Autonomous Operation of Attack-Resilient and Self-Organising Video-On-Demand Platform for Mobile Devices

by

P.W.G. Brussee

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday August 31, 2016 at 3:00 PM.

Student number:	1308025	
Project duration:	December 1, 2015 – August 31, 2016	
Thesis committee:	Assoc. Prof. dr. ir. J.A. Pouwelse,	TU Delft, supervisor
	Prof. dr. C. Witteveen,	TU Delft
	Dr. ir. C.C.S. Liem,	TU Delft

Abstract

The business model of social media directly conflicts with user privacy. Human rights like the protection of privacy, honour and reputation are endangered by this. An anonymous and fully distributed solution is required to solve this. Mobile devices with peer-to-peer communication technology can be used to build a server-less distributed system. The ubiquity of smart phones with the free and open source Android platform is ideal for this purpose. An attack-resilient and self-organizing video-on-demand platform called Tribler exists on desktop platforms but not yet on mobile devices. This report shows a solution by giving an implementation and evaluation of such a platform on an average smart phone. With this implementation all functionality of Tribler is now available on all mobile devices running Android version 4.3 or higher. This is done by enhancing the open source project python-for-android and retooling the entire code base of Tribler and its dependencies. Re-factoring to make Tribler more modular was also necessary to separate the GUI from reusable core functionality and introducing a representational state transfer (REST) API for interprocess communication. The implementation usability in terms of performance and latency is found to be good enough on an average smart phone like the Nexus 5. Based on the analysis and great fit in event based systems, the reactive programming paradigm is recommended for use in the entire current code base, making it fully asynchronous and shift from imperative to the functional programming paradigm.

Preface

I would like to thank my parents, teachers and colleagues. And above all God.

P.W.G. Brussee
Delft, August 2016

Contents

1 Introduction

In the era of social media people on earth are more connected then ever before.

However not every place on earth has an uncensored Internet connection, or has one that can be shut down with the push of a button.

Although smart phones have brought the Internet into the hands of people, this mobile device is not capable of overturning the power of the Internet-kill-switch, yet. This is about to change as the self-organising video-on-demand platform Tribler is going to make the jump to mobile devices. A big part of social media is video sharing and streaming. There is a great interest in this considering the amount of websites and apps available for streaming video-on-demand services. Huge video streaming providers like Youtube, Twitch, Periscope, etc. currently dominate the market. The problem with those is that none of them are server-less and do not provide anonymity in any shape or form.

With servers central to their design they create a single point of failure, even in a decentralized set-up. Several natural disasters have taken out the necessary infrastructure on numerous occasions for a prolonged period of time. Especially in situations like these, people need to communicate and coordinate their efforts to restore safety. Social media has played a major role in recent calamities when people could mark themselves as safe, effectively broadcasting that information to all their family and friends on social media, instead of contacting them one by one or not at all due to congestion in the communication channels. So the advantages are obvious, and the vulnerability of central elements underlying current social media too.

The lack of anonymity becomes a problem when the users privacy is being invaded. Revealing personal information can be deduced from search queries for example, or associations on social platforms. When this information can be used for targeted advertising it becomes very valuable, and creates an incentive for the parties that have access to this information to sell it to third parties. In fact the business model of social media appears to be serving targeted advertisements to its users on behalf of third parties. What's even worse is social media integrated into regular websites to de-anonymize and track the whereabouts of users even outside of the social media realm. Whenever users lose control over their privacy it becomes a serious problem.

1.1. Problem description

The business model of social media directly conflicts with user privacy. Targeted advertising becomes trivial when the user profile is provided by the user itself with accurate and current information. When the information is shared with parties outside of the specific social media website it effectively becomes a privacy leak. Subsequently users will be confronted with their information being misused in various ways, beyond their control. This lack of control over your own privacy is a problem and can lead to arbitrary interference, possibly even unknowingly. The Universal Declaration of Human Rights (UDHR) declares in article 12 that everyone has the right to protection against such interference.

