UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science

Jakob Mass

Automatic Mobile Device Pairing via Integrated Microphones

Bachelor's Thesis (6 ECTS)

Supervisors: Satish Narayana Srirama, PhD
Huber Flores, MSc

Tartu 2014

Automatic Mobile Device Pairing via Integrated Microphones

Abstract:

This thesis proposes a method for using integrated mobile device sensor data to gain information about users' social context. In-device microphones are utilized to determine which users are co-located (for example, participating in a business meeting). This is achieved by clustering the sensor data at a remote server which also notifies the co-located devices of each others' existance and provides instructions on how to communicate with one another using *Bluetooth* technology. The work contributes a description of an entire implementation for such a framework. Test results show an accuracy of 75% in recognizing user co-location.

Keywords:

Automatic pairing, Android, Bluetooth, Clustering, Audio analysis, Mobile Cloud

Automaatne Mobiilseadmete Ühendumine Integreeritud Mikrofonide Abil

Lühikokkuvõte:

Käesolev bakalaureusetöö kirjeldab meetodit, kuidas mobiilseadmete sensorite abil saada informatsiooni kasutajate sotsiaalse konteksti kohta. Seadmete mikrofone kasutades tehakse kindlaks, millised kasutajad paiknevad samas asukohas. Selle saavutamiseks viiakse sensorandmete peal läbi klasteranalüüs välisel serveril. Lisaks analüüsile annab server ka koos asuvatele seadmetele teada üksteise olemasolust ning varustab neid juhistega, kuidas teineteisega *Bluetooth*-i abil ühenduda. Käesolev töö pakub sellise platvormi teostuse kirjeldust. Testitulemused näitavad, et pakutaval lahendusel on koos asumise kindlaks tegemisel 75% täpsus.

Võtmesõnad:

Automaatne ühendumine, Android, Bluetooth, Klasteranalüüs, Heli analüüs, Mobiili- ja pilvearvutus

Contents

1 Introduction

Today, smartphones and other mobile technologies are impacting people's daily lives in a significant way. A growing number of persons are constantly connected to the internet and have access to a plethora of different services and software. These circumstances allow for the emergence of collaborative environments, which allow users to access, create and share information with little effort.

Yet today's most widely used mechanisms for close proximity inter-device communication establishment are often complex. They require a number of actions: to ensure security, users are prompted to enter PIN-codes, even though they might not realize the need for this. Most applications also require that users scan for a list of devices and then handpick each peer that communications are to be established with. These are steps which a person must go through before they can actually reach their goal – to exchange specific data with others.

Consequently, mobile users often prefer either traditional ways to share information, such as e-mail or turn to other popular applications which also support content sharing – for example social platforms such as WhatsApp, Line or Facebook Messenger.

These applications provide users with important social context, many of them provide the option of creating custom groups out of persons, thus enabling to share content with a specific group whenever necessary. However, these groups need to be formed manually, and any changes one wishes to make have to be stated in an explicit manner. In addition, the file transfers might rely on third party servers, which could bottleneck data transfer speeds. A new field called Mobile Social Network in Proximity (MSNP) [1] opposes itself to centralized social network services such as Facebook. MSNP aims to provide mobile users with the capability of forming proximity-based social networks on-the-go.

On the other hand, cloud computing is providing an ubiquitous communication platform, enabling augmentation of smartphone capabilities. Cloud computing allows to tap into several mobile devices to extract data from them with the intent of deriving conclusions from the collected information.

This creates the opportunity of providing each device with knowledge they could not have discovered on their own. Modern smartphones are equipped with a number of different sensors, such as an accelerometer, digital compass, gyroscope, GPS, microphone and a camera. All of these provide different ways to sense the device's and the user's surroundings.

Given this context, to tackle the challenge of setting up peer-to-peer connections, we have designed a framework that would provide a novel approach to share content to other mobile users close by. The framework allows a group of people, for example – attendants of a meeting or a class – to interact and share information with each other. The users are however relieved from the burden of setting up the

connections. The framework provides automatic setup for inter-device communications, with no input from the user.

To achieve this, a cloud service keeps track of a set of clients. Once the service determines that a set of devices are likely to be in the same context (ie. participating in the same meeting), it will provide instructions to them about how to pair up with one another.

