Ripple BT - The future of local social networking

Distributed Systems – Project Proposal

Carl Friess, Isaak Hanimann, 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 social networking application for sharing photos in close physical proximity, utilizing a combination of the Bluetooth Low Energy and BitTorrent protocols, as well as a custom RESTful HTTP API.

A user can take a picture, which is then broadcasted to all phones that are in the direct vicinity and part of the network.

The application use the Bluetooth Low Energy Protocol for close proximity device discovery and messaging. However, the fundamental transfer of image files leverages peer-to-peer technology to reduce operational costs and increase performance.

Every device runs a BitTorrent client, which seeds images it has already downloaded to other nodes within the network. The discovery of seeding devices is provided by a central BitTorrent tracker.

Another centralized server further allows devices to share torrent files, enabling the use of the BitTorrent protocol.

Our goal is to build a fully-working prototype with the proposed architecture.

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 and Motivation

There are numerous social networks, such as Instagram and Snapchat, allowing users to share moments with a specific group of people. These services are often criticized for removing the personal element from social communication, because users are sitting behind screens or using their phones, rather than actually spending time together.

Our approach to social networking is to augment social interaction, rather than redefining and possibly inhibiting it. Social Networks should help form connections between people and not just build on existing ones.

1.2 Concept

Considering the above problem statement, we propose Ripple: a social network allowing users to share moments with other users in their close proximity. The social network will consist of a freely available Android application with a focus on photos.

After a user takes a picture it is broadcasted to all other users of the app in their immediate vicinity, regardless of their affiliation with the broadcasting user. Communication is in principle anonymous (disregarding content) and always public.

Users receiving brodcasted images may view them once. They are then presented with the choice of either simply discarding the image or instead rebroadcasting ('re-rippling') it to other users in their proximity. Each user can view the ripple only once regardless of the number of re-ripples near them.

We believe this social network will encourage social interaction, as the communication can only take place when users are close to each other. Furthermore, the network allows interesting content to 'propagate', making it possible to share memorable moments in a wider radius.

1.3 Application Scenario

We will demonstrate a typical use case of the Android application in a group of people attending an event: *Alice*, *Bob*, *Charlie* and *Daniel*.

Alice takes a funny photo of a particular activity at the event and wants to raise awareness of the activity with other attendees. She therefore uses Ripple to share the image.

Bob, who is not far away receives the ripple and is intrigued by the picture. He thinks others might share his intrigue and re-ripples the image.

Charlie and Daniel are in close proximity of Bob but not of Alice. As Bob is also broadcasting the image and they have the Ripple app installed, they now also receive the ripple.

Following this scheme the ripple spreads out and propagates, maybe even beyond the event's location.

1.4 Background

This concept is related to an iOS application developed at the START Hack hackathon in March 2017. **[[Insert link to Devpost]]** It is a basic implementation of this concept using Amazon S3 to store images, so they can be downloaded by the receiving devices. Bluetooth Low Energy is used to advertise images by setting the value of a well known GATT characteristic to the UUID of the image in the S3 bucket.

While this implementation works, it has several flaws. Firstly, uploading every image to an Amazon S3 bucket means that operation costs for storage are bound to be high without heavy compression, leading to low quality content.

Similarly, repeatedly downloading the image from one central point, causes high bandwidth consumption and possibly unnecessary data transfer costs.

Beyond this, the method for announcing ripples is also suboptimal. It does not work well, when a user would like to broadcast multiple ripples in quick succession and (on

iOS) does not fully work when the application is in the background.

With Ripple BT we aim to redesign the system from the ground up with a focus on spreading the load of sending images among user devices, thus reducing operating costs and improving performance.

Figure 1: Screenshots from the original Ripple application. Includes a view to take photos (left), a list of all received photos (middle), and an image viewer for received ripples (right)

1.5 Challenges

This project has two main distributed challenges:

1. Finding users in close proximity
2. Sending 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

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 form the initial node and from other nodes, which have already completed the download.

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

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

2.2 BitTorrent File Transfer

Image files, which make up the largest portion of the data traffic, are transferred between devices peer-to-peer using the BitTorrent protocol.

We will use the ttorrent library**[[LINK]]**, a Java implementation of the BitTorrent protocol. With the help of this library, every device will run a BitTorrent client in an Android Service**[[LINK]]**.

Each phone seeds all images it creates and receives for 24 hours. This way many seeding devices should become available quickly allowing for the load to be shared effectively. Furthermore, there is no need for any device or server to store the full image file indefinitely.

The BitTorrent protocol relies on a central tracker to let clients know which other clients are seeding the file. We chose to use the JavaScript bittorrent-tracker library **[[LINK]]** to achieve this functionality. The tracker will run on an Amazon EC2 **[[LINK]]** instance.

2.3 Distribution of Torrent Files

One of the challenges that we face is the limited data rates of the Bluetooth Low Energy protocol.

This leads to the issue that we can't transmit torrent files via Bluetooth Low Energy. Instead we have decided to use a distribution server, which associates torrent files with UUIDs.

Before broadcasting a new Ripple to other devices, the application uploads the torrent file for the image to the distribution server by means of a HTTP POST request. The server stored the torrent file in a database and returns a UUID.

To retrieve a torrent file, the application submits a HTTP GET request containing the UUID. The server verifies the UUID by looking it up in a database and returns the corresponding torrent file. The application can then use it to

invoke the BitTorrent protocol and download the associated image file.

2.4 Bluetooth Low Energy

For the advertisement of Ripples we will use the Bluetooth Low Energy protocol as follows:

Each device will have a well known GATT service for Ripples. In this service there is a well known characteristic that stores a sequence number. In addition, the service will have a characteristic for each Ripple it is currently advertising. The UUID of that characteristic is used to query the torrent file from the central server.

The application is constantly scanning for devices with the Ripple GATT service and connects to them automatically.

If a device wants to send/broadcast a Ripple, it adds a characteristic with the respective UUID to the Ripple GATT service and increments the value of the sequence number characteristic. This will notify connected devices of potential changes and the existence of a new Ripple. The receiving device will then scan for new characteristics and proceed to retrieve the image for the Ripple.

2.5 Local SQLite Database

The application needs to track information about the ripples it creates and receives. Primarily, the application concept only allows users to receive and view ripples once, so the application must be able to check for duplicates. Furthermore, the embedded BitTorrent client should only seed ripples for 24h, which will be ensured by storing the time frame in a database. We have chosen to use an SQLite database for this task, as it provides all necessary features with good performance, while being easily embeddable in the application.

3. REQUIREMENTS

3.1 Hardware

As this is a social networking application, we want to ensure that there are no hardware dependencies beyond a typical Android smartphone.

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.

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.

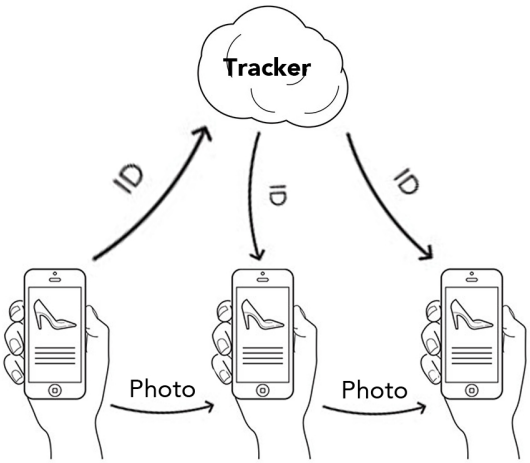

Figure 2: System Overview [1]

- **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 UUID 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.