Section 5.5 features an experiment to test the speed of the Shadow Internet File Transfer.

5.1. Implementation process

Development is planned using Scrum [33], an agile development strategy that divides planning into specified durations called sprints. The team used sprints that lasted two weeks, which allowed some time to be spent on other tasks such as integrating into the GitHub fork, or preparing for SIG. Scrum is designed to allow for changes in requirements. This made us quick to adapt to changes brought up during weekly meetings or informal status updates.

In order to make code collaboration easier, the project's code has been made available on GitHub. This open source versioning tool allows the team to work independently on code base and merge their changes into a single branch. GitHub also lets developers make a copy of an existing GitHub repository, make changes to it, and request it to be added back into the original. This would let future developers iterate upon our work. GitHub supports code branching, which allowed us to develop new functionality in separate branches. This meant that we always had a working build, regardless of any mishaps during development.

5.2. Sprint 1: Research and Orientation

The first sprint of our implementation phase is characterized by research and trial. The objective of this sprint was to get a feel for Kivy, a Python based GUI library suggested by the client, by implementing camera functionality and the ability to transfer files through NFC.

5.2.1. Kivy

Kivy is a Python based GUI library that was suggested to us. It's multi-platform, and runs on Android devices through the use of python4android. Kivy uses a CSS-like language to easily define GUI elements. Communication with android is done through the use of PyJnius, further discussed in chapter 5.2.2. Kivy comes with a number of easy to use libraries for oft-used functionality. Sadly, these libraries don't always work on Android, in which case the Android SDK will need to be used. In certain cases, objects returned by the Android SDK are incompatible with Kivy and will need to be converted. Kivy has painfully little documentation required to accomplish this.

5.2.2. PyJnius

As mentioned in the above section about Kivy, calling upon code from a different language (in this case, calling Java code from Python 2.7) is something that is not natively supported by Python 2.7. The Python for Android project solves this by a part of the project that is called Pyjnius. Pyjnius is a Python module that can access Java classes using the Java Native Interface (JNI). JNI is a framework within Java that was created to allow programmers to write native methods in case the standard Java class library did not support platform-specific features or libraries. In this case, it is used by PyJnius to easily implement existing Java classes in a

Kivy based application.

The main feature of PyJnius is the autoclass function. Using this function allows a specific Java class to be bound to a Python variable, from which it is then possible to create instances of said class and use the available Java methods of the class. The procedure used for assigning these 'autoclasses' looks similar to the way one would import classes in Java and the behavior of the autoclass is similar to a Java import. PyJnius also hosts several specific autoclass i.e. PythonActivity, which is a class that autoclasses the Android Activity class, but also adds in some additional functions to make the access to several key features of the Activity class a bit easier.

PyJnius also has a way to create a Java class within Python code. This feature is present in case a programmer needs to either extend or implement already existing parts of the Android API and wishes to use Python based modules and functions to fulfill parts of the class' workload. To do this, PyJnius features both a JavaClass and a PythonJavaClass module. With the JavaClass module, it is possible to reflect an exising Java class and it's methods, which makes a reflected Python function call the corresponding Java code. Using JavaClass, it is possible to extend an existing Java class within the class itself, rather than building a seperate class that calls upon the existing Java Class. The PythonJavaClass module is used to implement Java interfaces. The difference between this method of implementing an interface and creating a pure Java class is the ability to use Python modules within the overridden methods. When using the PythonJavaClass approach, methods are overridden by defining the methods (unlike Java, which uses @override). Both JavaClass and PythonJavaClass use Java signatures to identify which classes and methods are accessed or overridden. These signatures were obtained by the 'javap -s' command on the needed classes and methods.

5.2.3. CAMERA FUNCTIONALITY

Initially, adding camera functionality to the application seemed incredibly easy by use of Kivy's library. Kivy's camera functions are easy and straightforward, costing only a few minutes to get to work under Linux. Sadly, the camera did not function on an Android device. This meant that Android SDK would need to be used. There are two main ways to accomplish this. The first way would be to integrate a camera into the GUI, which requires calling the camera directly. According to a forum post online (which incidentally is about as good as documentation gets), there is a texture incompatibility between Android's camera output and Kivy's input. This quickly led to the discovery of the second way. It is possible to open the Android camera application, and let it do the heavy lifting. Sure enough, once the team figured out how PyJnius worked, implementing this was quick.

5.2.4. NFC FUNCTIONALITY

In this first sprint, we wanted to implement basic NFC support for Android Beam that allowed us to send files between devices by using our application. In order to understand and properly implement the Android Beam interface, we looked at the Android developer training database, the NFC API and several additional examples from both earlier projects (such as DroidStealh) and similar projects from the internet.

The first attempt at implementing NFC was by using purely Python and a few autoclassed Android classes in order to send a single file Universal Resource Identifier (URI) to another device. Whilst this approach did give us some insights in how Android processed the several calls that were needed to properly send a file through Android Beam, this approach would not be usable to send multiple files and react to user input regarding which files should be sent.

This led to our second attempt, which was to use the built-in Android Beam function that required a Callback interface. Using this method, it would be possible to first create a list of file URI's that the application should send within the callback class. Once the device would start an Android Beam whilst the application was in the foreground, it would then refer to the callback class instead of the default behaviour of the device (which is to send a link to the application in the Google Play Store). Using PyJnius' PythonJavaClass, we were able to properly implement the CreateBeamUrisCallback interface from the Android API, except for one critical point. The callbackmethod in question (CreateBeamUris()) should return an array of the type Uri. Whilst it was possible to return a List of type Uri, Python for Android and PyJnius would cast this to a generic Object array, which Android was unable to cast to an Uri array. Upon realizing the cast limitation, we converted the PythonJavaClass to an actual Java class, in which we were able to create an Uri array, which we filled with the requested file Uris, circumventing the need for casting.

5.2.5. THUMBNAIL AND FILE SUPPORT

While not originally part of the sprint, the ability to see and choose video files is an important aspect of the NFC functionality. It is after all convenient if the user does not need to recompile the application in order to send a specific video file. Setting up the list of files was relatively easy. Android's SDK nicely delivers the location of the video files, which can be explored natively with python. Thumbnails can be generated by Android as well, so that a developer can easily use it in their application.

Except it's not that easy, because of course it isn't. Kivy does not support Android's Bitmap class. Luckily the Bitmap class allows itself to be compressed into PNG or JPEG, which Kivy does support. Sadly, after doing a number of tricks to get Android to write the thumbnail file into memory in a way that Kivy could easily access it, Kivy still claimed an incompatibility error.

The final solution was to create an array of pixel colors, and using that as the source for kivy's texture. This was not without it's own problem, as Android's colors are in ARGB format while Kivy only accepts RGBA. A relatively simple byte-swap fixed that, though that is sadly harder in python than in C.

