TI3806 Bachelorproject

Anonymous HD Video Streaming for Android using Tribler

Final Report

Authors:
C. van Bruggen,
N. Feddes,
M. Vermeer

Supervisor:
Dr. Ir. Johan Pouwelse
Project coach:
Ir. Egbert Bouman

June 19, 2015

Abstract

Privacy is dying. Internet users have been trying more and more to keep governments' and security agencies' prying eyes away from their online activity. Anonymity - Internet anonymity, specifically - has therefore gained traction in recent years. In order to further contribute to the effort of Internet privacy and anonymity, this project's goal was to create an Android mobile application that allows users to anonymously stream videos from a decentralized network of peers. The main piece of software used to achieve this goal was Tribler, a Peer-to-Peer (P2P) file sharing program that provides users the ability to download their files anonymously. Since Tribler is not written for mobile devices, it, along with all of its dependencies, needed to be cross-compiled for the ARM CPU architecture. All of Tribler's dependencies were successfully ported, except for a cryptography library due to its lack of stability and maturity. As a result of this, an application was created that successfully streams video from a decentralized network of peers, but without the ability to do so anonymously. Even though anonymous streaming is not yet working, that functionality can be easily implemented in the application when the necessary cryptography library becomes mature and stable enough for cross-compilation.

Foreword

This document wraps up the bachelor project that we had the pleasure of working on this quarter, which was commissioned by the Tribler Team at the Parallel and Distributed Systems (PDS) group working on the seventh floor of the Electrical Engineering, Mathematics and Computer Science (EEMCS) faculty of the TU Delft.

Long before we started our project, we already had the knowledge that Tribler was being developed at TU Delft, and we have long been fascinated by it. When we saw that there was a bachelor project that would allow us to work with and contribute to the Tribler project, we, of course, jumped at the opportunity to do so. It was a great experience, and we want to thank everybody involved in making it possible. Notably, we would like to thank the following people for their help and participation throughout the duration of this project:

Johan Pouwelse, for his enthusiasm and exceptional guidance;

The Tribler team, for writing the Tribler code we used for our project;

Egbert Bouman, for helping us find our way through Tribler's codebase;

Raynor Vliegendhart, for filling in for Egbert when he was ill; and

Jaap van Touw, for helping us get our continuous integration working.

June 2015, Delft

C. van Bruggen,
N. Feddes,
M. Vermeer

Contents

Chapter 1

Introduction

We live in a strange world. Journalists are kidnapped and killed in countries like Syria to prevent the world from knowing what is going on. In Russia every blogger with more than 3,000 daily readers has to register at the government [1]. It is not to much surprise then that when Russians were surveyed, 80% of them believed that the MH17 flight was shot down by the Ukranian military [2]. Other countries such as China go to further extends and block the websites that many of us regularly use, like Google, Youtube and Twitter [3].

Many regimes want to control what their people see and what they can't see. It may perhaps be a positive thing when this control is used to block certain content that infringes on a persons rights, but as soon as it is used to prevent or limit journalists and ordinary people from spreading important messages, possibly against their government, then this can become an issue. The limitations in an online context can be on two sides: both the uploader of a message and the downloader can be traced and blocked, or have possibly worse things done to them. These messages regularly come in the form of video, especially since a video gives an element of trustworthiness when covering an important event.

With our project we aim to help the downloader of such a video to be anonymous, and because of the functionality we brought to Android smartphones it becomes easier for other teams to work on anonymous uploading. This anonymity is important as it mitigates the Pervasive Monitoring of the content a user downloads or uploads, as per RFC7258 [4]. We did all this by porting the Tribler functionality to Android. Tribler is being worked on by the Tribler Team at the TU Delft. It is a (big) piece of software that enables people to communicate with each other anonymously using a decentralized P2P network. Tribler was not meant to run on Android and that is why it was needed to port it. The advantage of having Tribler work on a mobile device is that it makes it easier to capture a video and spread it, and for others to watch the video from remote locations. Just last month Google stated that for the first time the searches made via mobile devices outnumbered the searches made via desktop computers [5]. And did you know that the country with the most mobile phones in use is China, where Internet censorship is quite aggressive [6]. All of these are good reasons to make anonymous video streaming available on the most used mobile operating system, Android.

In previous bachelor projects, students have already worked towards the goal of a P2P Video-on-Demand (VOD) censorship-resistant mobile application. Most of these were still prototype versions, from which we will have to extract the necessary functionality.

The process of how we achieved our goal is described in the following chapters. Chapter 2 starts with describing the problem, exploring the possible solutions and finally considering the best way to reach our goal. The next chapters, starting with chapter 3, are the result of the sprints we had according to the Scrum methodology. In the first sprint we worked on the essential software pieces of Tribler. During the second sprint, described in chapter 4, we significantly changed our approach to the project and continued to work on porting the updated Tribler codebase to Android. In the final sprint, chapter 5, we focused on integrating our code with the Shadow Internet Project, getting the previously created Tribler Streaming Android Project (TSAP) working with the new software and sending our contributions to the different open source projects that we used. The 6th and final chapter concludes the thesis by summarizing the work we have done, looking back at the goals that were set and giving recommendations to others that want to improve the work we have also contributed to.

Chapter 2

Problem description

The main goal of this project is to develop an anonymous Android P2P VOD streaming application that is ready to be released on Google Play. This application should allow a user to search and watch videos anonymously via Tribler. The full project description as provided by the client can be found in Appendix B.

Such an application is useful because of the anonymity it offers. In certain countries where there is Internet censorship, it can be used to send out important messages to the world. And the decentralized nature of the entire system makes it extremely hard to be taken down by governments for instance. The P2P aspect ensures that the user base can scale without generating overhead; it even makes the system faster when more users use it. The traditional way of providing video-on-demand was to store videos on one or more servers and let the clients, users, connect to view the videos. When the videos are stored among the peers in a P2P network, the central servers are not needed anymore, which is significantly more cost-effective.

2.1 Solution requirements

As part of the original project description, which can be found in the appendix on page 40, the requirements can be described in one sentence:

"[...] a prototype of an anonymous P2P-based video-on-demand mobile application will be realized, that will allow users to search for videos on the Internet."

Several important aspects in this sentence can be identified and will be discussed. The mobile aspect refers to the fact that a large number of people have mobile phones. There have been several anonymous, P2P solutions for desktops, however mobile platforms have yet to see such an application. In addition, smartphones with cameras can enable new ways this technology could be used.

When it comes to VOD capabilities on phones, there are numerous services and programs available [7][8][9][10]. This functionality allows users to directly watch a video without waiting for it to be completely downloaded.

Most existing VOD solutions [7][8][9] however, have a centralized approach allowing a single entity to control the information that can be viewed. In addition, this single entity needs to be able to concurrently serve all it's users the videos they want to see, which may lead to bandwidth scaling problems. Having centralized servers also allows adversaries that want to disrupt the service to do so by attacking the availability of the system. Using a distributed system this would be significantly harder, as there is no single entity that is relied upon.

As for search capabilities, this is also widely available in existing VOD solutions [7][8][9][10]. One should note however that the scope of the search and content availability differs quite a lot among these. In some cases the service does not allow users to upload their own content, see the next sections, greatly reducing the available content.

Anonymity is another important aspect of the application. Threats to this internet anonymity include ones such as tracing internet traffic back to the user and discovering their real identity, and having a middleman observer see all of a user's sent and received data packets. As the goal is to prevent others to find out a user's personal details and what exactly a user is downloading, the application should provide a level of anonymity that is able to do this.

2.2 Related work

Work on mobile P2P VOD applications is not new and several other related projects have worked towards this goal as well. We give a brief discussion on these systems and shortcomings and/or strengths towards our specific goals.

2.2.1 RapidStream

RapidStream [11] is a mobile P2P streaming application for Android; the first of its kind, as the authors claim. It uses a custom-build system designed for mobile use. Since it uses a 'hybrid P2P overlay maintenance design', it relies on central servers which facilitate and control the network. Using this prototype application, which was able to download at a little over 100 KByte/s, the authors have shown the feasibility of P2P VOD application on mobile phones. However, since this prototype network is separate from other networks, thus having limited available content, and still quite centralized, we consider the work relevant but not in a way that we are able to use parts of their application.

2.2.2 MicroCast

The MicroCast application [12] is a high performance cooperative download solution. In cases where multiple people want to watch the same video, it helps to increase the download speed by locally sharing the stream. Using their own all-to-all local dissemination scheme MicroNC-P2, they were able to get results outperforming BitTorrent on the total amount of local traffic sent. However due to the fact that the MicroNC-P2 algorithm uses Wifi overhearing to achieve these results and phones thus need to be near each other, it's application is rather limited. Perhaps it is possible to use the algorithm when this scenario occurs, in addition to other functions that are able to stream videos when users are not near each other.

2.2.3 3G-MOVi

The 3G-MOVi [13] application works in a similar way to MicroCast and allows users to cooperatively download content from central server. In contrast, the design has also integrated a trust scheme. Next to the consideration of the applicability of MicroCast, 3G-MOVi has another downside in that it requires modifications of lower operating system layers, to which 'normal' application have no rights. Since it then would not be possible to distribute our application to the general public, we cannot consider 3G-MOVi to play a role in achieving our goals.

2.2.4 Oghma

In Oghma, an Android application for accessing videos, a P2P content delivery network is used to deliver a streaming video to a user [14]. A central trust server functions as a tracker that keeps track of peers and their trustworthiness. Users authenticate to the trust server using OpenID and are then able to connect to other peers. All data in transit is encrypted to ensure that only the authenticated peers are able to download the video they requested.

Since it makes heavily use of a central server to which users authenticate and download from, its properties are quite different from what we need to reach our goals.

2.2.5 Saracen

Saracen is a framework for distributing real time user generated videos over P2P networks [15]. There is however no real world implementation of its system. All uploads of user generated videos are to be sent to a super node in the network, which thus serves as a dedicated server, and from there the video can be streamed to a P2P network.

Even though it has some interesting features, such as video distribution based on geolocation metrics, we decided not to take a similar approach to this, mainly because of the dedicated server which makes it easy to shut down by governments and the lack of implemented work.

2.2.6 StreamSmart

In StreamSmart [16] every user has their own smartphone software clone in the cloud and all communication happens in this P2P cloud environment. The application is built upon the Clone2Clone network [17]. All content has to be available in this Clone2Clone network before it can be accessed. Even though it's a decentralized system, it is not anonymous. And similar to I2P all content must be available and shared within the network, with no easy way to search content stored in other systems.

It should be noted however that the authors of Clone2Clone have interesting properties in their system. With the introduction of communication and computation offloading to private/public clouds, they were able to gain significant improvements in battery life. This gives a much better user experience compared to the alternative, doing all the work on the phone, at least when this has a significant impact on the user experience.

2.2.7 Bittorrent

BitTorrent clients are pieces of software that implement the BitTorrent protocol. This protocol is meant for P2P file sharing. A new file can be uploaded by creating a torrent file that describes the uploaded file and then making the file available through a node in the network that then functions as a seeder. The torrent description file can then be used to find the seeder. In this way other peers can download the original file. As the users download the file they become a source for a piece of the file to other peers. This makes sure that a file is propagated throughout the network and multiple sources are available. The new seeders with parts of the original file can then act as a swarm from which the individual pieces from the file are sent at once to a peer that is downloading, which improves the download speed. BitTorrent clients do not necessarily offer anonymity.