> No one shall be subjected to arbitrary interference with his privacy, family, home or correspondence, nor to attacks upon his honour and reputation. Everyone has the right to the protection of the law against such interference or attacks. (Article 12. UDHR)

Integration of social media on regular websites can easily de-anonymize the visiting user, directly benefiting the business model of targeted advertisements, aggravating this problem. This incentive not only causes

a lack of privacy, but also, paradoxically, a potential lack of freedom of expression. Every interest and opinion expressed on social media and beyond, whether that is by clicking on anything or publishing videos, is traceable to an individual due to this integration and business model. If the controlling party of any website connected to social media wants to silence a certain opinion, it can use this connection to trace the individual and any like minded people connected by social interaction. Particularly on social media do people exercise their right to freedom of opinion and expression. The UDHR declares in article 19 that everyone has the right to do so without any interference whatsoever.

> Everyone has the right to freedom of opinion and expression; this right includes freedom to hold opinions without interference and to seek, receive and impart information and ideas through any media and regardless of frontiers. (Article 19. UDHR)

To ensure that no controlling party can exercise censorship we **distribute authority** over all users, creating an *autonomous* system. If all information is located in one or a few places, the parties in charge of that location will still have control over it, so we must **distribute information** over all users, creating a *communication* system. Then if all users want to use this system to share, order and appreciate each others information, in other words the essence of social media: social interaction, with everyone being able to interact in the same way, we need to **distribute functionality** over all users, creating a *cooperation* system. Fully distributed systems capture these characteristics. Without any central component in the system it is no longer susceptible to censorship without everyone participating.

Peer-to-peer communication technology is essential for a server-less distributed system. Mobile devices typically do not require infrastructure to exchange information, like those equipped with Bluetooth or capable of ad hoc Wi-Fi. Smart phones are ubiquitous everywhere in the world and used to access social media and retrieve information from the Internet. Fortuitously these are also the type of mobile devices that can communicate peer-to-peer.

1.2. Thesis definition

The main question thus becomes: How to create a **self-organising video-on-demand** platform that is **attack-resilient** and can **operate autonomously** on a **mobile device**?

Self-organising in the sense that the platform coordinates the exchange of videos and meta-data fully automatically.

Video-on-demand in the sense that users can simply click and play videos in a streaming fashion, so without waiting for the entire video to be present on the device.

Attack-resilient in the sense that: First: censorship does not have an effect if the majority of users does not cooperate with the censor. Second: the privacy of users remains protected while they actively participate on the platform Third: no network infrastructure required for viral spreading of the entire video platform.

Autonomous operation in the sense that users do not have to manage any files or configuration manually at all to be active on the platform.

Mobile device in the sense that it is low-powered and portable including the network interface and power supply.

These properties will ensure social media with resilience against Internet kill switches, natural disasters and censorship.

1.3. Research Limitations

Software development / technical aspect only Not policy making, organisational perspective, decision making, normative, ethical, Time limit of 9 months

2

Methodology

The Delft Systems Approach, In 't Veld

Explore environment Define objectives

Define alternatives

Confront and tune

Develop and organize

Control and transform??

TOI single disciplinary approach, choosen, part of multi- : Tech, Org, Inf., focus on tech.

2.1. Explore environment

Various initiatives have been started to deal with one or both of these problems. Figure 2.1 shows a mapping of projects that are or have been working on that.

2.1.1. Tribler

Tribler introduces a server-less video-sharing platform with privacy enhancing technologies and giving a Youtube-like, social media experience at the same time. The capability of hiding your identity is greatly advantageous to the user if his or her human rights are violated, like free speech.

The server-less technique of Tribler is resistant to Internet kill-switches that are typically deployed for the purpose of censorship.

Social media on phones

You want to express freedom of expression with that all the time

Existing apps use central server design Vulnarable for Internet kill switches and censorship

Offline viral spreading image (hacking lab)

How I did the work What you have done told by what you did not do. No answer completely, with tests that I did, but what I tested says this.... and makes it reasonable to say probably yes.

Image of what I got in the end and what I got in the beginning next to each other.

Is the main question legitimate?

What is the problem this VOD platform solves?

Regulatory perspective, out of scope Ethical, out of scope

What do we want? VOD What is already there? p4a What is the gap? my work

How can we develop an autonomous and anonymous VOD platform with existing python code base that is ()user friendly /) working on a mobile platform and which is resistant to Internet kill-switches?

autonomous, not dependent on outside stuff user friendly, power draw, special permissions

multi-coin, mobile data, nodes/hops

Imperative programming has limitations Reactive fits mobile paradigm better

Iterations, waterfall? Delft system approach: Systeem -> Functional Design -> alternatives -> choose ->Prototype implementation |||—— validation <————————————————-

KPI 's, speed, availability, ... or App is done Now: multi-criteria analysis of all other apps and mine

What is VOD? What does working on mobile phone mean? What are the existing modules we re-use? What is missing? (gap) What are the possible solutions? Is this prototype a verified solution to main problem desc.?

this is depth first analysis now.