Lastly, the thumbnail functionality is slow to the point of freezing the application for several seconds per thumbnail, as shown in figure 5.1. This behavior happens even on the Nexus 6, so it is vital to create a multi-threaded solution in the upcoming sprint so that the user experience will not be impacted.

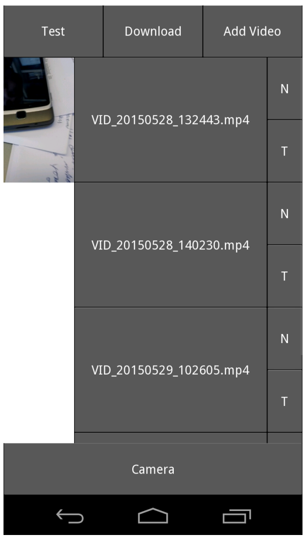


Figure 5.1: Loading the thumbnails noticably slowed down the application.

5.3. SPRINT 2: POLISHED PROTOTYPE

This second sprint of the implementation phase focused on polishing the features from the first sprint into a presentable product. This involved improving camera functionality, load times and redesigning the user interface for an overall better user experience.

5.3.1. THREADING

As noted in chapter 5.2.5, the loading of thumbnails was very slow and froze the application. During a test with the client's phone we also noticed that a sufficiently large video database would seemingly stop the application from loading up. The reason for this was a simple one: the application tried generating every thumbnail within it's first frame.

The first approach on multi-threading was straightforward. The application would create a separate thread for each video file, create a thumbnail and close the thread. This is slightly more complicated than it sounds, as Kivy requires it's UI data to be kept in the control thread. As such, a timer needs to be created to schedule the required interface commands.

There were two problems with this approach. Firstly, it would crash the HTC Desire Z. This was fixed by locking access to the Android thumbnail database to a single thread only. This however, proved to be a temporary fix, as this problem resurfaced when the Android Views were used to load thumbnails (refer to chapter 5.3.2). The other problem was that this approach used 100% CPU power, which meant that the application became slow and unresponsive.

During a test with the client's phone we also noticed that a sufficiently large video database would seemingly stop the application from loading up. The reason for this was simple; the application tried generating every thumbnail within it's first frame.

In order to fix the two main problems, the implementation was changed to make use of a single thread that handles the thumbnails sequentially. The thread makes use of a Python Queue object, which blocks the thread until the Queue is filled. Due to it's nature, the thread needs to be closed down manually upon application shutdown. This is accomplished through entering a manual "finished" object into the queue, after which the thread proceeds to closing down. In this implementation we also fixed the loading problem by limiting the amount of files that are loaded in a single frame to 10. By using 10 as a limit we ensure ourselves that the user is not made aware of the loading mechanism because the screen will be drawn in full when the application opens.

5.3.2. THUMBNAIL IMPROVEMENTS

When the first version of the multi-threading solution was implemented, it became apparent that loading thumbnails remained unacceptably slow. While the solution did allow the user to keep control of the application, it took up so much resources that the entire phone visibly slowed down. For a proper user experience, this would really need to be improved upon.

During implementation of the single button camera (subsection 5.3.4) it became clear that the Android interface could be displayed directly into the Kivy interface through the use of Views. This meant that android's bitmaps could be used directly, thus bypassing the costly conversion functions.

Implementation of the View functionality was, aside from the threading problem, relatively straightforward. Fully integrating the Views into the Kivy UI however proved to be a bigger problem. As Kivy does not have direct access to these Views, it became impossible to place the thumbnails within the scroll-able portion of the interface. This was a disappointment, as the thumbnails loaded extremely fast with this solution.

The final solution is a relatively simple one. Initially, the application requests android to save the thumbnail to local storage. While this isn't incredibly fast, it's very acceptable even on the HTC Desire Z. After conversion, Kivy can load the thumbnails quickly using it's own functions. This means that thumbnail loading is only slightly slower during the first run of the application, and very quick every subsequent start-up.

5.3.3. USER INTERFACE

The user interface was initially designed around Kivy's internal look and feel. This was done mostly from a practical aspect in order to allow the developers to access the application's functionality. This was a workable design but somewhat clunky and would not provide the user experience the client was looking for.

Prior to redesigning the interface, it was important to become acquainted with the android look and feel. As it turns out, there is very little consistency between popular applications and many of them use wildly different interfaces.

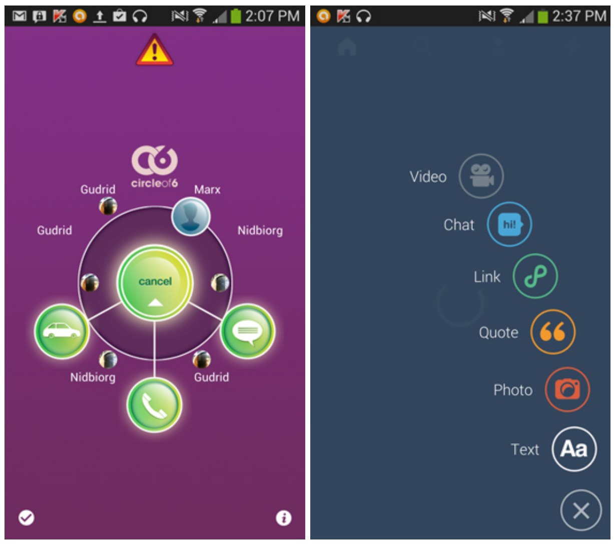
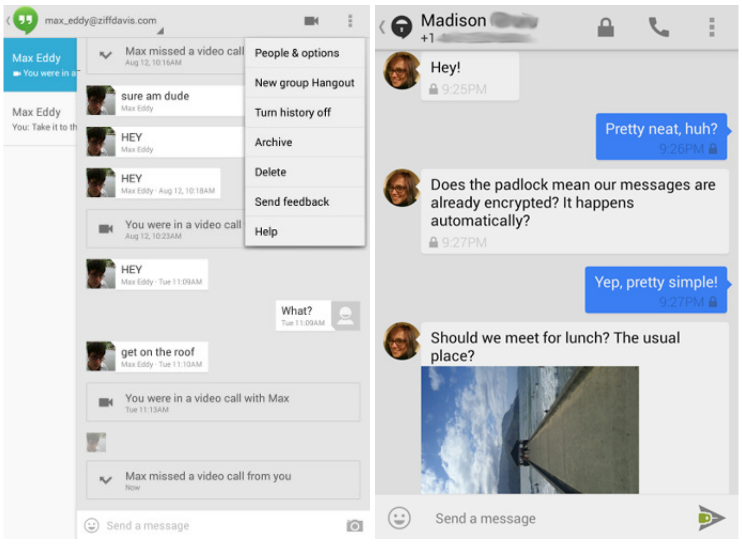


Figure 5.2: Two examples of android interfaces: Circleof6 and Tumblr

Image Sources: https://play.google.com/store/apps/details?id=com.circleof6.williamsuniversity and https://play.google.com/store/apps/details?id=com.tumblr

Evidently, the design of these interfaces have very little in common. Both interfaces rely heavily on iconography and large touch-screen friendly buttons.