To be able to classify whether or not some devices are in the same meeting, real-time audio data from each client is used. Each client is running a service which periodically utilizes the integrated microphone of the device to create a sequence of amplitude levels. Each sequence is of a predetermined length. This sequence is immediately transferred to the cloud service, where synchronously recorded sequences from different devices will be analyzed.

1.1 Benefits

Using audio data for presence detection provides some unique benefits when compared to other methods such as GPS. Acoustic information respects sound barriers such as walls. This is close to the way humans perceive things such as privacy and the notion of attending the same gathering [2]. Continuous audio analysis also means that participation detection is dynamic. For example, if somebody leaves a meeting early, their device stops being listed as a participant for the people still attending the meeting.

1.2 Outline

- **Section 2** covers the state of the art, providing a technological background relevant to this thesis. Mobile technologies, audio signals, clustering and computer supported collaborative work are characterized.
- **Section 3** presents my approach to the problem, describing the software solution. The approach is looked at from the client's perspective and the server's perspective. Clustering is explained, alongside audio analysis and data preprocessing.
- **Section 4** presents the effectiveness of the system, a series of tests are documented.
- **Section 5** looks at related works and similar solutions in the domain of mobile sensing applications, automatic wireless mobile connectivity.
- **Section 6** summarizes the work done and also describes possible future developments.

2 State of the Art

2.1 Sensing on Mobile Phones

Today's smart-phones offer a wide array of integrated sensors - like an accelerometer, a digital compass, gyroscope, GPS, microphone and camera. They enable the emergence of new applications across domains starting from healthcare, environmental networking, transportation to social networks and safety. Unlike before, the commodity smartphone of today provides an open and programmable environment, meaning that developers can easily create software and provide useful services using integrated sensor data [3]. Noteworthy smart-phone sensors are the microphone and the camera. Audio and video data are a rich source for deriving context or classifying activities of an individual's life.

One way to divide mobile phone sensing systems is following the involvement of the user. Should the user actively choose what, when and where to sample? Or should data sampling be automated, not requiring the user to provide input? Lane et. al. [3] label the first method *participatory sensing* and the second method *opportunistic sensing*. Opportunistic sensing lessens the burden of the user when using a sensing system.

2.1.1 Communication and Collaboration on Smart-Phones

As humans are highly social beings, technologies which help share ideas, transfer messages and other information with friends, business partners and family are continually challenged to evolve and offer new, improved ways to communicate.

Thanks to the immense popularity of smart-phones, researchers have recently started looking for ways to provide novel data transmission and interaction methods between people, using integrated sensors to augment traditional practices [3].

Currently, most platforms expect users to provide context manually, for instance: to share documents with the participants of a given ongoing business meeting, the user must first select which persons actually are attending the meeting.

2.2 Mobile CSCW

Computer Supported Cooperative Work (CSCW) is a field of Human Computer Interaction that looks into the phenomenon of using computer technology to assist group interaction and collaboration. As mobile phones have become ubiquitous, the branch of Mobile CSCW has also surfaced.

Although the definitions of CSCW and its primary goals are slightly varying [4], the core ideas are *supporting group work* and *designing* computer-based technolo-

gies. *Groupware*, a term coined in the 1980s by Peter and Trudy Johnson-Lenz was adopted by the CSCW community to label computer applications which support collaboration.

D. Johnson [4] examined a number of larger groupware applications, such as *Office Groove, BSCW, Apache Wave* and also looked at more general frameworks intended to be used for rapid development of groupware.

Some do have support for mobile users, yet mostly this has been a feature that has been added later on during the platform's lifetime, and was not a priority during development of the core application. While there are many technologies offering mobile cooperation support, there are no distinct, recognized tools which would support rapid engineering of mobile groupware.

Herskovic et al. [5] have proposed a set of requirements for mobile collaboration software. They are:

- **Users Interaction Flexibility**, meaning support for recurrent changes in group size and structure. One example of achieving this is *automatic user detection*, which involves automatic gathering and handling of information related to collaborator availability. An alternative is *User connection / disconnection*, meaning that users manually choose their level of involvement by switching to online use on-demand.

- **Users Interaction Protection**. The work done by each user must be protected against unauthorized and unintended access. Privacy measures are also necessary, users should be able to decide which data to share and to whom.

- **Communication**, users should be able to relay messages to one another. The system should provide means for synchronous/asynchronous communication. Typically there is also a local *shared* workspace. New members of a group need to be able to receive previously created shared data.

