Delft University of Technology

TI3800 Bachelorproject

Decentralized Media Streaming on Android using Tribler

Final Report

Authors:
Wendo Sabée
Dirk Schut
Niels Spruit

Client:
Dr. ir. J.A. Pouwelse
TU coach:
Ir. E. Bouman
Coordinator:
Dr. ir. F.F.J. Hermans

July 7, 2014

Preface

This report concludes the course TI3800, the final bachelor project. It was commissioned by the Tribler team at the Parallel and Distributed Systems (PDS) group at the faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) at the Delft University of Technology.

The report documents the eleven weeks that were spend on developing 'Tribler Play', an Android application to search and stream media in a decentralized way. Since this goal is similar to that of Tribler, a big part of those weeks were spend on getting the Tribler code to run on Android. The goal of this report is to educate the reader on the workings of the application and to explain the choices that led to this result, as well as documenting possible recommendations to continue this project.

We would like to thank the Tribler team for making the original Tribler code available and for helping us whenever we had a question. We would also like to thank the Android Tor Tribler Tunneling (AT3) bachelor project group for compiling libtorrent and their cheerful presence while working in the same room. Special thanks go to our supervisor Johan Pouwelse for his vision and enthusiasm during this project, to our coach Egbert Bouman for his coding tips, and to Jaap van Touw, for his extensive feedback.

Delft, June 2014
Wendo Sabée
Dirk Schut
Niels Spruit

Contents

List of Figures

List of Tables

Glossary

activity is a full screen building block in the Android interface.

artefacts are errors that occur when trying to show incomplete or corrupt image of video data.

BitTorrent is a protocol used for decentralized filesharing, used to distribute large amounts of data.

Bloom filter is a space efficient probabilistic data structure to test if an item is part of a set. False positives are possible, but not false negatives, so a Bloom filter check either returns that the item is definitely not part of the set or that it possibly is.

continuous integration is the software engineering practice of merging developer's branches with the main branch at specific times. Usually automated tests are run on the merged result afterwards.

cross-compiled is compiling code for a different instruction set than the one that is used on the computer the code is compiled on.

Dispersy is an elastic database system, or a database designed for peer-to-peers network where a lot of nodes share (parts of) a common database.

emulator is a piece of software that duplicates the functions of a computer system inside another computer system.

forking means making a copy of the source code of a project and start independent development on it.

headless is a term used to describe a system without any Graphical User Interface (GUI).

intent is the method of communicating between different activities in Android.

issue is a task to change something to the code on a repository, like fixing a bug or implementing a feature. Once an issue is completed it, the changes are often put into a pull request.

leeching means downloading files from peers without offering something in return, the opposite of seeding.

libswift is a multiparty transport protocol used to distribute data in a decentralized way, almost like BitTorrent but at the transport layer.

libtorrent is the most popular open source implementation of the BitTorrent protocol, written in C++.

magnet link is a link that includes a hash (a small unique identifier) of a torrent file. This hash can be used to ask other users to provide the full torrent file.

Makefile is a special file that describes how to automatically build and install the source code of a software project.

meshnet is a network topology where all nodes communicate directly with nearby nodes and relay data for the rest of the network, without any central servers.

metadata literally means data about data. In this context, it is used as data about files, such as the length or screenshot of a video.

morphing means seamlessly transforming from one state to another.

MoSCoW is a software development technique to rank requirements in one of four categories: Must have, Should have, Could have and Would/Won't have, which together make up the abbreviation.

node is a device connected to a network.

peer-to-peer is a term used to describe is a decentralized network in which individual nodes (peers) consume and supply resources, using direct connections between them.

polling is the process where the computer waits for external input by checking its availability on a regular interval.

porting means making a project run on a system or architecture that it was originally not designed to run on.

pull request is a request to adopt a collection of changes that were made in a fork of a code repository.

Python bindings are Application Programming Interfaces (APIs) providing 'glue code' to use a library in Python, that was originally not written for Python usage.

seeding is the process of making a torrent available to other peers when you have the complete file(s), the opposite of leeching.

segmentation fault is a fault raised by hardware to notify an operating system about a memory access violation.

socket is an endpoint of the communication flow inside a computer network.

sprint is a cycle of the Scrum software development process with predefined length of one to four weeks in which a couple of tasks will be completely finished.

streaming means processing data as it is received, instead of waiting to receive all data and then process it.

stubbing means replacing functionality with a piece of code that does not implement that functionality.

swarm is the collection of computers currently active in the uploading and downloading of a torrent.

thumbnail is a small image that represents a bigger image or video.

Tor is a free network that provides online anonymity by forwarding data through multiple nodes, available from `https://www.torproject.org/`.

torrent is a file that contains information about a collection of files so that it can be downloaded over the peer-to-peer BitTorrent protocol. Can also mean all the files that can be downloaded by a single torrent file.

upstream is used to describe the original project where a fork came from.

Acronyms

API Application Programming Interface.

APK Android application Package.

AT3 Android Tor Tribler Tunneling.

EEMCS Electrical Engineering, Mathematics and Computer Science.

GIMP GNU Image Manipulation Program.

GPLv3 GNU Public License version 3.

GUI Graphical User Interface.

HTTP Hyper Text Transfer Protocol.

I/O Input Output.

IDE Integrated Development Environment.

IP Internet Protocol.

ISP Internet Service Provider.

JNI Java Native Interface.

JSON-RPC JSON Remote Procedure Calls.

MHL Mobile High-Definition Link.

NDK Native Development Kit.

PDS Parallel and Distributed Systems.

PIL Python Image Library.

REST Representational State Transfer.

RPC Remote Procedure Calls.

SIG Software Improvement Group.

SOAP Simple Object Access Protocol.

TLS Transport Layer Security.

TSAP Tribler Streaming for Android Project.

UI User Interface.

UML Unified Modelling Language.

WWW World Wide Web.

XML-RPC XML Remote Procedure Calls.

Summary

The amount of people using smartphones to access shared content has rapidly grown in the last couple of years. Popular services to share the media all rely on a client/server model, which have scalability issues and are prone to censorship. A decentralized approach would make censorship more difficult and growth more sustainable. Thus the main research question of this project is: *How can Android users search and stream media in a decentralized way?* In answer of this question, an Android application was developed that enables decentralized searching and downloading of torrents and streaming of video torrents.

After a Research Phase and a Design Phase, which took a week each, the Scrum software development strategy was used during the Implementation Phase. This phase was divided over three sprints of two weeks, during which several requirements, as defined by the MoSCoW model, were implemented. These three sprints resulted in three iterations of prototypes, of which the last one implemented all *must have* requirements, three out of four *should haves* and even two *could haves*.

The resulting application is based on Tribler, a fully decentralized BitTorrent client, which was developed by the Parallel and Distributed Systems (PDS) research group at the Delft University of Technology. As Tribler is written in Python, a language not supported by Android, a modified version of the Kivy Python for Android framework was used. The reuse of the Tribler code results in a more maintainable codebase, as no additional modifications are needed.

To provide a native Android experience to the user, an Android Graphical User Interface (GUI) was developed in Java. Because the standard means of communication between the Tribler process (Python) and the Android GUI (Java) are limited, they communicate via XML Remote Procedure Calls (XML-RPC). This has the added benefit of providing a simple Application Programming Interface (API) to the Tribler process, which could be reused in versions for iOS or Windows Phone, or to run Tribler as a headless server.

Decentralized data exchange is handled by both libswift, used for small metadata (such as thumbnails), and libtorrent, which is used to download torrent files. Both these libraries had to be ported to Android, of which the latter was generously provided by the Android Tor Tribler Tunneling (AT3) team. In addition, the VLC for Android player was also integrated. The communication between Tribler and VLC goes through a Hyper Text Transfer Protocol (HTTP) server integrated in Tribler, which is used to stream media directly to the user.

The reuse of Tribler code also has some drawbacks: high definition content can not be streamed reliably on low end hardware, such as the Samsung Galaxy Nexus (2011). More recent hardware, such as the LG Nexus 5 (2013), has no such problems. With smartphones rapidly getting faster (following Moore's Law), this was deemed as an acceptable trade-off.

Chapter 1

Introduction

In the last decade the smartphone market has grown from scratch to a market in which more than one billion units are shipped in a single year[1], meaning that more and more people are in the possession of a smartphone. Most of these smartphones are used to access (social) media through the Internet[2]. The emergence of the smartphone market is therefore also related to the massive growth of social media like YouTube, Facebook and Twitter. However, as all these social media are based upon centralized servers, they are prone to be shut down through censorship by governments or Internet providers. Examples of this already exist, such as the censorship, or even complete blockade, of YouTube in many countries[3], and Turkey blocking access to Twitter during its elections[4]. The content index website The Pirate Bay is another example of a website getting blocked all around the world, this time by Internet providers. The availability of content providers and content indexers on smartphones without the usage of any central servers would therefore bring a big blow to censorship, by giving millions or even billions of smartphones owners the ability to access previously blocked content. Because the majority of smartphones sold worldwide run Android, with 79.0% in the second quarter of 2013[5], the client decided to focus on Android first.

Therefore, to fill this gap in the smartphone market, the main research question of this Bachelor thesis is: *How can Android users search and stream media in a decentralized way?* To answer this question, the problem will be analyzed further first. After that, the existing technology is researched to come up with a possible solution that will then be implemented. For a more detailed explanation of the approach to this project, as well as its conditions and restrictions, it is recommended to read the Plan of Action that is included in Appendix C before reading the rest of this report. Additionally, a glossary and a list of used acronyms is included with this report in which the definitions of some technical terms are explained.

The structure of this report is as follows. First, the problem is defined and analyzed in more detail in Chapter 2. Then the results of the research phase

[1] `http://www.idc.com/getdoc.jsp?containerId=prUS24645514`
[2] `http://www.nielsen.com/us/en/newswire/2013/tops-of-2013-digital.html`
[3] See `http://en.wikipedia.org/wiki/Censorship_of_YouTube`
[4] See `http://www.theguardian.com/world/2014/mar/21/turkey-blocks-twitter-prime-minister`
[5] `http://www.gartner.com/newsroom/id/2573415`

are discussed in Chapter 3. The requirements as well as the design and the quality control and implementation plan are addressed in Chapter 4. The results obtained in the three Scrum sprints are described in the Chapters 5, 6 and 7. The outcome of the project is explained in Chapter 8. Finally, the conclusions and recommendations are discussed in Chapter 9 and 10, respectively.

Part I

Orientation Phase

Chapter 2

Problem analysis

During the first meetings with the client, it became clear that the original project description of the *Shadow Internet* [3], a meshnet of devices which are able to share data anonymously, was too ambitious to be realized fully as a bachelor project. However, a first step towards realizing this vision of a Shadow Internet could be made by making it possible for Android devices to join an existing decentralized network to search for media files, and stream those using the BitTorrent protocol [1]. As such, the question that should be answered during this project is:

> Research question: *How can Android users search and stream media in a decentralized way?*

2.1 Current situation

Tribler[1] [4] is a BitTorrent compatible, social media sharing application that allows its users to share media in a distributed way using Dispersy, a distributed elastic database. It is developed at the Parallel and Distributed Systems (PDS)[2] research group at the Delft University of Technology. It is written in Python and currently runs on Linux, OSX and Windows.

Another project is the VLC-libtorrent streaming application[3] by Jaap van Touw. It is developed for Android, and a forking of the VLC for Android[4] application, with the libtorrent library integrated, enabling it to stream videos over the BitTorrent protocol in a non-centralized way. For a more detailed description of the existing technologies that were used in this project, see Chapter 3.

2.2 Project Goals

To answer the research question, several goals must be met:

[1] `https://www.tribler.org/`
[2] `http://pds.ewi.tudelft.nl/`
[3] `https://github.com/javto/Tribler-streaming`
[4] `https://www.videolan.org/vlc/download-android.html`

- Run (parts of) the Tribler code on Android, as this would provide the application with decentralized content discovery. However, as Tribler is written in Python, Python code needs to run on an Android device, which is not a supported language.
- Run libswift on Android. This is a dependency of Tribler, which would provide the application with decentralized metadata, enhancing the searching of media.
- Run libtorrent on Android with Python bindings. This is a dependency of Tribler, which would provide the application decentralized downloading via the BitTorrent protocol.
- Run and integrate VLC for Android into the application. This would provide the application the ability to play downloaded media files.

A more detailed description of the requirements can be found in Section 4.1. As the original problem description, as found in Appendix A, differs greatly, it is only included for the sake of completeness, and not as a description of this project.

2.3 AT3 Team

In parallel to this project, another team of three bachelor students under the name of Android Tor Tribler Tunneling (AT3) will work on bringing a still experimental part of the Tribler code, Tor-like anonymous tunnels, to Android. As both projects have some overlapping parts, some work can be shared between the groups. If time allows it, both the applications could be merged at the end of the projects, making anonymous video streaming on Android devices possible.

Chapter 3

Existing technology

In this chapter the results of the research phase will be presented. The chapter is divided into four sections that correspond with the main parts of the project. The first section will give some information about the prior work that has been done and that could be used for this project. Section 3.2 will present the features and drawbacks of the video decoding frameworks that are currently available for the Android platform. The workings of BitTorrent video streaming and the available choices for a BitTorrent library are discussed in Section 3.3. Finally, Section 3.4 will explain the available options in which content can be discovered.

3.1 Prior Work

In this section information will be given on projects that can be used to further move forward towards the vision of the 'Shadow Internet' [3]. These projects are:

- **Tribler**
  Tribler [4] is a BitTorrent compatible, social media sharing application that allows its users to share media through a peer-to-peer network. It is developed at the PDS research group at the Delft University of Technology. It offers true decentralized searching to replace the traditional BitTorrent index sites. To do this they developed a protocol called Dispersy [6]. More information on Dispersy can be found in Section 3.4.2.
- **VLC-libtorrent streaming application**
  Last year, another Bachelor thesis group developed an application to stream videos over BitTorrent[1]. It uses a forked version of VLC for Android, the libtorrent library and it uses piece picking (see Section 3.3) to buffer and stream media to the user, given a valid torrent file or magnet link.
- **Python for Android**
  To run Python code on Android, the Python for Android framework can be used. The Python for Android framework is based on the Scripting Layer for Android project[2], which is also used by other scripting languages. The

[1] `https://github.com/javto/Tribler-streaming`
[2] `https://code.google.com/p/android-scripting/`

original project has not been updated since August 2012, but luckily the developers of the Graphical User Interface (GUI) framework Kivy have a fork that is currently supported. This fork also includes support for Kivy GUI framework and a system for packaging libraries.

- **The Global Square social network**
  The Global Square application[3] is an application created by members of the Occupy movement[4], with the aim to be a decentralized social network using Dispersy. It uses a forked Python for Android runtime to run a few core parts of the original Tribler code (such as Dispersy and libswift), M2Crypto[5] and netifaces[6] on Android. However, not all code was written in Python, as the GUI was made in Java to use the standard Android interface.

- **Android stealth application**
  One of the groups from the Hacking Lab course IN4253ET created an application to store files in a virtual vault[7]. The application supports morphing to hide itself amongst other applications by changing its name and icon, and is able to hide itself from the launcher. To launch the application you can call a secret phone number or tap an invisible widget.

- **Popcorn Time and Flixtor**
  Popcorn Time[8] is an open source media streamer that streams its media over BitTorrent and sources the torrent files from various BitTorrent trackers. It aims to be an alternative of Netflix[9], free of content restrictions and with always the newest media.

  Flixtor[10] is a Popcorn Time fork which also provides an Android version. This Android version is (currently) not open source, but it provides an intuitive interface that could be used as inspiration.

- **Tor**
  Tor[11] is a privacy enhancing network that aims to provide strong anonymity on the web. Tor encrypts and forwards traffic through various hops, in such a way that a user cannot directly be linked to the data received by the destination. The path of hops that the traffic takes, is changed every 10 minutes.

  Tor does not provide end-to-end encryption for normal web browsing[12] and relies on existing technologies, such as Transport Layer Security (TLS), to encrypt traffic between the last hop and the destination.