NOT product description

maybe problem description in an image, like system diagram

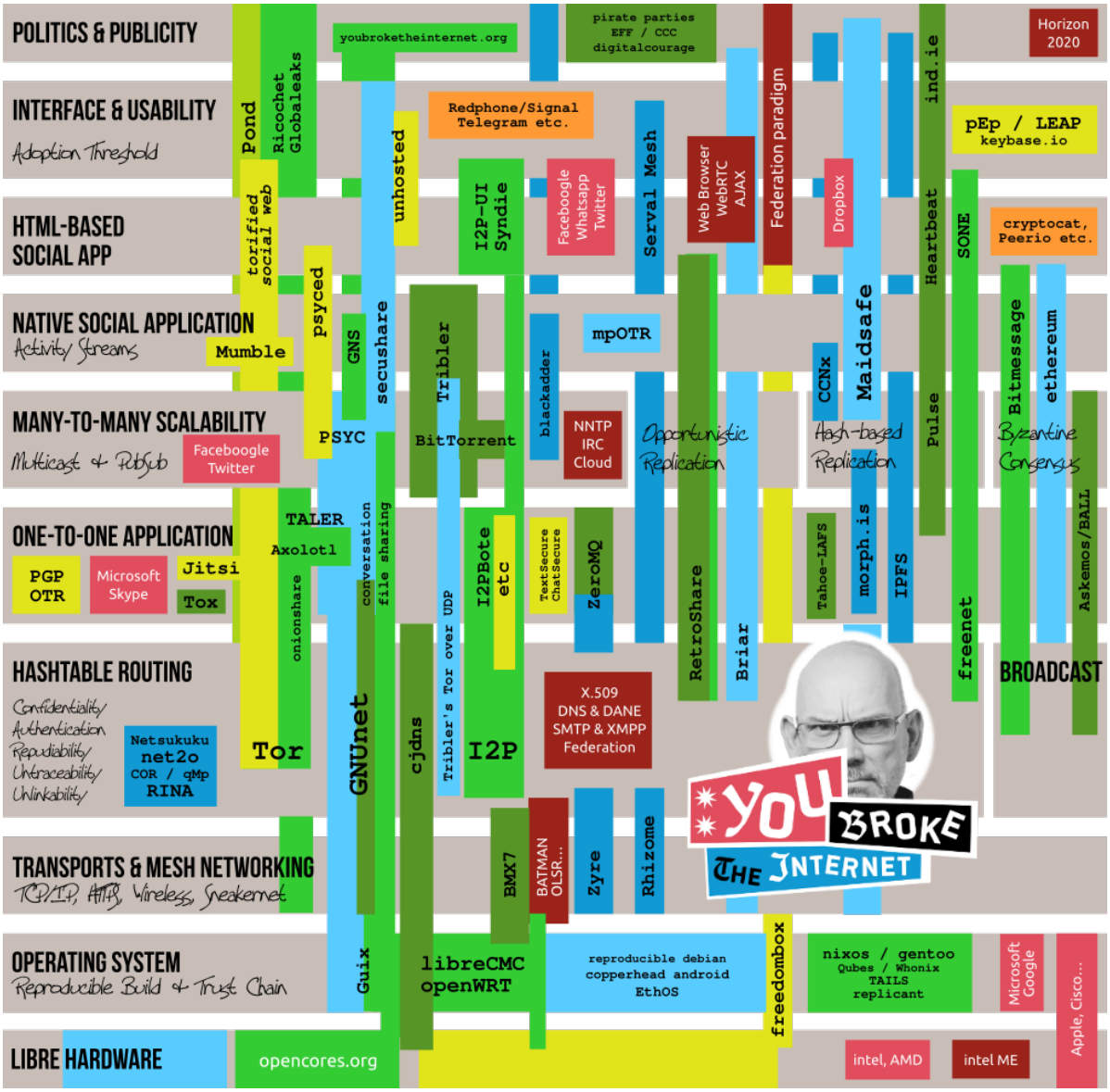

Figure 2.1: Map of projects trying to fix the Internet according to youbroketheinternet.org, last updated October 2015