- **Heterogeneity and interoperability**. Collaboration should support devices of different classes, manufacturers and different hardware specifications. The system should be able to handle data and services designed by different providers.

- **Autonomous Interaction-Support Services** . Networking issues should be invisible to the user. If a wireless network connection is not available, alternative means should exist which would restore the collaboration ability. Service and device discovery should be automated.

- **Users Awareness**. The system should provide both information about online awareness, for example: *lists of connected users, current activity*, as

well as offline awareness: *last modification to a document, text authorship*, etc.

- **Data Consistency and Availability**. The system must ensure data consistency when events such as disconnection occur. If an user shares data with another user, the data should be available to both parties after sharing has finished if one of them is no longer reachable. In addition, caching of data and data synchronization, conflict resolution methods for data inconsistencies should be considered.

2.3 Mobile Wireless Communication

Wireless communication is a fundamental part of smartphones. Thanks to the mobility that wireless communication technologies offer, users can dynamically join or leave a given network. Wireless networks are simple to set up.

There are however some disadvantages: reliability is lower because of the risk of radio interference, power consumption is higher when compared to wired connections and security is also a concern, because the transfer medium is more exposed when compared to traditional wired connections [6]. Bluetooth is a wireless telecommunication technology, which is supported by most modern Android smartphones. It was created for low-power devices to communicate over short distances. [7].

2.3.1 Bluetooth in Android

To use Bluetooth programmatically, the Android API provides RFCOMM (Radio frequency communication) sockets. RFCOMM is the most common socket type for Bluetooth [8]. To create a connection between two devices, one side must take the role of a server and create a listening socket - a *BluetoothServerSocket* object. The other side, the client, needs to create a *BluetoothSocket*. The *BluetoothServerSocket* listens for connections and once a client connects successfully, it returns a *BluetoothSocket*. The *BluetoothServerSocket* can listen for connections in two ways: using insecure RFCOMM socket or secure RFCOMM socket. The insecure method requires no user input (such as PIN entry) to create the actual connection, however, it can be vulnerable to Man In The Middle attacks [9].

After a connection is established, IO streams can be used to transmit data, via usual Java conventions - by calling *getInputStream()* and *getOutputStream()*.

2.4 Clustering

Clustering is the process of dividing a set of data into clusters, using some measure of similarity to decide which data points should belong to the same cluster. In [10],

the following methods are mentioned as the five main types of clustering methods: partitioning, hierarchical, density-based, grid-based and model-based clustering .

Partitioning algorithms partition a database D of n objects into k clusters, where k is an input parameter. This means that we need to know the number of existing clusters in advance. In addition, the partitioning approach regards the clusters as voronoi cells of a voronoi diagram, meaning that cluster shapes are restricted to being convex.

Hierarchical methods use trees to determine clusters, either building clusters from leafs by merging them up to some point or by moving top-down along the tree, splitting the data. This is especially useful for summarizing or visualizing data [11].

Grid-based methods divide the data space into a grid of cells. Clustering operations are then done directly on this grid, allowing for fast processing times.

Model-based form mathematical models to describe each cluster and try to fit the data to given models. These methods have the potential of considering noise and outliers in the data.

2.4.1 Density Based Clustering

In density-based clustering methods, the concept of density is used to form clusters.

Clusters are created from regions in the dataset, in which data points are spaced more densely. Clusters are separated from each other by regions where the density is lesser.

The main idea is to start growing a given cluster until the density in the region passes some given threshold. The advantages of density-based clustering methods are the ability to form arbitrarily shaped clusters and to disregard noise and outliers. A typical density based algorithms is DBSCAN.

2.4.2 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [12] is a clustering algorithm which requires only 2 input parameters. It can detect the number of clusters automatically.

As explained in [13], DBSCAN uses the following concept for clusters: *"a cluster is defined as containing at least a minimum number of points , every pair of points of which either lies within a user-specified distance (ε) of each other or is connected by a series of points in the cluster that each lie within a distance of ε of the next point in the chain."*

The algorithm can use any distance function for two points, meaning that for some given application, a fitting distance measure can be used [12].

2.5 Time Series

Time series is a category of data, a single time series is made up of a sequence of values, the values usually depict sequential measurements over time, often recorded at equal intervals. Time-series are popular in applications such as stock market analysis, economic and sales forecasting, observation of natural phenomena, etc. [10]

2.6 Distance Measures

As mentioned above, clustering algorithms require a distance measure to do classification. A well-known distance measure is the Euclidean distance. The Euclidean distance between two points is defined as the square root of the sum of the squares of the differences between the corresponding coordinates of the points. In two-dimensional Euclidean geometry, the Euclidean distance d between points $a = (a_x, a_y)$ and $b = (b_x, b_y)$ is defined as $d(a, b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}$.