The BitTorrent protocol and many of its implementations, by default do not give the user any anonymity, as this requires extra software. Several implementations are available as mobile applications and allow the streaming of videos [18][19][20][21], also see Popcorn Time. None of these support a decentralized search however.

2.2.8 Tor

Tor's goal is to provide their users online anonymity and privacy [22]. It does this by letting traffic of users who use their software go through relays. Relays are routers in the Tor network. There exist middle relays, bridges and exit relays. Bridges are just secret relays that are part of the Tor network but not made know. This is useful to prevent governments from blocking publicly known relays. By letting users' traffic go through at least two middle relays and then an exit relay, it seems as if the source of the traffic is the exit relay, thus providing anonymity to the users. The relays themselves are offered voluntarily and for free by people throughout the world who agree with Tor's vision of anonymity. The word Tor is an acronym for the original project name 'The Onion Router'. This is because data sent over the Tor network is encrypted and each relay it goes through decrypts a layer of this encryption, much like peeling an onion.

The Tor project has stated in the past, and this still holds true today, that using BitTorrent over the Tor network is not a good idea [23]. In part because it might break anonymity, but also because the network cannot handle the traffic load. Tor is available as an Android application under the name of Orbot.

2.2.9 I2P

I2P, the Invisible Internet Project, is a P2P anonymity network. It achieves this by using four layers of encryption when sending a message. Users can set the number of hops to use when sending data over the network. A maximum of seven hops can be used, but the default is set to two. Therefore they use a Tor-like network: garlic routing. One difference, however, is that I2P uses a distributed network database and it supports distributed peer selection. The latter allows the protocol to find reliable routers over which a message will travel. One of the other advantages of I2P is that it's already available on Android. It does however lack essential features here, such as video streaming. Besides that, the anonymity for their Android application is not fully done yet. I2P was not designed for anonymous World Wide Web browsing, instead it uses eepsites which are websites hosted anonymously inside I2P's network. Other websites or other internet services cannot be connected to unless a specific outproxy has been set up for this. This is also one of the reasons that their network is called a network within a network. Outproxies serve a similar purpose as Tor's exit nodes.

2.2.10 Popcorn Time

One popular video streaming service is called Popcorn Time. It differs from other streaming services in that it is P2P and uses a BitTorrent client with an integrated video player; VLC media player (VLC). The BitTorrent client uses the BitTorrent tracker called YIFY, but it can also use different trackers. These trackers contain torrents for pirated movies, and because Popcorn Time is encouraging users to download pirated content (and distribute since a downloaded movie will be seeded back into the swarm), they are conducting illegal business. Furthermore, Popcorn Time lacks the anonymity feature that we require. Users can use a VPN or a proxy but this is not anonymous enough. Furthermore their system is not decentralized because

they make use of centralized trackers. It did however become open source after the original makers took it down because of pressure from the Motion Picture Association of America. It is also available for Android.

2.2.11 Freenet

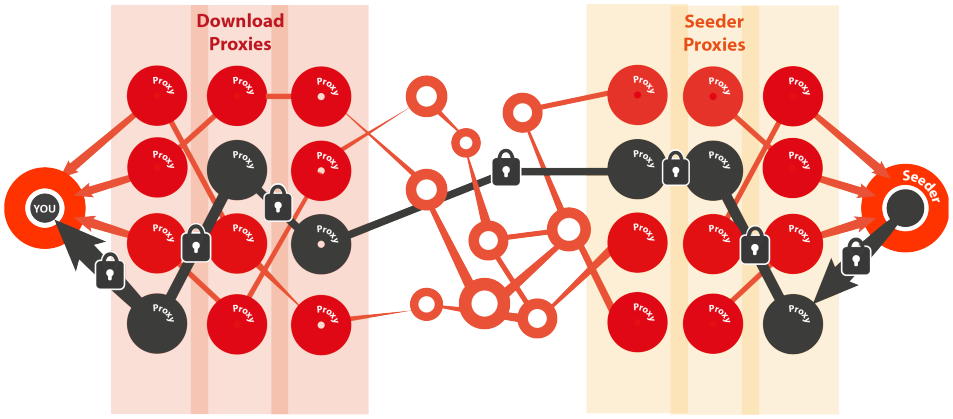
Freenet is a P2P platform for publishing and reading data anonymously [24]. By using Freenet one can connect with websites and download files hosted in Freenet, content outside the Freenet network is not accessible. It is however fully decentralized because of its distributed datastore. Freenet is also searchable, which means it offers functionality to search through the content in their network. This is quite essential for our project, since it allows users to easily find a video to watch anonymously. Freenet does not work on Android though. It also is not that fast unless the program is ran for some time, but then it still has performance issues which would just be too much for VOD on a mobile device.

2.2.12 Tribler

Tribler is a BitTorrent based client, but it also uses an enhanced version of the Tor protocol. This makes it different from other BitTorrent clients since it offers anonymity when downloading through this Tor-like protocol. It does not protect the anonymity of the seeders of as yet but this is something that is being worked on. The enhanced Tor protocol makes use of UDP instead of TCP (which the original Tor protocol uses) and every anonymous downloader becomes a relay (instead of the voluntarily relays which Tor uses). Below is a picture that represents how data travels through the network when seeders would also have anonymity. As can be seen in figure 2.1, Tribler uses three layers of proxies for the recipients and sender in the future.

Another difference between Tribler and other BitTorrent clients is that Tribler is

Figure 2.1: A graphical representation of the tunnel functionality



fully decentralized: it does not use torrent sites (which act as BitTorrent trackers) to find and download content. Instead it uses their own P2P database system called Dispersy, which will be discussed in the next section. Because of this Tribler will still work when all central BitTorrent trackers go down, which inherently means that taking down Tribler is as hard as taking down the Internet.

Currently, Tribler has a fully functioning decentralized search through Dispersy and provides VOD using libtorrent's piece picking. Anonymity can be provided by

the tunnels created within the community. The latest version of Tribler lacks mobile support. More on this can be found in the prior work section on page 11.

Dispersy

Dispersy is the elastic database system that Tribler uses. It has peer discovery and data synchronization mechanisms. The different distributed databases that Dispersy uses are called communities and are implemented in Tribler inside the Tribler.community package. Examples of these communities are the tunnel community (used for discovering proxies; these were previously called anontunnels), the search community (allows peers to discover .torrent files) and the channel community. Tribler uses channels to let users manage a bunch of torrent files. Other users can then mark the channel as favorite to be kept up to date with all new content of this channel. When a lot of users favorite a channel it can become more trustworthy to other users. This feature, in combination with Tribler's current video streaming functionality, allows for a Youtube-like experience but in a P2P fashion, without using a central server.

2.3 Prior work

Bachelor students in previous years have already contributed towards achieving similar goals as stated in the solution requirements.

2.3.1 Tribler Streaming

Previous and early work on the topic of decentralized streaming video mobile applications was done by Jaap van Touw in his bachelor thesis *"A non-centralized approach to Video on Demand on mobile devices"* [25]. In the report several possible solutions for mobile devices were discussed and the research can be used to give more direction to software architectural decisions that need to be made. One of the solution spaces that was researched is the platform an application should be built upon. The platforms compared are Windows Phone [26], Apple iOS [27] and Google Android. Given the market share and personal preference, Android was chosen as the best platform to develop for. We note the absence of, albeit relatively new, cross-platform solutions, such as Kivy and Phonegap [28], and considerations such as the capability to access networking functions or the review guidelines of the primary distribution method for the platform.

Another consideration made was the choice of VOD service. Although in two years time numerous new services have been launched or are about to be launched [29][9], the main argument for a non-centralized approach remains and excludes many of the new services.

When one wants to have a VOD application, the video player is one of the main components. Given a extensive comparison between supported codecs, hardware decoding support and other features, VLC was chosen as the best video player to be used.

2.3.2 Android Tor Tribler Tunneling

Other previous work includes the Android Tor Tribler Tunneling (AT3) application made by Rolf Jagerman, Laurens Versluis and Martijn de Vos [30]. Much of their

project was focused on researching how to get the required Tribler functionality working for Android. They came to the conclusion to use Python for Android (P4A) and worked a lot on writing required recipes. To achieve anonymous downloads in their application they used Tribler's anontunnel functionality, which now have been renamed to tunnels. Their final product, an Android application, had a Graphical User Interface (GUI) made in Python using Kivy. It can be used to download a specific torrent file for testing purposes. While this download is going on, CPU speed and download speed is displayed. This project is therefore not ready to be used by everyday users yet, but it contains a lot of the functionality that is required to get anonymous VOD streaming working on Android devices.

2.3.3 Tribler Streaming Android Project

The TSAP team is composed of Wendo Sabe, Dirk Schut and Niels Spruit. They worked closely together with the AT3 team in getting the Tribler code working on Android [31]. Other than that they built a user-friendly GUIin Java. This GUI is a lot more advanced than the one built by the AT3 team but it is not written in Java and the communication between Java and Python might make the application a bit slower.

2.4 Solution considerations

Out of I2P, Popcorn Time, Freenet and Tribler, we decided to use Tribler to work towards anonymous streaming on Android. Although I2P works on Android, it has no support for streaming video (Tribler has a built-in video server for streaming) and their anonymity is not secure yet on Android. On top of that I2P has no migration path, which means that they cannot use the already existing BitTorrent swarm, and therefore a lot of video content that is available on BitTorrent clients, will not be available in I2P. It is also still in beta mode. Popcorn Time has some legal aspects that we would not want to associate the TU Delft with. It does however use an internal VLC player and libtorrent, which might be interesting to use. However, two projects at the TU Delft, namely TSAP and Libtorrent Streaming Android Project (Tribler-streaming), have already finished their implementation of an internal VLC player with Java bindings.

Freenet was not picked because of the lack of video content, not being available on Android and its performance issues. Tribler does have a version which is available on Android, but it's currently one year old and needs to be updated to work with among others the new dependencies of Tribler. It also is fully decentralized which makes it harder to be shut down in certain regimes. Furthermore it was also the client's wish that we would use Tribler, and another advantage is that Tribler is being actively developed in the same university, same building and even on the same floor as the office that we're currently utilizing. This means that when needed we could ask the Tribler development team for help or advice.

2.5 Challenges

2.5.1 Porting new Tribler functionality

Tribler is written in Python and therefore its codebase and libraries had to be ported before they could work on Android devices. To achieve this, a project called P4A was

used by the AT3 and TSAP teams which can create a Python distribution from all the required Python modules. This distribution can then be packaged into an Android Application Package (APK). P4A makes use of so-called recipes. These recipes are used to compile Python modules that require C extensions. P4A comes with premade recipes for certain libraries and the AT3 team created a lot of the recipes which were still missing but required for Tribler. However, Tribler and its dependencies have changed over the last year. This means that part of our project will be to update these recipes and even add new ones for the new dependencies, such as libsodium (for which the libnacl project exists as Python wrapper).

2.5.2 Anonymity through TOR-like tunnels

Besides this, the AT3 team used Tribler's tunnel functionality to handle anonymous downloads in their application. The newest version of Python has changed this functionality quite radically, which means that AT3's application does not work with the newest version of Tribler. This integration and updating of the code will be another part of the problem that has to be solved by us.