The two interfaces our application's interface is inspired on are Google Hangouts and Textsecure, both chat applications. Their monochrome and clean look makes the functionality easy to recognize and use.



Max Eddy
You were in a
Max Eddy
You: Take it to th

Max missed a video call
Aug 12, 10:16AM

sure am dude
Max Eddy

HEY
Max Eddy

HEY
Max Eddy · Aug 12, 10:18AM

You were in a video call
Aug 12, 10:23AM

HEY
Max Eddy · Tue 11:09AM

What?
Tue 11:09AM

get on the roof
Max Eddy · Tue 11:10AM

You were in a video call with Max
Tue 11:13AM

Max missed a video call from you
Now

Hey!
9:25PM

Pretty neat, huh?
9:26PM

Does the padlock mean our messages are already encrypted? It happens automatically?
9:27PM

Yep, pretty simple!
9:27PM

Should we meet for lunch? The usual place?

Figure 5.3: Inspirations for the interface: Google Hangouts and Textsecure

Image Sources: https://play.google.com/store/apps/details?id=com.google.android.talk
and https://play.google.com/store/apps/details?id=org.thoughtcrime.securesms

These applications feature, like the applications featured in figure 5.2, heavy use of iconography. Unlike those applications, they do not use text labels to accompany those icons making for a cleaner look and feel.

Figure 5.4 showcases the current design of our application. It keeps in line with the heavy usage of android specific iconography. This way, a user can easily recognize the functionality he or she is already familiar with. The monochrome design makes for a clean look, with almost no unused space. The current exception to this rule are the widgets used to display the files. These need to be updated with the appropriate meta-info of the video, instead of just their name.

Improving the user interface is an ongoing process. For instance, the triple dot menu is an outdated style that may get replaced by a hamburger menu in the next sprint. Additionally, improvements on the camera's interface needs to be made. Lastly, the video details and playback screen will need to be designed when that functionality gets implemented into the application.

5.3.4. SINGLE BUTTON CAMERA

A major part of this sprint was implementing a built-in camera within the application that could start with a single press of a button. In the first sprint, we had implemented a camera by using the Android MediaStore API to call the default camera application from within our application. While this allowed us to quickly and simply access the camera, it turned out that instantly turning on video recording in this manner was not supported by Android. To create the requested single button recording functionality, we searched the Android API and found that in order to implement this functionality, we needed to firstly create a Camera object, a PreviewSurface object for the camera preview and finally a MediaRecorder object to handle the recording and storage of the video.

Creating a surface for the camera preview turned out to be a challenge. In Android, creating a preview surface is not that difficult. You need to create a View object (or a subclass of View, i.e. Surface- or Texture-View) and attach this View to the chosen Camera object as it's preview output screen. It is then possible to adjust aspects of the preview and the preview screen by calling various methods of both the Camera and View classes to accommodate for issues such as the device's screen size and orientation. In our case it was a different story, because we used Kivy as our GUI language instead of the native Java supported by Android. Whilst Kivy supports a similar approach as Android for Kivy native camera, this approach was incompatible with Android. Two major issues were that the Android camera would not accept a different surface then an Android View as its preview output and the Kivy equivalent of the Android View did not accept the output of the camera due to its encoding. After searching around on the Kivy and Python for Android help forums, we found an application for a built-in Camera that found a way to circumvent these issues surrounding the camera preview.

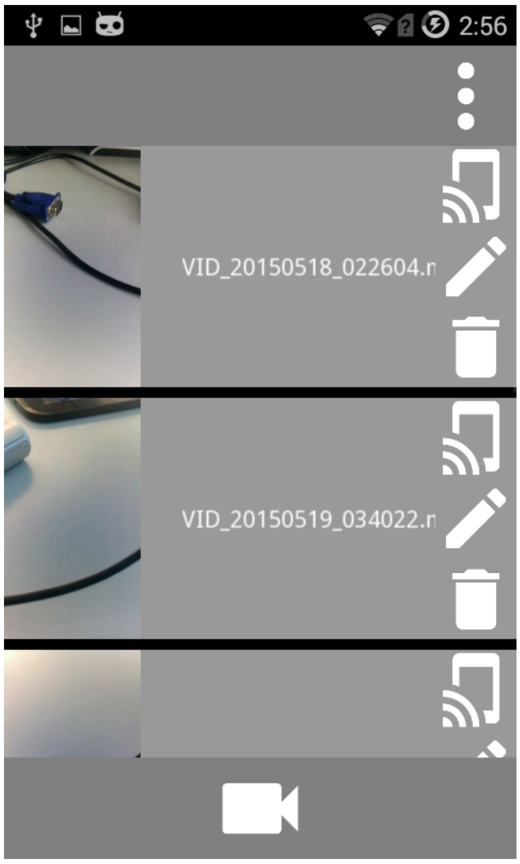


Figure 5.4: The homescreen of our application

The solution was to trick both Android and Kivy. For Kivy, this meant tricking it into believing that the Android View was a Widget and encapsulate this within a Kivy Widget whose only task is to display the View. For Android, we had to trick it to believe that the View was part of an Java GUI, whilst, in practice, it was part of the Kivy GUI. In order to do this, we made a Kivy Widget called 'AndroidWidgetHolder'. This Widget contains an Android View and places this inside a Kivy Widget which we are able to resize and move within the Kivy GUI. As the Widget does not interact with the camera or the preview itself in any way, it can be reused in other parts of the application should we need to use a different kind of Android View in the future. In order to properly provide a SurfaceView to our camera to use for the preview output, we created a 'fake' SurfaceView from our main Activity. Once the camera start outputting the preview onto this View, it will call the SurfaceHolder of the View in order to update it. To handle this call, we implemented the SurfaceHolderCallback interface from the Android API and 'hotwired' it, so that the callback refers to a function that we created, instead of its normal behavior. Thus, if the camera updates the preview, we manually supply the correct data to the correct View. By doing this, we are able to show the preview within Kivy without having to decode the preview data (which would increase the start-up time of the camera and potentially make the preview hang up at times), whilst providing the Android camera with the View it requires. The preview itself is converted from the received data using the standard decoding presented in the Android camera API.