For *time series*, Euclidean distance is defined as the *sum* of the squared distances from each nth point in one time series to the nth point in the other. However, the Euclidean distance is only fitting for time series which are perfectly aligned. The distance measure fails to recognize two highly similar time series if one is slightly shifted along the time axis.

2.6.1 Dynamic Time Warping

Dynamic time warping (DTW) is a distance measure which was proposed to tackle the shortcoming mentioned in 2.6. DTW is able to take into account global and local shifts in time.

For two time series, X and Y,

$$X = x_1, x_2, \ldots, x_n$$
$$Y = y_1, y_2, \ldots, y_m$$

an n-by-m cost matrix is constructed, where the (i^{th}, j^{th}) element of the matrix contains the distance between two points x_i and y_j. Most often, the Euclidean distance is used. Next, an optimal alignment between X and Y is found (an alignment which would have minimal overall cost according to the cost matrix).

However, the space of possible alignments between X and Y is large, thus DTW has a time and space complexity of $O(N^2)$. A modification of DTW, called FastDTW, was created to improve upon the complexity of DTW. Using a multilevel approach, which initially finds optimal alignments between time series at a coarser level and then refines the resolution, a linear time and space complexity was achieved [14].

This makes FastDTW a fitting time series distance measure for situations where the processing time is limited.

2.7 Machine Learning Libraries

Several machine learning libraries which include different clustering algorithms and distance measures exist. For example, a well-known machine learning software library is Weka [15]. Weka puts emphasis on user-friendliness and providing interactive use with the user, including graphical user interfaces, enabling users to try out algorithms quickly. On the other hand, Java-ML is a library which focuses on providing developers with the opportunity to use machine learning in their own software. Thus, Java-ML provides simple, basic and easy to understand interfaces which developers can integrate into their code. Java-ML also involves a number of similarity measures, among them FastDTW. [16]

2.8 Audio Signal Analysis

Audio representations are most often categorized into two: *a)* time domain characterizations, which use a time-amplitude representation or *b)* frequency domain characterizations, which use frequency-magnitude representation. Next, a brief overview of features which can be extracted from these two domains is presented.

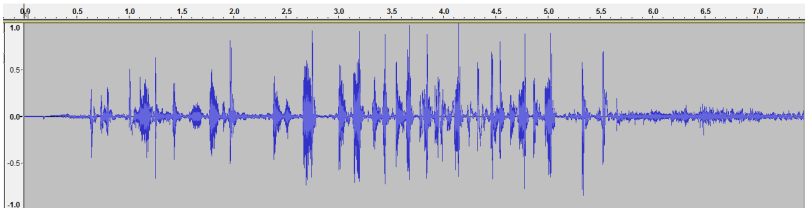


Figure 1: A time-amplitude representation of an audio signal

- **Features in the time domain.** In time-amplitude representation, a signal is depicted as amplitude varying in time. A typical audio signal in this domain is shown in Figure 1.

  In the time domain, features such as *average energy*, which indicates loudness of the signal; *volume distribution*, which is the variation of a signal's energy level; and *silence ratio*, which describes how big a part of the signal is silent, can be extracted.

- **Features in the frequency domain.** Using the Fourier Transform, a frequency domain representation can be derived from a time domain representation. In the frequency domain, common features used are: *bandwidth*,

which describes the range of frequencies present in a signal; energy distribution, which specifies which frequencies are more present in a given signal; harmonicity, a feature that is often of high value in music. Pitch is the fourth frequency domain feature, which is a much more subjective feature than the ones described before. Pitch ties audio signals with the musical notion of tones.

[17]

2.9 Literature Overview

Implementing the client-side mostly follows the official documents and API Guides of the Android Developer portal [18]. Some work on using the integrated microphone has been done following the book *Professional Android Sensor Programming* [19].

The book *Algorithms for Sound and Music Computing* [20] provides some information on the characteristics of human speech and techniques for extracting features from audio.