[3] `https://github.com/d3vgru/tgs-minimal`
[4] `http://www.occupytogether.org/`
[5] `https://github.com/martinpaljak/M2Crypto`
[6] `https://alastairs-place.net/projects/netifaces/`
[7] `https://github.com/AlexKolpa/AndroidStealth`
[8] `http://get-popcorn.com/`
[9] `https://www.netflix.com/`
[10] `http://www.flixtor.com/`
[11] `https://www.torproject.org/`
[12] Only up to the last hop, unless hidden services are used.

3.2 Video decoding frameworks

The Android application that will be developed during the project should be able to stream videos, therefore it is essential that it can play videos. As is not possible to create a video decoding framework in the timespan of this project and because some high quality video decoding frameworks for the Android platform already exist, an existing framework for video decoding will be incorporated into the application. To choose the framework that fits the project's needs best, this section will list the available options, including their features and drawbacks.

The available video decoding frameworks are:

- **VLC for Android**
  VLC is a well-known open source video player that many people already use on their desktop computer. Recently, the developers have also released a beta version for the Android platform[13] that is both available as an standalone application as well as in the form of a library. The application supports the playback of most video formats[14], streaming videos over network streams[15], subtitles and hardware decoding.

- **Stagefright**
  Stagefright is the native video decoding framework that comes along with Android. It supports video streaming through the network with the Android Application Programming Interface (API). Using this framework is quite easy as no external libraries for video decoding are required, only the Android API needs to be used. However, this framework does not support that many video formats, for instance a MKV video encoded with a H.264 encoder is not supported[16] while it is a combination that is used for a lot of videos on the BitTorrent network. Another disadvantage of Stagefright is that is does not support hardware decoding out of the box[17].

- **Other frameworks**
  Many other video decoding frameworks are available for the Android platform, like for instance the popular application MX Player[18]. However, this player is not open source and is not available in library form and therefore not suited for this project. Most other video players are not useful for this project as well, for the same reasons as MX Player or because they do not support enough of the popular video formats.

After looking at the features and drawbacks of the video decoding frameworks described above, VLC and Stagefright seem to be best suited for the project. After installing the VLC for Android application on a Samsung Galaxy Nexus, a test has been performed to check how VLC and Stagefright handle some popular video formats. The test has been done with several 720p videos and with both MP4 videos with MPEG-4 encoding and MKV videos with H.264 encoding, which where top three video formats used in a sample of the most popular torrents on several BitTorrent content index sites. Both Stagefright and

[13] `http://www.videolan.org/vlc/download-android.html`
[14] `https://www.videolan.org/vlc/features.html`
[15] `https://www.videolan.org/streaming-features.html`
[16] `http://developer.android.com/guide/appendix/media-formats.html`
[17] `https://source.android.com/devices/media.html`
[18] `https://sites.google.com/site/mxvpen/`

VLC were able to play the MP4 videos smoothly. However, the MKV videos could only be played smoothly with VLC, playing these videos with Stagefright was laggy or not even possible for some videos.

In addition, VLC has support for streaming media files over a network connection using various protocols, of which Hyper Text Transfer Protocol (HTTP) is the most basic. It uses the Content-range header[19] to support seeking. Using this feature, This means that VLC can be used as a standalone application in the project, having a vastly smaller codebase in comparison to forking VLC for Android and building upon that.

Therefore, VLC for Android seems to be the most appropriate choice as video decoding framework for this project.

3.3 BitTorrent video streaming

To stream media files, BitTorrent [1] will be used. By default, BitTorrent works by downloading the rarest pieces first with the aim to make all pieces evenly available by all nodes in the swarm.

Tribler uses BitTorrent for streaming videos by having multiple priority levels [5]: all pieces at the current playback position are given the highest priority, with the priority level getting gradually lower as they are further away in time. Pieces before the current playback position have the lowest priority and are only downloaded after all pieces with a higher priority are downloaded. To support video streaming, the BitTorrent library that will be used needs to support per piece priority levels, or piece picking.

According to Wikipedia, the three popular BitTorrent libraries[20] are:

- **libtorrent (rasterbar)**
  The original version of libtorrent, developed by Arvid Norberg[21]. This is the library that both Tribler and the VLC-libtorrent application use. It has support for piece picking and there is a (relatively old) port available for Android[22]. In addition, the AT3 team will try to compile the latest version as part of their project.

- **MonoTorrent**
  This is a BitTorrent library written in C#. The website for this library is defunct[23] and the last stable release is 4 years old. It is unknown if it supports piece picking.

- **rTorrent (rakshasa)**
  A fork of libtorrent (rasterbar), used by rTorrent. It introduces memory mapped file pages to speed up down- and uploading. Currently, it has not been ported to Android.

In the final report of the VLC-libtorrent streaming project, it was discussed how a lot of time was lost trying to compile libtorrent for Android. The project

[19] `https://en.wikipedia.org/wiki/Byte_serving`
[20] `https://en.wikipedia.org/wiki/Comparison_of_BitTorrent_software#Libraries`
[21] `http://www.rasterbar.com/products/libtorrent/`
[22] `https://softwarrior.googlecode.com/svn/tags/RutrackerDownloader/2.6.5.5/jni/libtorrent/`
[23] `http://monotorrent.com/`

ended up using a Android port of libtorrent by RuTracker[24]. A drawback of this port is that it is a 4 year old version, lacking various features, that only works on the ARM architecture, not MIPS or x86. No definitive numbers could be found about the architecture market share for Android, but as all popular and flagship devices[25] are ARM based, this is seen as an acceptable trade off, if the AT3 team will not succeed in compiling a newer version.

MonoTorrent is not actively maintained anymore and would add the Mono Framework as an extra dependency, thus this is not seen as a serious option. As both Tribler and the VLC-libtorrent application use the rasterbar version of libtorrent, and there is an available port for Android for this version of libtorrent, it seems to be the most sensible choice to choose this BitTorrent library.

3.4 Content discovery

In order to share media with other users, there must be a way to discover the content that others are willing to share with you. Traditional, for-profit media sharing services are usually fully centralized. The most popular, more decentralized technologies still rely on centralized websites to index the content.

In contrast, the use of decentralized and nomadic content discovery cannot be disrupted by blocking the access to one or a few central servers, which makes it harder to prevent their use.

3.4.1 Centralized content discovery

Current technologies rely on centralized systems to discover content. The most popular video sharing services, such as YouTube and Vimeo, are fully centralized and are susceptible to blocking by governments or can easily be forced to cooperate with censorship, as most recently has been made clear in Turkey[26]. They do provide a very easy and intuitive interface with a search bar and a list of videos that start playing as soon as you click on them.

As BitTorrent does not provide a way to index and search torrent files, it relies on websites, such as The Pirate Bay, to index them and provide torrent files or magnet links to its users. These BitTorrent index sites are usually hosted in countries with a more liberal view regarding copyright laws to prevent organizations representing the entertainment industry, such as the MPAA[27] or BREIN[28], from taking them down. A lot of countries[29] have started blocking The Pirate Bay and other torrent index websites at the Internet Service Provider (ISP) level however, making them inaccessible for a majority of users.

3.4.2 Decentralized content discovery

Dispersy [6] is a protocol that is used for sending messages through a decentralized network. A group of nodes can be clustered in a community, where messages can be used to share a database in that community.

[24] `http://rutracker.org/forum/index.php`
[25] `http://www.appbrain.com/stats/top-android-phones`
[26] `http://www.theguardian.com/world/2014/mar/21/turkey-blocks-twitter-prime-minister`
[27] `http://www.mpaa.org/`
[28] `http://www.anti-piracy.nl/english.php`
[29] `https://en.wikipedia.org/wiki/Countries_blocking_access_to_The_Pirate_Bay`

This works by letting the node first looks for a similar node in the community by comparing the Bloom filter of his own characteristics with the same Bloom filters of several candidate nodes. The node selects the part of the database it wants to synchronize and sends the most similar node a Bloom filter that contains the parts of the database that the node already has. The receiving node checks for parts that are not in the received Bloom filter and sends those parts back. As each node executes the same algorithm, they fill up the gaps in their database until they all have the same data.

Tribler uses Dispersy communities to share information about torrent files. The three most important communities are:

- `AllChannelCommunity` is used to share torrent files amongst users. This is done by broadcasting random torrent files, and rebroadcasting torrent files that are received.
- `SearchCommunity` is used to search torrent files among the other nodes. It does so by broadcasting your search term to which other nodes reply with a collection of info about relevant torrent files.
- `MetadataCommunity` is used to share thumbnails or movie posters that correspond to a certain torrent file. Tribler automatically generates and broadcasts thumbnails for torrent files that are downloaded, but a user can also select a thumbnail or poster manually.

The first two channels are most relevant. They provide the search and torrent sharing that are used to replace traditional BitTorrent index websites. The last channel is a useful addition that can be used to make the GUI more intuitive.

3.4.3 Nomadic content discovery

In a nomadic system, there is no simple way to have a full overview of the content in a network because communication between nodes is sparse. To help getting a more complete overview of the network, this method relies on the nodes to more actively cache the content they receive from other nodes, even if they do not plan on using it themselves. This way, content can spread without a direct connection between two nodes.

There are two ways to share content information with other nodes, using nomadic content discovery:

- **Broadcasting**
  If a device uses broadcasting, it is always active and broadcasting its presence to other devices, for example via WiFi or Bluetooth. Once two devices find each other, they exchange information. This makes it easy to share information with people without interacting with them personally, creating a much larger network. The drawback of this method is that, in very oppressive environments, it might not be in your best interest to constantly broadcast information as this could be used to track or possibly identify the user.
- **Bumping**
  Bumping is a form of manual synchronization referring to the 'bumping' of two phones when sharing something via NFC, but could also include Wifi

or Bluetooth sharing. Because this requires manual action and personal interaction with the owner of the other device, the network is vastly smaller than with broadcasting. On the plus side, in repressive environments this method is far more secure, because only the people the user chooses to interact with know about his participation in the network.

If this method of content discovery is used, it is probably best to give the user the choice between broadcasting and bumping.

Part II

Design Phase

Chapter 4

Design

Before starting the Implementation Phase, it is useful to think about the structure of the Implementation Phase and of the features and limitations of the application. Therefore this chapter will first list the requirements and constraints in Section 4.1 and 4.2, respectively. After that, Section 4.3 will present the global software architecture that will serve as the backbone of the Implementation Phase. To make sure the quality of the software remains high throughout the project and to structure the Implementation Phase, a Test and Implementation Plan is given in Section 4.4. Finally, the last section of this chapter will present the initial GUI in combination with the most important use cases of the final product. The working title of the application that will be developed is Tribler Streaming for Android Project (TSAP).

4.1 Requirements

To define what the end product of this project should be capable of and what not, this section will list the requirements of the end product. To prioritize the requirements into several categories, the MoSCoW method[1] is used. In this way it is easier to see which features have a high priority and have to be implemented first, over features with a lower priority.

4.1.1 Must haves

1. **Decentralized content discovery**
   When the user types something in the search bar and presses the search button, the application must search for that string on a decentralized network without any central servers. As a result a list of videos must appear.

2. **Stream video files via BitTorrent**
   When the user selects a video he wants to watch, the program must start streaming that video over BitTorrent.

[1] `http://en.wikipedia.org/wiki/MoSCoW_method`

3. **Stream videos with H.264 and MPEG-4 encoding**
   The application should be able to stream videos with a H.264 or MPEG-4 encoding in both a MKV or MP4 container.

4. **Pause the video**
   While the program is playing a video the user must be able to pause the video. After pausing, the user must be able to resume the video.

5. **Seek in the video**
   A slider must be located at the bottom of the screen, which the user can set to a certain position in the video. The video must start buffering from that point and resume to play once enough of the video has been buffered.

4.1.2 Should haves

1. **Support numerous video formats as Tribler**
   The user should be able to play most video formats that can be played with Tribler.

2. **Single tap installer**
   The application can be packed into a single Android application Package (APK) file which can be installed by just tapping on it.

3. **Anonymous onion routing suppport**
   If the other group gets anonymous Tor like onion routing support stable, we should integrate it to enable anonymous streaming.

4. **Bandwidth management**
   The user should be able to manage the bandwidth that the application uses (limit up- and download).

4.1.3 Could haves

1. **Audio streaming**
   The user could be able to download and stream audio files, such as MP3 files.

2. **Search and download file types other than media files**
   The user could search for other file types on the Dispersy network and download them with the program.

3. **Secondary screen support using MHL**
   The user could display the application on a secondary screen, like a computer screen or TV by using an Mobile High-Definition Link (MHL) HDMI cable.

4. **Wireless playback on TV**
   The user could display the media on a TV without using a cable. One way to achieve this would be to use Chromecast or Miracast.

5. **Distribute local media**
   The user could be able to distribute its own media to the network, by creating a torrent file and upload it into the Dispersy network.

6. **Disk space management**
   The user should be able to specify how much disk space the application can use. The user should also be able to specify which downloaded media to keep and which media to purge.

7. **Nomadic content discovery**
   The user must be able to find and distribute data without any internet connection, as described in Section 3.4.3.

4.1.4 Would haves

1. **Multiplatform support**
   The application would work on other platforms like iOS, Windows Phone or Bada.

4.2 Architectural constraints

One constraint is the software version and API level of Android. Fragmentation and supporting older versions of Android is often cited as one of the biggest problems regarding developing for Android. Google provides statistics about the Android versions that the devices using the Google Play Store run on their Android Developer Dashboard[2]. At the time of writing, 82.7% of the devices run Android 4.0.3 or higher, supporting at least API level 15. Given that a lot of extra work comes from supporting versions lower than 4.0 (specifically 2.3), Android 4.0.3 with API level 15 will be the minimum supported version for this project.

Another constraint is the hardware architecture. Currently, both the version of libtorrent that we use and the Python for Android runtime only run on the ARM architecture. This means that the application will not work on any device using the MIPS or x86 architecture. No statistics could be found about the division of the market regarding hardware architecture, but as all major brands use ARM for their flagship and other high selling devices[3], targeting only the ARM architecture should make the application run on a vast majority of Android devices.

4.3 Software Architecture

The application will be divided into several components, as illustrated in Figure 4.1. This is done because code from different projects is used and because these projects were written in different programming languages. It is also done to separate dependencies. If a part uses a library, its dependencies should stay within that part.

The libswift and libtorrent components already exist as libraries, with only the latter available as an older version without any Python bindings. As part of the project, libswift will be compiled for Android and the AT3 team will compile libtorrent for Android with Python bindings. A big part of Tribler will also be ported to Android. The existing implementation of Tribler is written in

[2] `https://developer.android.com/about/dashboards/index.html`
[3] `http://www.appbrain.com/stats/top-android-phones`

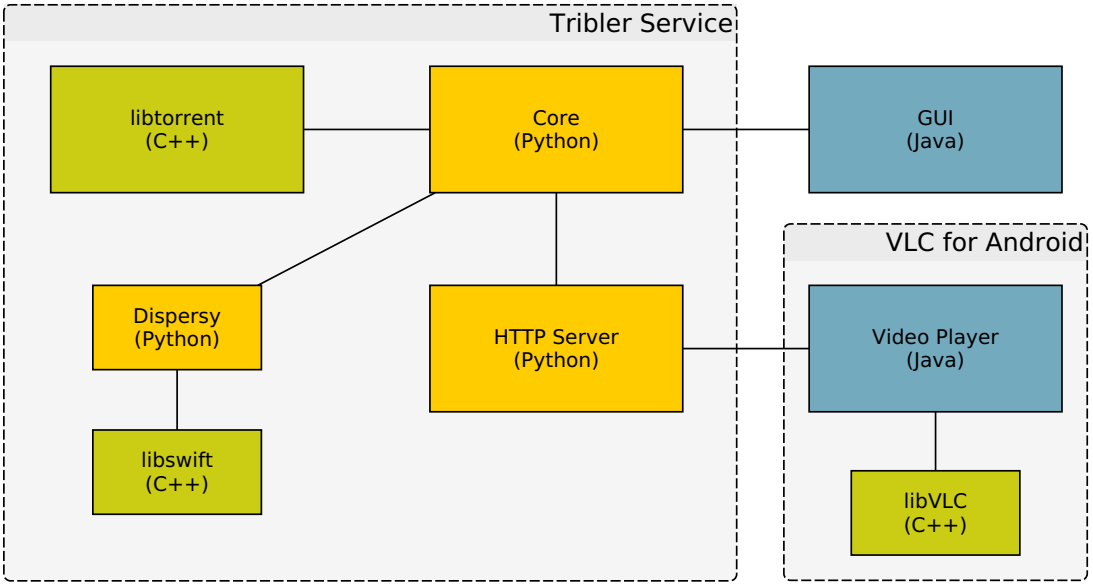


Figure 4.1: The different components that are used in the application, including all major libraries.