Implementing the Camera and MediaRecorder were much easier compared to the PreviewSurface. Creating a camera object is done with a single method, which opens the camera corresponding to the input given (we chose to default to the rear camera). The MediaRecorder took a bit more time to understand, as it was more of a manager class. Whilst to commands to start and stop recording were simple, it took a bit of time to fully understand which parts of the MediaRecorder had to be set up. For our application we chose to go for the highest available quality of recording that the device supports, so that the recording are as good as possible. Afterwards, we must link the recording camera and the output path of the file, so that the video is properly stored. Storing the recording revealed a long-standing bug within Android to us (first recorded around 2012), namely that files created in this manner are not automatically added to the internal MediaStore and are therefore invisible to everything in Android but the application that created the file. Luckily, a workaround exists by using the Android MediaScanner to add the new file to the internal MediaStore. Because the MediaRecorder features a explicit method to both start and stop the recording, it is possible to create the instant recording functionality, as we can make a button that both opens the preview and starts recording at the same time and, once finished, can stop recording and return to the main screen with a single button as well.

5.3.5. NFC INTEGRATION

The original plan for the NFC integration was to activate the ability to select and send one or several videos and to make the application able to retrieve the beamed files and store these in its own storage. Due to the difficulty of implementing the Single Button Camera, we did not have enough time to do all of these things. We were able to activate and test the additional file transfer functionality, as the essential functions were already written in the previous sprint. At the start of the application, it checks if NFC is available, to avoid crashing by accessing non-existing hardware. After setting up the callback, as described in Sprint 1, it keeps track of an array which is filled the Android Package (APK) of the application, so that it can send the application to another device. Once the user begins adding files, the application removes the APK from the array and adds the corresponding file Uris. Should no video files by selected for the Android Beam transfer, the application will re-insert the APK, so that the application will always send its APK if the Android Beam transfer is started without any files selected.

5.4. SPRINT 3: PRODUCT RELEASE SPRINT

The final sprint focuses on making a marketable product, this includes creating the text for the Google Play store, polishing the GUI, and adding Tribler integration. This sprint also had integration day, where the code from our team was merged with the code from the Anonymous HD video streaming for Android team. Additionally, this sprint was plagued by a number of incidents that severely impaired productivity.

5.4.1. SIG FEEDBACK

The software improvement group (SIG), was kind enough to provide us with feedback regarding the maintainability of our code. While they couldn't calculate a reliable score for it, on account of the code base being too small, they did bring up two points of improvement.

The first of this was that most of our code, namely all python code, resided in a single file. This would become a problem when multiple developers work on it simultaneously. While subversion systems like svn or git mitigate this problem somewhat, it will still be a huge cause of merge errors. We improved this by putting the code's denser sections into separate files. This required some reprogramming, which took longer than anticipated.

SIG also pointed out that our code lacks unit tests. Having unit tests would allow a higher degree of maintainability as it will help prevent existing functionality from breaking when other sections of the code are modified. The team looked into this for a full day, but Kivy's documentation is extremely lacking on the topic of testing. It does have some support for it, but this did not seem compatible with an android application. It appears that it's possible to run unit tests on a desktop, but our application can not be run on a desktop, due to it's reliance on android's functionality and libraries. In the end, we chose not to look into it further, and instead focus our efforts into other important aspects of the project.

5.4.2. INTEGRATION DAY

The client wishes to create a single application capable of housing all of Tribler's functionality, so creating two applications that implement different aspects of it is at odds with that wish. Integration day was created in order to merge our code with that of the Anonymous HD video streaming for Android team. This is one of the general programming tasks that can take an extreme amount of time, depending on the code bases involved. In our case we got off extremely light. Our application, although planned, did not have any Tribler functionality yet. Likewise, the other team's project existed mainly of the libraries we would need to use and some throwaway code to test if the libraries performed. Merging this is as simple as gaining access to the libraries, importing them into our code and providing the other team access to our repository. In practice this took more time than anticipated for third reasons. Firstly, slight delays were caused due to incomplete build-requirements. This meant that the code base needed to be recompiled several times as those problems were discovered and fixed.

Buildozer, in combination with Android, was the cause of the second problem. Google had updated the Android SDK to version 24.3.2, which Buildozer eagerly updated to. This update made Buildozer entirely unable to build either the new or the old version of the code. We had seen similar errors this week related to the SDK updates. Google had changed their version numbering from a simple number to appending "_rc01" or "_rc02" to it, and Buildozer could not parse this. Luckily this was a known issue, and both a workaround and a buildozer update existed.

The final problem was more serious, and caused notably more delay. Integration day was performed on the oldest laptop in the office, and while it had not been an issue up to this point, it suddenly became one. LibTorrent is a huge library, needing six gigabytes worth of memory to compile. Said laptop only had two gigabytes worth of ram and another two as cache. This resulted in the system slowing down until eventually freezing. We finished the merge on another laptop once we came to the conclusion that it was a hardware problem. The laptop was upgraded in order to be able to meet the memory demands.

5.4.3. Buildozer and Android SDK trouble

One of the Tribler developers asked us to move our code into a fork of the Tribler repository, instead of it's own. He felt this was important in order to avoid that this project, like others before it, would be abandoned. This was a good argument and should have been no trouble to implement, so we eagerly obliged.

Murphy's law states that if something can go wrong, it will. Moving the code into another repository means moving the code into another folder and letting Buildozer recreate it's .buildozer folder so that it can compile. Ordinarily this is no problem as it reuses the existing Python-for-Android packages, updating them where needed, and it will be building the application again in a few moments. This all sounds easy enough, so obviously it took a huge turn for the worse. Google had updated the Android SDK to version 24.3.2, which Buildozer eagerly updated to. This update made Buildozer entirely unable to build either the new or the old version of the code. We had seen similar errors this week related to the sdk updates. So we thought it was the same issue as described in section 5.4.2. This versioning problem was supposed to be fixed in Buildozer 0.29, but we couldn't be sure as Google had started using yet another naming standard, this time appending "preview". There existed a workaround for the version problem in earlier versions of Buildozer, so we downgraded to it and modified the workaround to be sure. This didn't work, so we concluded that it must be a Buildozer problem, and issued a support ticket at their GitHub repository. This yielded one potential fix, which proved ineffective.

As we still had no fix, it was time to dig into the inner workings of Buildozer and Android. The other team managed to get their code to work by disabling Buildozer's update functionality and reverting to an old version of the SDK. This fix, while effective, would lead to other problems and would not solve the core issue. This meant that later developers would not be able to build our code, so a proper fix was preferred. The team started to suspect it was not a Buildozer problem but rather an Android problem. Upon inspection of the error codes it seemed to have been a problem with locating the Android Interface Definition Language (AIDL). It's location should have been given by ANT, the java build tool android uses. Hard-coding it's location into ANT successfully fixed fix one error but led to another, more cryptic, error. Replacing ANT with a version from a previous sdk worked, and Buildozer did not update it automatically. This meant we had a working build environment. After updating the support ticket with our workaround, a more targeted fix was discovered by the community; crucial variables were completely undefined in this version of android, and redefining them fixed it.