2.5.3 Polishing the Graphical User Interface

Both the AT3 and TSAP projects have their own GUI. AT3 created their GUI in Python by making use of Kivy. Kivy also happens to be the creator of P4A. The GUI of TSAP is made in Java and makes use of a specific Remote Procedure Call (RPC) protocol, namely XML-RPC, which uses XML for encoding function calls and HTTP for sending these calls and receiving their results. Tribler is then run as a web service and the Java code functions as the client. On the other hand, the AT3 wrote their interface in Python by using Kivy's design features. This allows for easier interaction with the Tribler functionality. Deciding which of these two GUI's (or another new approach) will be used for the final version is a part of the main problem. One of the main considerations is that the XML-RPC protocol used by TSAP might give a lot of overhead, but, on the other hand, TSAP's GUI is the most complete and useful contribution towards the final goal. We consider the approach taken by the TSAP team the right choice, as this gives the user the best looking GUI available. There is however room for improvement on the actual implementation. It might be useful to combine this with work on an actual Application Programming Interface (API) for Tribler as described in Tribler issue #1107.

2.5.4 Automating Cross Compilation

When trying to get the ported Tribler of last year's projects AT3 and TSAP to work, we noticed that the different bash files provided to first make a distribution of the libraries and then package them into one APK file using P4A did not work anymore. We made numerous attempts to get them to work and we finally did, but it took us a lot of time.

To make it easier on future groups that work on this project and for ourselves to maintain the project we decided to switch from the bash files to Kivy's build automation tool, called Buildozer. This tool also installs any requirements for P4A, such as the Android Software Development Kit (SDK) and Native Development Kit (NDK), and makes use of the Android Debug Bridge (ADB) to deploy the built application to an Android device and then run it. Furthermore it makes use of a buildozer.spec file to store all the different configurations that can be used when deploying an application to a device, such as the Android API level to use, the permissions the application

requires and any app icons to show while it is starting up. Buildozer also has support for Apple's mobile operating system iOS.

We decided to use Buildozer for several reasons. As we wanted an easy system that could automatically generate the P4A distribution and APK we considered buildozer against the custom systems made by TSAP and AT3, and writing our own, possibly with a Makefile. The custom systems have the problem that they were tailored to the specific setup of the other projects. This would mean that we always have to reconfigure the not well documented system as we work on the project. Writing our own system has the drawback that others using our code need to maintain the system and in general we have to put some time in to testing and documenting how it works. As Buildozer has out-of-the-box support for everything we needed as well as an easy to use configuration file where we could specify how to build our application, it was chosen as the best system compared to the alternatives.

2.5.5 Common Tribler Core Wrapper with Shadow Internet

Concurrent with our project, the Shadow Internet Team is working on another application that enables a user to record a video and immediately share it over a local ad-hoc connection or anonymously online using Tribler. As the Tribler part overlaps with our application, the client wants us to write some common code that allows for this functionality to work in both applications. Since this code also wraps Tribler and any Android specific initialization code, we have named this the Common Tribler Core Wrapper. The wrapper should provide an easy interface to start Tribler, download a new torrent, allow a keyword search in Dispersy and start playing a video while it is still downloading (VOD).

2.5.6 Tribler software stack complexity

Over the 10 years of Tribler's existence, it has accumulated a large number of features. Since these features are often dependent on other software libraries to provide basic functionality, this complicates the process of porting the software to other platforms. Given the fact that it has been almost a year since anyone touched the previous project code, several dependencies have been outdated. This might not be problematic if newer software versions only contain added features, but when there are security issues fixed, it is highly recommended to update these dependencies. In addition, in order to allow developers to release new versions of the application, they need to be able to re-create the dependent libraries in an easy manner. Since the build system downloads the newest version of pure-python modules using pip, we are only interested in the compiled python modules.

Name	P4A Version	Latest Version	Used by	Comments
netifaces	0.10.3	0.10.4		
libtorrent	0.16.16	1.0.5	Download Engine	
apsw	3.8.4.1-r1	3.8.10.1-r1	Persistent Storage	Sqlite bindings
M2Crypto	0.21.1	0.22.3	Tribler, Dispersy	
OpenSSL	1.0.1g	1.0.2b	M2Crypto, cryptography	Several CVE's
zope.interface	3.8.0	4.1.2	twisted	
twisted	14.0.0	15.1.0		
libsodium	N/A	15.1.0	libnacl	
libnacl	14.0.0	15.1.0	Dispersy	
LevelDB	N/A	15.1.0	TorrentStore	Plyvel as binding
cryptography	N/A	0.9.0		No stable release yet
CFFI	N/A	1.1.1	1.1+ by cryptography	1.0 released in May
CherryPy	N/A	3.7.0	Web server/VOD streaming	

Chapter 3

First sprint: Laying the foundations

At the very start of our project we wrote a project plan which is included in Appendix C. In this document we described the agile software development methodology called Scrum that we used when working on the requirements for the project. Every development cycle is called a sprint and in the following three chapters the work done during each of the three sprints will be discussed. In the project plan we also discussed MoSCoW; a technique for prioritizing software requirements. At the start of each sprint chapter we will give some of the requirements that we tried to complete in the MoSCoW format.

For the most part the first sprint was focused on getting Tribler to work on Android using Kivy/P4A. We could build upon the work done by other groups, see 2.3 on page 11, but since the Tribler code base has undergone some changes [32] and new dependencies were introduced, additional work needed to be done to make this work.

For the first sprint the following goals were set:

1. Must have: Run AT3 with the new Tribler codebase
2. Must have: Run TSAP with the new Tribler codebase
3. Must have: New P4A for the newly introduced libraries
4. Could have: Rewrite P4A recipe for libtorrent

Over time we found out that it would take quite some more work to get the project fully working, mainly because Tribler has seen quite some active development over the last year. This resulted in a different approach in the second sprint.

3.1 Recipe for CherryPy

CherryPy supplies an object orientated framework for Python applications that have a web front-end. Tribler uses the CherryPy engine to provide a web interface that can directly deliver data (in for instance the JavaScript Object Notation (JSON) format) over the Hypertext Transfer Protocol (HTTP) protocol. Besides that it is used for its library functions. This was one of the first recipes for P4A that we wrote. Other than that this recipe did not cause much problems when writing it.

3.2 Recipes for libnacl and libsodium

One of the new dependencies introduced is libnacl. This pure-python module contains the bindings to interface with the libsodium library. These cryptographic functions are used in the anonymous tunnel functionality of Tribler. More on the cryptographic libraries can be found in section 4.3 on page 24.

Since Kivy supports pure-python modules such as libnacl by automatically downloading them using pip, it should not be necessary to include a recipe for this module. However, it is part of the Tribler code base, as a submodule, but not referred to as such.

Compiling libsodium for Android was an easy task as it already has support for Android. Our recipe automates this task by compiling libsodium for Android and making it available for libnacl. Later in the project we intend to submit a pull request upstream to have this recipe merged so that other users can enjoy state-of-the-art cryptography in their projects.

Any project that wants to use the libnacl library in their project needs to include both libnacl and libsodium in the Buildozer requirements, as libnacl does not have libsodium as a dependency.

3.3 Recipes for libtorrent and boost

The libtorrent library is used by Tribler to download torrents, optionally using the anontunnel functionality. In prior work done by the AT3 team, a recipe was written which can be used by Kivy to create Android applications that use it is functionality. The recipe, however, contains a pre-compiled binary which does not contribute to the maintainability of the project, as more work is needed to upgrade to a newer version of the library. This is already the case as can be seen in the section 2.5.6 on page 14. We intend to write a recipe which compiles the library for the given platform. Given the way the library is compiled, it is quite difficult to do this. As the library depends on the Boost C++ Libraries and the NDK supplied by Google has these installed by default, these need to be compiled first. Moreover, the Boost C++ Libraries need to be compiled with Python bindings, as libtorrent depends on these. The compilation of Boost C++ Libraries with Python bindings requires Python configuration settings which are generated by Buildozer.

The compilation then can be done using two different build systems, either using Makefile or using Boost.Build. The benefit of the first approach is that the Boost C++ Libraries can be build outside of the build process of libtorrent and the benefit of the second approach is that only one targets needs to be compiled as the Boost C++ Libraries are also build as a dependency.

3.4 Tribler Main Interface

Another library that Tribler uses is wxPython. This library provides a GUI toolkit for Python. Even though we only want to use Tribler for its functionality without the user interface, since the Android application will feature a completely different interface, we still noticed that the anonymous tunnel code in Tribler that we wanted to use made a call to a libtorrent test download function that required wxPython.

Because of this dependency, we started writing a P4A recipe for wxPython to enable usage of the anonymous tunnel code. After fixing multiple problems that occurred while cross compiling via the recipe we found that it seemed to work except

for the fact that it still required GIMP Toolkit (GTK+). So we went on to write a recipe for GTK+. This recipe did not work because P4A did not support archives with a tar.xz extension. However, instead of reporting that it did not support this extension, it went on with the cross-compilation steps specified in our GTK+ recipe, but on a different archive than GTK+. This issue was fixed by adding support for unpacking of archives with tar.xz extensions to P4A, once we found the cause.

Sadly, however, after resolving other problems with GTK+, we came to the conclusion that GTK+ itself had too many dependencies. Our new approach was to change Tribler itself to make sure that it did not require wxPython anymore in the section of the code that we needed to download anonymously. This means that we do not have to write a recipe for wxPython anymore.

During this sprint the changes made to Tribler have not been included in the main Tribler code base yet, but instead we managed our own version with our changes that we hope to merge with Tribler at a later point. The reason for waiting is that we first want to know all necessary changes that we have to make (there might be changes needed for other recipes), so that we can send all of our changes at once as neatly as possible so it becomes easier for the Tribler team to review our changes before using them.

3.5 Recipe for Tribler

A recipe for Tribler was made by the AT3 team. This recipe downloaded an older version of Tribler with a few changes to the source code to make it work. We updated this recipe with Tribler's new dependencies and made it download our own version of Tribler. It does this by making a recursive clone of our Tribler repository on GitHub. This is to make sure that our latest changes are included in the download. The reason for a recursive clone is to include Dispersy, Pymdht and libnacl's sources in the download. It then runs the setup.py file which will be discussed in the next section.

Tribler is multi-platform software. It does this in part by detecting on startup which Operating system (OS) it is being run on. Since Android is based on Linux, Tribler detects it as such. But because Android and Linux do not have the same directory structure, the correct directories were not used when running Tribler on Android.

In order to have Tribler correctly detect when it was running on Android, the file `Tribler.Core.osutils.py` needed to be updated. Kivy's `android` module would be imported, and if the import succeeds, it would mean that Tribler is running on Android. Otherwise, it would be running on Linux (or Mac).

3.6 Tribler setup.py

By using the Distutils package in Python, one can write a setup script for their Python program so that it can be built and installed as a Python package from which modules can later be used. Such a script is by convention located in a file called setup.py. Tribler did not have such a setup script, instead it uses different scripts for different operating systems. It also does not have a generic script for the different Linux distributions, which is something that we do however need.

So a new setup script had to be written. The script makes sure that after the full version of Tribler is downloaded, only the necessary packages are installed. It also makes sure that some files that are not part of the code base but are needed (think

about data files) are included at their proper locations. We currently only include packages that we need in our application. In another sprint we might write a generic script so that all packages are included or make it possible to specify via the script parameters which packages should be included. This could then be submitted to the Tribler team to solve issue #1086. This item is however not that high on the backlog.