Python, so it will run on the Kivy Python for Android runtime. This runtime uses a version of Cython, compiled with the Android Native Development Kit (NDK), to run Python code on Android.

Instead of creating a video player from scratch, the application will use the existing video player from the VLC for Android application and integrate it in the APK. To send the video stream to the video player local HTTP streaming will be used. Both Tribler and the VLC player already support this feature.

The GUI will be written in Java and will use the native Android layout. The communication between the Tribler service in Python and the GUI in Java will be further examined in the first sprint.

4.4 Test and implementation plan

To ensure the code quality of the software remains high throughout the project, it is essential to properly test the software. A test and quality control plan is useful to clarify the strategy used to accomplish this and to get all group members on the same page. The same clarification is also useful to have for the general structure of the implementation phase. Therefore an implementation plan is explained in Section 4.4.2.

4.4.1 Test and quality control plan

During the project, unit testing will be used as well as integration testing and manual testing. The unit tests serve to test whether the behaviour of the small units, in particular classes and their methods, corresponds to their specification.

The integration tests will test the system 'at a higher level', meaning that they will not test individual units like the unit tests, but instead they will test the behaviour of several units at once. In this way the dependencies between the individual units will be tested to ensure the system works as specified. Manual testing will test the system at the highest level as it will test whether the software behaves as specified when a user interacts with the system.

To check whether the unit and integration tests cover most parts of the system, automated code coverage analysis tools will be used to check whether the software requires additional unit and/or integration tests. Documentation within the code will be used to give the people that did not write the code an overview of what a certain piece of code is supposed to do without the necessity to study the code itself. To make sure the code keeps behaving properly after changes are made, a continuous integration server will be used to ensure this. Finally, it is desirable to have a consistent format in the source files. Therefore, one single code formatter will be used to accomplish this.

As the project will eventually create an Android application, tools for the Java programming language have to be used for the quality control mechanisms described above. As Python code for Tribler have to be included in the project as well, the same kind of tools will also be used for code programmed in Python. An overview of the tools, frameworks and Integrated Development Environments (IDEs) that will be used during the project is given in Table 4.1.

	Android	**Python**
IDE	Eclipse incl. ADT[4]	PyCharm[5]
Unit testing framework	Android JUnit	PyUnit
Code coverage analysis tool	Emma[6]	Included in PyCharm
Code documentation	JavaDoc	Doxygen[7]
Continuous Integration	Jenkins[8]	Jenkins
Formatting	Eclipse	PyCharm

Table 4.1: The tools that will be used for quality control

4.4.2 Implementation plan

During the implementation phase the Scrum[9] software development strategy will be used. Scrum was chosen because no members of the team had any prior experience with Android development, which made it hard to plan everything in advance. However, it is expected that the scope of the project is narrow enough for multiple requirements to be completed in a single sprint. The implementation phase lasts six weeks and will be divided over three sprints of two weeks. Each sprint has the goal to create a prototype of the final application that has implemented a certain functionality. To make it easier to refer to

[4] http://developer.android.com/tools/index.html
[5] http://www.jetbrains.com/pycharm/
[6] http://emma.sourceforge.net/
[7] http://www.stack.nl/~dimitri/doxygen/
[8] http://jenkins-ci.org/
[9] http://en.wikipedia.org/wiki/Scrum_(software_development)

these prototypes created in the three sprints, they will be labeled the Foundation Prototype, the Decentralized Search Prototype and the Search and Stream Prototype. For each of these prototypes, a milestone will be created on GitHub.

Each sprint will start with a relatively long meeting in which a detailed analysis of the requirements of the prototype belonging to the sprint will be performed. After that, a broad structure of the code for the new functionality will be created, after which a task division and sprint planning can be made. This division, including deadlines, will be put on GitHub in the form of issues belonging to the milestone of the sprint. This system of issues and milestones will make it easy to check the progress of the tasks as well as the progress of the project in general. It can also be used to track the contributions that are made by the individual team members. At the end of a sprint the prototype has to be finished and tested according to the strategy described in Section 4.4.1. The documentation of the prototype will have to be completely updated as well.

4.5 Use cases

The GUI was first created using empty placeholders and mock data for Input Output (I/O) functions. This facilitates early development by giving a clearer direction and communicates the product design to others, such as the client. To amend this, several use cases are presented below. They describe the major functionality of the application, supported by several screenshots.

4.5.1 Searching for torrents

As the user first starts the application, he is presented with a list of currently popular torrents, as can be seen in Figure 4.2a. The torrents that have an attached thumbnail show their thumbnail, in addition to the name of the torrent, the size of the torrent and the health of the torrent.

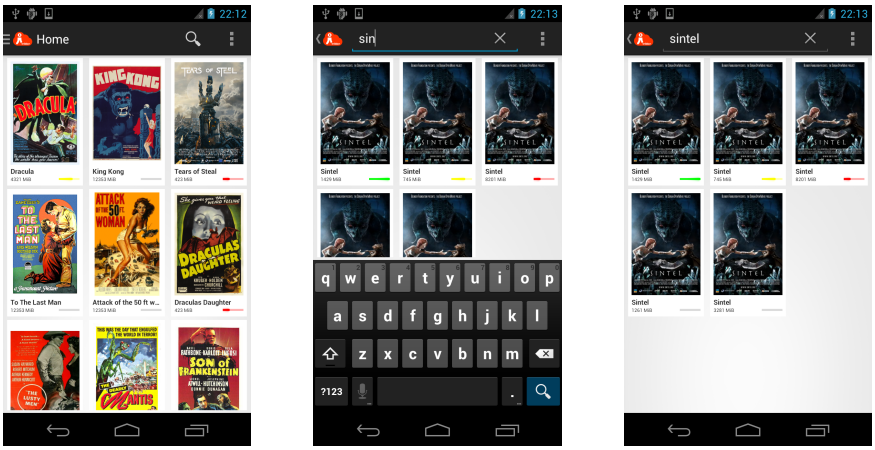
If the user presses the search icon on the action bar, it will expand into a textfield. The user can use the keyboard to type in his search query, as can be seen in Figure 4.2b where the user is starting to search for a movie called Sintel. While typing, already known torrents are filtered to give the impression of a faster search.

After the user taps the search icon on their keyboard to indicate they are done, the application launches a search for more torrents among its peers. As results come in, the user is presented with them, as can be seen in Figure 4.2c.

4.5.2 Streaming a video

Starting from the previous use case, the user taps on one of the videos from the list, in this case the movie Sintel. The user is presented with a screen that shows the information about the selected torrent, as shown in Figure 4.3a.

When the user taps the 'Play video' button on this screen, the phone presents the user with a list of available video players, of which the user picks the one he wants to use. The chosen video player application will then be opened and it will start streaming the selected video. In this case, the user picked the recommended VLC for Android application, which can be seen in Figure 4.3b.

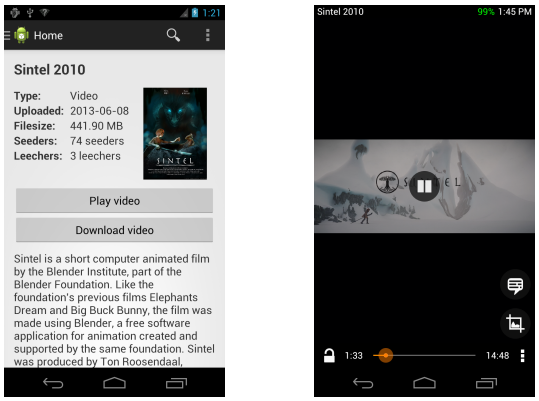


(a) The home screen of the application. It shows currently popular torrents.

(b) The results found while the user is still typing.

(c) The results found after pressing the search button

Figure 4.2: Screenshots showing searching for a torrent.



(a) Information screen of a certain video.

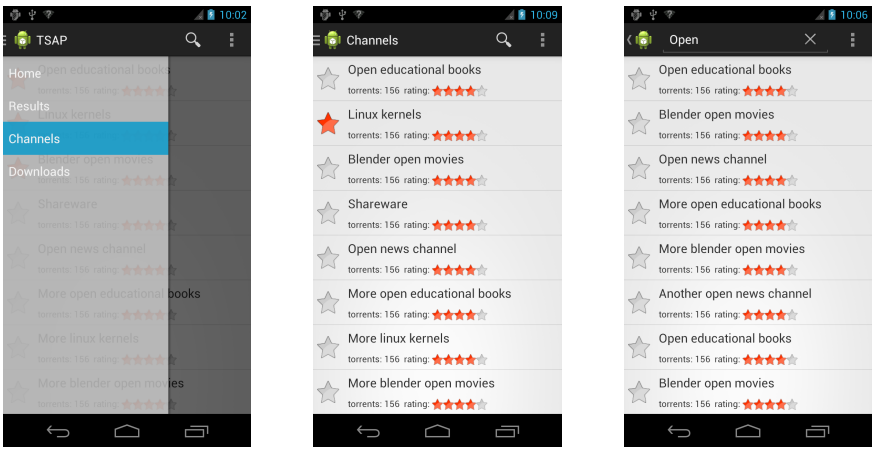
(b) Streaming the video in VLC for Android.

Figure 4.3: Screenshots showing information and playback of a video.

4.5.3 Channels

The user opens the navigation drawer using the icon in the top left corner or by swiping from the left, after which the user picks the 'Channels' option. This is shown in Figure 4.4a. A list of channels appears in which the user marks a channel as favourite by tapping the star icon next to the channel.

The user can also search for channels by tapping the search icon in the action bar, and typing his search query in the textfield that expands at the top of the screen. If the user taps on a channel, a screen with video thumbnails that looks very similar to Figure 4.2a will appear which will contain the videos contained in the selected channel.



(a) The navigation drawer.

(b) List of channels. One is marked as favorite.

(c) Searching for the word 'open' in the channel list.

Figure 4.4: Screenshots showing searching for and favoriting channels.

Part III

Implementation Phase

Chapter 5

First Sprint: Foundation prototype

In this first sprint of the implementation phase, it was necessary to lay the foundation for the rest of the project. A complete list of things that were done during this sprint can be found under the relevant milestone in the GitHub issue tracker[1]. The most important things that were implemented are discussed in this chapter, such as getting Tribler to run on Android in Section 5.1, the communication between the Python runtime and the Java GUI in Section 5.2, the integration of VLC into the GUI in Section 5.4 and how a continuous integration system was set up to maintain quality in Section 5.3.

In the implementation plan (see Section 4.4.2), it was specified that at the start of each sprint a broad structure of the code would be created. It was decided not to do this for this sprint as, for a big part, this sprint consisted of setting things up and getting them to work. In addition to that, the code for the GUI using empty placeholders was already created in the previous phase and functioned as the code structure of the Java part of the application.

5.1 Running Tribler on Android

One of the core parts of the project is getting the Tribler core running on Android using the Python for Android framework. Tribler and its submodules are written in Python, but it also uses several libraries written in C and C++. This means that in addition to getting the Tribler Python code[2] to run, these libraries have to be cross-compiled for Android as well. Therefore this section is divided into two subsections: Section 5.1.1 is about the Tribler code and Section 5.1.2 about its dependencies.

5.1.1 Tribler core package

Tribler is divided into eleven Python packages. Several interesting packages are:

[1] See `https://github.com/wtud/TSAP/issues?milestone=2&state=closed`

[2] Found on `https://www.github.com/tribler/tribler`.

1. `Tribler.Core` is the core package that includes subpackages that do most of the heavy lifting.
2. `Tribler.dispersy` is the Dispersy package that handles all the decentralized database communication.
3. `Tribler.community` is the package that includes all the Dispersy communities.
4. `Tribler.SwiftEngine` is the package that communicates with the libswift binary.
5. `Tribler.Main` is the package that includes all the GUI code.

At first the plan was to take the `Tribler.Core` package and every package it depends on, and create a separate Python package for each of them. After some experimentation it was discovered that at most two or three packages could be excluded this way. In addition to that, a lot of references between the packages had to be modified to make everything work. Therefore the decision was made to make a single package that includes all the Tribler code.

This decision also makes it easier to keep up with the upstream Tribler source code as only a few small additions to the Tribler source code are needed, which can be contributed upstream. Updating the Tribler package is as easy as downloading the latest Tribler code and use a simple script to create a package.

5.1.2 Tribler dependencies

In addition to the Tribler Python code, several other libraries are needed. Some are written in Python, such as Dispersy and pymdht[3], but most of them are written in C or C++ which makes cross compilation for Android necessary. Several of these dependencies were already provided by the Python for Android framework, which includes 'recipes' that describe how to build each library. These include the Cython python interpreter and libraries such as OpenSSL[4], PyCrypto[5], M2Crypto[6], pyasn1[7], SQLite3[8] and netifaces[9]. Compiling and including those in the project was as easy as adding their name to a list of included packages.

libtorrent

Another substantial library that Tribler uses is libtorrent. As compiling libtorrent for Android is not a trivial task, it was originally planned to use a precompiled version that shipped with the VLC-libtorrent streaming application described in Section 3.1. This is a version from a Russian BitTorrent application that can be called from Java. As the design moved to one where Tribler does all the heavy lifting and where the GUI is mostly a shell around it, a version with

[3] `https://github.com/rauljim/pymdht`
[4] `https://www.openssl.org/`
[5] `https://www.dlitz.net/software/pycrypto/`
[6] `http://chandlerproject.org/Projects/MeTooCrypto`
[7] `http://pyasn1.sourceforge.net/`
[8] `https://sqlite.org/`
[9] `https://alastairs-place.net/projects/netifaces/`

Python bindings[10] was needed. After two weeks of hard work, Rolf Jagerman of the AT3 team was able to compile a mostly functional version of libtorrent for Android including these bindings.

libswift

Another dependency that Tribler has is libswift[11]. The newest version of libswift depends on libevent[12] and libcryptossl[13], but the version used by Tribler depends only on libevent. Because of this, and to maintain the highest compatibility with the current Tribler source, the older version is used. Because compiling libevent is also not that trivial, a pre-compiled library was used. Another issue with libswift is that Tribler expects it to be a regular binary, but the included Makefiles only build a library by default. The included Android Makefile had to be modified to include the shell interface and build a binary instead.

APSW

The APSW library[14] is also used in Tribler to save all the channels, torrents and other objects that Tribler collects. This library is a wrapper around SQLite3, providing higher level Python bindings. The build script of APSW downloads the corresponding SQLite3 release and compiles it together with the APSW source.

The standard version specified in the recipe[15] gave I/O errors, most likely because of the big SQL transactions that Tribler uses. After trying numerous other versions, including the same version number as the SQLite3 shipped with Python for Android, it was concluded that the SQLite3 version included by Python for Android was a heavily modified version, with over 2000 differences from the source of the version it claims to be. Compiling APSW against this modified version made the I/O errors disappear.

5.2 Communication between Python and Java

To run Python code, the Kivy Python for Android framework is used, which is currently the only supported Python runtime on Android. Since it is designed to make platform independent applications in pure Python with a Kivy GUI, it does not offer support for two way communication with the native language of the platform, in this case Java. It does offer support for calling Java code from Python, using the PyJNIus library, but it has no support for calling Python code from Java.

