Technische Universiteit Delft

TI3800 Bachelorproject

A non-centralized approach to Video on Demand on mobile devices

Orientation Report

Authors:
Martijn Breet
(1265458)
Jaap van Touw
(1380753)

Supervisor:
Cor-Paul Bezemer

August 4, 2013

Contents

Abstract

Preface

Chapter 1

Subdivision of the Research Question

1.1 Introduction

This chapter will describe how the research question, first submitted in the Plan of Action, can be subdivided into different subquestions. Research into different solutions will follow in the next chapters.

1.2 Subdivision of the Research Question

The project is focused on answering the following main research question:

"How can we make video-on-demand available for mobile devices using a non-centralized approach?"

This can be broken down into the following questions:

- *"How can we facilitate sharing through a Peer-to-Peer network?"*
- *"How can we stream video to Android?"*
- *"How can we play any type of video on Android?"*

Chapter 2

Tribler

2.1 Introduction

Tribler is an application that enables its users to find, enjoy and share content through a P2P network. The application is currently available for Windows, Mac and Linux[1]. This section will provide an overview of Tribler and describer related software.

2.2 An overview of Tribler

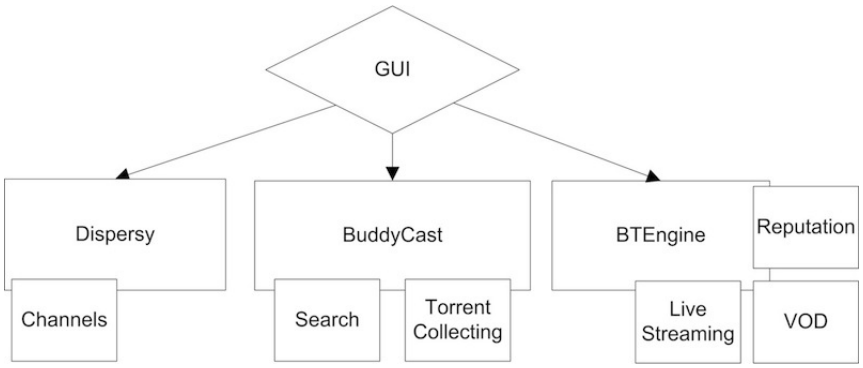

Figure 2.1: The architecture of Tribler

Tribler consists out of four major components as can also be seen in figure 2.1[2]:

- GUI: the graphical user interface.
- BTengine: the bittorrent engine.
- BuddyCast: a protocol used to find peers with the same taste as the user [3].

[1] www.tribler.org
[2] http://sigmm.org/records/records1201/featured03.html
[3] http://iptps06.cs.ucsb.edu/papers/Pouw-Tribler06.pdf

- Dispersy: a fully decentralized system for data bundle synchronization[4].

2.2.1 GUI

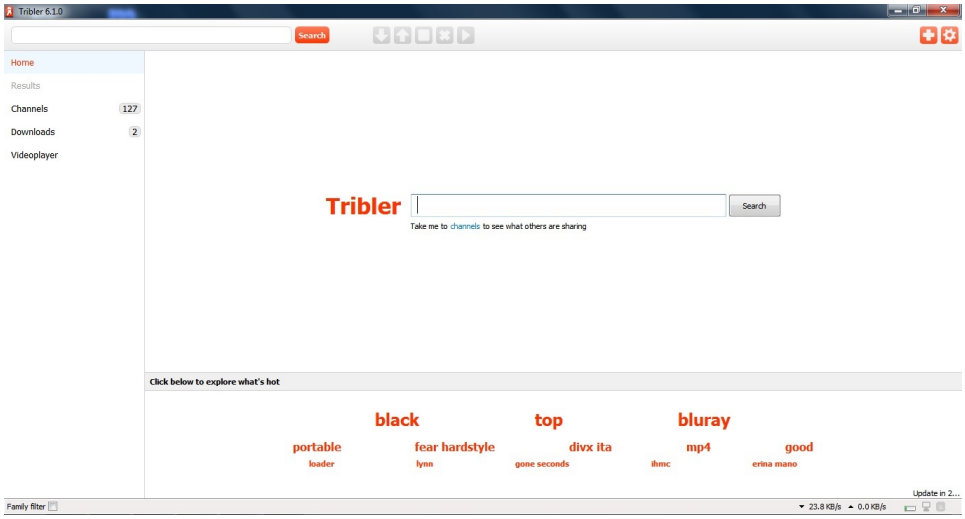

Figure 2.2: Tribler's GUI

The GUI has a number of elements to it, resembling Tribler's main features:

