Ripple - The future of local social networking

Distributed Systems – Project Proposal

Laurin Paech, Sven Knobloch, David Yenicelik
ETH ID-2 XX-XXX-XXX, ETH ID-3 XX-XXX-XXX, yedavid 15-944-366
one@student.ethz.ch, two@student.ethz.ch, three@student.ethz.ch

ABSTRACT

We propose a novel social networking application where photos are shared with neighboring devices. The Android app does not rely on high-bandwidth connections from a centralized server, but rather uses a distributed network and bluetooth technology to transmit photos between individual nodes.
Our proposed app uses bluetooth to transmit photos between nodes. These transmissions are tracked through a centralized Amazon AWS server which helps coordinate transmissions if photos are not yet sent or received. This tracker acts as a coordinator, but does not rely on a high-bandwith and high-load connection. Through this architecture, we also allow the network to be self-reliant, and use the tracker only as a helping entity which only intervenes when mistakes in transmission have been made. 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 (as determined by the tracker). We possibly want to extend this to a many-to-one connection to transmit files, if the time allows us to do so.

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. AWS: Amazon Web Services (the cloud)

4. P2P: Peer to peer

5. BLE: Bluetooth low-energy

6. n: The number of nodes / telephones in observed network

7. tracker: A central server handling the distribution logic

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 another local group of people determined by geographical proximity. As such, we propose Ripple as a local social networking app where people can share their appreciated moments with anyone near them.

1.2 Application scenario

We propose a novel Android application and demostrate it's application scenario by first defining a group of people, namely *Alice*, *Bob*, *Charlie*, *Daniel*. Alice sees a demonstration, and captures a symbolic and emotionally photo of a police officer shouting at demonstrator. Bob, who is at his home but nearby the demonstrations, receives this photo and deeply appreicates this moment. He decides to share this photo again, with anyone near 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 behing the above descriped 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 over the entire planet, as (by everyone propagating the ripple on) the photo will reach all ripple users. This allows the entire userbase to experience moments, that are meaningful to the user, as he is 'in proximity' to the event. And the more important the event is, the more 'close' a person is to this event. If, however, a photo is of bad quality, local users will quickly stop the photo to be propagated, forming a natural blockade for any low quality photos.

1.4 Challenges

This task proposes a challenge because 1.) we work with a theoretically exponentially increasing number of users receiving a single ripple, and 2.) because our userbase is spread across the world. Furthermore, 3.) we must find a dynamic way for users to receive enough ripples to be engaged, but at the same time not to be spammed with photos.
We intend to solve 1.) and 2.) by a *P2P* (peer to peer) network which takes over the tasks which a central server would need to do. As such, instead of download all photos from a central server, each phone is a node, and transmits it's captured image directly to it's neighbors. This saves both space and time. 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. 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. 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.

Challenge 3.) Can easily 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 bluetooth-receivable range, making bluetooth a viable option for distirbution of the photos. P2P technology has been used in torrenting applications for a long time, and has proven to work well and reliably. As such, we assume that it proposes a highly-scalable technol-

ogy with the ability to be rapidly deployable and efficient-to-run with regards to our setting.

2. SYSTEM OVERVIEW

2.1 System Architecture

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 logic of the photos, and n number of **nodes**, which will form the phones that can receive the photos.

2.2 Components

Given the above system architecture, we have two components.

1. **Tracker:** Handles the distribution logic of the photos, but does not store and photos. As such, only location, permissions etc. are uploaded and downloaded from the server, allowing this to have an average low load at all times.

2. **Nodes:** Are the phones that receive the photos. These nodes communicate with each other using bluetooth, 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, as there is no single point of failure, and one phone can receive data-packets (forming a photo) from multiple other phones that can provide it with.