When this was discussed with the client, several solutions were proposed. The client also noted that they would like Tribler to be some sort of standalone package or service, so it could be more easily ported or run on a server with an external client. In this section the different solutions for calling Python code

[10] Which enables Python to call C++ functions in a compiled library.
[11] `https://github.com/libswift/libswift`
[12] `https://github.com/libevent/libevent`
[13] `IncludedinOpenSSL.`
[14] `https://github.com/rogerbinns/apsw/`
[15] Found on the Python for Android mailing list.

from Java will be looked into. In Section 5.2.3, the decision to use XML Remote Procedure Calls (XML-RPC) is motivated.

5.2.1 PyJNIus

PyJNIus is a library designed to enable Python code running under Cython to access Java libraries. It is developed by the Kivy team for their Python for Android framework and it automates the process of making Java Native Interface (JNI) bindings that are used to communicate between Java and C and making Python bindings that are used for communication between C and Python, which would normally be a tedious process to do manually.

With this library a Java class can be exported to Python by using the `autoclass` function. Calls to the resulting class will be converted into calls of its Java original. This can be very useful for accessing Android specific functions, like accessing the hardware or GUI, from Python code. In our case it has few uses, because our GUI already runs in Java. For most uses, Tribler should not access Android functionality and the GUI will call Tribler instead of the other way around.

One way to achieve communication from Java to Python with PyJNIus would be polling. This scheme is used in the TGS-minimal application, which was used as an example throughout the project. With polling, the Java part creates a queue in which it puts messages. The Python part calls a Java function with a constant frequency, which returns the contents of the queue. Python receives these messages and executes the associated functions, optionally sending results back via PyJNIus.

This method is easy to implement and can work well with a few different types of messages. The system does not scale well when more messages are added, because each message has to be implemented manually in both languages. It does not scale to any other languages because pyJNIus only supports calling Java code from Python.

5.2.2 Web services

A more generalized way of making the Tribler code accessible to Java, is by turning it into a web service. A web service exposes data to other processes by specifying a format for storing data and a protocol for sending the formatted data. The format is not bound to a programming language so communication between different languages is possible. The protocol is often build on top of existing protocols like HTTP, which makes communication across the Internet possible.

Protobuffers and ØMQ

One combination of format and protocol is protobuffers[16] and ØMQ[17], which were both developed fairly recently compared to the alternatives discussed in this section. Protobuffers are used internally by Google to efficiently store information that is used between different programs. The format is specified in a text based .proto file, which can be compiled to code in multiple languages

[16] `https://developers.google.com/protocol-buffers/`
[17] `http://zeromq.org/`

to read and write the actual data. The data itself is stored in a binary format, which makes reading and writing the data very efficient[18] [19].

ØMQ is a framework that is used to send messages over sockets (which can include Unix domain sockets or traditional Internet Protocol (IP) sockets) that represent a many-to-many connection. The big advantage over the other options is that ØMQ is bidirectional and thus supports push messages.

REST

A term that is also often heard in the world of web services is Representational State Transfer (REST) [2]. REST is not a clearly defined protocol but an architectural style originally described by Roy Thomas Fielding. When a web service complies with this style it is said to be RESTful. REST is based on the World Wide Web (WWW) and because of that the WWW is said to be RESTful.

A RESTful system has a very clear client server model, where the client pulls data from the server via a symbolic link. The client does not know about the implementation of the server. The pull requests do not change the state of the server, so the state of the session is only defined by the representation of the data that the client has pulled already. A REST style system could useful for handling search requests and receiving thumbnails, where the Volley framework[20] could aid the Java side of the requests.

RPC

A different approach to web services is Remote Procedure Calls (RPC). With RPC it is possible to expose functions on the server which can then be called by connected clients. Several standards exist for formatting the call and the result. The most popular standards are XML-RPC, Simple Object Access Protocol (SOAP) and JSON Remote Procedure Calls (JSON-RPC). XML-RPC is a relatively simple and old standard which is supported on many systems. Python has XML-RPC support in its standard library.

SOAP extends XML-RPC with many features including named variables and classes. It used to be the industry standard, but it is less widely used now because it is seen as bloated. JSON-RPC is a newer standard that is about as complicated as XML-RPC. JSON is more compact than XML and is by many perceived as more readable, which can be usefull during debugging. This is however a personal preference and the performance is also mostly based on the interpreter used.

5.2.3 Selected approach

Because the client wanted Tribler to be some sort of standalone object it was decided to run Tribler as a web service. For communication the following three aspects were identified as most important: The ease of implementation, the robustness of the protocol and how future proof the protocol is. Because relatively small amounts of data are sent over the network, performance and support for

[18] `https://code.google.com/p/thrift-protobuf-compare/wiki/Benchmarking`
[19] `http://damienbod.wordpress.com/2014/01/09/comparing-protobuf-json-bson-xml-with-net-for-file-streams/`
[20] `https://developers.google.com/events/io/sessions/325304728`

advanced features were seen as less important. Because XML-RPC scored well on all three aspects it was decided to use XML-RPC.

Ease of implementation was seen as important because it makes the service more useful to future projects, which was the main reason to turn Tribler into a web service. RPC scored best on this aspect because it only requires code for specifying which functions can be called on the server side and code for handling the calls asynchronously on the client side. Both ØMQ and REST need a user specified format, which requires custom code on the server and the client.

Robustness of the protocol was seen as important because this reduces code maintenance. Newer libraries such as ØMQmight still evolve, while the XML-RPC specification has not been changed since 2003. Of the many libraries that exist for XML-RPC, most will very likely continue to work, because it's unlikely that the specification will be changed at this point.

Future proofness was seen as important, for similar reasons as with the previous two aspects. If the technology and used libraries stay popular longer, the service will be useful in other projects for a longer time, and if the libraries are maintained longer, less work is needed to keep the existing applications running. ØMQ, REST and JSON-RPC seem like the most popular options to use currently across different languages. However, XML-RPC is supported in the Python standard library, so it will probably be supported and used for a long time too. For Java the XML-RPC libraries seem less future proof, but the aXMLRPC library is currently maintained and written specifically for Android.

5.3 Quality Control

To execute the quality control plan described in Section 4.4.1, several mechanisms had to be set up in this sprint. In particular, an Android test project had to be set up as well as a Jenkins continuous integration server that is able to show the test results and the Emma code coverage report. Setting up the test project is described in Section 5.3.1 and setting up the Jenkins server is explained in Section 5.3.2.

5.3.1 Test project set-up

To write tests for the Android application project, an Android test project had to be created and set up by following the steps on the Android developer website[21]. After performing these steps, an empty test project was created and it could be executed from Eclipse. The next step was to actually create test fixtures and add tests to them. At the end of this sprint, all classes from the application project have their own test fixture filled with some basic tests. In practice this means that functional test cases have been written for the `MainActivity` class and all the `Fragment` classes using the `ActivityInstrumentationTestCase2` class from the `android.test` package. The standard Android JUnit `TestCase` class could not be used for testing these classes, as the application's activity has to be started to run the tests and this could not be achieved with the Android JUnit `TestCase` class. For all other classes it was not necessary to launch the application, therefore the default Android `TestCase` class from JUnit has been used. The test project including the test fixtures that has been set up in this

[21] `http://developer.android.com/tools/testing/index.html`

sprint serves as a basis for the rest of the project, as tests for new functionality can easily be added to the existing test fixtures and new test fixtures can easily be added to the test project as well.

5.3.2 Jenkins set-up

To set up a continuous integration server, first an Amazon EC2[22] instance was launched and set up to function as a web server. After that, Jenkins was installed and set up using the tutorial about installing Jenkins on Ubuntu[23]. As the Amazon instance now functioned as a Jenkins continuous integration server, it was time to configure a build job that is able to check out the source code from GitHub, build the Android application project, run the tests on an Android emulator and publish the test and Emma code coverage reports. However, after trying to start an emulator with several different configurations using the Android Emulator Plugin[24], it was discovered that the Amazon EC2 instance was not powerful enough to start an emulator as it did not have enough main memory.

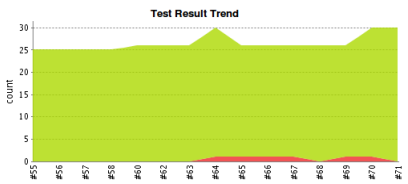
However, the client provided us with an account on the Tribler continuous integration server[25]. As this server is more powerful than the created Amazon instance, it should be able to run an Android emulator with ease. However, after trying numerous emulator configurations, the Android Emulator Plugin kept giving a time-out error when waiting for the emulator to start. As this time-out period could not be changed manually, another solution needed to be found. Therefore, an emulator was created and run on a laptop and a snapshot was saved, which means that the emulator's state was saved so that it can be started much quicker the next time the emulator was run. After changing the configuration files of this snapshot so that it could be run on the Jenkins server, the snapshot was put on the server and it is was expected that the emulator would now be able to start within the time-out period. However, this was not the case, as the plugin did not recognize the snapshot and therefore tried to start a 'normal'emulator. After searching the web, it was found out that this was caused by a bug in the code of the Android Emulator Plugin. Luckily enough the developers released an update for this plugin just two days before, on 19 May 2014, that included a fix for the time-out error as well as the snapshot error. Installing this update on the Jenkins server fixed the problems and the emulator could now be started within the time-out period.

Now that the emulator could finally run, the build job needed to be set up to run the tests on the emulator and publish the test and code coverage reports afterwards. When building the job, the tests were executed but the build was marked as successful while one of the test failed, which is not the desired behaviour. After looking for a solution to this problem, following another tutorial[26] made it possible to retrieve a JUnit XML test report from the emulator and this report could then be published on Jenkins after the build, and mark the build as unstable when one or more tests failed.

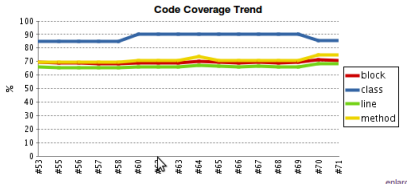
After reading through the list of installed plugins on the Jenkins server, it

[22] http://aws.amazon.com/ec2/
[23] https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins+on+Ubuntu
[24] https://wiki.jenkins-ci.org/display/JENKINS/Android+Emulator+Plugin
[25] http://jenkins.tribler.org/
[26] http://blackriver.to/2012/08/android-continuous-integration-with-ant-and-jenkins-part-2-2/

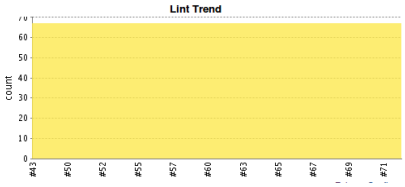
was discovered that the Android Lint Plugin[27] was installed as well. This plugin analyses the code statically for possible bugs and improvements and publishes the results afterwards. As this tool seems to be useful for the project, it was included to the build job as well. At the end of this sprint the test results, code coverage and Lint trends on Jenkins looked as shown in Figure 5.1a, 5.1b and 5.1c, respectively.



(a) The test results trend.



(b) The Emma code coverage trend.



(c) The Android Lint trend.

Figure 5.1: Jenkins trend figures.

5.4 VLC Integration

Last Year, Jaap van Touw made an application for streaming videos from torrents with the VLC player[28]. He had a lot of trouble with seeking, because he was accessing the data of the torrents directly from within VLC, instead of buffering it before playback. Learning from his experience, it was decided to send the data to the VLC player through a local HTTP server using the code of this feature from the desktop version of Tribler. Originally the plan was to install VLC separately and send an intent every time the user wanted to play a movie. This would give the user the option to pick another player and it would be easier to maintain the application, since the video player would be maintained by the VLC team. However, having to install another application could be confusing to the user and it would break the flow of the user interface. Because of this usability issue it was decided to include the VLC player in the APK.

Because only functionality for playing videos was needed from VLC, it was decided to remove the other parts of the VLC-for-Android application. To make sure the application would still run, some files had to be modified. If possible, files were either copied without modifications or stubbed, so that it would be easier to update to a newer version of VLC. Which files are used in what way is shown in Table 5.1. In the end only the `Util` class and the

[27] `http://developer.android.com/tools/help/lint.html`
[28] `https://github.com/javto/Tribler-streaming`

Copied	Stubbed	Modified	Removed

`MediaDatabase`
`VlcCrashHandler`
`WeakHandler`
All 9 files from the `libVLC` package
	`VLCApplication`
	`PreferencesActivity`
		`VideoPlayerActivity`
		`Util`
			All 70 other files

Table 5.1: The usage of VLC files in the application.

`VideoPlayerActivity` class needed to be slightly modified to remove dependencies in unused features. These changes are marked in the code with a TODO comment.

To stream a video with VLC, an intent with the video URL can be sent to the `VideoPlayerActivity`. This was also the behaviour of the VLC for Android application, so no modifications were needed in the code. This functionality is already tested by streaming from a web server. In the final version of the application, the video will be streamed from a local HTTP server hosted by the Tribler part of the application.

5.5 Reflection

In this sprint several major decisions were made about the foundation of the project. The overall architecture of the application shifted from a Java core that communicated with a few Tribler Python packages to a full Tribler process running with a native Android GUI written in Java around it. For communication between the two parts it was decided to use XML-RPC because of its ease and robustness. VLC moved from a full, external application to an integrated player. Tests were written and integrated with the Tribler Jenkins server to enable continuous integration.

In conclusion, the application now consists of three parts: a native Android GUI in Java, a slightly modified and integrated VLC player in Java and a Tribler session in Python that is able to access Dispersy communities and expose its functionality through XML-RPC. The next step would be to integrate XML-RPC in the Java GUI and merge the two parts to create a single Android APK.

Chapter 6

Second Sprint: Decentralized search prototype

In the previous sprint the foundation of the project was laid. In this sprint, which was a few days shorter than the previous one due to holidays, many of the features were added on top of that foundation. All changes made during this sprint are again listed in the GitHub issue tracker under the relevant milestone[1]. The most significant changes will be discussed in more detail in this chapter.

In Section 6.1, the information that is retrieved from Dispersy will be discussed. How this information is communicated to the Android GUI using XML-RPC is shown in Section 6.2. The creation of a single Android package containing both the Tribler service and the GUI is treated in Section 6.3. Streaming a video directly from a torrent is examined in Section 6.4. How all code is tested is explained in Section 6.5. Finally, Section 6.6 will contain a reflection on this sprint.

6.1 Dispersy communities

As explained in Section 5.1, the Tribler package consists of several packages, of which `Tribler.community` contains all the Dispersy communities. Each of these communities is a separate distributed database, in which participating peers can send each other messages to share torrents, channels, votes, comments, etcetera.

In this sprint, two communities were implemented and exposed to the Java GUI over XML-RPC:

1. `SearchCommunity` is used by peers to search for torrents. Each peer keeps a local database in which it stores (actively) collected torrent files. Whenever someone performs a keyword search, it sends a `search-request` message to this community and each of its peer responds with a `search-response` message containing info about the matches from their local torrent database. Whenever a client wants to use the torrent file, it uses the `torrent-request` message to request the actual torrent file.

[1] See `https://github.com/wtud/tsap/issues?milestone=3&state=closed`

2. `AllChannelCommunity` is the community in which the channels are shared. These channels can be used to create a collection of torrents. Whenever a peer subscribes to a channel, it downloads all the torrents that the channel contains and thus helps seeding the channel contents for other peers.

For the specific functionality that was exposed to the Java GUI, see Section 6.2.

6.2 XML-RPC communication

With Tribler it is possible to search locally and remotely in Dispersy communities. On a local search, Tribler will search in its local databases that contain collected Dispersy data such as torrents and channels. Whenever a remote search is launched, Tribler will use Dispersy to ask its peers for search results. Since searching locally is faster than searching remotely, but searching remotely provides far more results, because both modes are used simultaneously.

On the Tribler side of this project, a wrapper containing a `TorrentManager`, which uses the `SearchCommunity`, and a `ChannelManager`, which uses the `AllChannelCommunity`, provide access to Tribler from the GUI. The managers do all the search actions, cache the results, and serve them to the GUI on request. They register their functions with the `XMLRpcManager`, which starts a XML-RPC server and provides the actual communication to the GUI. The functions that are exposed to the client line up with the functions that are shown in Figure 6.1.