- Top bar
  This includes a search bar, a number of controls, and buttons to go into the settings and download external torrents. The controls include a button to start streaming the video instead of having to wait for it to be completely downloaded.

- Left pane
  The left pane gives an overview of the different pages to visit. Channels is a page where collections of content are bundled so the user can browse through them. The page below, the downloads page, shows which torrents have been downloaded and what the status is of the current download. The next page is the videoplayer where videos can be watched and streamed.

- Bottom bar
  At the bottom, information can be found how fast the download is going, as well as the upload and more information about how much peers the user is connected to.

2.2.2 BitTorrent

BitTorrent is a protocol which allows for P2P file sharing, section 3 will describe more about other P2P file sharing protocols that might be used in the Mobile VoD project. The protocol allows users to join a swarm of hosts to download and upload files. For a user to share a file it can create a torrent descriptor file

[4]http://www.pds.ewi.tudelft.nl/fileadmin/pds/reports/2013/PDS-2013-002.pdf

which contains information about the file. The torrent can then be distributed over the internet via e-mail, a link on a website, etc. Other users with the torrent can connect to this host, called a seeder, and ask for pieces of this file. After all pieces are collected, the leecher becomes a seeder and other leechers can download from the new seeder. This way the files is distributed over the Internet and no central server is needed.
The BTengine in Tribler uses the BitTorrent protocol and also includes a reputation system, where the user is rated for their upload to download ratio. This helps to minimize the effects of free riding, where users only download, because the user with a low ratio will be given lower speed peers to connect to.

2.2.3 P2P communication

The BuddyCast protocol is used to find peers with similar taste so Tribler can give recommendations to what the user might like and thus discover new content. Dispersy is used to spread data bundles over the Internet in a fully decentralized way. This could potentially remove the need for central servers for services as Facebook or Wikipedia. In Tribler it is used for creating the channels.

2.3 Related Software

A lot of BitTorrent clients exist, such as uTorrent, MLDonkey, Miro, LimeWire, etc. The team needs to implement such a client to make VoD available for mobile devices. Not all clients support VoD, and the ones that do still need to be modified to be able to operate on Android, so they have to be open source in order to use and modify them. Because the aim of the project is to create VoD for Android powered devices, not all functionality of the open source BitTorrent clients that support VoD is needed. Because the team works at the department that created Tribler, the team can get a lot of information about how Tribler operates, which subsystems are useful for VoD and more. Tribler is therefore currently the best choice to implement as client.

Chapter 3

P2P Streaming Protocol

3.1 Introduction

This chapter describes different protocols that facilitate sharing files over a P2P network. First it descrbes three different protocols that coud be suited for VoD on mobile devices, then the different protocols are compared. Finally a conlusion, to which protocol is best suited, is reached.

3.2 Protocols

3.2.1 Libtorrent

Libtorrent is a C++ implementation of BitTorrent for embedded devices as well as desktops[1].The interface is well documented and is designed to be cpu and memory efficient as well as easy to use.

3.2.2 BitTorrent

The protocol was released in July 2001 by Bram Cohen[2] and aimed to facilitate sharing files over a P2P network. In 2012, it surpassed 150 million active users [3] and remains the base for all BitTorrent clients.

3.2.3 Libswift

The libswift protocol is designed to operate at the transport layer, it is designed to be very lightweight, have non-intrusive congestion control and automatic disk space management.[4] It is developed by the PDS department as part of the P2P-Next project[5].