3.7 Recipe for leveldb

LevelDB is a Not Only Structured Query Language (NoSQL) database made by two employees at Google that can store keys and their values on-disk. It also provides compression of the associative (key-value) array if the optional compression library Snappy is used. Because the keys and values are stored on-disk, it can be used for persistent storage.

Because LevelDB is dependent on Snappy, we attempted to write a recipe that would cross-compile both LevelDB and Snappy. After spending much time on getting Snappy to work, however, we decided to disable Snappy, since it is only an optional requirement for LevelDB. This was done by creating a patch that removed compile arguments from LevelDB's setup.py file that instructed the compiler to use Snappy's libraries during compilation.

Afterwards, the challenge was to get LevelDB compiling for Android. LevelDB uses the GNU standard C++ library, which is not used by default by Buildozer's compilation process. This library is contained within the used NDK, and the path to this library must be passed as an entry in the compiler's CFLAGS and CXXFLAGS. After getting the compiler to use the correct library, the compiler found that LevelDB was using non-locking I/O functions that are not available for ARM architectures. Another patch was therefore created to replace these functions with their locking equivalents. LevelDB was able to compile successfully, and its Python bindings installed, after this last patch.

3.8 Download test with newer Tribler version

The anontunnel functionality, which was used last year by the AT3 and TSAP projects to achieve anonymous downloads, is no longer available in Tribler. Since anonymous downloads are required, we made a test script in Python that could achieve anonymous downloads. We first did this by creating a session object (`Tribler.core`) and then using this to download a torrent file. This did however not give us any control on how the download would be conducted, and since we needed an anonymous download, we used the tunnel object (`Tribler.community.tunnel.main`) instead. The tunnel object then made a session object itself which could be used for an anonymous download. This script that we made was still meant to only run locally on a Ubuntu machine. When trying to get it working on Android once all of Tribler's dependencies have been ported in a future sprint, we might run into new problems.

3.9 Research GUI methods for final app

Currently we are in the phase where we still have to get Tribler working on Android, but we would like to already look ahead and think about what we are going to use for our GUI once it works. We looked at the two different directions taken by the AT3 and TSAP teams and saw that these two approaches were not compatible with

each other. AT3 went for a type of test GUI created in Python by using Kivy. It did not have much features other than showing the statistics of a torrent that was being downloaded. TSAP, on the other hand, was not writing a test GUI. They were working on an application that went towards being ready to be distributed on Google Play. To do this they wrote their GUI in Java, so that it could make use of all the features in the Android SDK. Kivy also has some support for using features from the Android SDK but this is very limited. On top of that, it is also very hard to make good-looking interfaces in Kivy.

Before making a decision on which route to take, we also had to look at the underlying system that makes both the Kivy method and the Java method work. For Kivy it was rather straight forward, since Tribler is made in Python and so is Kivy, therefore communicating between the two has no limitations at all. When we would need to call some sort of Java code from Kivy, we can use Pyjnius. TSAP makes use of XML-RPC to let Java communicate with Python, as discussed in Chapter 2. It involves running a local web server so that Java and Python can communicate via HTTP. No real good alternatives could be found for this. It should be noted that Pyjnius itself cannot be used to let Python interact with already running code in Java.

Other solutions that can do this were discarded, because they only work with Java running in the Java Virtual Machine (JVM) and not in Android's Dalvik Virtual Machine (DVM). In the end we decided to go with the XML-RPC approach, mainly because this approach is the one that best helps us achieve our final goal. Additionally, if we decide not to use it, their work would go to waste and we would just end up redoing it in a different way. The possible overhead of XML-RPC was not enough to be a game breaker, since no large amounts of data would be sent over XML-RPC.

3.10 Switching from Trello to GitLab issues

Originally, we had decided to use Trello, a project management application, to manage our progress during the course of this project, such as information about Scrum sprint cycles, coming meetings, problems encountered, and general information about the project. The reason for this decision was simply that every member of the team was familiar with the application, since its usage was mandatory for last year's context project. See figure 3.1 for a screenshot of Trello's interface.

After using Trello for roughly a month, we made the decision to switch to GitLab issues to manage our project progress. A reason for the switch is that, since we use mainly GitLab for source control, we can easily manage these issues from commits, as well as reference those commits, and comment on them in a place we have to look anyway. By using issues instead of Trello, we reduce the number of different tools we have to use and combine, which made keeping track of our project a lot easier and more pleasant. Figure 5.5 is a screenshot of GitLab's issues page.

Figure 3.1: Screenshot of Trello's interface

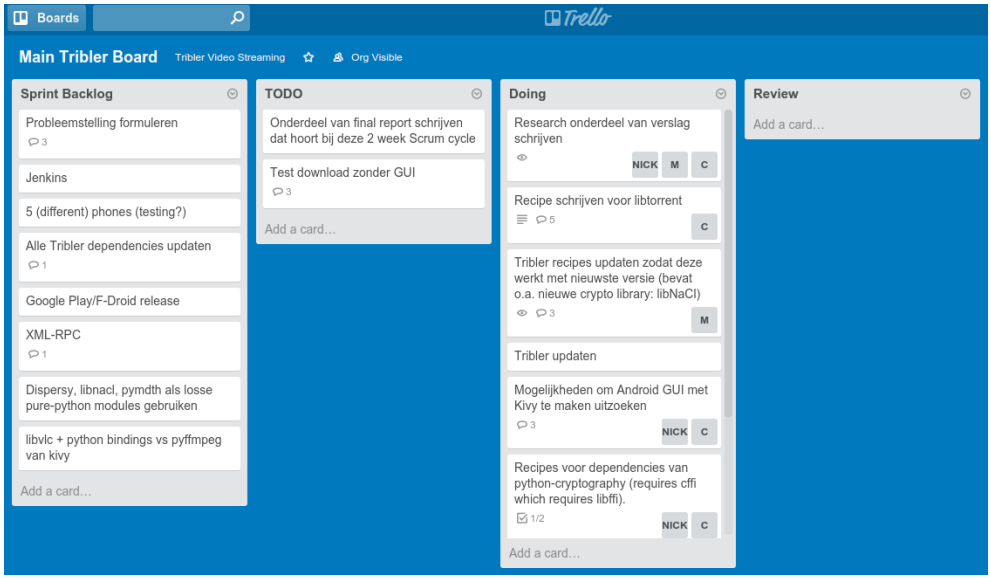


Figure 3.2: Screenshot of GitLab's issue interface

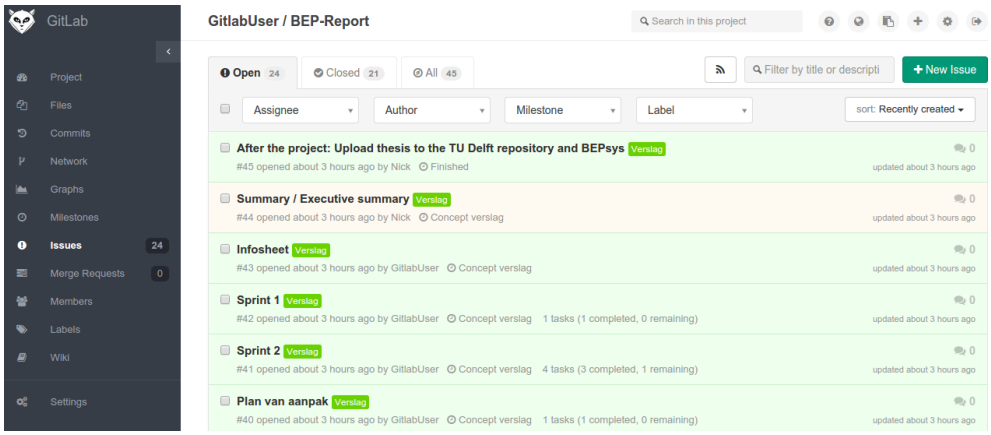
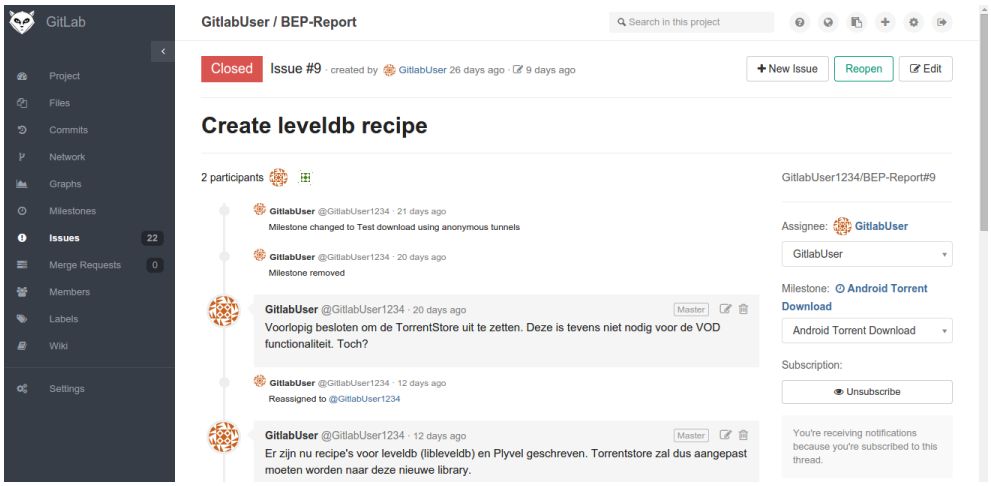


Figure 3.3: Screenshot of an issue page on GitLab



Chapter 4

Second sprint: The skeleton approach

Getting all of Tribler's dependencies to work proved to be a time consuming task. The reason for this is that all sorts of bugs had to be fixed when cross-compiling these dependencies and some of these fixes meant editing these with the use of a patch. So during this sprint more time was spent on some of the recipes from the previous sprint.

During this sprint we thus had the follow requirements:

1. Must have: Produce a minimal application with download functionality from Tribler
2. Must have: New P4A recipes for the newly introduced libraries
3. Should have: Setup Continuous Integration testing
4. Could have: Rewrite P4A recipe for libtorrent

Since we were unable to produce a demo application during the first sprint, as not all dependencies were ported yet, we decided to turn off some functionality in Tribler and our application so that we could develop the Android app and add new features as they were ported. This proved to be very useful as at the end of the sprint we had a running Tribler application capable of streaming a video.

4.1 Continuation of recipe for libtorrent and boost

During this sprint we were able to compile libtorrent. The P4A recipe is now able to generate a static libtorrent library with Python bindings. We have chosen for this approach as Boost is only used in libtorrent and dynamically linked gave problems due to the fact that Android does not support versioned library names. This is hard to disable in Boost, so we decided to statically link and avoid these problems. The current approach uses Boost.Build to build libtorrent, its Python bindings and all Boost libraries it needs. The Makefile approach gives the same problems as creating a dynamically linked library, namely the library version numbers.

This recipe is a much better alternative to the out-dated pre-compiled binary as supplied by the previous projects. It now allows anyone to compile the software from source.

4.2 Continuation of recipe for leveldb

In the previous sprint, a recipe for LevelDB was written that successfully cross-compiled it for Android. Getting the library to actually function on a mobile device, however, was a completely different job on its own. The Python bindings worked, and the library was compiled and placed in its corresponding directory, the application kept displaying an error stating that file libleveldb.so.1 was nowhere to be found, even though it was placed in its directory.