From the Android GUI side, searching for torrents or channels is very similar. The items are retrieved from a manager derived from `AbstractXMLRPCManager`, which calls remote functions through tasks derived from `XMLRPCCallTask` and the items are stored in an adapter derived from `AbstractArrayListAdapter`. The communication between these three elements is displayed in the sequence diagram in Figure 6.1. Conform the Unified Modelling Language (UML) standard, synchronous messages are displayed with a closed arrow, and asynchronous messages are displayed with an open arrow.

First, the search function is called which starts searching both locally and remotely. Once the local search results are found, they will be added to the adapter. Because the remote results do not arrive all at once, another system is used for retrieving those results. A poller is started which checks for new search results at a constant interval. If new results are found, they are requested and then added to the adapter as well.

6.3 Creating a single APK

At the start of this sprint, the Android part of the project was in one APK file and the Python part running Tribler was in another APK file. So, the functionality of the application was distributed over two APKs, meaning the user needed to install two applications before he could actually use the application. This was of course undesirable and therefore it was needed to find a way to integrate both parts into a single APK containing all the functionality.

When looking at the code in the distribution that was created by Python for Android, it was discovered that the Python for Android project contains

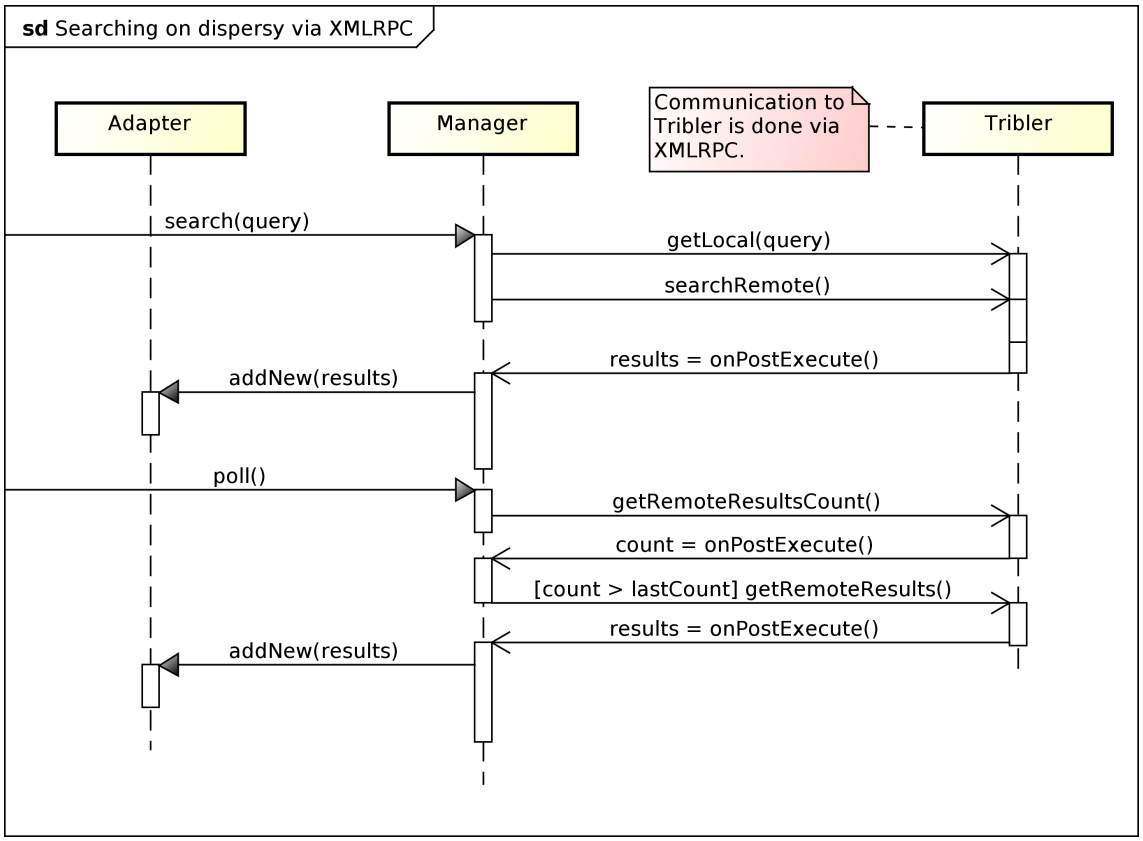


Figure 6.1: Sequence diagram showing the flow of information at the client side when searching. Open arrows represent asynchronous messages, closed arrows represent synchronous messages. The `onPostExecute()` function is called to send the result of a request.

code for what they call the 'Java bootstrap'. When you open an application that has been built using Python for Android, the first activity that is started is the `PythonActivity` class. In the `run` method of this class the following assets are extracted first: `private.mp3` and `public.mp3`. These two files, which are actually gzip compressed TAR files instead of MP3 files, contain all the Python code. After these assets are extracted, the necessary libraries are loaded, the GUI is started and the Python code is executed.

The GUI that is started in the process described above is actually a GUI created with the Kivy framework. As this project will use a native Android GUI instead of a Kivy GUI, all the bootstrap code related to creating and updating the Kivy GUI has been removed to clean up the bootstrap code and to remove the dependency on the Kivy framework. As the project will not use billing, all the code related to this was removed as well. Finally, the bootstrap code also contained a `ResourceManager` class. After removing the Kivy related code, this class was not needed anymore and the standard Android `R` class could be used to manage the application's resources instead.

As the Tribler Python code needs to run in the background, an Android

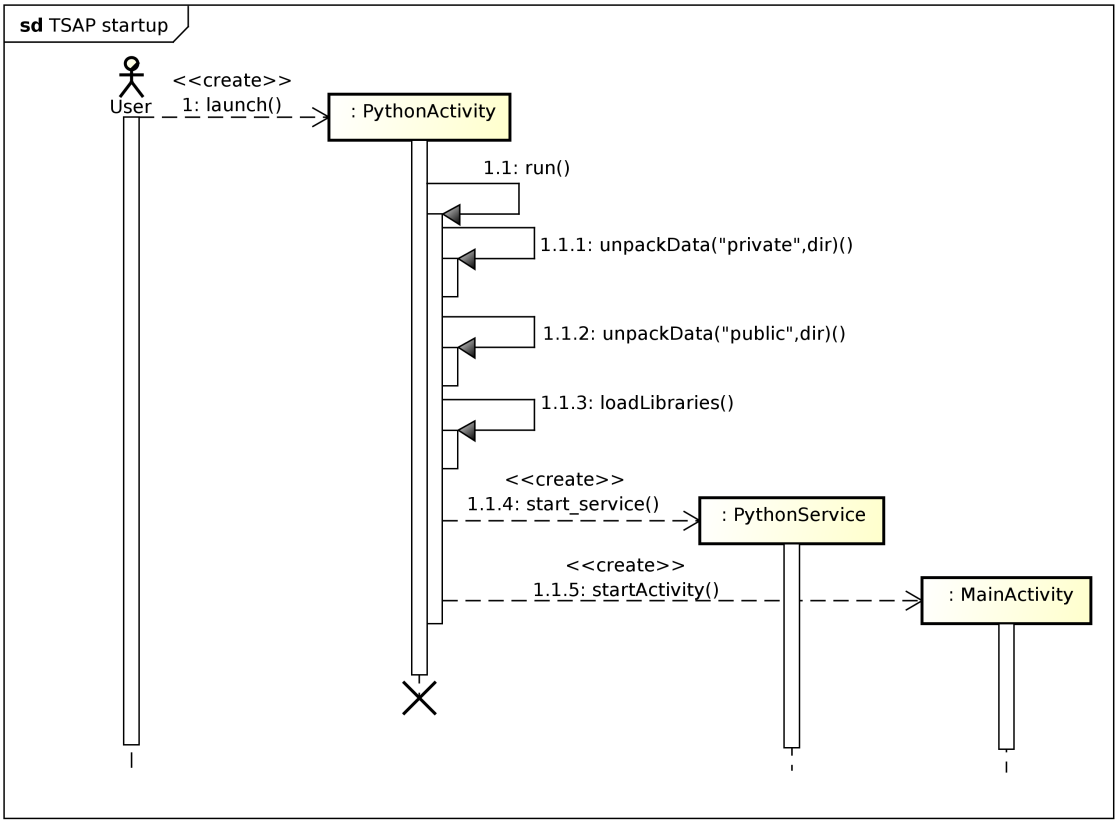


Figure 6.2: A sequence diagram showing the startup process of the application.

service needs to be created. The Python for Android project also includes a `PythonService` class that can execute the Python code in the background as a standard Android service. Therefore, this class will be used to run the Tribler code as a service. After the Python code has been extracted and the necessary libraries are loaded in the `run` method mentioned above, an instance of this `PythonService` class will need to be started. Therefore the `run` method has been adapted to start this service instead of starting the Kivy GUI at the end of the method.

The final issue that had to be resolved was to integrate the Java code of this project into the remaining bootstrap code from the Python for Android project. Basically, this means that after the assets have been extracted, the libraries are loaded and the `PythonService` is started, the `MainActivity` of the project needs to be to launched to start the GUI. Therefore, the code necessary to perform this was appended to the end of the `run` method. A sequence diagram showing the current startup process of the application in a clearer way is shown in Figure 6.2. In the next sprint, the service will be adapted to actually run Tribler instead of the Python code that was used to adapt the startup process as described above, as this code only printed a text message every second.

6.4 BitTorrent streaming

One of the main features of this project is streaming videos via BitTorrent. This means that a video should be downloaded sequentially via BitTorrent and that it starts playing before it is fully downloaded. In the previous sprint, VLC for Android was integrated into the Java GUI (see Section 5.4), and shown to be able to stream and seek video files over a normal HTTP connection.

In this sprint, the first steps were taken to run the HTTP server in the `Tribler.Core.Video` package. This package had a few dependencies on `wx`, the GUI framework used by the Tribler desktop client, which cannot run on Android. Because of this, some unused functions had to be disabled or rewritten to use other libraries such as Python Image Library (PIL)[2]. At the end of the sprint, it was possible to stream and seek a hardcoded torrent via the Tribler HTTP server to VLC for Android.

The modifications that were needed to get this working, plus the modifications that were made during the previous sprint (see Section 5.1), were contributed upstream and merged into the main Tribler codebase[3].

6.5 User Interface (UI) tests

At the start of this second sprint, the code for the Android part mainly consisted of code for creating and updating the GUI. It is not easy to test this particular code with unit tests, as creating and updating the GUI is mainly done within Android specific callback functions like the `onCreate` function. These callback functions do not have return values that can easily be checked with regular unit tests. Excluding the GUI code from the tests did not seem to be a viable option for this project however, because they form a large part of the Java codebase. Therefore, it was decided to write UI tests so the GUI specific code, at least the results that these functions need to have, would be tested as well.

By following the tutorial on UI testing on the Android developer website[4], a new project containing the UI tests was created. After that, tests were written for each fragment in the main project, as these fragments are responsible for updating the GUI with the correct content. As the GUI layout is different in landscape mode than in portrait mode, subclasses of the created test classes were made so that the same tests would also be run when the phone is in landscape mode. Finally, a test class was made that tests the application's behaviour when 'special' events occur, like when the user presses the home button or when he opens the notifications bar.

To integrate the UI tests in the continuous integration process, the tests would also need to run on the Jenkins server. This was more difficult than it sounds as when the tests were initially run, they all failed because the application could not be found when the tests tried to open it. This issue was not present when running the tests on a laptop, either with a real device or an emulator, which made it hard to find out why the application could not be found. After trying several other ways to open the application, the tests kept failing. Finally it was discovered that when using an emulator with the exact same configuration

[2] `http://www.pythonware.com/products/pil/`
[3] `https://github.com/Tribler/tribler/pull/619`
[4] `http://developer.android.com/tools/testing/testing_ui.html`

as used by Jenkins, the tests were failing on the laptop as well. After playing around with the emulator options, it was discovered that the UI tests failed because of the `no-window` option that caused the emulator to run without a display, which is necessary as the Jenkins server is headless. This issue was fixed by designating an Android device as test device, and letting the Jenkins server connect with it via an `adb` connection over the Internet instead of running on an emulator. This has the added benefit having a testing environment much closer to the real usage, plus faster running tests.

The final issue that needed to be solved was that when the tests fail, the build needs to be marked as unstable or failed. As this was also the case with the unit tests created in the previous sprint (see Section 5.3.2), the method to fix this issue was already known. Using the `automator-log-converter` library[5], a JUnit XML report could be created from the output of the test run. This report could then be published after the build and it could give the build the appropriate mark based on the test results. As Jenkins has the ability to merge multiple JUnit XML reports, the report of the UI tests was merged with the unit test report resulting in one test result report and one figure showing the test result trend.

6.6 Reflection

As a continuation of the previous sprint, many features were added or polished during this sprint. Searching for torrents and channels from the Java GUI using Dispersy is now functional. The Python for Android loader that executes the Python code was integrated in the project, making a single APK within arm's reach. UI tests were created and integrated into the Jenkins server and an Android device was designated as test device instead of using an emulator. It was shown that streaming a video over BitTorrent to VLC for Android using the Tribler HTTP server works as well. At the end of the sprint the code was submitted to the Software Improvement Group (SIG) for quality control.

In conclusion, the application now almost consists of a single APK, which should not take up much time in the next sprint. The search functionality works, but still needs some minor improvements. A step for the next sprint would be to create a manager for the download and streaming process, and download and integrate metadata, such as thumbnails, for torrents.

[5] `https://github.com/dpreussler/automator-log-converter`

Chapter 7

Third Sprint: Search and stream prototype

In the third and final sprint of this project, many features were implemented and polished. This means that many GitHub issues have been processed and closed. The complete list of these closed issues are listed under the relevant milestone in the GitHub issue tracker[1]. As it is not practical to address all these issues in this chapter, only the most significant changes will be addressed. During the sprint, the title of the application has also been changed from the working title TSAP into Tribler Play.

The most important use cases of the application, downloading and streaming, will be addressed in Section 7.1 and 7.2, respectively. The addition of Tribler's `MetadataCommunity` is examined in Section 7.3 and the integration of Tribler into a single APK file is explained in Section 7.4. The addition of settings, the enhancements made to the GUI and the feedback given by SIG are discussed in Section 7.5, 7.6 and 7.7, respectively. Finally, a reflection on this final sprint is given in Section 7.8.

7.1 Downloading

At the end of the previous sprint, it was possible to download and stream a single hardcoded video over BitTorrent (see Section 6.4). The libtorrent library that was used was only able to handle one download at a time. If another download was started, it causes a segmentation fault and crashes the whole application.

During this sprint, the AT3 team[2] was able to compile a libtorrent version that was largely free of segmentation faults, which paved the way for a `DownloadManager` that would expose the download functionality to the Android GUI via XML-RPC calls. The first implementation was quite simple, because the Tribler session provides simple functions to add, remove and list downloads. In addition to working fine on a computer, a simple test which added, downloaded and removed a single torrent also worked fine on an Android device.

However, after the first tests with the actual Android GUI, it was discovered that Tribler would stop responding after calling some of the download func-

[1] `https://github.com/wtud/tsap/issues?milestone=4&state=closed`
[2] `https://github.com/rjagerman/AT3`

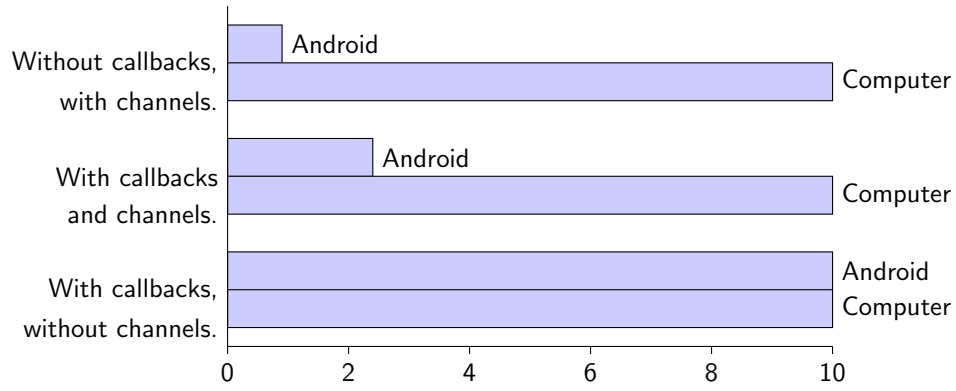


Figure 7.1: Number of consecutive finished downloads (out of 10) of a 102 MB torrent. The numbers are an average of 10 runs on a Samsung Galaxy Nexus and on a Linux computer.

tionality several times. To debug this functionality, a stress test was written. The test tries to add, fully download and then remove a torrent ten times in a row. During the actual downloading, the status is requested in one second intervals. As can be seen in Figure 7.1, the first implementation was usually able to download a single torrent, and would freeze during the second. As a control experiment, the same code was run on a computer, which would do ten consecutive downloads without a hitch.