[1] http://www.rasterbar.com/products/libtorrent/
[2] http://finance.groups.yahoo.com/group/decentralization/message/3160
[3] http://www.bittorrent.com/intl/es/company/about/ces_2012_150m_users
[4] http://libswift.org/
[5] http://p2p-next.org/

3.3 Trade-off

The following comparison is aimed to see which protocol is best suited for VoD on mobile devices. The following criteria are considered:

- start-up time: how long it takes for a video stream to start playing.
- stalls experienced: how often does the playback stop because the next part of the video is not ready.

Both criteria are heavily dependant on the ratio of leechers to seeders in the swarm. First, the start-up time is compared between the three different protocols, which is followed by a comparison of different download methods for VoD. The full analysis can be found in "Performance Analysis of the Libswift P2P Streaming Protocol" by R. Petrocco, J. Pouwelse and D.H.J. Epema submitted to IEEE P2P in 2012.

3.3.1 Start-up time

For the following comparison the BitTorrent client: uTorrent is chosen as implementation of the BitTorrent protocol. The start-up times are measured in two scenarios: a flashcrowd scenario where a lot of leechers come all at almost the same time and the steady-state scenario where leechers appear at Poisson inter-arrival process with a rate of 1 peer per second.

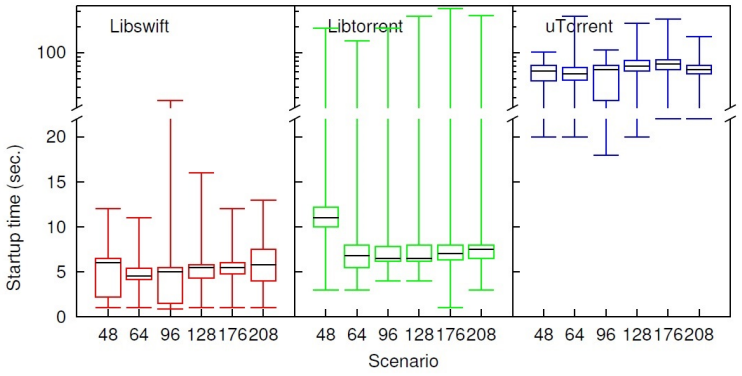

Figure 3.1: start-up times in a flashcrowd scenario

As can be seen in figure 3.1 the average start-up time of libtorrent is only a bit higher than with libswift but has a lot of outliers in the high regions, whereas uTorrent has a significantly higher average than the rest.

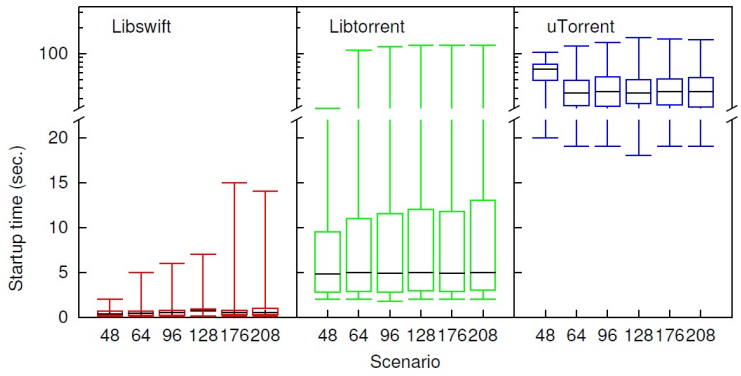

Figure 3.2: start-up times in a steady-state scenario

In a steady-state scenario, Libswift outperforms the other two protocols quite obviously as can be seen in figure 3.2.