During a meeting with our client and coach, we were informed that LevelDB was needed only for Tribler's TorrentStore class. Since the functionality of this class is not required for VOD, it was decided that LevelDB would be removed. In order to prevent import errors caused by not having a working LevelDB, the Tribler source code had to be modified. Specifically, in the file LaunchManyCore.py, TorrentStore should only be imported if usage of the torrent store has been enabled in the Tribler session.

However, after some technical discussions we concluded that the TorrentStore functionality is needed by Dispersy and more specifically, to resolve magnet links. Since this is needed not for VOD but for the other team, priority was given again to implement this.

It was decided to change the Python bindings of LevelDB from py-leveldb to Plyvel, as the former automatically compiles both LevelDB and Snappy and we wanted more control over this process. Plyvel does not do this and is better suited for our needs. Note that py-leveldb might be useful for the Tribler desktop client as users need not to install LevelDB themselves. Also, this change has minimal impact on the Tribler codebase as the TorrentStore is a simple wrapper around it and LevelDB itself has very few functions. For now in our Tribler fork we exclusively use Plyvel but we intend to submit patches to Tribler to support both.

4.3 Usage of various cryptographic libraries

The following cryptographic libraries are used directly by Tribler: libnacl, cryptography and M2Crypto. These libraries themselves use other libraries that handle cryptographic functionality, such as OpenSSL which is required by M2Crypto, but these libraries will not be discussed here as only the functionality directly available to Tribler is of importance when determining what is still needed.

In this section we will explain how each of these libraries are used so that we might find which ones are redundant. Because if we find redundant libraries, it might mean that we have to write less recipes and therefore save time. First of all we found that the pycrypto library (not to be confused with the cryptography library) and the gmpy library are never used in Tribler (or Dispersy). The README of Tribler does however state these as dependencies.

libnacl, the python wrapper for libsodium, is used by Tribler and Dispersy. Dispersy uses libnacl for its dual key functionality. A DualKey object can be used to manage public and private keys, also called a key pair. These public and private keys can then be used to send encrypted messages to others in the network and to decrypt them again. This is called an asymmetric key algorithm since the key used to encrypt or decrypt by the first party is not the same as the key used to reverse the encryption or decryption by the second party. However, having a single shared key (for symmetric key algorithms) allows for faster communication. This can be done using a Diffie-Hellman key exchange function, which computes a shared secret between the parties. This computation is done by letting both parties choose a private

key and the corresponding public key. The public keys are then exchanged and then both parties use their received public key in combination with their own private key to come up with the shared secret (because of the way Diffie-Hellman is designed, both will arrive at the same shared secret). Because it uses a shared key it is a case of a symmetric key algorithm. The specific function used by the DualKey object in libnacl is a variant of Diffie-Hellman since it makes use of elliptic curve cryptography (ECC), and is called Curve25519. The DualKey object maintains a key pair based on the Curve25519 function. It also maintains a different key pair used for signing and verifying digital signatures. Digital signatures are a way to proof the authenticity of a message. The DualKey object uses Ed25519 to generate this different key pair and to sign and verify messages.

Tribler uses libnacl as well. The cryptowrapper.py file (located in the directory at Tribler/community/tunnel/crypto) is the only place where the libnacl and cryptography modules are imported. This crypto wrapper is used in the TunnelCrypto class (tunnelcrypto.py in the same directory). libnacl is used here exclusively for computing shared secrets (compatible with Curve25519) and verifying them. The cryptography library is then used for encrypting and decrypting messages with this shared key. To allow for this it provides an implementation of the sha256 hashing algorithm and the HKDFExpand functionality to generate expanded keys derived from an existing key (the shared key in this case). Furthermore a cipher, used to decrypt and encrypt messages, is used. This cipher is based on a block cipher (such as Advanced Encryption Standard (AES)) and a mode of operation (such as Galois Counter Mode (GCM)). There are some block ciphers and mode of operations used which are supported by cryptography but not by any of the other cryptographic libraries, making it therefore an essential library. Dispersy does not use this library.

We should note that libtorrent also depends on OpenSSL to enable HTTP over SSL (HTTPS) connections and BitTorrent connection encryption (BEP 8). Considering that the connection encryption is considered weak and redundant to Tribler's anontunnels, as well as the fact that Tribler strives to work without trackers, the encryption in libtorrent can be completely disabled and remove the need for OpenSSL in this component. It should be noted that some hashing functionality is then provided by libtorrent itself, which might impact the performance.

The final library we will discuss is M2Crypto. It is used by both Tribler and Dispersy. Tribler uses its Rand module to generate random seeds and its EC module for generating key pairs from curves (they only use the NIST-sect233k1 curve though). Dispersy uses M2Crypto to generate key pairs, create and verify signatures and for quite a lot of supported curves (which can be used in Tribler). It should be noted however that any change in the cryptographic primitives used by Dispersy may render it completely backwards incompatible, as newer clients are then unable to verify previously sent messages.

In time it may prove useful to move away from M2Crypto to cryptography as it stabilizes and adds new functionality. This is our recommendation as it reduces the number of dependencies which need to be kept up-to-date and given the fact that M2Crypto has seen no active development for over more than a year.

4.4 Recipe for cryptography

As porting multiple complex Python modules did not produce a lot of progress, it was decided that we strip some functionality and focus on getting an initial application running Tribler first. Since the cryptographic libraries changed quite a bit since last year, we have seen the addition of cryptography and libnacl, we decided that for the

time being the application will not use the new Tribler anontunnels. This removes the need for cryptography.

The Python module itself proved difficult to port. This is related to the, now old, build system that it uses. This system compiled the Python/CFFI bindings for the OpenSSL backend itself. Given that we needed to cross-compile to Android, we would have to change how this system works. In addition, the new CFFI changed how this process worked but cryptography did not yet support this. Getting cryptography to work on the older version would mean that our code had to be changed as soon as cryptography was ported to CFFI 1.1 . Since these patches were applied during this sprint, we would now be able to try porting the module again.

4.5 Non-anonymous download test with newer Tribler version

As we decided to drop some functionality in order to get a working demo with the newest Tribler code; we also decided to drop the tunnel functionality and focus just on getting VOD to work. In order to do this, we wrote some code that creates a new Tribler session and streams a predefined torrent. The VOD function was able to detect when the torrent was downloaded to a certain percentage of its size and then started playing it from the local file. The Tribler desktop version supports also seeking within the file and some logic to then start downloading the necessary pieces to play the requested part of the file. For this to work we need to be able to stream over HTTP to a video player and this requires a bit more work.

4.6 Continuous Integration

One way of improving the stability of code, is to implement a Continuous Integration solution. This has several purposes. First, it allows developers to always know if the latest code they added breaks any other functionality. It also allows for the automatic testing of code that is submitted to be included in the application. Third, it allows for the generation of automatic releasable applications. Specifically in our case, it is possible to automatically upload a new version to Google Play. Since Tribler already has a Jenkins instance running for the same purpose, we want to make sure our code also gets tested there. This is the most easy way compared to setting up our own testing infrastructure. It was discussed with the client that this will be arranged for us.

4.7 First code feedback processed

During this sprint we also received feedback on our code from Software Improvement Group (SIG). The overall score was quite good, 4 out of 5 stars. Since the team thinks maintainability is an important part of the project, we were quite content with this result. We also received some advice to improve the score even further. Given that the Unit Size and Unit Complexity of our code scored a little bit lower than the rest, we intend to take this into account when working on the application. We have also decided not to change the P4A recipes, as suggested. Changing these would result in unidiomatic code, as the current structure conforms to the structure all other recipes use. One might argue that this lacks important aspects, such as testing, but changing this would be out of the scope of our project.

Chapter 5

Third sprint: Contributions to the Open Source Community

We have several goals in mind for this third and final sprint.

1. Must have: Streaming videos within the application should be possible, along with seek functionality.
2. Must have: Integration of this application with the Shadow Internet project.
3. Could have: Contribute to the open-source community by submitting our work to its corresponding open source project.
4. Would have: Getting TSAP to work with the new code.
5. Would have: Porting the cryptography module

5.1 Recipe for cryptography

As in sprint 2 we decided to disable the tunnel functionality to avoid packaging the cryptography library, we have decided to drop this functionality altogether, as this task is beyond the scope of a time-constrained bachelor project. This decision was made with our client based on the fact that both cryptography and the Python bindings it uses, CFFI have made several big changes during our project. This did not give us much time to port these to Android. In addition, the CFFI library was only recently marked stable, while cryptography has not seen a 1.0 release yet. Our advice is to wait with porting these software libraries until they are stabilized and have decent cross-compile support.

5.2 Key roadmap decisions

During this sprint have given some thought on what components to use in the final application. Given that it took some time to get the new Tribler codebase to work on Android, we have to carefully consider what is possible to achieve in the remaining time of the project. Some of the decisions that had to be made were writing our

own application from scratch or using any of the previous projects, what video player would we be using and how would we write the common wrapper so that the Shadow Internet Team can also use it. While testing the search functionality, it became clear, with a little help from our coach, that the Tribler GUI code and core code were more intertwined than we thought. Since the Tribler.Core code does not properly initialize Dispersy, which is done when starting the GUI, our simple common wrapper had to be significantly extended to include all of this. In addition, there were several options available as to what player we would be using. It was possible to modify the provided Kivy video player which would provide a very easy integration with our code, but has an interface which contains some severe user experience issues. Another option, preferred by our client as well as TSAP and Tribler-streaming, was to use VLC. This player provides an extensive array of video codecs, more than Kivy or the Android built-in player. The downside of this player, as well as the built-in player, is that they are not easily accessible from Python and thus require us to include some Java classes with our project. This however is somewhat complicated as Buildozer does not have much support to do so. Because of this we have decided that it would be easier to reuse TSAP as the project already solved the missing initialization of Dispersy, and possible other related issues, and also has a built-in video player integrated with the rest of the functionality. The downside of this approach is that we need to rewrite their interaction with Tribler as Tribler has changed. We estimated however that this rewrite would be significantly faster than to write our own application from scratch.

5.3 Upgrading TSAP

The TSAP project used a one year old version of Tribler which contained some of their own modifications. We have rewritten TSAP to work with the newest version of Tribler and we plan on getting the changes that were needed in Tribler to be merged upstream. Since Tribler does not offer an API for the functionality used by TSAP, a lot of code had changed and did not work anymore in the state that it was. We had to go through the commit history of Tribler to find out to the best of our ability how we could update without breaking functionality. By using the TSAP code base we did not have to redo any work other than rewriting some of it. This saved us time and made sure that progress from last year was not in vain, since last year some Tribler projects that were supposed to be integrated with each other did not get to do so. Another reason to use the TSAP code base is that they had an internal VLC video player with Java wrapper working, something that they got from the project a year before them: Tribler-streaming. Since VLC used a Java wrapper and we did not have enough time to write a recipe to compile it with Python bindings (furthermore there might be issues with the look and feel of the player that we have to solve if we use Python bindings), we decided to use TSAP's Java GUI. This meant that we had to build the application with a Java bootstrap that replaced the bootstrap used by Kivy's P4A. Some of the build process had to be changed. We decided to write a script that among other things would run Buildozer and move generated files to the Android Studio project in Java.

The latest version of our application is able to search Dispersy for torrents. Once a user downloads a torrent and this happens to be a video file, the application is able to start playing the video while it is still downloading. Using the built-in VLC player we have seen several full HD videos play and noted download speeds well over 4 MB/s.

