

Ripple - The future of local social networking

Distributed Systems – Project Proposal

Carl Friess, Isaak Hanemann, Sven Knobloch, Laurin Paech, Sebastian Winberg, David Yenicelik
cfriess 15-943-111, isaakh 15-913-312, knsven 14-945-166, lpaech 15-944-242, winbergs 15-941-222, yedavid
15-944-366
cfriess@student.ethz.ch, isaakh@student.ethz.ch, knsven@student.ethz.ch, lpaech@student.ethz.ch,
winbergs@student.ethz.ch, yedavid@student.ethz.ch

ABSTRACT

We propose a novel social networking application for sharing photos in close physical proximity. A user can take a picture, which is then broadcasted to all phones that are part of the network and in the direct vicinity.

To increase speed and reduce operating costs, the Android application does not solely rely on high-bandwidth connections with a centralized server, but rather uses a distributed peer to peer network to transmit photos between individual nodes. Our proposed app utilizes a combination of the Bluetooth Low Energy and BitTorrent protocols, as well as a custom RESTful HTTP API.

The fundamental transfer of image files is done using BitTorrent. A central tracker will facilitate the discovery of nodes in accordance with the BitTorrent protocol. A receiving node will then download the image from the initial node and from other nodes, which have already completed the download. The tracker will use a JavaScript BitTorrent implementation [[Link]] executed in a node.js environment hosted on a AWS EC2 instance.

Devices use the Bluetooth Low Energy Protocol for close proximity device discovery and for transmitting identifying information of the broadcasted images.

A centralized server further allows devices to retrieve torrent files, which are necessary for the BitTorrent protocol.

Our goal is to have a fully-working prototype with the proposed architecture, where a single node delivers to another node, and where a node can ping other nodes if it is 'missing' a photo. [[??]] All participating, delivering and receiving nodes are determined by the tracker.

1. INTRODUCTION

We first introduce a few definitions.

1. **ripple (noun):** A photo that is being shared with anyone nearby from a central person's smartphone. We will use the words 'ripple' and 'photo' interchangeably.
2. **to ripple (verb):** When person A shares a photo (the ripple), and person B decides to share it again, the photo has 'rippled' once as it propagated over one person since the initial share.
3. **tracker:** A central server handling the distribution logic
4. **AWS:** Amazon Web Services. The cloud, which will act as the tracker.
5. **P2P:** Peer to peer
6. **n:** The number of nodes in observed network. In our case, these nodes will be android devices.

1.1 Problem statement

There have been numerous social networking applications that connect people over a range of topics. Such services include Instagram and Snapchat, where a person shares his

or her moments with a local group of people, namely their friends. However, none of these services share their appreciated moments with people nearby (measured by geographical proximity). As such, we propose Ripple: a local social networking app where people share their appreciated moments with anyone near them.

1.2 Application scenario

We propose a novel Android application and demonstrate its application scenario by first defining a group of people, namely *Alice, Bob, Charlie, Daniel*. Alice sees a human rights demonstration, and captures a symbolic and emotionally photo of a police officer shouting at a demonstrator. Bob, who is home but nearby the demonstrations receives this photo and deeply appreciates this moment. He decides to share this photo again. Due to the rules of our network, this picture is shared with anyone close to Bob. Because Charlie and Daniel also have the Ripple app installed, and are physically close enough to Bob, Charlie and Daniel also receive this photo. They again have the opportunity to further share this photo, or simply 'drop' it and block further 'propagation' of this photo.

1.3 Motivation

The idea behind the above described mechanism is that the mass forms a selective filter over what content to share. If a ripple is 'good', it has the chance to be propagated over all ripple users, as the photo will be propagated repeatedly. This allows the entire userbase to experience pictures that are meaningful to the user, for the reason that the receiver is 'in proximity' to the event. More specifically, if the first share indicates appreciation of the picture, and each additional share indicates consistence in this appreciation, then - by induction - everyone feels appreciation of this picture. Generally, the more important the event is, the more 'close' a person feels to this to this event. If, however, a photo is of bad quality or is not appreciated, local users will quickly stop the photo to be propagated, forming a natural blockade for any non-appreciated pictures.

1.4 Challenges

This project has two main challenges:

1. Find users in close proximity
2. Send the image data

For the first challenge of user discovery we investigated using GPS data to track the user location. Since GPS has a fairly weak accuracy with uncertainties as high as a few hundred meters when inside a building, we couldn't achieve sufficient locality using this technology. We therefore turned our focus to Bluetooth Low Energy. The range of 10-20 meters is optimal considering the desired behavior and less power consuming than GPS. Furthermore Bluetooth Low Energy provides a well established protocol with wide-spread hardware compatibility.