The following days, one team member was fully devoted to fixing this problem. After some debugging and studying the Tribler code, the `DownloadManager` was rewritten to use callbacks. This meant that instead of calling a function that would return every download on a set interval, the Tribler session would inform the `DownloadManager` about any changed downloads every two seconds. These changes were cached and served over XML-RPC on request. The results can be seen in Figure 7.1, showing that the application would now usually freeze after finishing the second or third download. An improvement, but far from acceptable.

What followed were two full days of debugging, rewriting code and running download tests. The XML-RPC server was first rewritten to use a different thread for every request, then it was rewritten to use the Twisted framework (that was already used by Dispersy) to handle the XML-RPC calls. The tests were even rewritten to bypass the XML-RPC calls altogether and directly call the `DownloadManager` functions, and then the Tribler sessions functions itself, all without making any significant improvement.

On the third day, the project coach suggested that the `AllChannelCommunity`, the community that handles everything relating to channels (see Section 6.1), is quite heavy to run, and it might help to disable it. As a lot of code was already written for the channel functionality, and quite an integral part of the application, this was not considered before. However, as Figure 7.1 shows, disabling the `AllChannelCommunity` fixed the issue completely. In addition, any other Dispersy related actions, such as searching, were sped up significantly. Despite channels being an important feature of Tribler, reliable downloading was more important and as such, it was decided to keep the `AllChannelCommunity`

disabled and hereby any channel and voting functionality was dropped.

7.2 Streaming

As the main use case of this project is to stream videos on an Android phone, functionality had to be added to stream videos from BitTorrent from inside the application. At the end of the previous sprint it was already possible to stream a hardcoded torrent from BitTorrent (see Section 6.4). Besides, the GUI of the application already contains a 'Play video' button since the design phase, but this button did not have any functionality attached to it until this sprint.

Now, the 'Play video' button provides a single click play experience: whenever the user presses the button, a download for the torrent is started as described in Section 7.1. After the download is started, the application opens a dialog that shows the current status of the download to the user. When the torrent is actually being downloaded, the dialog shows the user how much time it will take before the video can be streamed. When the video is ready to be streamed, meaning that 5% of the video has been buffered, the built-in VLC player is opened and it will start to stream the selected torrent from the integrated HTTP server.

When this new streaming functionality was tested on the Galaxy Nexus phones that have been used as testing devices throughout this project, it was found out that they were not able to stream HD videos smoothly. Instead, streaming HD videos on these phones was glitchy as the CPU was not powerful enough to stream HD videos. Therefore, the client provided us with a Nexus 5 phone, which is more powerful and therefore able to stream HD videos smoothly.

7.3 Distributed thumbnail discovery

A fairly recent addition to the Tribler codebase is the introduction of the `MetadataCommunity`. This Dispersy community is used to share extra information about torrents, in particular torrents that contain videos. The metadata of these torrents includes thumbnails (posters, as well as screenshots from the actual video) and information about video files such as play length and resolution. The image files are shared via libswift (see Section 5.1.2) peer-to-peer connections.

Since the first interface design in Section 4.5, the Android GUI relied heavily on the presence of thumbnails to make the interface more intuitive to use. During the previous sprints however, placeholders were used instead of actual thumbnails. At the start of the sprint a small effort was made to implement this community, but when this did not work immediately, priority was given to other features.

Near the end of the sprint, several hundreds of additions were made to the Tribler code base, including to libswift. As it turned out, the version of libswift used previously did not work consistently on Android and would fail without throwing any errors. This resulted in no metadata being downloaded at all. However, after recompiling this newer version for Android, Tribler soon started collecting thumbnails from its peers and storing them in a hidden directory on the SD card, each torrent in its own subdirectory. After running for several

days, and collecting nearly 250 thumbnails, it was found that for every 1 MB storage used, about 24 thumbnails could be stored. This was deemed well within an acceptable range.

The way that Tribler saves the thumbnails made it quite simple to use them in the Android GUI. Whenever a search result is presented to the user, the GUI checks if there is a directory with thumbnails available for each result. If found, it looks for a thumbnail file and loads it asynchronously using the open source library Picasso[3]. If not, a placeholder thumbnail is shown.

As this is feature is not yet released in the main Tribler client, the number of torrents that have thumbnails associated with them is still quite low, with the exception of some keywords such as "vodo". As this feature makes its way into a main Tribler release, it is planned to run a bot that will automatically inject metadata for a large amount of torrents which will significantly boost the amount of available thumbnails.

7.4 Creating a single APK

At the end of the previous sprint it was possible to build a single APK including both the Android code and the Python code. In this APK the Python code ran inside a background service. However, the Python code that was used for this only contained testing code, that printed some text every second. In the real application, it should instead run a Tribler session in the background. To accomplish this, the build script used to create a Python for Android distribution needed to be adapted.

To simplify the build process, the decision was made to add the Python for Android repository as a Git submodule to the repository of this project. This means that people that want to build the code do not need to clone the Python for Android repository themselves and then give the path to it as a command-line argument to the build script. After the submodule was added, the path to the Python for Android project could be put into the build script so that the user does not need to pass it as a command-line argument.

The next step was to let the build script automatically copy the required libraries and assets from the created Python for Android distribution to the project code itself. This was done by adding commands to the build script that run after the Python for Android distribution is created. One new command simply copies all the generated static libraries from the distribution's `/libs` folder to the main project's `/libs` folder to ensure the project has all the necessary libraries at its disposal. A similar command was added to copy the generated MP3 files containing the Python code from the distribution to the project's `/assets` folder.

Finally, two string resources containing the version number of the generated assets needed to be copied to the main project's resources. This was done by putting these resources into a separate file, that is updated every time the build script is run. At the end of this sprint it was possible to build a single APK file containing the Android code and the Python code running a Tribler session, simply by executing the build script and by building the project with Eclipse or Ant afterwards. This new build process has also been put into the Jenkins server to ensure the Python code is up-to-date when the application is tested.

[3] `https://github.com/square/picasso`

7.5 Settings

At the end of this sprint the application has two types of settings: settings that apply to the Android GUI or ones that apply to the Tribler service. However, this distinction is of no interest to the user and therefore they should both be presented inside a single screen. Programmatically, it is also nice to store any settings that are presented to the user in a single place, to prevent them from getting out of sync from each other. Even though most settings apply to the Tribler service, it was decided to let the Android GUI store these settings. This has the advantage that whenever the settings screen is loaded, the settings are immediately presented to the user, without loading time caused by XML-RPC calls. In addition, the native Android settings screen automatically stores its settings without any additional code.

The Tribler service has a special `SettingsManager` that deals with pushing and reading settings from the service via XML-RPC calls. Whenever the application is started, the Python unpacker reads any settings applying the to Tribler service, and passes them to the service via environment variables. They are then loaded by the `SettingsManager`, which makes them available to the rest of the service.

On the GUI side, there is a static `Settings` class, that can be used to read any settings applying to the GUI. This class is also in charge of providing access to any read only settings that can be read from the Tribler service via an XML-RPC call, such as the location of the folder containing the thumbnails. Whenever a setting is changed that also applies to the Tribler service, the `Settings` class is in charge of doing an XML-RPC call to inform Tribler.

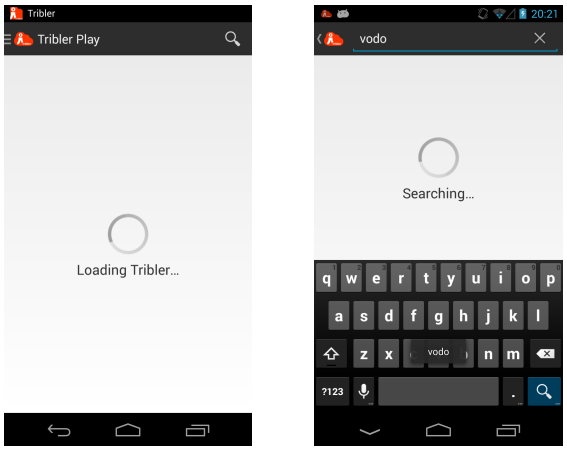
The settings fragment also hosts the about screen and a screen with the licenses of all used open source software. The open source licenses setting opens a dialog that loads a HTML file containing all the licenses of the software used for the application, like the Python for Android project. The about screen is also a dialog that shows some information about the project, like the version of the application and a link to the GitHub page of the project.

7.6 GUI enhancements

The Android GUI sometimes takes a while to respond to an action. For example, when the user starts a search for torrents it takes a while before any torrents are found. To provide the user with more feedback about whether the application is still working properly, loading screens were added in the torrents and download screens. These screens show the user progress information whenever the GUI is waiting for the Tribler service to start, as shown in Figure 7.2a, or when Tribler is busy searching for a torrents, see Figure 7.2b. In addition, messages were added to invite the user to perform an action, such as searching or downloading, whenever they encounter an otherwise empty screen.

An enhancement was also made to the overall color of GUI elements. The Tribler logo uses a very distinct shade of orange (`#FF3300`), which is also the dominant color on the Tribler website and in the desktop version of Tribler. To relate to the Tribler brand it was decided to use this color in the GUI. To change the color of the layout elements, the Android Holo colors generator[4] and

[4] `http://android-holo-colors.com/`



(a) The screen showing that Tribler is being loaded.

(b) The screen showing that Tribler is searching for a torrent.

Figure 7.2: Screenshots showing progress information.

the action bar style generator[5] were used. These two websites generate a set of images to replace the images of the Android standard layout with similar looking images, but with a different color.

The icons used by the integrated VLC player are not from the standard set, so they were not changed by these tools. A GNU Image Manipulation Program (GIMP) script was modified[6] to shift the colors of these icons from VLC orange to Tribler orange using the 'hue saturation' tool.

7.7 SIG Feedback

The SIG was positive about the code that was written at this point and awarded it with the above average score of four out of five stars. The score of five stars was not achieved because of a lower score on code duplication. They noted the class `XMLRPCChannelManager` as an example of this. At the moment the feedback was received it was already decided to remove that class, because channel functionality was no longer needed. Still, similar `XMLRPCManager` classes were refactored, which reduced code duplication amongst other improvements. The SIG was also pleased to see that test code was available for the project. The complete feedback can be found in Appendix B.

7.8 Reflection

In the previous sprints most of the back end of the application was made. Because of that, many visible features could be implemented in this sprint. Downloading was implemented and when a movie is downloading, it can already be streamed. The GUI now shows more info on torrents and downloads and even

[5] `http://jgilfelt.github.io/android-actionbarstylegenerator`
[6] From `http://www.gimptalk.com/index.php?/topic/52571-hue-saturation-batch-script/`

retrieves thumbnails. All features are combined into one package that is easy to build and trivial to install. The SIG also had positive feedback on the code, and the code has been improved according to the single negative point they pointed out.

In conclusion, the project is almost finished. All the main features are working and are integrated into an easily installable package. With a little more testing and tweaking the application would be ready for release.

Part IV

Final Phase

Chapter 8

Project outcome

After the implementation phase consisting of three Scrum sprints, the project is nearly finished. To get a better overview of what has been achieved in this project, this chapter will discuss the main features of the application that was created during the six week lasting implementation phase. This will be done by examining each main feature one by one. For each feature, the technology behind it will be addressed in short and screenshots will be shown to see how the user will experience the feature.

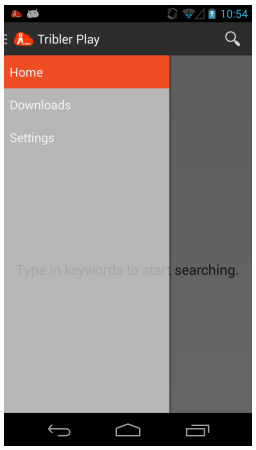
First, Section 8.1 will examine the feature of searching for torrents in more detail. Downloading and streaming these torrents will be discussed in Section 8.2 and 8.3, respectively. Viewing the running downloads and editing them is further examined in Section 8.4. Finally, Section 8.5 will address the feature of viewing and updating the application settings.

8.1 Decentralized content discovery

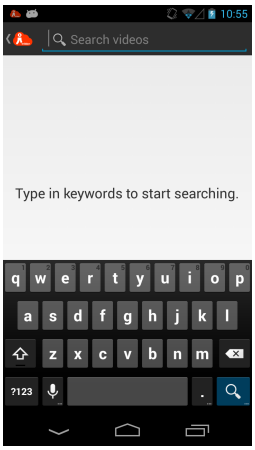
When the user opens the application, he or she is welcomed with the home screen, as shown in Figure 8.1a. This figure also shows the navigation drawer on the left which is used to quickly navigate between the most important screens of the application. The 'Home' option opens a screen in which the user can search for torrents. The 'Downloads' option shows a screen containing the information of all torrents the user is downloading. Finally, the 'Settings' option opens the settings menu in which the settings can be viewed and changed. For this section all actions are performed in the screen selected with the 'Home' option, the other options will be addressed in the sections below.

After the user has opened the application and selected the 'Home' option, the search screen is opened. When the search icon in the top right of the screen is clicked, the user can enter a query to search for, as shown in Figure 8.1b. For instance, when the user enters the query 'vodo' and presses the search button, an XML-RPC call to the Tribler service will be made and it will launch a search for 'vodo' in Dispersy's `SearchCommunity`. When the results are found, they will be shown on the screen with their title, size, health and thumbnail (if available). The screen containing the results for the query 'vodo' is shown in Figure 8.1c. Clicking on a specific result will open a new screen showing more information about the torrent and the ability to play and download it. An example of the

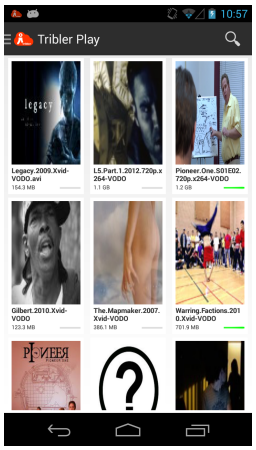
screen that is shown whenever clicking the top right result in the previous figure, is shown in Figure 8.1d.



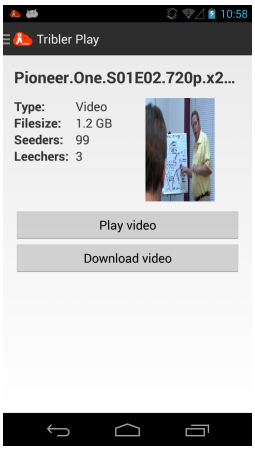
(a) The home screen of the application. It also shows the navigation drawer.



(b) The home screen when the search option is clicked.



(c) The results found when searching for 'vodo'.



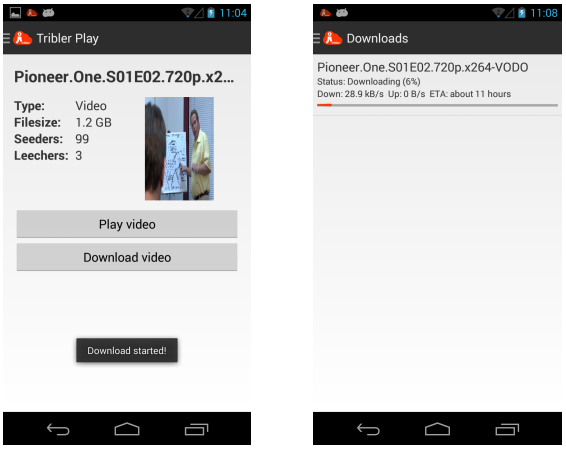
(d) The screen that is showed when clicking on a result.