5.4.4. GUI

The client was not yet happy with the look of the interface, so it's in the process of being overhauled. We will be basing our design on Youtube's application for Android, which can be seen in figure /reffig:youtube

5.4.5. Tribler Integration

As previously mentioned in section 3.3.2, the client wishes for us to implement Tribler support. Implementing Tribler support takes a significant amount of time, previously being dedicated bachelor projects. Thankfully, there is a lot of existing code in both Tribler Play and in the Anonymous HD Video Streaming project which can be reused. This should make it easier to reimplement the Tribler functionality into our code.

5.4.6. NFC File Relocation

Files that were sent by through Android Beam would be deposited in the default 'beam' folder in the Android storage. In order to properly receive these files, an Intent filter for the 'ACTION_VIEW' intent was created. This filter triggers once the user accesses the Android Beam notification after the transfer is complete while

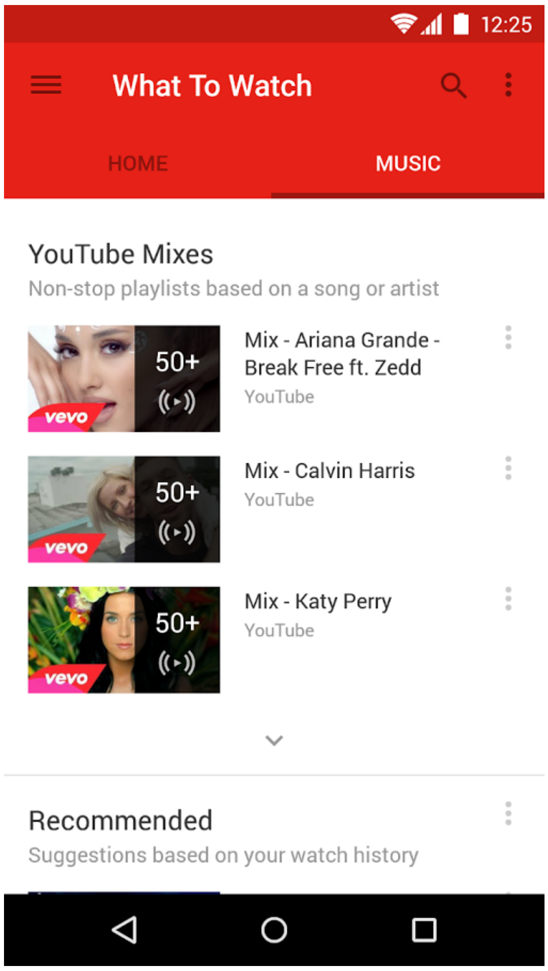

12:25
What To Watch
HOME
MUSIC
YouTube Mixes
Non-stop playlists based on a song or artist
Mix - Ariana Grande - Break Free ft. Zedd
YouTube
Mix - Calvin Harris
YouTube
Mix - Katy Perry
YouTube
50+
vevo
Recommended
Suggestions based on your watch history

Figure 5.5: Youtube's look on android

Image Source: https://play.google.com/store/apps/details?id=com.google.android.youtube

the application is running. By accessing this intent it is possible to access the file's location in the default beam folder. Then it is possible to move the file to a different location and delete the file from the default beam folder.

5.4.7. File Scanning

A problem of the second sprint (section 5.3) was that newly created files were not correctly added to the Android MediaStore, which resulted in the files being 'hidden' from the Android system (after forcefully shutting down the MediaStore (e.g. a system reboot), these files were properly added to the MediaStore). With a FileScanner object, it is possible to manually add the new files to the MediaStore, solving this issue.

5.4.8. Own Storage

The application was using the standard output folder for media files created by the camera (DCIM) for testing purposes. In order to make it easier for users to identify which files were created by the application, the code was changed so that the video files created by the application's own camera are stored in a folder specific to the application. This change will also affect the Tribler download functionality, as it will now point towards this newly implemented folder structure.

5.4.9. Jenkins

During the final parts of the sprint, the Jenkins server was finally made available for use for the project. With Jenkins, it is possible to build the application and then deploy it on several Android devices on the same time for testing purposes. Jenkins can see whether or not a testing suite and run it accordingly. Jenkins also support several methods to create graphs to indicate which tests succeed and fail.

5.5. The Shadow Internet Experiment

This section describes an experiment to determine the speed of the file transfer speed. It was performed with a Google Nexus 6 phone as a host. It would transfer a file to a Google Nexus 10 tablet while in close proximity of each other. This was to simulate the phones being left on a table.

We chose to transfer the application itself, as it had a nice size of 10.9MB. This transfer cost 3 minutes and 43 seconds to complete. This puts the speed of our Shadow Internet at 391Kbit/s or 48KB/s. 100MB is a reasonable size for a small video, and would take roughly 34 minutes to complete. Future development could consider alternative methods like Wi-Fi Direct in order to improve the speed of our Shadow Internet implementation.

6 Conclusions

This chapter presents the conclusion of the project and addresses whether or not the goals and requirements of the final project are fulfilled by the final product. This will be done in section 6.1. In section 6.2 recommendations and further work to the project will be presented. In section 6.3 the team will each individually present their opinion on the project.

6.1. Conclusion

The goal of this project was to create an application that is able to record videos and spread these videos without the use of the Internet, in order to make it impossible for the government to censor these videos (as they control such public services). To produce an application to fulfill this goal, research was performed to investigate potential solutions together with a planning of the development.

Afterwards, key requirements were identified and specified to be: wireless file transfer, the uploading of videos and the user interface with a corresponding programming framework. For the wireless file transfer, Android Beam was chosen, as it is easy to use. For the uploading of videos Tribler was chosen, per the client's request. For the User Interface and programming framework, Kivy was chosen. This would allow for easy access to Tribler.

As of writing, most of the main requirements are implemented in the final product. With the new Shadow Internet application, users can record videos, store them and, if both devices support Android Beam, use it to send videos to each other. All of these features are accessible through simple button presses from the main screen of the application. The last major requirement, being able to upload videos to the Internet using Tribler, is currently unavailable due to time constraints, but the team will attempt to incorporate this last feature in the week leading up to the final presentation of this project as described in section 5.4.

The greatest challenge the team faced was that the Kivy framework combined with PyJnius had little to no documentation. This resulted in increased development time of certain features such as the 'single button camera', as it took more time to either find a proper solution or create one ourselves. Another challenge resulted from incompatibilities between Kivy and Android for media output, as both frameworks accepted different kinds of encoding for similar features. This was solved through the use of Android Views. These View objects could be displayed as though they were a Kivy Widget, which made it possible to display various media outputs without having to convert these to a proper Android format.