We considered the following aspects for the second challenge of sending the image data. It became clear early on

that Bluetooth Low Energy would not have high enough data rates to send an entire image from one device to another in a reasonable time frame and acceptable reliability. For this reason we first looked into a centralized structure (using Amazon S3) where images are stored with a UUID and the UUID is sent to the receiver over Bluetooth. The image can then be retrieved from the central server using the UUID.

While this implementation works, storage and bandwidth costs especially for a later support of video files would be fairly high. With the intentions of shifting most of the load to user devices and not storing the image data on a central server we looked into peer to peer technology. We considered the use of IPFS but decided to use the BitTorrent file sharing protocol since it is well established and commonly used with a large amount of software support. **[[java library]]**

With BitTorrent the devices can distribute the content between themselves. However, this requires a torrent file, leading to the question of how to send this file to other devices. Our first idea was to send the torrent file over Bluetooth Low Energy. But since torrent files can have a significant size themselves and BLE only allows for low data rates, we chose to only pass an identifier over Bluetooth and retrieve the torrent file from the central server using a custom RESTful HTTP API. Since one torrent file is only a few kilobytes big, we have significantly lowered the cost by several orders of magnitude.

2. SYSTEM OVERVIEW

2.1 System Architecture

We describe the distributed system architecture with regards to the above presented challenges.

We intend to solve 1.) by a *P2P* (peer to peer) network which makes a central server redundant. Instead of downloading all photos from a central server, each phone is a node and independently transmits its captured image directly to its close neighbours. It also has the benefit that we remove any single point of failure within the system, as there are multiple possible paths to a receiver from a sender in the network of phone-nodes (assuming multiple people are in physical proximity to the sender and the receiver). We must handle some logic tasks though, for which we use a central server through *AWS* (Amazon Web Services), which decides which phone should receive which photos, and which other nodes to contact for this data. It is important to emphasize that there are no photos saved on the server, and as such, no major load to be handled uploading and downloading. This effectively solves the problem of data-congestion in a single point, and reduces points of failures.

Challenge 3.) can be solved by using dynamic algorithms that work on the position of a user. However, for this project, we assume that we are in a network where more than 5 people are in physical proximity, making any technology a viable option for distribution of the photos.

P2P technology has been used extensively in torrenting applications, and has proven to work reliably and efficiently. As such, we assume that it proposes a highly-scalable technology with the ability to be rapidly deployable and efficient-to-run with regards to our application scenario. Our proposed system architecture is a well-established solution in a novel setting. Conventional *P2P* don't use mobile devices but stationary, reliably and well-connected agents. Our system architecture is simple, as we will use one central **tracker**, which will handle the distribution and communication logic of the photos and nodes, and n number of **nodes**, which will form the phones that can upload and receive the pictures.

2.2 Components

Given the above system architecture, we have two components.

1. **Tracker:** Handles the distribution logic of the photos, without storing the pictures. Only IP, location, permissions etc. are uploaded and downloaded from the server, allowing this to have a low worst-case and average load.
2. **Nodes:** Are the phones that receive the photos. These nodes communicate with each other using over simple internet protocols, and are responsible for sharing the files. As such, the load is taken away from the cloud. This also makes the network more reliable and fast. There is no single point of failure, and one phone can receive small data-packets (which combined forms a picture) from multiple other phones that can provide the data packets.

2.3 Distributed system components and their interaction

[[[WRITE THIS SHIT]]] Think about possible additional technical details. Think about problems that might arise during development.

Difficulties are:

1. Partial data-package-delivery

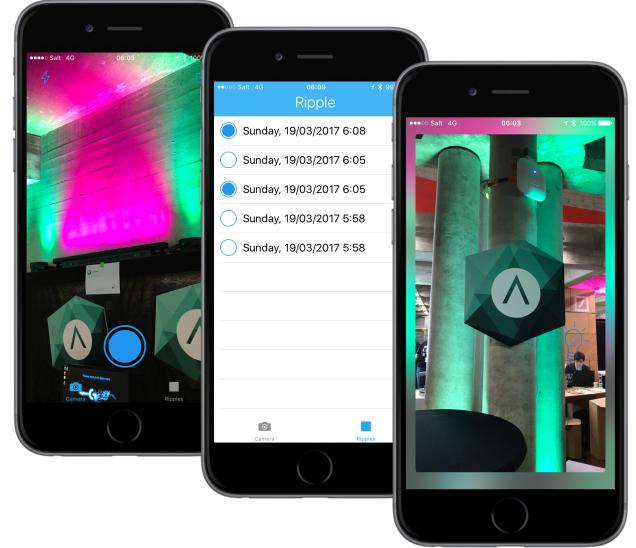


Figure 1: What our final product could look like. Includes a view to take photos (left), a list of all received and opened photos (middle), and a screenshot of what a photo looks like right after when it's taken and before it's shared (right)

3. REQUIREMENTS

3.1 Hardware

Our proposed app is a social networking app, relying on no additional hardware but the android device at hand.