Figure 8.1: Screenshots showing searching for a torrent.

8.2 Torrent downloading

Following the user's navigation explained in the previous section, the screen with more info about the selected torrent contains a 'Download video' button. When this button is pressed, another XML-RPC to the Tribler service is made. This call basically makes sure that Tribler starts downloading the torrent using the libtorrent library. The result of clicking this button is a small pop-up showing

the user that the download has started, as can be seen in Figure 8.2a. When the download has started, the user can open the navigation drawer and select the 'Downloads' option to monitor the progress of the started download, including the download status, the download speed and the estimated remaining time, as shown in Figure 8.2b. This information is continuously updated by performing XML-RPC calls to Tribler every two seconds.



(a) The screen with more info about a torrent when the download button is pressed.

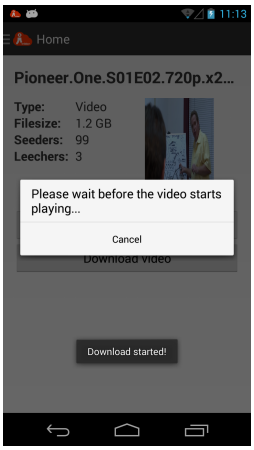
(b) Downloads screen showing the progress of the download.

Figure 8.2: Screenshots showing downloading a torrent.

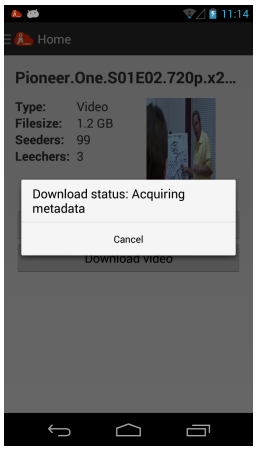
8.3 BitTorrent streaming

Continuing the user's navigation as explained in Section 8.1, the information screen also contains a 'Play video' button. Pressing this button first starts the download process of the torrent as discussed in Section 8.2 and then it opens a dialog that asks the user to wait, like in Figure 8.3a. When the download is added to Tribler, the message of dialog is replaced by the current download status as illustrated in Figure 8.3b. At the moment the torrent is actually being downloaded, the dialog message will show how long it will take until the video can and will be streamed, as in Figure 8.3c.

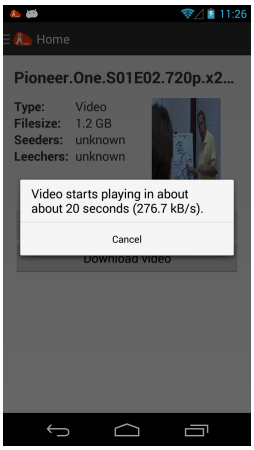
When the video is ready to be streamed, meaning that 5% of the video frames have been buffered, the integrated VLC player is automatically started as shown in Figure 8.3d. After a few seconds of loading, VLC will start playing the selected video, see Figure 8.3e. Once the video is playing, the user can pause or stop the video and also seek within the video. All of this while the torrent is not even entirely downloaded.



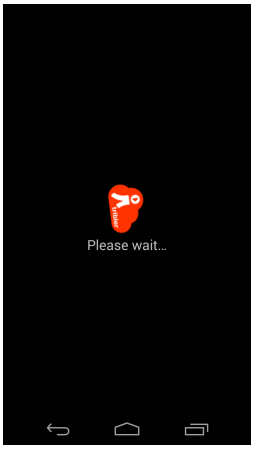
(a) The dialog shown when the play button is pressed.



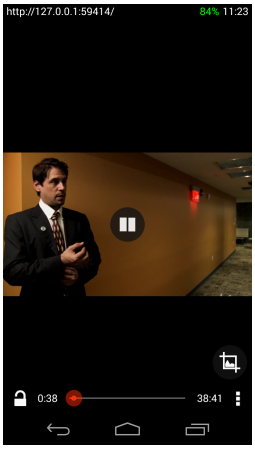
(b) The dialog showing the download status.



(c) The dialog showing how long it will take until the video can be streamed.



(d) The screen shown when VLC is started.



(e) The screen shown when VLC is streaming the video.

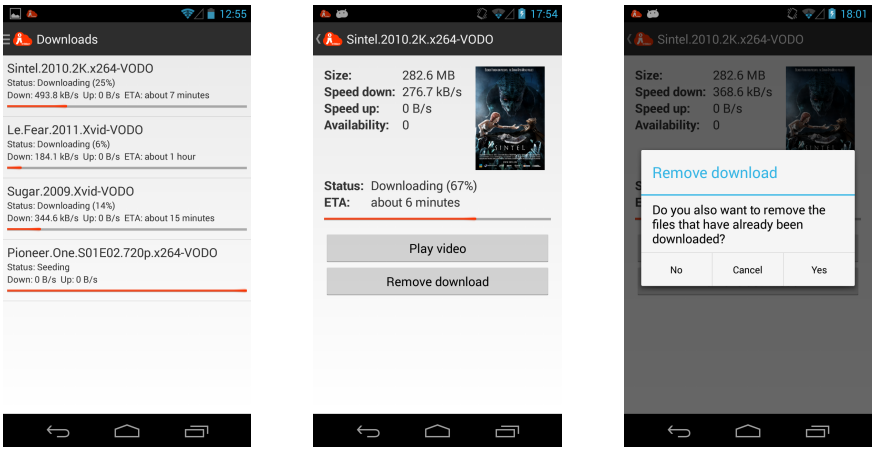
Figure 8.3: Screenshots showing streaming a torrent.

8.4 Viewing and managing downloads

When the user has started one or more downloads and wants to see the progress of these downloads, he can select the 'Downloads' option from the navigation drawer. This will open a screen containing a list of all started downloads including their status, download speed and estimated remaining time, as in Figure 8.4a. Clicking one of these downloads opens a screen with more info about the torrent, which also contains buttons for playing the video and removing the torrent, see Figure 8.4b.

The 'Play video' button in the download information screen behaves exactly the same as the one explained in Section 8.3. The 'Remove download' button

opens a dialog asking the user whether he or she wants to remove the download and the files associated with it, as in Figure 8.4c. Pressing 'No' in this dialog will only remove the download from the list of downloads. Pressing 'Yes' will also remove the already downloaded files from the phone.



(a) The screen showing the progress of all started downloads.

(b) The info screen shown when clicking a download from the list.

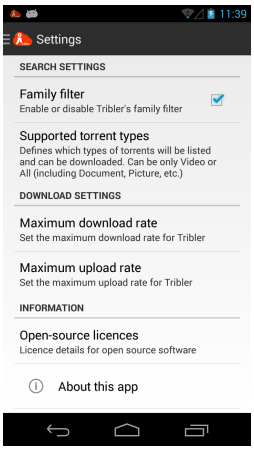
(c) The dialog shown when the remove button in the info screen is pressed .

Figure 8.4: Screenshots showing how to manage downloads.

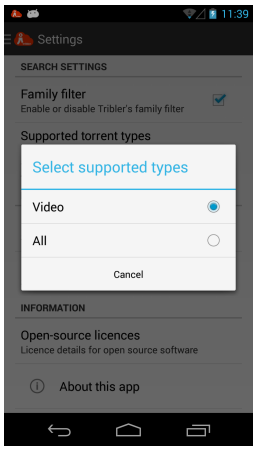
8.5 Viewing and modifying settings

Selecting the 'Settings' option in the navigation drawer opens a screen in which the user can view and edit the application's settings, as in Figure 8.5a. The 'Family filter' setting, which is enabled by default, is used to enable or disable Tribler's family filter. Clicking the 'Supported torrent types' setting opens a dialog in which the user can set the supported torrent types to either Video (default) or All, see Figure 8.5b. The maximum download and upload rate used by libtorrent can also be set within the settings screen. Selecting one of these two settings opens a dialog in which the user can enter a number that represents the rate in kilobytes per second, see Figure 8.5c. By default these rates are set to an unlimited speed.

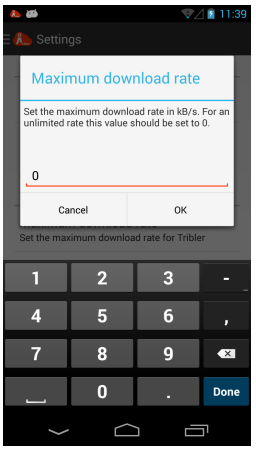
Finally, the list of settings also contains two items that are not really settings, namely the 'Open-source licences' and 'About this app' items. The first one opens a dialog that displays the licences of all open-source software that is used for creating the application (see Figure 8.5d) as it is legally required to include these licences. The second one opens a dialog showing some useful information about the application including which licence this application uses (GNU Public License version 3 (GPLv3)), as shown in Figure 8.5e.



(a) The screen showing the available settings.

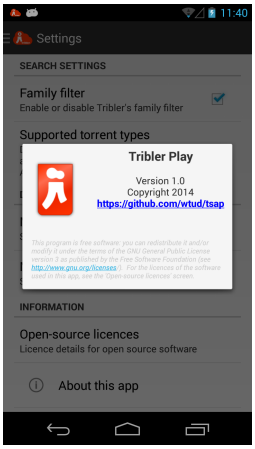


(b) The dialog shown when the supported torrent types setting is pressed.



(c) The dialog shown when the maximum download rate setting is pressed.

(d) The dialog shown when the open-source licences setting is pressed.



(e) The dialog shown when about this application setting is pressed.

Figure 8.5: Screenshots showing viewing and editing settings.

Chapter 9

Conclusion

When comparing the outcome of this project with the original problem description, it can be said that the problem has been solved and therefore the project can be marked as successful. This is the case because the Android application that has been developed provides the answer to the research question, as the application is able to search and stream videos in a decentralized way. In this way the application contributes to society as a whole by contributing to the fight against censorship.

Besides, the project's goals have all been realized as Tribler and its dependencies, as well as VLC for Android, have been integrated in the application. Any changes made to the Tribler code were contributed back upstream. Finally, the project can also be called successful because all *must have* requirements have been met as well as three out of four *should haves* and even two *could haves*.

During the development of the application, as much existing software as possible was reused to avoid re-inventing the wheel. This resulted into a design consisting of a Tribler service written in Python, an Android Graphical User Interface (GUI) written in Java and a modified version of the open source VLC video player which is also written in Java. The GUI communicates with Tribler through XML Remote Procedure Calls (XML-RPC) using the Hyper Text Transfer Protocol (HTTP) and the modified VLC player communicates with Tribler through HTTP streaming.

Combining Python and Java code was the main challenge of the project as Python is not natively supported by Android, so a third party framework was needed. The Kivy Python for Android framework was used for this purpose, because it is the only Python runtime for Android that is currently being maintained. It also has a convenient system for managing libraries written in C and C++. However, the framework was designed for applications that are written in Python only, using Kivy for the GUI. To allow for an Android GUI written in Java, the framework had to be modified. Also communication between Tribler and the GUI was a challenge, but that was fixed by using XML-RPC, which uses HTTP as transport protocol.

The main advantage of having a separate Tribler service is that the Tribler code that is run as a service in the background is the same as in the PC version, which makes maintaining and updating the code easy. An additional benefit of having all communication with Tribler over HTTP is that the Tribler service can be run on a different device. This was useful for testing and it could be the

starting point of the development of new applications, like applications for iOS and Windows Phone or running Tribler on a headless server.

However, sending all information over HTTP has its downsides too, as it introduces overhead that can be a problem while streaming videos. When the application was used to stream a 720p video on a Galaxy Nexus smartphone, which was first released on 19 October 2011, it showed artefacts. When the same video was displayed from its internal storage, without using HTTP streaming, the video would display without problems. Lower quality videos and the rest of the application work without problems. On a newer Nexus 5 smartphone, which was first released on 31 October 2013, HD videos could be streamed without problems.

Concluding, this project resulted into an Android application that solves the client's problem and that is ready to be published in the Google Play Store. The focus on reusing existing code resulted in an application that is maintainable, but might be less responsive on lower-end devices. Since smartphones are getting faster rapidly, this seems like a reasonable trade-off for maintainable code. Currently, the application provides decentralized content discovery and streaming in a single Android application Package (APK), which is a good step in the fight against censorship.

Chapter 10

Recommendations

During the weeks of the project, a lot of work was spend on researching, implementing features and making the application as stable and user friendly as possible. Looking back at the MoSCoW requirements set in Section 4.1, all must have requirements have been met, as well as the majority of the should have requirements. Given the limited time however, there is still room for improvement. Some of these recommendations for future work are described in the following sections.

10.1 Integrate anonymous tunnels

As described in Section 2.3, the Android Tor Tribler Tunneling (AT3) team was working in parallel to this project on getting an experimental feature of Tribler working on Android devices. As both teams did not have enough time at the end of the project to bring both projects together, this idea was dropped for the moment.

Integrating the two projects should not be a whole lot of work however, as all fixes by the AT3 team were contributed back into the main Tribler repository. Importing the `ProxyCommunity` when starting the Tribler sessions, plus exposing its functionality in the Android GUI are the main steps that should be taken to get this feature working.

10.2 Channels and vote support

During the last sprint of the development cycle, it was found out that the `AllChannelCommunity` required a lot of resources and would bring the whole application to a halt during downloads (see Section 7.1). If a way could be found to make this Dispersy community run reliably and without taking up too much resources, bringing back channel management would be a big feature.

In addition to channel management, the `AllChannelCommunity` also enables users to vote on the quality of torrent. Distributed quality control in the form of voting is an effective way to filter out broken torrents or spam, greatly improving the quality of the found search results.

10.3 Integrate with the main Tribler repository

Currently, the application is a standalone product, that uses the Tribler code as a package, and wraps a Python service around it, which interfaces with the Tribler core and exposes some of its functionality over XML-RPC to the Android GUI. The reasons to do this were that the functionality of the application is a subset of the full Tribler functionality and that this makes keeping up with the main Tribler code trivial, as this requires no additional modifications to the ever evolving Tribler code.

The downside to this approach is that there is some code duplication, in particular in the `Tribler.Main.vwxGUI.SearchGridManager` package and the various Manager classes in the Python service. The code in this `SearchGridManager` package was written with the Tribler `wx` GUI in mind, and thus relies heavily upon it and cannot be used on Android at all without modifications.

One approach would be to take this code and create a more generalized version of it, containing all functionality not specific to the `wx` GUI, such as the searching and downloading of torrents and the management of channels. The current `wx` GUI should be able to plug into this code, and use it for its main functionality. The current XML-RPC server, as used by the applications Python service, could then use this same generalized version and could be integrated into the main Tribler repository. This would make running Tribler on other platforms, with different kind of GUIs, or no GUI at all, a possibility.

10.4 Add support for additional Android devices

Currently, the application was only run and tested on Samsung Galaxy Nexus and Google Nexus 5 devices. A simple improvement would be to test it on various other Android devices, ranging from small screen phones to big screen Android TVs to increase the amount of potential users.

In addition to testing on these extra devices, support for tablet screens can be added. As tablet penetration is nowhere near as high as that of smartphones, no time was spend on developing a tablet interface. It is currently not known how the interface looks on a tablet device, but it is expected that the usage of screen estate is not optimal.

Bibliography

[1] B. Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.

[2] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[3] J. Pouwelse. The shadow internet: liberation from surveillance, censorship and servers. Internet-Draft draft-pouwelse-perpass-shadow-internet-00, IETF Secretariat, Feb 2014.

[4] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. Epema, M. Reinders, M. R. Van Steen, and H. J. Sips. Tribler: a social-based peer-to-peer system. *Concurrency and Computation: Practice and Experience*, 20(2):127–138, 2008.

[5] N. Zeilemaker, M. Capotă, A. Bakker, and J. Pouwelse. Tribler: P2p media search and sharing. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 739–742. ACM, 2011.