Colour coding:

Green: Projects that are available today.
Dark green: Projects that are available, but are not fully protective of meta-data.
Blue: Projects in development.
Dark blue: Projects in development which will have little or no protection of meta-data (but that does not mean they can't be an excellent piece in the general puzzle).
Yellow: Projects that may be okay but depend too much on the security of servers.
Orange: Products whose end-to-end encrypting client side has been open-sourced but whose server side remains proprietary.
Red: Brands that currently occupy the respective layers with unsafe technology.
Dark red: Possibly cool but unsafe technologies that we need to replace.

3 Design

knowledge of requirements knowledge of design as created so far knowledge of technology available knowledge of software design principles and best practices knowledge of what has worked well in the past

3.1. Design principles

Top down: high-level issues like software architecture and kind of database, down to details like format of data items and individual algorithms

Bottom up: decide reusable low-level utilities and then how these are put together to create high-level constructs

3.1.1. Divide and conquer

3.1.2. Increase cohesion where possible

functional layer communicational sequential procedural temporal utility

3.1.3. Reduce coupling where possible

content common control stamp data routine call type use inclusion/import external

3.1.4. Keep the level of abstraction as high as possible

3.1.5. Increase re-usability where possible

design for reuse

3.1.6. Reuse existing designs and code where possible

design with reuse

reuse of expertise reuse of standard designs and algorithms reuse of libraries of classes or procedures, of of powerful commands built into languages and operating systems reuse of frameworks reuse of complete applications

3.1.7. Design for flexibility

reduce coupling and increase cohesion create abstractions no hard-coding leave all options open, like exception handling by caller instead of callee use reusable code and make code reusable, like hooks

3.1.8. Anticipate obsolescence

avoid early releases avoid environment dependencies avoid undocumented features or little used features, like libtorrent geo avoid smaller companies/projects due to lack of long term support use standard languages and technologies supported by multiple vendors

3.1.9. Design for portability

3.1.10. Design for testability

avoid static

3.1.11. Design defensively

check all inputs and preconditions design by contract !!! preconditions postconditions invariants -> assertions

3.2. Define objectives

The key performance indicators (KPI) will be *scalability*, *resource usage* of cpu, memory and power, and *usability* in terms of latency.

3.2.1. Accessibility

3.2.2. Availability and resilience

3.2.3. Development resource

3.2.4. Evolution

3.2.5. Internationalization

3.2.6. Location

3.2.7. Performance and scalability

Just like the current server-centric design of social media A fully distributed system needs to scale to potentially all smart phones in the world, just like the current architecture of social media with central server locations.

Resource usage

3.2.8. Regulation

3.2.9. Security

3.2.10. Usability

3.3. Define alternatives

Why no iOS? No routing, no blutooth p2p, mesh networking: that is done by Serval. Out of scope of this work. Add as enhancement later.

3.4. Architecture design

better understanding individual pieces worked on in isolation prepare for extension facilitate reuse and reusability

logical breakdown into subsystems, package diagrams dynamics of interaction among components at run time data shared among subsystems components existing at runtime and machines/devices on which they are located

Architectural patterns:

3.4.1. Client-Server

REST API

3.4.2. Broker

objects are actually remote

3.4.3. Transaction processing

atomicity

3.4.4. Pipe-and-Filter

Rx data flow pipeline

3.4.5. Model-View-Controller

model: JSON de-serialized objects view: xml layout controller: activity, fragment, adapter

3.4.6. Service-Oriented
app is collection of services that communicate with each other through well defined interfaces http request behind the scenes JSON REST API videoserver

3.4.7. Message-Oriented
message-oriented middle-ware (MOM) communicating apps do not have to be available at the same time virtual channels, topics publishers, broadcasters subscribe to topic android intents, intent filters, broadcasts, bundle, messages, handlers

3.5. Class design
3.6. User interface design
3.7. Database design
3.8. Algorithm design
3.9. Protocol design
3.10. Requirements
3.10.1. Functional requirements
inputs, commands and conditions outputs, conditions store, reuse data, backup computations timing and synchronization

3.10.2. Quality requirements
response time throughput resource usage reliability availability recover from failure maintainability and enhancement re-usability