These challenges were introduced through our choice of using Kivy to develop the application. Whilst these problems did result in an inability to implement all of our should have requirements, the final application still has the intended benefit of being written in Python. This makes it simple to access Tribler, which is a feature heavily requested by the client.

The team believes that the future use of the application depends on two factors: Google Play Store access and continued integration of Tribler. Should the application be accepted to be published on the Google Play Store, it will be able to spread quicker and gain enough online presence to be considered for use by those who would need the application. The application is currently stable, but the Tribler integration may result in unexpected behavior. Furthermore, Google has a history of banning bittorrent clients from the Play Store, which could happen to our application due to the Tribler access. As section 3.1 showed, there is a market for an application that can spread media and cannot be easily censored and we hope that this application will be considered a good alternative to similar existing applications.

6.2. FUTURE WORK AND RECOMMENDATIONS

As mentioned in section 6.1, the application still has features that could be implementation to further enhance the capabilities and user experience of the application. These additional features include the following:

- **Advanced Tribler Support**
  Due to time constraints, Tribler features such as MyChannel and Channel search were not implemented in the application. These features can be implemented to increase the options the user has while using the Tribler portion of the application to search for other videos and reviewing their own uploaded videos. To make this process easier, the GitHub repository of the project was setup in a manner so that it could be added to the main Tribler GitHub repository.
- **Camera Improvements**
  While the application currently has a camera that can be started with the press of a button, it does not have all features that a camera application can have on Android. Some users may want to have these additional features (e.g. auto-focus).
- **DroidStealth Morphing**
  The application may be able to avoid the censorship that occurs on the Internet, it can still be censored in a more physical way through device inspections. Once the DroidStealth morphing engine is completed, it could be integrated into the application to provide the user with the option to hide the application altogether, thus preventing someone from confiscating the device when they see the presence of the application on said device.
- **File Encryption**
  To increase the safety of the user, file encryption could be implemented in the application. With encryption, it will become harder for a third party, e.g. a government hacker, to access the video files made by the application.
- **Testing Suite**
  As mentioned in section 5.4, the SIG evaluation was negative about the lack of a proper testing suite for the application. This suite was not incorporated due to a lack of information of the workings of testing for a Kivy application for Android and a lack of time to properly investigate such a setup. While the application was continuously tested on several Android devices during development, to properly aid the continued development of the application a testing suite must be added to the project.
- **Wi-Fi Direct**
  Currently, the application uses Android Beam to transfer files from one device to another. Users testing has indicated that the current transfer speed lies around 391 Kbit/s or 48 KB/s. It is possible to use Android Beam to setup a Wi-Fi Direct connection instead of transferring the files with Bluetooth. This would allow the application to send files faster, which can improve the user experience.

6.3. REFLECTIONS

This section contains the personal reflections of the project team members on the project as a whole. These reflections contain thoughts on the project including thoughts on the team's progress during the project, all participating actors of the project and the final product itself.

6.3.1. MARK VAN BEUSEKOM

Project Shadow internet was an interesting project which allowed me to hone my old skillset as a designer. While at times it was hard to translate my vision of the interface into Kivy, it is considerably easier to use than QT or CSS. The client's regular informal meetings helped tremendously in keeping the project on track as adaptations could be made according to the feedback.

Time management was the biggest challenge during the course of this project. The team was understaffed and forced to work under a tighter deadline than is the norm. This meant that often a choice between functionality and documentation had to be made. We also saw this with the testing suite, as we could not justify the time needed to implement it.

6.3.2. Nicolaas Herckenrath

The Shadow Internet project has, while not progressing as smoothly as it should have at times, been quite successful. The final application fulfills the main requirement of being able to share videos without the use of an Internet connection and it does so without requiring users to have extra knowledge of the application, as it uses the standard Android Beam method. There are of course area's of the application that could see improvement. The GUI still needs some tweaking to fully resemble a native Android interface and the buffering process of the camera preview could benefit from some optimization (it currently uses an implementation suggested by the official Android API, but it should be possible to optimize it further as other existing applications are faster).

While we believed that the coding itself would not be as challenging as it should be, it did turn out to pose a difficult challenge. Both Mark and me knew the basics of programming in Python, but a lack of documentation for both Kivy, Python-for-Android and PyJnius made it difficult to implement several features. At several point, we had to look into the source code of the Python-for-Android and PyJnius in order to understand what functions did and if they were usable for our project. I believe the project could have been much easier to complete if we had used the source code of Tribler Play, but our approach will, if the Tribler Team is to be believed, be easier to maintain in the future, as they intend to transform Tribler into a full fledged Python module (in the current situation, a lot of the setup steps of Tribler are handled in the GUI).

It is regrettable that we started with a delay of about a week. Around the last third of the project we noticed that while the development of the code for functionality the application was on schedule, we were only able to keep track on that front by sacrificing time that were originally planned for other parts of the project, such as the final report. A feature that is sorely missed is the (unit) testing suite, which was not implemented due to the fact that it was unclear how to properly implement tests for a Kivy application for Android and how to build this test version with Buildozer. There simply was not enough time to investigate these issues (but I will try to make an attempt, should the Tribler integration for the demo go smoothly). I believe these issues would have been much less severe if we had managed to find a third student, as these tasks could have been properly addressed with by a third team member.

A
Original Project Description

A.1. Project description

The shadow Internet is an alternative communication infrastructure. Under active development for several years, it's specifically crafted to be resilient to sniffing, blocking, filtering and shutdown. A place for free expression and innovation. Censorship is a key threat to The Internet, with the Shadow Internet this project will start to protect you. Android-based smartphones, the TOR protocol, Bittorrent and a novel reputation system form the Internet-deployed technical foundations. For the past years we worked hard with a group of dozens of scientists and engineers to realize this vision (including 3 phd-level cryptographers). The team have come a long way and with additional support we can make this project self-sustainable and ready for release. Your BEP assignment is to contribute to this work and build a fully usable Android prototype from existing pieces of code and publish this on the Android Market.

A.2. Company description

Tribler Team. Read about the team at these URLs:

http://www.reddit.com/r/tribler, http://www.ee.princeton.edu/events/anonymous-hd-video-streaming-and-reputations, http://news.harvard.edu/gazette/story/2007/08/creating-a-computer-currency, http://tweakers.net/nieuws/981
tribler-gebruikt-onderdelen-tor-voor-anonieme-downloads.html, http://github.com/Tribler/tribler/issues/1066

A.3. Auxiliary information

WARNING: this project is challenging and recommended for students experienced in software development and/or honor students.