3.3.2 Stalls

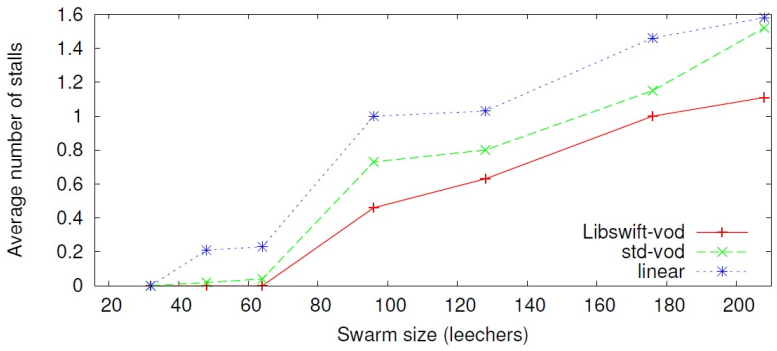

Figure 3.3: average number of stalls experienced during playback

In figure 3.3 three different download algorithms are compared:

- Linear
  Only peers further on in the playback are able to provide useful data for newcomers, so there is no global knowledge about data availability in the swarm, making it the worst algorithm in terms of playback stalls.

- Standard
  The standard algorithm holds global knowledge of data availability in the swarm and will download pieces in a rarity-first fashion. It improves on the linear approach as shown in figure 3.3.

- Libswift
  Libswift has three sets: a low-, medium- and a high-priority set. The high-priority set is the part just after the playback position, and the protocol

requests data in-order as these pieces are considered urgent for video playback. The medium- and low-priority set are downloaded in a rarity-first fashion when no pieces from the high-priority set are needed anymore. This can be done because, just like the standard algorithm, Libswift holds global knowledge about data availability. Because the content just after playback position is downloaded first, it scores best on the average number of playback stalls.

3.4 Conclusion

Concluding, Libswift has the best start-up time, especially in the case of a steady-state scenerio, and the least number of average stalls per playback. This makes Libswift the best choice to implement as transport protocol, it is also developed by the PDS department where the team works, meaning that specialist information can be obtained when needed.

Chapter 4

Video Decoding

4.1 Introduction

This chapter describes different open-source video decoding frameworks that exist for the Android Platform. The architecture of each of the of the frameworks is briefly described, as well as the built-in features. The dependencies of each framework are also briefly analyzed. Finally, a trade-off will be made between these frameworks upon which a framework will be chosen for use in the project.

4.2 VLC

VLC for Android Beta is an open-source multimedia player and framework[1], that is immensely popular and has been downloaded over 10 million times.[2] It currently is in a BETA phase, meaning that it is not completely finished, lacks features, and can still contain unknown issues.

4.2.1 Features

VLC for Android plays most multimedia files both locally and through a network stream. The current version of VLC for Android has support for devices with an ARMv7 CPU and a x86 CPU.

4.2.2 Architecture

Core

VLC is a highly complex modular framework that consists of one central core (LibVLCcore) written in C++. This core manages the threads, loading/unloading modules (codecs, multiplexers, demultiplexers, etc.) and all low-level control in VLC.[3]

[1] http://www.videolan.org
[2] https://play.google.com/store/apps/details?id=org.videolan.vlc.betav7neon&hl=nl
[3] http://wiki.videolan.org/Hacker_Guide/Core/

LibVLC

On top of libVLCcore, a singleton class libVLC acts as a wrapper class, that gives external applications access to all features of the core. Modules on the other hand communicate directly with the core.

Modules

VLC comes with more than 200 modules including various multiplexers, demultiplexers, decoders and filters. These modules are loaded at runtime depending on the necessity. Given the modular nature of VLC's architecture, unnecessary modules can be taken out in order to reduce the footprint of the framework.

Multi-threading

VLC is a multi-threaded framework. One of the main reasons for using multi-threading is used to warrent that an audio or video frame will be played at the exact presentation time without blocking the decoder threads.

4.3 Native Android media player

This chapter describes different open-source video decoding frameworks that exist for the Android Platform. The architecture of each of the of the frameworks is briefly described, as well as the built-in features. The dependencies of each framework are also briefly analyzed. Finally, a trade-off will be made between these frameworks upon which a framework will be chosen for use in the project.

4.4 Dolphin

This chapter describes different open-source video decoding frameworks that exist for the Android Platform. The architecture of each of the of the frameworks is briefly described, as well as the built-in features. The dependencies of each framework are also briefly analyzed. Finally, a trade-off will be made between these frameworks upon which a framework will be chosen for use in the project.

Chapter 5

Risk Analysis

5.1 Introduction

Appendices