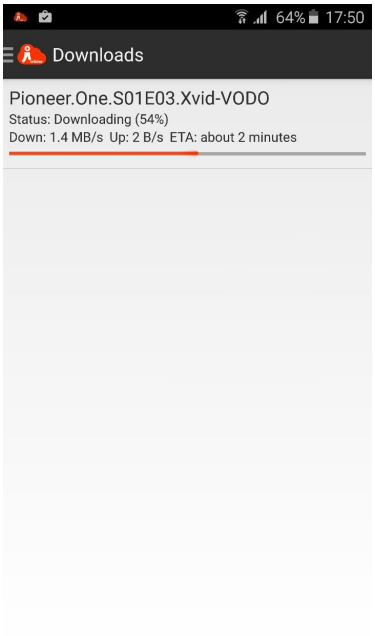


Figure 5.1: TSAP download screen

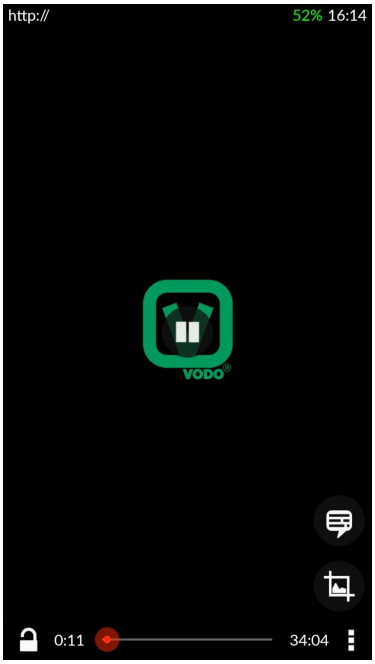


Figure 5.2: Streaming with an internal VLC player in the TSAP application

5.4 Integration with Shadow Internet group

Concurrent with our project, the Shadow Internet project was working on an application that allows users to create and distribute content without centralized servers. To solve the distribution part, they decided to use Tribler to share videos, in addition to an ad-hoc local connection.

Since we were working on updating Tribler to work on Android, they would need to use our code to make this work. Due to the way both of the projects are set up (both using P4A and Buildozer), making their application able to use Tribler was as easy as giving them our Tribler recipe.

Getting Tribler to actually work on Android, however, requires setting of specific environment variables to make Tribler detect that it is running on Android, as well as explicit initialization and configuration of some of Tribler's components to make Tribler's core function correctly.

5.5 Update older recipes

During this sprint we also updated a few components that Tribler depends on. As can be seen in de dependencies section on page 14, some of the components that are used in P4A were heavily outdated. Updating these components was very much needed, in order to, for example, fix security issues, as was the case with OpenSSL.

All of these components were updated, with the exception of M2Crypto. We decided to drop updating this, as our recommendation for Tribler and Dispersy is to move to cryptography. Additionally, updating M2Crypto to the latest version was more complicated than expected and since we gave more priority to other goals, we have decided to postpone this to when these other goals have met.

5.6 Continuous Integration

We continued with setting up the continuous integration after Jaap provided us with a little system that builds the application, and then deploys and runs it on several devices. Currently these are a Nexus 5, Nexus 6, 2 Galaxy Nexus' and a Nexus 10. This wide variety of devices, operating systems, screen sizes and other properties, allows us to test the application and make sure that it not just works on our test devices, but on others as well. We should note however that most of these phones are relatively powerful, all run ARM and either have stock Android or a custom rom. In the future, it would be advisable to add other less powerful non-stock Android phones and see if the application still works in a more constraint environment.

Figure 5.3: Devices for continuous integration with Jenkins

5.7 Final application integration

Part of this final sprint was devoted to working on the Common Tribler Core Wrapper. Due to the decisions we made this work largely involved rewriting TSAP to the new Tribler code. As part of integration the Tribler Play application with the Shadow Internet Team's work, we will make sure these play well together. We foresee that the wrapper will not have any issues as this is not yet part of the Shadow Internet application. The single roadblock for a Google Play release is a GUI integration between the two. We are confident that major steps will have been taken before we show our first public demo. The result of the integration will be hosted on GitHub instead of GitLab. This decision was made by the client since GitHub is more widespread than GitLab and has a much larger userbase. This allows us to reach a much broader audience which will hopefully attract more contributors that can help make this a better piece of software. The repository can be found at `https://github.com/Tribler/shadow-Internet`.

5.8 Contributions to Tribler

As part of making Tribler ready to be installed as a Python module, we created a setup.py to do this. We will submit this to the Tribler Team for inclusion in the codebase. This will avoid the need for a separate file in the Tribler recipe.
In addition, since we use a different module for LevelDB, Plyvel, a pull request was created that adds support for this new module to the `TorrentStore` class. This allows for either py-leveldb and Plyvel to be used as Python bindings, depending on which module has been installed on the machine. A screenshot of this pull request is found in figure 5.4.

5.9 Contributions to python-for-android

During the course of these three sprints, different recipes needed to be updated or created so that all the necessary components of the application could be compiled. These include recipes for:

- OpenSSL: update to newest version;
- zope.interface: update to newest version;
- apsw: avoid hard-coded version numbers;
- netifaces: update to newest version;
- CherryPy: create new recipe;
- LevelDB: create new recipe;
- Plyvel: create new recipe;
- Boost: fix recipe from last year's bachelor project;
- libtorrent: fix recipe from last year's bachelor project;
- libsodium: create new recipe

As a contribution to further improve the open-source project that P4A is, these updated and created recipes were submitted to the project's GitHub page as pull requests. At the time of writing, all but the libsodium recipe have been merged into P4A's master branch.

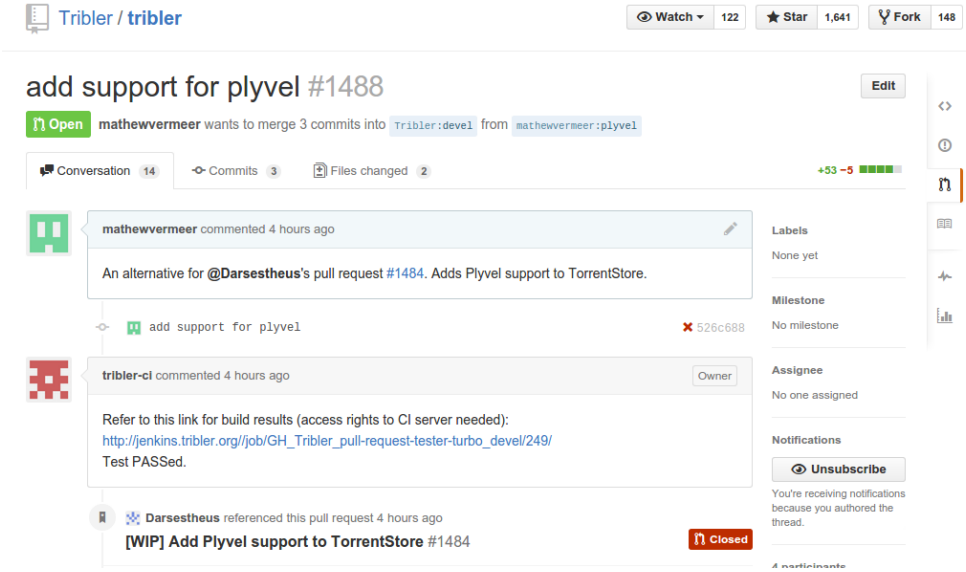
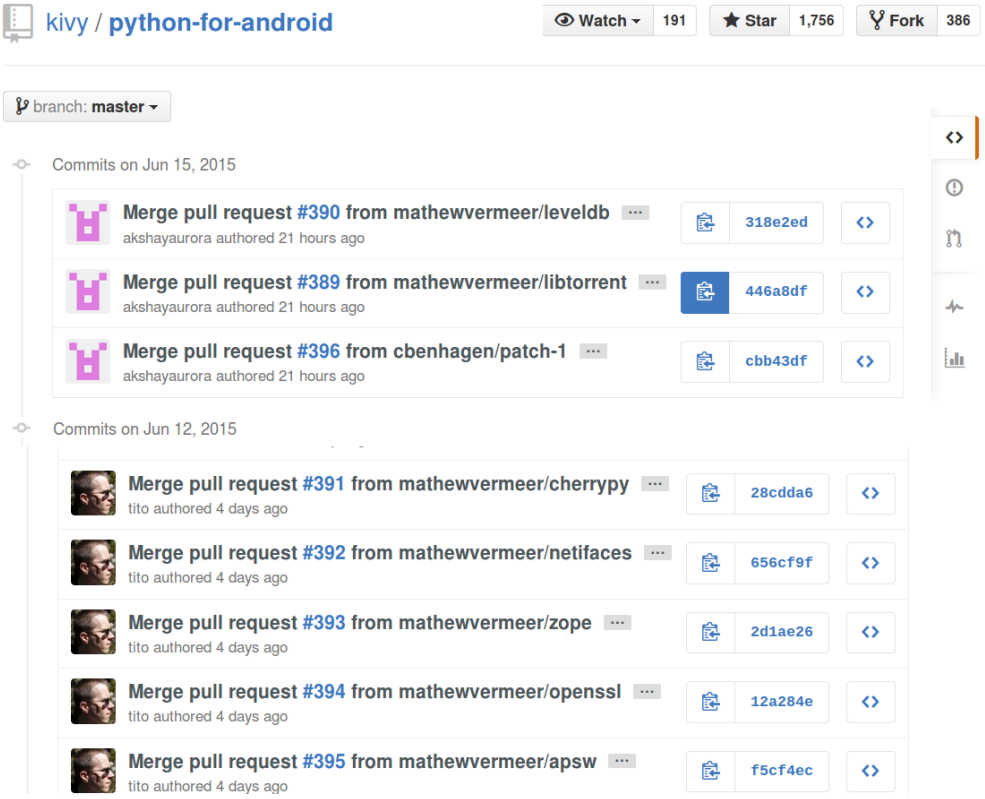


Figure 5.4: Plyvel pull request on Tribler repository

Figure 5.5: Merged pull requests on P4A's GitHub repository.



Chapter 6

Conclusion

This chapter will look back at what has been achieved and how these achievements contributed towards the problems that needed to be solved. It will mention the mistakes that were made, and what should have been done to prevent them. Throughout the conclusion some lessons learned will be shared. Finally, recommendations will be discussed for anyone who wants to improve the work done or the work that was built upon.

The biggest achievement that was accomplished during the project is getting the new Tribler codebase to work on Android. This includes the downloading and streaming parts, search and other communities using Dispersy and the TorrentStore. In addition, any modifications that were needed to be made to the Tribler code were submitted to the project. This removed the need for a separate Tribler fork specifically made to work for Android.

Part of getting Tribler to work was making its dependent software libraries available. This involved creating P4A recipes for several of these libraries. Recipes were created for the following dependencies: CherryPy, libsodium, Plyvel, libtorrent and update the recipes of OpenSSL, zope.interface, apsw and netifaces.

Looking back on the project's initial goals, the project was rather successful. Tribler's new codebase, along with a simple wrapper, was used to create common Tribler code that could be run on Android. One of the goals that was not able to be reached at the end of the project was adding the anonymity functionality using tunnels. After looking at the complexity of the cryptographic library that needed to be ported, it was decided that before porting this, it needs to be more stable and mature. Spending more time on trying to port this library would endanger other goals of the project.

