WiFi Direct Message Flooding API

Distributed Systems – Project Proposal

Student One, Student Two, Student Three
ETH ID-1 XX-XXX-XXX, ETH ID-2 XX-XXX-XXX, ETH ID-3 XX-XXX-XXX
one@student.ethz.ch, two@student.ethz.ch, three@student.ethz.ch

ABSTRACT

1. INTRODUCTION

Our message flooding API can be useful to many future projects that involve several Android devices which should be connected even without a working internet connection. For some applications, the API might simply provide an alternetive communication channel that can be used when the device does not have a connection to the internet, but for other applications it can be the core of the communication between several devices.
A simple example application will be distributed along with the API as a demo. The demo is an SOS forwarding app that uses our API to propagate an emergency call between devices which are not connected with the internet, until it reaches a device with a working internet connection that can send the call to a webserver.
Of course the full power of the API will only be visible in more complex systems. In principle, the API will be powerful enough to support a document editor which is synchronized over many users, all without the need of a working internet connection. That could be interesting for a military office outside, but also for a working team that wants to keep working on the same files while travelling together in an airplane.
To demonstrate how the API is used for more complex applications, we will develop a messenger app. The app will support multiple secure chats that users can join.
As the name suggest, the API provides nothing but a message flooding interface, therefore most of the copmplexitiy will be in the client's code outside of the API, namely in the client's application. However, the API solves most of the problems of a ditributed systems and hides them from the client. The features available in the API are:

- **Dynamic local network**: Devices can form a local network and new devices can enter it dynamically.
- **Message flooding**: A device can easily send a message to all other devices in the local network.
- **Message buffering**: A device which loses connection to the other devices will receive all sent messages when it connects to the local network again.
- **Message reordering**: The ordering of messages sent by one device is preserved on the receiver side.

2. SYSTEM OVERVIEW

2.1 API

Jakob, Manuel
Last Contact Table:

N_1	T_1
N_2	T_2
$\vdots$	$\vdots$
N_n	T_n

ACK-Table:

Sender \ Receiver	N_1	N_2	...	N_n
N_1			...	
N_2			...	
$\vdots$			$\ddots$	
N_n			...	

Message:

Last Contact Table
ACK-Table
Content

Acknowledgement:

Last Contact Table
ACK-Table

2.2 Emergency App

Claude, Alessandro

2.3 Chat App

Joel, Pascal
The Chat App ensures end to end encrypted messages via peer-to-peer connection through the flooding API. Encrypting and Decrypting messages is done public key cryptography. The keys are generated by the user and shared by QR codes that have to be scanned from the receiver.
If the receiver's network is not connected to the sender's network the messages are buffered and will be sent to the receiver later when the receiver's and the sender's network are connected. The receiver is able to get as many messages as are stored in the buffer.

When first starting the App the user has to enter his name and generate his public and private key. After generating the key the user is able to scan public keys from other members or provide his own public key for scanning. Upon scanning a new chat is displayed in the chat-list and a reminder appears to scan the public key of the chat partner.

For group chats symmetric keys are used. Every chat will have an administrator who generates a symmetric key and can distribute the key to other group members.

Pressing on a chat in the chat-list opens a chat to write and read messages.

3. REQUIREMENTS

3.1 API

Jakob, Manuel
Last Contact Table:

N_1	T_1
N_2	T_2
$\vdots$	$\vdots$
N_n	T_n

ACK-Table:

Sender \ Receiver	N_1	N_2	...	N_n
N_1			...	
N_2			...	
$\vdots$			$\ddots$	
N_n			...	

Message:

Last Contact Table
ACK-Table
Content

Acknowledgement:

Last Contact Table
ACK-Table

3.2 Emergency App

Claude, Alessandro

3.3 Chat App

Joel, Pascal

The Chat App ensures end to end encrypted messages via peer-to-peer connection through the flooding API. Encrypting and Decrypting messages is done public key cryptography. The keys are generated by the user and shared by QR codes that have to be scanned from the receiver.
If the receiver's network is not connected to the sender's network the messages are buffered and will be sent to the receiver later when the receiver's and the sender's network are connected. The receiver is able to get as many messages as are stored in the buffer.

When first starting the App the user has to enter his name and generate his public and private key. After generating the key the user is able to scan public keys from other members or provide his own public key for scanning. Upon scanning a new chat is displayed in the chat-list and a reminder appears to scan the public key of the chat partner.

For group chats symmetric keys are used. Every chat will have an administrator who generates a symmetric key and can distribute the key to other group members.

Pressing on a chat in the chat-list opens a chat to write and read messages.
Joel

4. WORK PACKAGES

4.1 API

Jakob, Manuel

4.2 Emergency App

Claude, Alessandro

4.3 Chat App

Joel, Pascal

5. MILESTONES

Pascal

6. REFERENCES