3 Approach

The platform is built following a client-server model. Each client is expected to create a time series of audio amplitude data, and transmit it to the server. Alongside the time series, other details such as the device's *Bluetooth* MAC-address are provided. Continuously, the client records and transmits these sequences throughout the application's lifetime. In addition, all clients begin and end the capture of audio in synchronization, meaning that the time series recordings are started and stopped on every device at the same time.

When the server receives a set of client data, all recorded at some specific time, it runs a clustering algorithm on the time series in order to group them based on audio similarity. Then, the server provides each member of a specific group with the information needed to create a *Bluetooth* network with other members in that same group.

After receiving this information, the clients in each group create wireless connections with other members of the same group, with no input from the user needed. Once the wireless network has been established in the group, users can share files and text messages with other users within the group.

3.1 Technologies and Methods Used

The client-side for this platform has been implemented as an Android application. For communication with the server, the internet connection provided by the Android platform is used. This connection might be based on Wi-Fi or a GSM data transmission protocol, either way, the connection type here is no concern to the client application and is handled by the operating system.

In addition, an in-device microphone is utilized by the application. Microphones are present in practically every newer Android phone, some of which might even have more than one microphone. Wireless inter-device connections are established using *Bluetooth*. Messages to the server are sent as HTTP requests.

The server-side is implemented using Java Servlets technology, the server itself is run by the open-source *Apache Tomcat version 7* web server. The data analysis on the server uses the *Java-ML* machine learning library, which includes an improved version of the *Dynamic Time Warping* distance measure algorithm - *FastDTW* [14].

3.2 The Client

3.2.1 Client Application Startup and Initialization

This section describes the platfrom from the client's point of view.

As the client application is started, the software first runs a few checks to determine whether the device is configured properly for running this application. This includes determining whether an internet connection is available and whether the *Bluetooth* radio is enabled.

After this, the difference between the device's internal clock and a remote Network Time Protocol (NTP) time server clock is estimated. The same NTP server is used by every client in the platform. This allows different devices to reference a common time and use it to start recordings in unison. The difference between the device's local time and the server's time is stored and the NTP time server is not used during the rest of the client software's lifetime. A screenshot of the application during this state is shown in Appendix B, figure 7a.

At this point, the application can enter it's main state: producing sequences of audio data and transmitting them to the platform's cloud service.

3.2.2 Sequence Production and Transmission

The client periodically transmits data to a server for processing. This data is formatted as a *JSON* string, in which the following is included:

- the client device's *Bluetooth* MAC address;
- a user-created string depicting the user's (nick)name;
- a time series of successively recorded amplitude values;
- a timestamp depicting the time at which the recording of the time series began

The creation of the amplitude time series is explained in more detail in the next section. Once the time series creation is complete, each *JSON* object is transmitted immediately to the server using a HTTP POST request.

3.2.3 Using Integrated Microphones

A time series is created from samples gathered from the physical device's microphone. The Android platform allows to utilize the microphone via the *MediaRecorder* API. *MediaRecorder* allows specifying the audio encoder, output format and audio source used (often, phones have two microphones- one in the front, used for calls, and one in the back, for video capture).

The work behind the creation of a single sequence is as follows. The *getMaxAmplitude()* method of *MediaRecorder* returns the maximum amplitude heard since the last call to the method. This method is periodically called in a finite loop. A set interval determines the time between method calls, the method return values

are stored into an array during this loop. The resulting array contains numeric data about the amplitudes heard during the loop, no information about audio frequencies is stored.

The process of recording is reflected to the user in the form of a progressbar, which can be seen at the top of figures in7, Appendix B.

My approach uses the following constants when creating the audio time series. The number of samples gathered per series is 50. The sampling interval is 140 ms. Thus, one sequence represents a $50 * 140ms = 7000ms$ time period. These numbers are reasonable, as in normal human speech, acoustic transformations (syllable changes, letter expressions) happen within 100-200 ms [20] and sentences in speech generally last a few seconds.

3.2.4 Handling Server Instructions

The client receives a response for each POST request it sent to the server. The server determines which group of devices the client who sent the request belongs to, if any. The response, also a JSON object, contains nicknames of other devices belonging to the same group and a MAC address of the device to connect to as a client. A flag which tells the device whether it should listen for incoming connections is also included. An example JSON object is provided in figure 2.

```
{"listento" : true,
 "connectto" : "CC:FA:00:16:2B:9A",
 "group" : ["Luthien","Beren"] }
```

Figure 2: An example of an instructions JSON object

The Bluetooth network is further detailed in section 3.2.5. As each response contains a list of devices, the client can keep the list of group members up-to-date, and display this information to the user dynamically. If an user leaves the group, his device will stop appearing in the member lists of others still in the group.