3.2 Software (libraries and frameworks)

[[[WRITE THIS SHIT]]] We plan on using existing *P2P* sharing networks, such as **ttorrent** [2]. This library is a java framework that allows for all clients to register to a *P2P* network, and also allows a tracker to manage all clients in the network. The clients have following functions in the framework: The tracker has following functions in the framework:

3.3 3rd party services

We also intend on using Amazon AWS EC2 as a tracker, allowing the individual nodes to ping for all photos that should be delivered, and for the senders to actually deliver them. This instance has free plans for low usage-tiers, and as such, is sufficient for our needs.

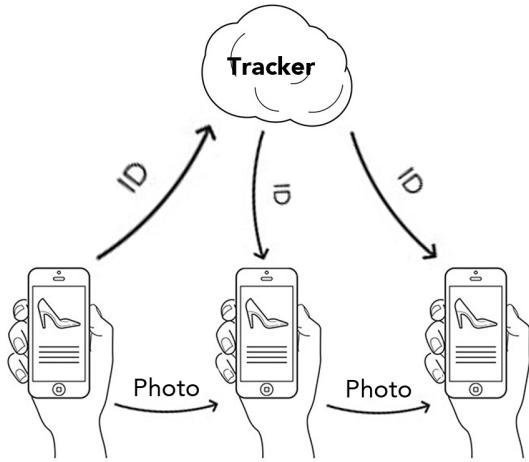


Figure 2: System Overview [1]

4. WORK PACKAGES

We figured out that working in the past group structures allowed developers to be flexible but also effective in their work. As such, we define broad tasks with specific goals. In the following, we present tasks, that when done, result in a robust and well-working app.

- **WP1:** Build an Android front-end that can take pictures and temporarily saves it in a local storage.
- **WP2:** Build an Android back-end which acts as a node. It should be able to transfer files from one device to another, and save it to the local storage.
- **WP3:** Build the transition between WP1 and WP2 such that all transmitted files are displayed on the receiver using the front-end.
- **WP4:** Initialize and configure backend server which is supposed to serve as a tracker.
- **WP5:** Create all API endpoints which accept device ID's and picture-identifiers, and checks which photos are delivered to which devices. The uid can act as an unique identifier for each node.
- **WP6:** Implement P2P logic in the node backend that pings other phones for a photo, if a photo was not yet received (which is determined by the received photo id's).
- **WP7:** Extensive usage testing and debugging.

5. MILESTONES

5.1 Schedule

We form a temporal schedule on what tasks should be achieved, sorted by upcoming dates.

1. **24. Nov. 2017:** We plan on finishing the work packages that are considered part of the app, namely *WP1* and *WP2*.
2. **1. Dec. 2017:** We intend to finish setting up the central tracker, and all it's API functions. These are part of *WP3* and *WP4*.
3. **8. Dec. 2017:** We plan to do a naive implementation of the entire system, where the tracker communicates with the phones the photos to be downloaded, and the photos are communicated between the individual nodes. This is concluded by *WP5* and *WP6* 'putting the components together'.
4. **15. Dec. 2017:** Assuming all went as planned, we would try to test the network on a multi-node connection ($n < 3$), identifying bugs and generally debugging the system. We would possibly try to identify network congestion behaviour and try to optimize these (*WP7*).

5.2 Assignment to team members

Luckily, all tasks are extensive enough that a majority of team members can work on the individual working packages individually. As such, for each deadline (*24. Nov. 2017*, *1. Dec. 2017*, *8. Dec. 2017*, *15. Dec. 2017*), we plan to split the team into two sub-sections, each of which handles one working package. Within each working package, we intend to have a flexible schedule amongst team-members, as this has proven to be effective in the past few Android projects. The last part (*WP7*) is not well-predictable as of now. Our previous experience has shown us that collectively listing issues and working on these issues one-by-one has proven to be an effective way to debug and optimize our program.

6. REFERENCES

- [1] Estimote. <http://estimote.com/>. Accessed on 26 Oct 2015.
- [2] Ttorrent, a Java implementation of the BitTorrent protocol. <https://github.com/mpetazzoni/ttorrent>. Accessed on 15 Nov 2017.