2.3 Distributed system components and their interaction

Think about possible additional technical details. Think about problems that might arise during development.
Difficulties are:

1. Partial data-package-delivery

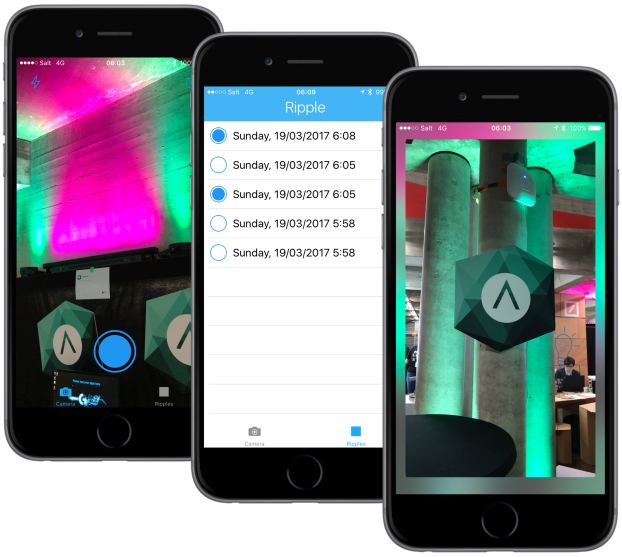

Figure 1: What our final product could look like. Includes a view to take photos (Left), a list of all received and opened photos (middle), and a screenshot of what a photo looks like right after when it's taken and before it's shared (right)

3. REQUIREMENTS

Describe system setup, components, external libraries, hardware etc.

3.1 Hardware

Our proposed app is a social networking app, relying on no additional hardware.

3.2 Software (libraries and frameworks)

We plan on implementing our own P2P logic, as most existing solutions build P2P technology that heavily relies on web, and not bluetooth. In addition to that, our proposed architecture introduces a rather centralized element, *the Tracker*, which has a vastly different role than a node, and as such, calls to a different underlying P2P technology. However, if suitable, we will try to implement

3.3 3rd party services

We will use bluetooth technology to transmit photos between the individual nodes. We also intend on using Amazon AWS EC2 to use as a tracker, allowing the individual nodes to ping for all photos that should be delivered, and for the senders to actually deliver them.

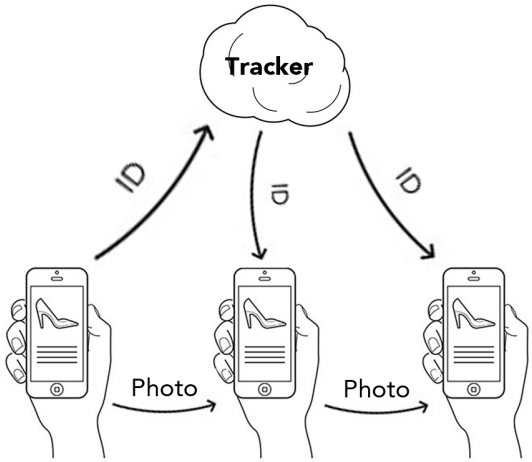

Figure 2: System Overview [?]

4. WORK PACKAGES

Breakdown the work to subtasks to meet the project requirements. Define and describe these tasks.

- **WP1**: Build an Android front-end that can take pictures and temporally 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 using bluetooth, and save it with the local storage.
- **WP3**: Connect WP1 and WP2 such that all transmitted files are displayed on the receiver using the front-end.
- **WP4**: Initialize and configure backend server.

- **WP5**: Create all API endpoints which accept ID's, and checks which photos are delivered to which devices. The uuid can act as an unique identifier for each node.
- **WP6**: Implement 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).
- **Optional WP7**: The final part would be to change the data-transmission through packages, instead of using a one-to-one connection, where only a single node communicates with another node.

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 be able to implement some more efficient partial-data-delivery system as modern system like bittorent do. If this is not possible, however, we plan on using this time as a buffer, finishing any tasks from the previous weeks that are left undone. Assuming we have the time, we would continue with *Optional 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 package 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.