3.2.5 Bluetooth Network

The traditional Bluetooth network model, the so-called *piconet* supports only up to 8 devices. Due to the complexity of creating an efficient network out of multiple piconets, a *linked list*-based structure was chosen, as it has virtually no size limitations and is very easy to implement. A tradeoff, however, is reduced efficiency in data routing.

As explained in 2.3.1, there are two roles devices can take when creating Bluetooth connections. Each device connects to 0-1 other devices and acts as a server

for up to 1 device. Any client that isn't first or last in the linked list network will act both as a client and a server.

For example, consider figure 3 where we have a network of 3 devices. Devices which have an arrowhead entering them are listening for connections. Nodes from which arrows start are acting as clients, connecting to a provided address. In this figure only device B is acting the two roles (server and client) at once. Because

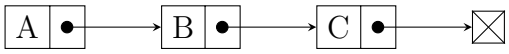


Figure 3: A 3-node example of the wireless network structure

of the client-server model used for creating a Bluetooth connection between two devices, in order for one device to share data to multiple devices, a session with each device must be created separately.

When programmatically creating Bluetooth sockets, the methods *createInsecureRfcommSocketToServiceRecord(UUID uuid)*, *listenUsingInsecureRfcommSocketToServiceRecord(String mac, UUID uuid)* take an argument Universally unique identifier (UUID).

In this approach, each client creates an UUID using their own Bluetooth MAC-address and a string which is common to the entire platform. This means that if some device D has created a listening RFCOMM socket with an UUID generated using it's own MAC-address, then another device E will be able to reconstruct the same UUID and use it to connect to device D, because E has been provided the MAC-address of D.

Because these UUIDs must match in order for the connection to be accepted, a level of security is provided.

3.2.6 File Sharing

Once a Bluetooth network has been established, group members may share files within the group. The client supports sending files to all network participators or a single, specific recipient.

To allow a file to reach the intended destination(s), a custom class (*BTMessage*) is used which contains fields that help transmit the file across the network. The fields of the class are as follows:

1. **The initial sender of the file**
2. **The intended destination of the file**
3. **The MAC address of the last transmitter**

4. **A byte array of the actual contents of the file**

In the general scenario, the initial sender will create a *BTMessage* object and send it to all of its open *BluetoothSockets* (see 2.3.1). As the *BTMessage* is received by the immediate neighbours in the network, a given receiver will first check if it was the intended recipient for this *BTMessage*. If so, then the byte array inside *BTMessage* will be constructed into an actual file and saved onto the device's internal storage. A request is then made to view the file using Android *Intents* [21].

The file is not stored if the current device is not the recipient. In this case (or if the recipient is the entire group), the next step is to rebroadcast the message.

To do this, the client goes through all of its open *BluetoothSocket*s and passes on the *BTMessage* to any socket which doesn't correspond to the MAC in the third field of the *BTMessage* (as listed above). This ensures that we won't retransmit the *BTMessage* to a client that already sent us the message.

3.3 The Server

3.3.1 Initialization

As the server is started, first some global objects are created, which will be accessible to different threads. The first object is a queue for incoming data (henceforth called the ***data queue***). As clients post audio sequences to the server, the sequences are stored in this queue to be accessed later.

The second object is a hash map (hash table), which is used to store instructions relevant to individual clients (hereafter referenced to as the ***instructions map***). The keys in the hash map are MAC-addresses of the clients. The values corresponding to the keys are JSON objects which contain Bluetooth connectivity instructions and information relevant to the group the client has been clustered into.

Before the servlets which handle HTTP requests are enabled, the server is also synchronized to an NTP time server clock. This same time server is used by the all the clients as well, meaning that each party in the system has a reference to a common clock.

A new thread, called the *WorkThread* is also executed. The *WorkThread* is responsible for periodically creating another type of thread - *ClustererThread* - and running instances of this thread. The ClustererThread is described in section 3.4.

3.3.2 Request Handling

As mentioned earlier in 3.1, the server is implemented as an Apache Tomcat web server with Java Servlets. The following process takes care of handling POST-

requests which the clients send.