3.10.3. Platform requirements
computing platform: min. system specs and features, api level technology to be used: programming language, db

3.10.4. Process requirements
development process / methodology cost delivery date

Top/System level: common core, user friendly, not vulnerable to kill switches and censorship, ...
Functional level
Design level
Prototype level
Alternatives: choices, decision process know what you don't know Aware of what you did not do.
describe: chosen based on time investment, other P4A only app OR java + p4a app imperative OR reactive

Common Core
10 years of Python code reusable multi-platform: Windows Mac Linux

Native Android app for Tribler with Java+xml GUI. Using https://github.com/kivy/python-for-android to run Tribler python code as a native Android service. The app communicates with the python service via the new rest api.

Picture of the technology stack and all components.

4

Architecture

Intro per chapter, refer to problem definition each time, where are we now, what now (for reader) Why this chapter Paragraphs Conclusion of contents of this chapter Link to next chapter

4.1. Confront and tune

What can go wrong?

Viral spreading Multi user view Autonomous region Spreading app Single leak to Internet Solution to problem description

Show complete technology stack

5 Implementation

5.1. Develop and organize

Verkopen: app build with reactive programming paradigm, Rx founded by TU Delft professor
code example modular functional programming

5.2. Tool-chain architecture

we transfered the first generation bash script into a more involved python tool chain ecosystem the p4a project accepted our first contribution the same day our approach for this .. we focused on the low hanging fruit first to benefit from the learning effect the first task consisted of porting the python bindings for 15 - 30+ commonly used libraries to Android [recipe example code block]

5.2.1. Python-for-android

Open source project p4a uses python toolchain with bootstraps, recipes, ant, make, Android SDK (aapt), Android NDK, zipalign, ..., setuptools, ...

5.2.2. Gradle experimental

Use Android SDK, jni, Android NDK, ...

5.3. System architecture

table of lines of code modules app

5.3.1. Python on Android

Open source project Python-for-Android is capable of running python code.
no shell acces from Popen() means only single binaries instead of normal commands
My contributions:

- add service only bootstrap with templates
- recipe (missing) for all dependencies
- bug fixes in build tool chain
- service interface binder

5.3.2. Rest API

My contributions:

- create channel
- create torrent

- add torrent to channel
- async download and add torrent to channel from url and magnet links
- remote shutdown

5.3.3. Common python core

My contributions:

- correct paths for Android
- setup.py rewrite
- improve error handling in test runner
- various threading bug fixes

5.3.4. Native Android GUI

Gui for common core back-end and rest api in between.

My contributions:

- Native service wrapper for Python process
- GUI:
- create own channel
- record video
- create torrent file
- add torrent to own channel
- search torrents and channels
- voice search integration
- view channel contents
- download torrent
- launch video viewer
- mark and un-mark channel as favorite

6 Performance Analysis

Performance experiments:

6.1. Startup time
Startup experience

6.2. GUI Responose Time
Latency during first hour, after first hour What does it say about the design?

6.3. Content Discovery Performance
First discovered content, latency, cdf

6.4. Search Performance
Search results response time, latency, cdf

6.5. Typical System Load
y: cpu usage x: time 0-3 uur

Graph showing cpu and memory consumption from fresh install to an hour idling. 0 tunnels.

Graph showing cpu and memory consumption with max. 10 tunnels after an hour idling.

Graph showing cpu and memory consumption during single download after an hour idling. 0 tunnels.

Graph showing cpu and memory consumption during streaming HD video after an hour idling. 0 tunnels.

stacking of cost, turn all off, turn on one by one or peel off some functionality, look at resulting workload / impact on performace / give relative cost of component

6.6. CProfiler Analysis
Graph showing wall clock time spend on functions running 10 minutes during first half hour with max. 10 tunnels. flame graph

6.7. Test Suite Performance
Graph showing total run time and average run time per test.

Code coverage?

6.8. Multi-chain Scalability Experiment
Multichain record creation cost Graph showing scalability or lack thereof of the multi-chain record creation cost.

6.9. Phone-to-Phone new content discover time
6.10. Transfer time of app
6.11. DAS5 1 to 1000000

7

Conclusions and Future Work

app needs wifi access points

integreren met self compile, stealth app, thumbnail navigation, no live streaming still periscope central server

gui socked shared with network ? if BAD else

Bibliography