Unfortunately, too much time was spent on trying to write recipes for certain dependencies that were unrealistic or unnecessary to complete. These libraries include cryptography and GTK+. The complication with cryptography is that it has not seen a stable 1.0 release yet. As such, it is not mature enough to consider porting to Android. This should have been realized before so much time was wasted on its recipe, considering the library went though some major updates in the span of this project. And instead of trying to get a GUI library ported that was not going to be used any, Tribler itself should have been refactored in order to prevent GTK+ from becoming a dependency for the core Tribler code.

During the project some problems were encountered with the tools and libraries that were worked with. Improving all of these would be out of the scope of our project, but they are mentioned here in case readers might want to improve them themselves.

Just like the recommendations of previous projects were taken into account in this current project, hopefully these recommendations prove useful for future work.

First of all, it was noticed that Tribler and Dispersy use a wide variety of cryptographic libraries for an ever wider variety of functions and primitives. A recommendation is that both projects move away from the old and not up-to-date M2Crypto OpenSSL Python bindings. In several places this has already been done by switching to cryptography. A second important recommendation is to separate the desktop GUI from the core functionality, e.g. searching and streaming. Work on this is already underway. Hopefully in time the Core Wrapper might not be needed anymore, as the Tribler Core exposes all the functionality that is needed.

As for the build system that was used, Buildozer, it must firstly be acknowledged that it is still in an alpha development stage. But even though it is currently in this stage, the addition of support for 'hybrid' projects, projects that use both a standard Android Java project, as well as some P4A functions, is very possible in the future. Currently, it is tailored to applications completely written in Python, and the app developers have to write a script to work around this limitation themselves. Support for these 'hybrid' projects would allow for a better GUI, as this can be easier built using Android design standards, ultimately resulting in a much more aesthetic interface. It must be noted, though, that Buildozer is much easier to work with than manually performing the separate steps needed to build an APK. Having a single specification file to configure made the build system pleasant to work with.

Bibliography

[1] BBC, "Russia enacts 'draconian' law for bloggers and online media," August 2014. Accessed: 17-06-2015.

[2] J. Hruska, "Russia might ban tor and virtual private networks," February 2015. Accessed: 17-06-2015.

[3] GreatFire.org, "Online censorship in china." Accessed: 17-06-2015.

[4] S. Farrell and H. Tschofenig, "Pervasive monitoring is an attack," 2014.

[5] D. Mail, "Google reveals more searches are made from mobile devices than pcs for the first time," May 2015. Accessed: 17-06-2015.

[6] "List of countries by number of mobile phones in use." Accessed: 17-06-2015.

[7] "Netflix." `https://www.netflix.com/`. Accessed: 7-05-2015.

[8] "Youtube." `http://youtube.com/`. Accessed: 19-05-2015.

[9] "Hbo now." `http://hbonow.com/`. Accessed: 7-05-2015.

[10] "Popcorn time." `https://popcorn-time.se/`. Accessed: 7-05-2015.

[11] P. M. Eittenberger, M. Herbst, and U. R. Krieger, "Rapidstream: P2p streaming on android," in *Packet Video Workshop (PV), 2012 19th International*, pp. 125–130, IEEE, 2012.

[12] L. Keller, A. Le, B. Cici, H. Seferoglu, C. Fragouli, and A. Markopoulou, "Microcast: cooperative video streaming on smartphones," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pp. 57–70, ACM, 2012.

[13] T. Iyer, R. Hsieh, N. B. Rizvandi, B. Varghese, and R. Boreli, "Mobile p2p trusted on-demand video streaming," *arXiv preprint arXiv:1204.0094*, 2012.

[14] R. K. Ege, "Securing video delivery to the android platform," *Journal of Systemics, Cybernetics & Informatics*, vol. 10, no. 5, 2012.

[15] N. Papaoulakis, C. Z. Patrikakis, C. Androulaki, L. Argyriou, and I. Schmidt, "Distributing real time user generated video over p2p networks," in *Proceedings of Third International Conference on Computational Aspects of Social Networks*, 2011.

[16] A. Gaeta, S. Kosta, J. Stefa, and A. Mei, "Streamsmart: P2p video streaming for smartphones through the cloud," in *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2013 10th Annual IEEE Communications Society Conference on*, pp. 233–235, IEEE, 2013.

[17] S. Kosta, V. C. Perta, J. Stefa, P. Hui, and A. Mei, "Clone2clone (c2c): Peer-to-peer networking of smartphones on the cloud," in *5th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud13)*, 2013.

[18] "Bittorrent android." `http://www.bittorrent.com/bittorrent-android`. Accessed: 15-06-2015.

[19] "torrent for android." `http://www.utorrent.com/mobile`. Accessed: 15-06-2015.

[20] "Flud." `https://play.google.com/store/apps/details?id=com.delphicoder.flud`. Accessed: 15-06-2015.

[21] "ttorrent." `http://www.ttorrent.org/`. Accessed: 15-06-2015.

[22] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," Free Haven.

[23] "Bittorrent over tor isn't a good idea." `https://blog.torproject.org/blog/bittorrent-over-tor-isnt-good-idea`. Accessed: 15-06-2015.

[24] I. Clarke, S. G. Miller, T. W. Hong, O. Sandberg, and B. Wiley, "Protecting free expression online with freenet," IEEE.

[25] J. Van Touw, "A non-centralized approach to video on demand on mobile devices," bachelor thesis.

[26] "Windows phone." `http://www.windowsphone.com`. Accessed: 7-05-2015.

[27] "Apple ios." `http://www.apple.com/ios/`. Accessed: 7-05-2015.

[28] "Phonegap." `http://phonegap.com/`. Accessed: 7-05-2015.

[29] "Phonegap." `http://www.nlziet.nl/`. Accessed: 7-05-2015.

[30] J. R. De Vos, M.A. and L. Versluis, "Android tor tribler tunneling (at3): Ti3800 bachelorproject," bachelor thesis.

[31] S. N. Sabe, W.F. and D. Schut, "Tribler play: decentralized media streaming on android using tribler," bachelor thesis.

[32] R. Plak, "Anonymous internet: Anonymizing peer-to-peer traffic using applied cryptography," master thesis.

[33] N. Zeilemaker, B. Schoon, and J. Pouwelse, "Dispersy bundle synchronization," *TU Delft, Parallel and Distributed Systems*, 2013.

Appendix A

Glossary

ADB Android Debug Bridge. 13

AES Advanced Encryption Standard. 25

Android `http://www.android.com/`. 1, 4–14, 16–20, 23, 26–30, 33, 34, 37, 44–46

API Application Programming Interface. 13, 28

APK Android Application Package. 13, 14, 34, 45

apsw Python bindings for SQLite. `http://github.com/rogerbinns/apsw`. 15, 33

AT3 Android Tor Tribler Tunneling [30]. 11–14, 16–20, 43

AVD Android Virtual Device. 45

BitTorrent `http://www.bittorrent.org/`. 7–10, 12, 25

Boost C++ Libraries `http://www.boost.org/`. 17

Buildozer Generic Python build-system for Android and iOS. `http://github.com/kivy/buildozer`. 13, 14, 17, 19, 28, 29, 34

CFFI `http://cffi.readthedocs.org`. 15, 26, 27

CherryPy Python framework for web application. `http://www.cherrypy.org/`. 15, 16, 33

cryptography Python library providing both high-level constructs and low level cryptographic primitives, using OpenSSL or other backends. `http://cryptography.io/`. 15, 24–27, 29, 33, 34

DHT Distributed Hash Table. 39

Dispersy See *"Dispersy bundle synchronization"*[33]. `http://github.com/tribler/Dispersy`. 10, 11, 14, 15, 18, 24, 25, 28, 29, 33, 34

DVM Dalvik Virtual Machine. 20

EEMCS Electrical Engineering, Mathematics and Computer Science. `http://www.ewi.tudelft.nl/`. 1

Freenet http://freenetproject.org/. 10, 12

GCM Galois Counter Mode. 25

GitHub http://github.com/. 18, 30

GitLab http://gitlab.com/. 20–22, 30

Google http://google.com/. 4

Google Play http://play.google.com/. 6, 20, 26, 30, 43

GTK+ GIMP Toolkit. 18, 33

GUI Graphical User Interface. 12, 13, 17, 19, 20, 28, 30, 33, 34

HTTP Hypertext Transfer Protocol. 16, 20, 26

HTTPS HTTP over SSL. 25

I2P http://geti2p.net/. 8, 9, 12

Jenkins http://jenkins-ci.org/. 26, 45

JSON JavaScript Object Notation. 16

JVM Java Virtual Machine. 20

Kivy Cross-platform Python Framework for Natural User Interface Development. http://kivy.org/. 11–13, 16–18, 20, 28, 45

LevelDB A key-value storage database. http://github.com/google/leveldb. 15, 19, 24, 31, 39

libnacl Python bindings for libsodium. http://github.com/saltstack/libnacl/. 13, 15, 17, 18, 24, 25

libsodium Cross-platform port of NaCl. http://libsodium.org/. 13, 15, 17, 24, 33, 38

libtorrent http://libtorrent.org/. 10, 12, 15–17, 23, 25, 33

M2Crypto Python bindings for OpenSSL. http://github.com/martinpaljak/M2Crypto/. 15, 24, 25, 29, 34

Makefile http://www.gnu.org/software/make/manual/make.html. 14, 17, 23

NaCl Cryptographic library as created by Daniel J. Bernstein, Tanja Lange and Peter Schwabe, exposing several primitives such as Salsa20, Curve25519 and Poly1305. http://nacl.cr.yp.to. 38

NDK Native Development Kit. http://developer.android.com/tools/sdk/ndk/. 13, 17

netifaces http://bitbucket.org/al45tair/netifaces. 15, 33

NoSQL Not Only Structured Query Language. 19

OpenSSL http://www.openssl.org/. 15, 24–26, 29, 33, 34, 37, 38

OS Operating system. 18

P2P Peer-to-Peer. 1, 4, 6–11, 42, 46

P4A Python for Android. http://github.com/python-for-android. 12–18, 23, 26, 28, 29, 31–34

PDS Parallel and Distributed Systems. http://pds.ewi.tudelft.nl/. 1

pip Tool for installing Python packages. http://pip.pypa.io/. 14, 17

Plyvel Python module with bindings to LevelDB. http://plyvel.readthedocs.org/. 15, 24, 31, 33

Popcorn Time https://popcorntime.io. 9, 12

Pyjnius Python library which provides access to Java classes. http://pyjnius.readthedocs.org/. 20

Pymdht Python implementation of the Mainline Distributed Hash Table (DHT) protocol. http://github.com/rauljim/pymdht. 18

Python http://www.python.org/. 12, 13, 16–20, 23–28, 31, 34, 43–45

RPC Remote Procedure Call. 13

SDK Software Development Kit. http://developer.android.com/tools/sdk/. 13, 20

SIG Software Improvement Group. https://www.sig.eu/. 26, 47

Snappy Compression library. http://github.com/google/snappy. 19, 24

Tor http://www.torproject.org/. 9

Trello http://trello.com/. 20, 21

Tribler http://tribler.org/. 1, 4–6, 10–20, 23–34, 42, 43, 45, 46

Tribler-streaming Libtorrent Streaming Android Project [25]. 12, 28

TSAP Tribler Streaming Android Project [31], also known as Tribler Play. 5, 12–14, 16, 19, 20, 27, 28, 30, 43, 46

twisted http://twistedmatrix.com. 15

Twitter http://twitter.com/. 4

VLC VLC media player. http://www.videolan.org/vlc/. 9, 11, 12, 28

VOD Video-on-Demand. 4, 6, 7, 10–12, 14, 15, 24, 26, 42

wxPython Python wrapper for wxWidgets. http://www.wxpython.org/. 17, 18

Youtube See Youtube [8]. 4, 11

zope.interface http://docs.zope.org/zope.interface/. 15, 33

Appendix B

Original Project Description

The original project description as stated by the client on BEBSys is supplied below.