The smartphone app you will develop enables people to distribute videos by copying them from phone to phone wirelessly. So even without an Internet connection you can share videos and other content. This is specifically targeted for recording and spreading of protest videos. Work on easy-to-use cryptography for protecting content on your phone and masquerading it as innocent content is ongoing. The Shadow Internet ensures people no longer are reliant on websites like YouTube or Facebook to view and share content with friends. Many smartphones have data limits and these deter people from uploading video files. We will let you share content with friends simply by holding your phones against each other. The existing foundations you can use for you work can be found here:

B
Project Plan

B.1. Introduction

B.2. Project Assignment

B.2.1. Project Environment

The project is issued by Dr. Ir. Johan Pouwelse of the Parallel and Distributed Systems group, a section of the Department of Software and Computer Technology (SCT) at the Faculty Electrical Engineering, Mathematics, and Computer Science (EEMCS) of the Delft University of Technology. The Parallel and Distributed Systems group focuses on research into the fields of P2P systems and online social networks, massively multiplayer online games, grids and clouds, and multi-core architectures and parallel programming. This project is part of the realization of the 'Shadow Internet', which is an attempt to make it impossible to censor the Internet, as it is an important place for free and innovative ideas.

B.2.2. Project Goals

The Tribler Team/Distributed Systems group requested an Android application that is able to distribute media files (i.e. pictures and videos) between Android enabled phones wirelessly, so that the user is not dependent on existing sites as Facebook or YouTube. This is important, as the core concept of this project is supporting amateur-journalism in areas where internet is either heavily monitored or shut off entirely. Therefore, the goal of this project is to create an Android application that allows file sharing over a short range by using existing knowledge and projects. The application should preferably be able to be launched on the Android Play store.

B.2.3. Project Description

This project extends an existing Android application named 'DroidStealth'. DroidStealth is an application that is able to hide the presence of both itself and multimedia files on your phone. Additionally, it is capable of using Near Field Communication (NFC) to install itself onto another Android phone, but cannot do the same for multimedia files. The client wishes for the project team to extend this application with the ability to also transfer media files between two phones that have Droidstealth installed. This project will aim to extend Droidstealth to include file sharing capabilities, whilst retaining the concealing nature of the application.

B.2.4. Deliverables

During the course of the project, The team will deliver the following number of artifacts:

Project plan
: This document intended to define the approach and scope of this project

Research document
: Wherein the project team will explore the possible solutions to the assignment as defined in section B.2.3

Implemented Solution
The source code and compiled android application which addresses the assignment as defined in section B.2.3

SIG feedback
The Project team will need to submit the project's code to the Software Improvement group, and discuss their feedback.

Final Report
The last artifact which will detail the entire will be detailed in this report

B.2.5. REQUIREMENTS
The goal of this project is extending an Android application, DroidStealth, with ad-hoc file transfer functionality. It is pivotal that this functionality is both secure and independent of internet access. Further Project requirements will be detailed within the research document.

B.2.6. CONDITIONS
All project documents and code must be handed in by June 19th, 2015. This is one week prior to the presentations scheduled on June 26th.

B.3. APPROACH AND PLANNING
This chapter describes how the project team will approach the assignment as described in section B.2, and provides a rough planning of each phase. A more detailed version of the planning can be found within the research document.

B.3.1. RESEARCH PHASE
During the research phase, the team will explore the possible solutions to the assignment defined in section B.2. This phase will take approximately one week of time.

B.3.2. DESIGN PHASE
The team will focus on the system design during this phase. To assist in this, the team will create a MoSCoW [34] planning of the desired functionality, as well as a Scrum [35] planning detailing the project. The Design Phase will take approximately one week.

B.3.3. IMPLEMENTATION PHASE
The Implementation phase will be characterized by weekly Scrum[35] cycles. At the end of every Scrum cycle, the team will dedicate some time for the reporting process. This phase will take approximately four weeks of time.

B.4. PROJECT DESIGN
B.4.1. ORGANIZATION
The members within this project do not have any specific roles during the span of the project. The client is Johan Pouwelse, Associate Professor of the Parallel and Distributed Systems group. Raynor Vliegendhart is the coach during the project.

B.4.2. PERSONNEL
The project team consists of two Bachelor students in Computer Science at the TU Delft. The members will work full time on the project to ensure both its quality and completion. Both members of the project group have experience with Java programming, but not with Android programming. The contact information of the group is provided below:
Mark van Beusekom
m.l.j.vanbeusekom@student.tudelft.nl
Nicolaas Herckenrath
n.c.herckenrath@student.tudelft.nl

B.4.3. ADMINISTRATIVE PROCEDURES

The Project Team will branch the existing DroidStealth GitHub repository and will use it to expand DroidStealth's functionality. After each Scrum cycle, an evaluation will be recorded for use in the final report.

B.4.4. REPORTING

The project team will regularly schedule meetings with the client, and arrange a meeting with the Coach whenever an artifact has made sufficient progress in order to receive feedback.

B.5. QUALITY CONTROL

In order to assure a high degree of quality within the project's code, the team will make use of Jenkins, a continuous-integration server. Additionally, in order to test the functionality, the team will use jUnit Tests. Lastly, to improve maintainability, the code will be submitted to the Software Improvement Group for feedback.

C
SIG FEEDBACK

The first SIG feedback is as follows:

"Met 197 regels code is deze code-base te klein om een betrouwbare score voor onderhoudbaarheid uit te rekenen.

Wat opvalt in de code is dat vrijwel alle code en logica in 'main.py' zit. Zodra het systeem gaat groeien is het verstandig om de verschillende functionaliteiten onder te brengen in aparte bestanden om ervoor te zorgen dat verschillende ontwikkelaars elkaar niet in de weg gaan zitten.

Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen. "

During the third sprint, we received another set of feedback:

In de tweede upload zien we dat het codevolume is gestegen, van ongeveer 200 regels code in de vorige upload tot 700 regels nu. Dat is in vergelijking met de andere groepjes nog steeds erg weinig, maar we kunnen nu in ieder geval een score voor onderhoudbaarheid uitrekenen. Jullie zitten op 4 sterren. Dat is bovengemiddeld, maar we maken daarbij de kanttekening dat het onderhoudbaar houden bij zo'n klein systeem redelijk eenvoudig is.

Daarnaast hebben jullie nog steeds geen unit test-code geschreven. Jullie benutten de voordelen van test-driven werken hierdoor niet, waardoor het onderhoud en de betrouwbaarheid van het systeem onnodig worden beperkt.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie nauwelijks zijn meegenomen in het ontwikkeltraject.