As some given request is received, the JSON object described inside the request body is added to the data queue. Secondly, the instructions map is accessed using the *Bluetooth* MAC-address of the request author as the key, determining whether there are any instructions currently corresponding to it. If instructions matching this key are found, they are added to the response of the POST request. Otherwise, the response will contain no instructions.

This concludes the process of receiving POST requests and responding to them.

3.4 Clustering

The WorkThread schedules the creation and execution of ClustererThread instances. A new ClustererThread is executed every m time units, the thread's execution is always started n time units after the clients start recording audio. This is illustrated in figure 4, as the *cluster()* method is periodically called.

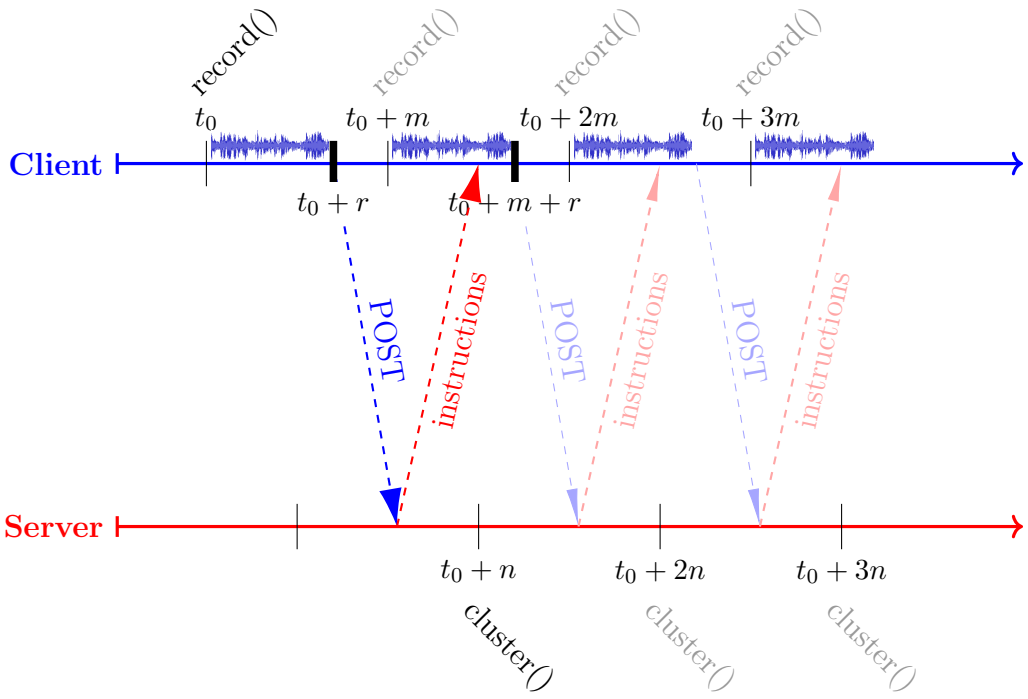


Figure 4: A timeline of interactions between the client and server

Two things must be ensured when choosing when to call the clustering method (parameter n). First, only data from one recording session must be clustered. Second, the clustering must finished by the time the client makes the next POST

request and expects a response containing the results of the clustering. For the example in figure 4,this can be ensured by requiring that:

$$\begin{cases} t_0 + n & > & t_0 + r + d_1, \textit{ where } d_1 \textit{ is the POST request network delay} \\ t_0 + n + d_2 & < & t_0 + m + r, \textit{ where } d_2 \textit{ is the POST response network delay} \end{cases}$$

More generally, we require that

$$\begin{cases} n & > & r + d_1, \textit{ where } d_1 \textit{ is the POST request network delay} \\ n + d_2 & < & m + r, \textit{ where } d_2 \textit{ is the POST response network delay} \end{cases}$$

The following actions are done by the the ClustererThread. The server's data queue is emptied, the audio time series are preprocessed and a *dataset* object is formed out of all the data. Preprocessing is documented in detail in section 3.5. The dataset is then given as input to the clustering algorithm itself.

3.4.1 DBSCAN in Java-ML

The clustering algorithm, DBSCAN, is provided by the Java-ML library. DBSCAN is a fitting choice for the purpose of clustering audio amplitude data with the intent of forming social groups on-the-go, as it can determine the number of groups automatically and allows to use a distance measure which is fitting for audio data.