[6] N. Zeilemaker, B. Schoon, and J. Pouwelse. Dispersy bundle synchronization. *IFIP Networking 2013*, 4820:203–214, 2013.

Appendix A

Original project description

Included here is the project description, as specified on BEPSys[1].

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.

A.2 Auxiliary information

The smartphone application 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.

[1] `https://bepsys.herokuapp.com/projects/view/49`

The existing foundations you can use for you work can be found here: https://github.com/AlexKolpa/AndroidStealth/issues/ (this project will finish 15 April 2014)

Appendix B

SIG feedback

The feedback as provided by the SIG is included below.

> De code van het systeem scoort 4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Duplication.
>
> Voor Duplication wordt er gekeken naar het percentage van de code welke redundant is, oftewel de code die meerdere keren in het systeem voorkomt en in principe verwijderd zou kunnen worden. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om een laag percentage redundantie te hebben omdat aanpassingen aan deze stukken code doorgaans op meerdere plaatsen moet gebeuren. In jullie geval komt dit voornamelijk door XMLRPCChannelManager.java, waar de verschillende XMLRPCCallTask implementaties veel herhaling bevatten. Het is aan te raden om dit soort duplicaten op te sporen en te verwijderen.
>
> Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase.
>
> De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.

Appendix C

Plan of action

Plan of Action

Wendo Sabée, Dirk Schut, Niels Spruit

Contents

Preface

This document serves as the plan of action for the project that will be conducted for the course TI3800 and it will serve as a foundation for the remainder of the project. The problem that will be solved during this project, as well as the responsibilities of the client, the supervisor and the project team and the structure of the project will be established in this plan.

Summary

There is a pre-existing Android application that streams media files over BitTorrent. The aim of this project is to extend that application to use the search technology used by Tribler (Dispersy) to enable users to easily search for files, in a decentralized way.

The project will take eleven weeks, during which there are four phases: the Research phase, the Design phase, the Implementation phase and the Release phase. During these phases there will be weekly meetings between the group and the client to discuss the progress of the project. In the final week there will be a presentation and demo of the final product. The group works on a voluntary basis, but client is expected to provide the necessary Android hardware.

Quality control will be enforced with the use of Scrum, GitHub Issues and SIG. The first two will encourage the use of branches and maintaining a working master branch, while the latter will provide feedback during the last twee weeks of the Implementation phase on how to improve the overall code quality of the final product.

1 Introduction

The security and robustness of current media and information sharing applications are at best flawed. They usually incorporate a central component in their design and do not provide strong anonymity. This is bad when their users reside in a hostile environment. Services such as Twitter can be completely blocked by governments or forced to collaborate in censorship because of their central nature, as has been done in Turkey[1]. More decentralized technologies such as BitTorrent are fairly easy to disrupt by blocking their central trackers, as has been done with The Pirate Bay in for instance Australia[2].

Services that especially focus on providing strong anonymity to their users are not well maintained. More than a week after the discovery of one of the most severe security bugs in the history of the Internet, Heartbleed[3], more than 12% of the exit and guard nodes of the most popular privacy enhancing network, Tor, are still leaking private keys and information about their users[4], with thousands of relay nodes still vulnerable too.

1.1 Current state

To combat the shortcomings of these technologies, a fully decentralized application providing strong anonymity is needed. Tribler, a fully decentralized and BitTorrent compatible media sharing application developed by the Parallel and Distributed Systems (PDS) research group of the Delft University of Technology, is currently in the process of implementing Tor-like privacy enhancing technology[5].

[1] http://www.theguardian.com/world/2014/mar/21/turkey-blocks-twitter-prime-minister

[2] http://au.ibtimes.com/articles/551214/20140506/pirate-bay-blockade-australia.htm

[3] https://en.wikipedia.org/wiki/Heartbleed

[4] https://lists.torproject.org/pipermail/tor-relays/2014-April/004336.html

[5] http://github-pages.tribler.org/anonymity.html

As part of the effort to bring this technology to Android, an application was developed that enables users to stream media files on their phone or tablet using BitTorrent. After several meetings with Johan Pouwelse, the client, it was decided to extent this application with Dispersy, the technology behind Triblers decentralized search.

1.2 Accordance and adjustments

This report will be discussed with our client before starting the project, and amended where needed. If during the project it is found that the planning is insufficient, adjustments can be made in agreement with the client.

1.3 Document structure

In section 2, the assignment will be discussed. A global planning is provided in section 3. The project design is discussed in section 4 and finally, the plan to maintain high quality code is discussed in section 5.

2 Project assignment

In this section will be explained what the actual project consists of and what the deliverables, restrictions and conditions associated with this project are. First, the environment of the project will be examined, followed by the goals and the description of the project. Finally, the project's deliverables, demands, restrictions and conditions will be explained.

2.1 Project environment

The project is an initiative of the Parallel and Distributed Systems (PDS) research group of the Delft University of Technology, which focuses on the research areas of "P2P systems and online social networks, massively multiplayer online games, grids and clouds, and multicore architectures and parallel programming"[6]. The project can be viewed in particular as a part of the research area of P2P systems and online social networking, as the project is initiated by the Tribler team and Tribler is an example of a P2P system using online social networking. The long-term vision of the Tribler team is to create the 'Shadow internet', which is defined as "an infrastructure in which the ability of governments to conduct indiscriminate eavesdropping or censor media dissemination is reduced"[7]. To facilitate the evolution of this 'Shadow internet', Tribler has been created and it uses "the Dispersy elastic database for providing: keyword search, content discovery, content voting and spam prevention using crowd sourcing"[8].

2.2 Project goals

The Tribler team wants an Android application that is capable of streaming videos anonymously using the Dispersy elastic database to further create the

[6] http://www.pds.ewi.tudelft.nl/

[7] http://tools.ietf.org/html/draft-pouwelse-perpass-shadow-internet-00

[8] http://tools.ietf.org/html/draft-pouwelse-perpass-shadow-internet-00#ref-TRIBLER

'Shadow internet'. Therefore, the goal of this project is to develop an Android application that is able to stream videos from the Dispersy database using existing knowledge and projects.

2.3 Project description

There is a pre-existing Android application that can stream videos using BitTorrent[9]. This application uses traditional torrent files or magnet links[10]. The client wants us to integrate the distributed Dispersy database, which is also used by Tribler, for finding these torrents and the metadata of these torrents. This project will try to link the Dispersy database to the existing video streaming application to create the Android application that the client wants to have. The project will not focus on providing Tor-like anonymity, as this is done by another group (see section 4.1).

2.4 Deliverables

Eventually, the project will result in an Android application with the features described above. Apart from the final product, the following reports, excluding this Plan of Action, will be written during the project: a Research Report, a Design Document and a Final Report. The Research Report will explain the existing knowledge, technologies and projects in more detail as well as the problems and challenges of these projects. The Design Document will show the results of the requirements analysis as well as the proposed software architecture and a Test and Implementation Plan. The Final Report will contain everything that is necessary to get a potential new project team informed about the project and the choices that have been made during the project so that they are able to continue the project without much trouble.

2.5 Requirements

As described in the previous sections, the final product will be a mobile application specifically targeted to the Android platform. This application can be considered as a prototype and it will contain the features described in section 2.3. The exact requirements of this prototype will be determined during the design phase and will therefore be documented in the design document.

2.6 Conditions

The project team will deliver all deliverables described in section 2.4 during the fourth quarter (Q4) of the academic year 2013-2014. This quarter will last from Tuesday 22 April 2014 until Friday 4 July 2014 and this period will therefore be the exact timespan of the project.

The client, Johan Pouwelse, and the supervisor, Egbert Bouman will help the team in getting the necessary resources for the project, like some Android devices to test the software.

During the weekly meetings they will also discuss the progress of the project and they will provide feedback on the work that has been done.

[9] `https://github.com/javto/Tribler-streaming`
[10] `https://en.wikipedia.org/wiki/Magnet_links`

3 Approach and planning

The project is divided into four phases: the Orientation Phase, the Design Phase, the Implementation Phase and the Release Phase. These phases will be explained in more detail in the sections of this chapter. The overall global planning of the project has been put in a Gantt chart[11] and can be found as an attachment in Section 6. In this section a timeline (Figure 2) of the project can be found as well to give a better overview on the project and its phases.

3.1 Orientation Phase

The first phase lasts two weeks and is divided into two parts. In the first part of the first week, this document, the Plan of Action, is written. In this document the project and our proposed approach to it are explained.

The second part of this phase is for research. The currently existing projects and literature will be examined and decisions about which technologies to use will be made. The results of this research will be included in a Research Report.

3.2 Design Phase

The second phase lasts a week and will be divided into four parts. During the first part of this phase a requirements analysis will be made, according to the MoSCoW method[12]. The GitHub Issues system will be used to keep track of the requirements during the Implementation phase.

The second part of phase two will be spend on defining the software architecture. In this phase all components that will be used in our project will be defined, as well as how they will be used.

During the third part of this phase, a Test and Implementation plan will be written and in the last part a GUI mock-up will be created. The result of the four parts of this phase will be included in a Design Report.

3.3 Implementation Phase

In this phase, consisting of six weeks, the actual coding and implementation will be done. The project team will use Scrum[13] to gradually expand the code base, while still maintaining a working master branch, with sprints consisting of two weeks.

At the beginning of each sprint there will be a meeting to make plans and divide tasks. At the end of each sprint there will be a meeting with the client. At the beginning of each day there will also be a short meeting where the group members discuss what they have done and decide what they are doing that day.

At the end of the fourth week of this phase, the code will be submitted to SIG (See section 5). Depending on when they provide us with the feedback, parts of the fifth or sixth week will be spent improving the quality of the code according to their recommendations.

[11] `http://www.gantt.com/`
[12] `https://en.wikipedia.org/wiki/MoSCoW_Method`
[13] `https://en.wikipedia.org/wiki/Scrum_(software_development)`

3.4 Release Phase

This final phase consists of two weeks, which will be spent on presenting and distributing the application. All the implementation work should already be finished in the previous phase. A Final Report will be written, which describes the process and how the application works. At the end of this phase, a presentation will be given to the client, supervisor, course coordinators and other interested parties. The application itself will be made into an APK file, which can be released in the Play Store.

4 Project design

In this part of the plan of action the organization of the project is shown in more detail, including the personnel, administrative procedures, resources and reporting. These plans have been made in agreement with the client and the supervisor.

4.1 Organization

Members of the group do not have any specific roles throughout the project. The client is Johan Pouwelse, co-founder of Tribler. During the timespan of the project, two other groups are working on other Android applications in parallel to this project in order to make additional steps towards the 'Shadow internet'. These groups, another Bachelor End Project (BEP) group and a group of MSc students, are responsible for adding Tor-like anonymity to the video streaming application and making a stealth application[14].

The end goal is to merge the work done by all groups into one application to create the application the client desires. It will depend on the progress of all groups whether this will happen during or after the project.

4.2 Personnel

The members of the group are working on this project full time. All members have prior experience with Linux and Java, but no real experience with Android development, Tribler or video streaming applications. The contact information of the group members is shown below.

Wendo Sabée
✉ w.f.sabee@student.tudelft.nl

Dirk Schut
✉ d.e.schut@student.tudelft.nl

Niels Spruit
✉ n.spruit@student.tudelft.nl

[14] `https://github.com/AlexKolpa/AndroidStealth`

4.3 Administrative procedures

To monitor the project, GitHub[15] will be used extensively. If a group member encounters a serious issue or if he wants to see a new feature, an issue can be created on GitHub. The group members will try to commit often and with clear accompanying messages. Apart from the GitHub repository containing the code of the project, another repository will be used for the reports.

4.4 Resources and Finances

Currently there are no costs related to this project. The group members work on a voluntary basis on their own hardware, with Android phones provided by the client. The client provided the group with a room to work in, room EWI-HB 07.250, which is part of the Parallel and Distributed Systems Group at the faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS).

4.5 Reporting

The group will have weekly meetings with the client and the supervisor in which the current progress of the project and the future plans will be discussed. The team will also deliver the required reports and other deliverables as described in section 2.4.

5 Quality control

There are several ways in which quality is monitored throughout the project. As the project is open source and openly available on GitHub, both the team members and the project supervisor have constant access to the most recent code. After the initial start of the project, non-trivial changes to the codebase will be done via GitHub issues. In this way, the use of branches is enforced and the process of each change can be reviewed by the other team members before merging it into the production code.

A more formal way of quality control will be provided by SIG[16]. They will judge the code twice on several factors during the project. Their first code review will be done at last on 13 June 2014. Feedback on this review will be provided before 20 June 2014, and will be used to improve the quality of the code where possible. Their second review will be done over the final product and will count towards the final grade.

[15] `https://github.com/`
[16] `http://www.sig.eu`

6 Attachments

Id	Taaknaam	Duur	Begindatum
1	**Plan of Action**	1 wk	maa 21-4-14
2	**Orientation phase**	2 wkn	maa 21-4-14
3	**Design phase**	**5 dagen**	**maa 5-5-14**
4	Requirements analysis	2 dagen	maa 5-5-14
5	Software architecture	2 dagen	woe 7-5-14
6	Test - and implementation plan	2 dagen	vri 9-5-14
7	**Implementation phase**	**30 dagen**	**maa 12-5-14**
8	Implement features	6,2 wkn	maa 12-5-14
9	First SIG evaluation	0 dagen	vri 6-6-14
10	Process feedback SIG	3 dagen	maa 16-6-14
11	**Release phase**	**10 dagen**	**maa 23-6-14**
12	Write Final Report	1,2 wkn	maa 23-6-14
13	Second SIG evaluation	0 dagen	vri 27-6-14
14	Prepare presentation	1,2 wkn	maa 30-6-14
15	Final presentation	0 dagen	maa 7-7-14
16	Deadline Final Report	0 dagen	din 1-7-14

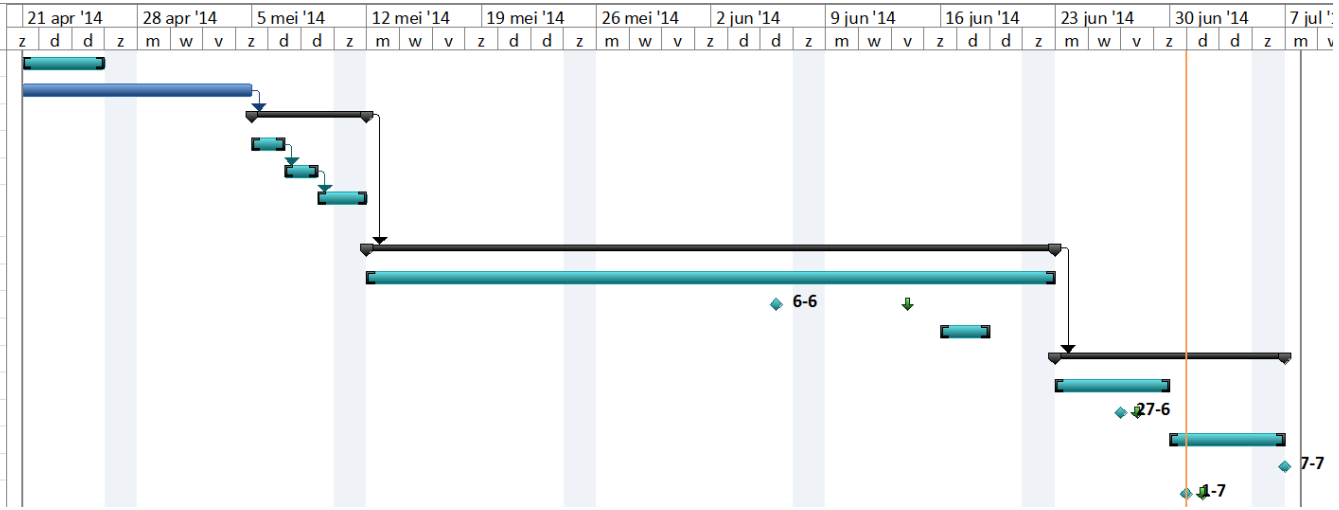


Figure 1: The projected planning of the project.

Figure 2: The timeline corresponding to the planning in figure 1.