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

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 accuarcy with uncertanties 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 retreived 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 librabry]]**

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 retreive 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 it's captured image directly to it's 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 emphasis 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. We use a tracker instead of distributed hash table (DHT) to improve the speed of peer discovery and lower the overhead.

2.2 Distribution of Torrent Files

One of the challenges that we face is the limited data rates using BLE. That leads to the issue that we can't distribute the torrent files via BLE but instead we use a Server that distributes the torrent files to nodes that request them by sending a UUID of a picture.
Node A sends the UUID to Node B after uploading it to the server and notifying the tracker of a new file. Node B requests the torrent file by sending the UUID to the server. The server verifies the UUID by look up in a database, retrieving the corresponding torrent and sending it to Node B. Node B can now use the torrent file to download the picture over the BitTorrent Protocol by contacting the Tracker, which provides a list of seeds.

2.3 Bluetooth Low Energy

For the Bluetooth Low Energy we have come up with the following protocol: Each device will have a well known GATT service for Ripples. In this service there is a well known characterisitic 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 for querying the torrent file from the central server.

The device 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 existance of a new Ripple. The receiving device will then scan for new characteristics and continue to retreiving the image for the Ripple.

2.4 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.

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.

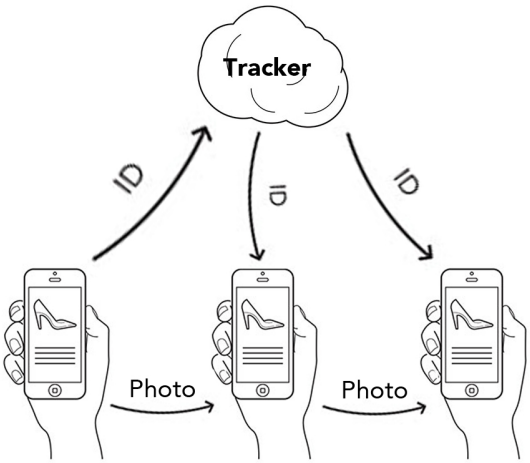

Figure 2: System Overview [1]

4. WORK PACKAGES

We identified and sectioned the project in the following work packages:

- **WP1 - BitTorrent Client**: Implementing the management of the embedded BitTorrent Client.
- **WP2 - Distribution Server**: Implementing a RESTful API for torrent file distribution.
- **WP3 - Camera UI**: Implementing the camera and photo capture functionality of the app using the Android API.
- **WP4 - Receive UI**: Implementing the receive stack of Ripples including the reripple and discard functionality.
- **WP5 - BLE Sending**: Implementing the sending/broadcasting of Ripples via Bluetooth Low Energy.
- **WP6 - BLE Receiving**: Implementing the receiving and scanning of Ripples via Bluetooth Low Energy.
- **WP7 - Local Database**: Implementing an interface to a local SQLite database for the storage of send and received Ripples.

Each working package is intended to be done by an individual team member. Due to the dependencies of some of the packages it is necessary for certain group members to work together. Since some work packages are more extensive than others, team members are encouraged to help their fellow team mates when done with their own part. In the later stages of the project more and more interaction will be required in order to have a working and coherent application for the final submission.

5. MILESTONES

We plan to uphold the following schedule:

- **24. Nov. 2017**: First working versions of the individual work packages
- **4. Dec. 2017**: Integrated all work packages into one coherent app

- **8. Dec. 2017**: Testing application on multiple nodes, identifying bugs and generally debugging the system
- **14. Dec. 2017**: Finished presentation and Ripple logo.
- **15. Dec. 2017**: Deadline for submission of presentation slides and project logo.
- **16. Dec. 2017**: Finished debugging entire system and fully working version of the app.
- **17. Dec. 2017**: Deadline for submission of code.
- **18. Dec. 2017**: Presentation and demo of project.

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.