D
Shadow Internet Infosheet

Title

Shadow Internet

Organization

Tribler Team

Date of presentation

26 June 2015 at 14:30

Description

The Shadow Internet bachelor project focused on creating an Android application with peer-2-peer video transferring through NFC to be released on the Google Play Store. Tribler functionality was also implemented. In order to properly make use of Tribler libraries, the team made use of the Kivy GUI framework. Due to a lack of documentation, it proved challenging to get the Python based Kivy framework to communicate with the Java based Android functionality. Through several formalized meetings and daily informal meetings, the look and feel of the application was constantly iterated upon until release. Unexpected new releases of the Google SDK halted development a number of times, but workarounds were found or created. The final application allows a user to record a video and share it without the use of internet. This means that, for instance, valuable evidence can be passed around in areas where the internet use is restricted or unsafe.

Members of the Project Team

Mark van Beusekom

Mark is an avid gamer and flash animator studying Computer Science at the TU Delft. As an old student of Graphics design, Mark focused mainly on designing the user interface.

Nicolaas Herckenrath

Nicolaas is a Computer Science bachelor student at the TU Delft, who enjoys both puzzles and video games in his spare time. Nicolaas was responsible for the connection between the application and the Android environment.

Both team members have contributed to preparing the report and final project presentation.

Coach:	Ir. Egbert Bouman	TU Delft
Client:	Dr. Ir. Johan Pouwelse	TU Delft
Contact Person:	Dr. Ir. Johan Pouwelse	J.A.Pouwelse@tudelft.nl
Contact Person:	Mark van Beusekom	mark.vanbeusekom@gmail.com

The final report for this project can be found on: `http://repository.tudelft.nl/`.

Bibliography

[1] J. Pouwelse, *Moving toward a censorship-free internet,* IETF Journal **8** (2012).

[2] C. Arthur, *Egypt blocks social media websites in attempted clampdown on unrest,* http://www.theguardian.com/world/2011/jan/26/egypt-blocks-social-media-websites (2011).

[3] M. Chulov, *Syria shuts off internet access across the country,* http://www.theguardian.com/world/2012/nov/29/syria-blocks-internet (2012).

[4] TU Delft Departement EWI/PDS/Tribler, *Tribler website,* http://tribler.org/.

[5] W. Sabée, N. Spruit, and D. Schut, *Tribler play: decentralized media streaming on android using tribler,* (2014).

[6] wtud, *Tribler play repository,* https://github.com/wtud/tsap.

[7] M. De Vos, R. Jagerman, and L. Versluis, *Android tor tribler tunneling (at3): Ti3800 bachelorproject,* (2014).

[8] rjagerman, *Android tor tribler tunneling repository,* https://github.com/rjagerman/AT3.

[9] C. van Bruggen, N. Feddes, and M. Vermeer, *Ti3806 bachelorproject; anonymous hd video streaming for android using tribler,* (2015).

[10] O. Hokke, A. Kolpa, J. van den Oever, A. Walterbos, and J. Pouwelse, *A Self-Compiling Android Data Obfuscation Tool,* ArXiv e-prints (2015), arXiv:1502.01625 [cs.CR] .

[11] droidstealth, *Droidstealth github repository,* https://github.com/droidstealth/droid-stealth/.

[12] kivy, *Kivy python-for-android github repository,* https://github.com/kivy/python-for-android.

[13] LiveQoS, *Superbeam,* http://superbe.am/.

[14] Open Garden, *Firechat,* https://play.google.com/store/apps/details?id=com.opengarden.firechat&hl=nl.

[15] N. Cohen, *Hong kong protests propel firechat phone-to-phone app,* www.nytimes.com/2014/10/06/technology/hong-kong-protests-propel-a-phone-to-phone-app-.html (2014).

[16] Eyewitness Technologies Ltd., *eyewitness,* https://play.google.com/store/apps/details?id=com.changeagency.eyewitness.

[17] eyeWitness, *eyewitness to atrocities,* https://play.google.com/store/apps/details?id=com.camera.easy&hl=en_GB.

[18] K. Curran, A. Millar, and C. Mc Garvey, *Near field communication,* International Journal of Electrical and Computer Engineering (IJECE) **2**, 371 (2012).

[19] IHS Inc., *Nfc-enabled cellphone shipments to soar fourfold in next five years,* http://press.ihs.com/press-release/design-supply-chain/nfc-enabled-cellphone-shipments-soar-fourfold-next-five-years (2014).

[20] P. Kieseberg, M. Leithner, M. Mulazzani, L. Munroe, S. Schrittwieser, M. Sinha, and E. Weippl, *Qr code security,* in *Proceedings of the 8th International Conference on Advances in Mobile Computing and Multimedia* (ACM, 2010) pp. 430–435.

[21] Denso Wave Incorporated, *Information capacity and versions of the qr code,* http://www.qrcode.com/en/about/version.html.

[22] Bluetooth SIG, Inc., *Bluetooth basics,* http://www.bluetooth.com/Pages/Basics.aspx.

[23] S. Carlaw, *Emerging bluetooth verticals,* https://www.bluetooth.org/ja-jp/Documents/BW13_DayOne_Session3_BluetoothTrends.pdf.

[24] P. Chaudhari and H. Diwanji, *Enhanced safer+ algorithm for bluetooth to withstand against key pairing attack,* in *Advances in Computing and Information Technology*, Advances in Intelligent Systems and Computing, Vol. 176, edited by N. Meghanathan, D. Nagamalai, and N. Chaki (Springer Berlin Heidelberg, 2012) pp. 651–660.

[25] Wi-Fi Alliance, *Wi-fi direct,* http://www.wi-fi.org/discover-wi-fi/wi-fi-direct ().

[26] Wi-Fi Alliance, *How fast is wi-fi direct?* http://www.wi-fi.org/knowledge-center/faq/how-fast-is-wi-fi-direct ().

[27] Wi-Fi Alliance, *How far does a wi-fi direct connection travel?* http://www.wi-fi.org/knowledge-center/faq/how-far-does-a-wi-fi-direct-connection-travel ().

[28] Kivy Organization, *Kivy website,* http://kivy.org/#home.

[29] The Qt Company, *Qt website,* http://www.qt.io/ ().

[30] The Qt Company, *Pyside on the qt wiki,* http://wiki.qt.io/PySideDevelopment ().

[31] naranjomanuel, *Python-for-android google code page,* https://code.google.com/p/python-for-android/.

[32] BitTorrent, *Bittorrent and µtorrent software surpass 150 million user milestone; announce new consumer electronics partnerships,* http://www.bittorrent.com/intl/es/company/about/ces_2012_150m_users.

[33] K. Schwaber and M. Beedle, *Agilè software development with scrum,* (2002).

[34] D. Clegg and R. Barker, *Case method fast-track: a RAD approach* (Addison-Wesley Longman Publishing Co., Inc., 1994).

[35] K. Schwaber, *Agile project management with Scrum* (Microsoft Press, 2004).