B.1 Project description

In this project, a prototype of an anonymous P2P-based video-on-demand mobile application will be realized, that will allow users to search for videos on the Internet. Once the user has found a video to his or her liking, and issued a play command, the application will play the video by means of streaming it through a P2P-based network. TOR is the market leading solution for anonymous web access. The Tribler team has created a P2P implementation of the TOR prototocol and enhanced it with NAT puncturing and decentralized the directory service. Your task is to take the existing operational Python code, initial Android code and make it ready for deployment towards the Google Android Play app marketplace. Challenges are understanding the available complex codebase, integration of some existing libaries, Jenkins continuous build setup, working with other ongoing team efforts on the same codebase, multi-device compatibility, User Interface realization and polishing for Google Market readyness.

B.2 Company description

Tribler Team. Read about the team at these URLs:
http://www.reddit.com/r/tribler,
http://www.ee.princeton.edu/events/anonymous-hd-video-streaming-and-reputations,
http://news.harvard.edu/gazette/story/2007/08/creating-a-computer-currency,
http://tweakers.net/nieuws/98175/torrentclient-tribler-gebruikt-onderdelen-tor-voor-anonieme-downloads.html,
http://github.com/Tribler/tribler/issues/1066

B.3 Auxilary information

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

Appendix C

Project Plan

In the previous year, work has been done on using existing Tribler code for anonymous torrent downloads on mobile devices, as well as video streaming via torrent on mobile devices. The goal of this project is to combine these two previous projects and create a mobile application that allows the user to anonymously download and stream videos on their mobile device.

In this appendix we will describe the specifics of this project. Furthermore, we will sketch out how we will approach the problem and how solutions will be implemented.

C.1 Project

In this section, we will describe this bachelor project, client, contacts and the problem the group will focus on. We will also outline the final product that we must deliver to the client at the end of the project. Some of the most important requirements will also be discussed in this section.

C.1.1 Environment

The vision of Tribler is to create a decentralized internet infrastructure that cannot be shutdown by governments and is hard to be eavesdropped on. This project will make use of Tribler to allow for anonymous video streaming on Android. Websites like YouTube [8] make use of a central server which makes it expensive to maintain and harder to scale. Peer-to-peer video streaming could hold a lot of potential in the future of video sharing.

C.1.2 Description

In this project, we will implement an anonymous P2P VOD mobile application and make it ready for deployment on Google Play. Last year, two different projects were completed: one that allows anonymous P2P downloads on an Android device, and another one that allows peer-to-peer video streaming on an Android device. We will take these final products of last year's bachelor projects, adapt and combine them in order to create a working application. This will be an Android mobile application that allows the user to anonymously download and stream video files from a peer-to-peer network.

C.1.3 Client

Our client is Dr. Ir. Johan Pouwelse, the head of the Tribler group and Assistant Professor at the Parallel and Distributed Systems Group of the Faculty of EEMCS, Delft University of Technology. Pouwelse has measured and researched peer-to-peer networks for years and has been working on Tribler for ten years.

C.1.4 Contacts

Client:

Dr. Ir. Johan Pouwelse
J.A.Pouwelse@tudelft.nl
+31 (0)15 27 82539
Room: HB 07.290
Mekelweg 4
2628 CD Delft

TU Delft coach:

Ir. Egbert Bouman
E.Bouman@tudelft.nl
Mekelweg 4
2628 CD Delft

Bachelor Project Coordinator:

Dr. Martha A. Larson
M.A.Larson@tudelft.nl
+31 (0)15 27 87357
Room: HB 11.040
Mekelweg 4
2628 CD Delft

C.1.5 Final Product

When finalizing our bachelor project, we will deliver the final product to the client. This application will be able to anonymously download and stream videos from a peer-to-peer network. The application will be a combination of last year's projects: AT3 and TSAP. Since Tribler is written in Python, AT3 and TSAP have been (partially) written in Python as well. Seeing as how we need to combine AT3 and TSAP into a single application, our application must be built with Python too.

The AT3 project will be used to download the videos anonymously, and TSAP will be used to actually stream the selected video.

The client would prefer an application that is ready to be distributed on Google Play.

C.1.6 Requirements

As was discussed in the above sections, this will be a mobile application targeting the Android operating system. Android applications are usually written in Java, but since Python was used for the previous projects, Pythonn must be used for our project as well. This application will contain the functionality described in section Description. Functional and non-functional requirements are not yet fully known and will be identified after meeting with the client.

C.2 Approach

In order to work effectively and efficiently so that we have a working production grade version of the application at the end of the bachelor project, we must work with certain tools and techniques that allow us to do so. These include Scrum, the MoSCoW method, and all the software tools that make the development of this application possible.

C.2.1 Scrum

The agile software development methodology that we will use is Scrum. Because this software development methodology is agile, it allows for quick adaptation of changed software requirements (something which happens all to often in real world scenarios). It also forces us to have a shippable deliverable at the end of each sprint session.

Within Scrum there are three roles assigned to certain people associated to the project. The first one is called the product owner and it's attributed to the person being the voice of the client, in our case Johan Pouwelse. The second role is given to every member of the development team; the role of developer. For this project that will be all students working on the project. The final role is that of Scrum master. We decided to collectively take on this role as there is no one else able to take this role.

C.2.2 MoSCoW

The MoSCow method is a technique that is often used in software development. It is used to reach a common understanding with stakeholders about the importance of every requirement of the piece of software set by the stakeholders. Requirements can then be prioritized according to their importance. The MoSCoW categories are the following:

- **M**: MUST. A requirement that must be satisfied.
- **S**: SHOULD. A requirement that should be included in the implementation if possible or if time permits.
- **C**: COULD. A requirement that is desirable by the stakeholders or developers, but not necessary for a successful product.
- **W**: WOULD/WON'T. A requirement for which the stakeholders have agreed not to include in the next release, but may possibly be implemented in the future.

We will use the MoSCoW method to categorize and prioritize the requirements set by our client. This will allow us to focus on and complete the most important requirements first to ensure a successful product.

C.2.3 Tools

Since Tribler is written in Python, a tool called Python for Android (by Kivy), is used to port the Python code base to an APK. This tool is created for Linux so that is the environment that is currently used.

Programming of the app-specific requirements (which are not in Tribler) is done in Java and inside the Android Studio. Debugging of the app is done using the Android Virtual Device (AVD) or a physical device. A Jenkins server is utilized so that continuous integration can be used.

C.3 Planning

At the beginning of the project a meeting with our client Johan Pouwelse is scheduled in order to discuss the specific requirements of the project. After that a planning can be made. How this planning developed can be found in the different sprint sections.

Appendix D

Infosheet

Tribler Play
Client organization: Tribler Team at TU Delft
GitHub: https://github.com/tribler
Website: https://tribler.org
Presentation Date: 26 June 2015
Description:

Privacy is dying. Internet users have been trying more and more to keep governments' and security agencies' prying eyes away from their online activity. Anonymity - Internet anonymity, specifically - has therefore gained traction in recent years. In order to further contribute to the effort of Internet privacy and anonymity, this project's goal was to create an Android mobile application that allows users to anonymously stream videos from a decentralized network of peers. The main piece of software used to achieve this goal was Tribler, a P2P file sharing program that provides users the ability to download their files anonymously. Since Tribler is not written for mobile devices, it, along with all of its dependencies, needed to be cross-compiled for the ARM CPU architecture. All of Tribler's dependencies were successfully ported, except for a cryptography library due to its lack of stability and maturity. As a result of this, an application was created that successfully streams video from a decentralized network of peers, but without the ability to do so anonymously. Even though anonymous streaming is not yet working, that functionality can be easily implemented in the application when the necessary cryptography library becomes mature and stable enough for cross-compilation.

Members of the project team
Name: C. van Bruggen (C.vanBruggen@student.tudelft.nl)
Contributions: Libtorrent recipe
Name: N. Feddes (N.F.Feddes@student.tudelft.nl)
Contributions: Rewrote TSAP
Name: M. Vermeer (M.Vermeer-1@student.tudelft.nl)
Contributions: Video Streaming Prototype

All team members contributed to all the work done that has not been mentioned in the specific contributions above.

Name and affiliation of the client:
Dr. Ir. Johan Pouwelse, Parallel and Distributed Systems Group, TU Delft (peer2peer@gmail.com)
Name and affiliation of the project coach:
Ir. Egbert Bouman, TU Delft (E.Bouman@tudelft.nl)

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

Appendix E

SIG Code feedback (DUTCH)

Dit deel bevat informatie over de SIG code terugkoppeling die we hebben gehad. Er zijn twee upload momenten van onze code geweest, waarbij het de bedoeling was om de feedback van de eerste upload the verwerken in code van de tweede upload. Hoe we dit gedaan hebben is in sprint 2 terug te lezen, en of dit goed gelukt is kunt u lezen bij de feedback over de tweede upload in deze appendix. Een opmerking bij de feedback van de tweede upload is dat de hoeveelheid code ongeveer gelijk is gebleven. Dit heeft er mee te maken dat oude prototypes niet opnieuw zijn opgestuurd en veel code is toegevoegd aan al bestaande projecten wat dus niet apart opgestuurd kon worden.

E.1 Feedback eerste upload

"De code van het systeem scoort vier sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Unit Size en Unit Complexity.
Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld de 'status'-methode binnen 'AnonTunnelService', zijn in de indentatie aparte stukken functionaliteit te herkennen welke gerefactored kunnen worden naar aparte methodes. Een ander voorbeeld is het gebruik van commentaar zoals '# Add new flags', dit geeft over het algemeen ook aan dat er een apart stuk functionaliteit begint. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.
Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Ook hier geldt dat het opsplitsen van dit soort methodes in kleinere stukken ervoor zorgt dat elk onderdeel makkelijker te begrijpen, makkelijker te testen en daardoor eenvoudiger te onderhouden wordt. In dit geval komen de meest complexe methoden ook naar voren als de langste methoden, waardoor het oplossen van het eerste probleem ook dit probleem zal verhelpen.
Het opnemen van een README met daarin een duidelijke uitleg van de structuur van de code is trouwens erg goed.

Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase. Als laatste nog de opmerking dat er geen unittest-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen."

E.2 Feedback tweede upload

"In de tweede upload zien we dat de hoeveelheid code ongeveer gelijk is gebleven (er is evenveel code toegevoegd als verwijderd). De score voor onderhoudbaarheid is licht gestegen. De stijging wordt voornamelijk veroorzaakt door een verbetering op Unit Size en Unit Complexity, die in de analyse van de eerste upload als verbeterpunt werden aangemerkt. Jullie scoren voor die twee deelscores nu allebei 5 sterren, dus complimenten voor de grondigheid waarmee jullie de refactoring hebben doorgevoerd.
Daarnaast hebben jullie wat testcode toegevoegd. Aan zowel de hoeveelheid tests als de kwaliteit van de testcode valt nog wel het een en ander te verbeteren, maar het is goed om te zien dat jullie deze aanbeveling hebben opgevolgd.
Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject."

Appendix F

Dependency Graph