The Java-ML implementation of DBSCAN uses 3 parameters: a distance measure, the minimum no. of points from which a cluster can be formed; and the ε-value. The meaning of the parameters has been explained in 2.4.2, 2.6. In my approach, DBSCAN is configured to run with the following parameter values:

1. **FastDTW** is given as the distance measure to use.

   As described in 2.6.1, Dynamic Time Warping (DTW) offers the ability of measuring the similarity of signals which might be misaligned. Because perfect time synchronization is an extremely challenging task, we expect that the recordings done by different clients are slightly shifted on the time axis. FastDTW is an optimization of DTW that is included in the Java-ML library.

2. **The minimum points** for a single cluster is 2.

   We wish to support work groups of sizes 2 persons and up, thus the choice for this parameter's value is clear.

3. **The ε-value** is set to 15.5.

   In the course of the practical work in my approach, I came to fix ε at this value, as it showed the best results in comparison to other values during initial tests. A more detailed methodology of determining an appropriate ε-value is described in [12], section 4.2.

When the clustering algorithm finishes its work, it returns a collection of datasets each representing a group of clients that are in proximity.

3.5 Preprocessing

Before clustering data, it is desirable to process the dataset in such a way that the features on which the distance measures are based on are better distinguished.

With time-domain audio data, one of the fundamental steps is feature scaling: ensuring that the values of each time series are of the same magnitude. A simple example illustrating this is the case where two microphones are recording audio which comes from the same source, however one device is significantly closer to the audio source than the other. This results in the two recordings having different average energy (amplitude) levels, even though the shape of the time-amplitude curves would look similar. In figure 5, this is the case when comparing the 2nd and 4th rows of the 1st column. The signals have a similar pattern, yet the magnitude differs greatly.

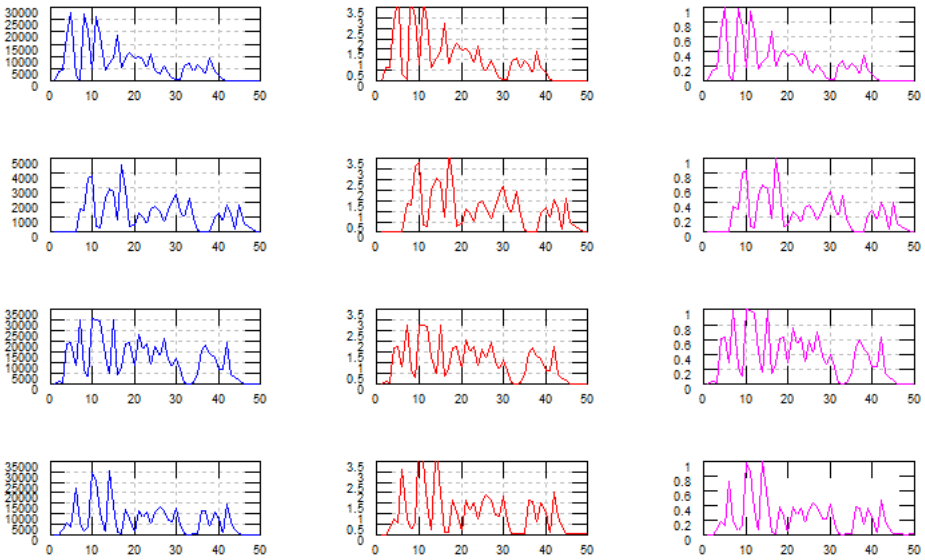


Figure 5: Left column: original data, center column: mean-normalization, right column: max-normalization

3.5.1 Mean Normalization and Alternatives

In my approach, I overcome the above-mentioned magnitude differences by taking the mean of a given amplitude series and dividing each value in the series with that mean value.

Alternative methods were also tried. One simple approach would be to rescale the values using the maximum value of the series, giving a range of values within $[0 \cdots 1]$. Initial tests however showed a higher error rate compared to using the mean. Using the mean gives more weight to peaks in the signal if the rest of the signal is not very noisy.

Emphasizing peaks is beneficial, as peaks (high amplitude sounds) are more likely to be heard on all devices at the same time.

3.5.2 Signal Smoothing

In addition to feature scaling, signal smoothing was also tried. Using a window function (the Hamming function) and Short-Time Average Energy analysis, the data was smoothed. This approach is discussed in detail in the book *Algorithms for Sound and Music Computing* [20]. Figure 6 shows a smoothed set of signals and the original audio. Tests showed no benefits compared to using mean scaling alone.

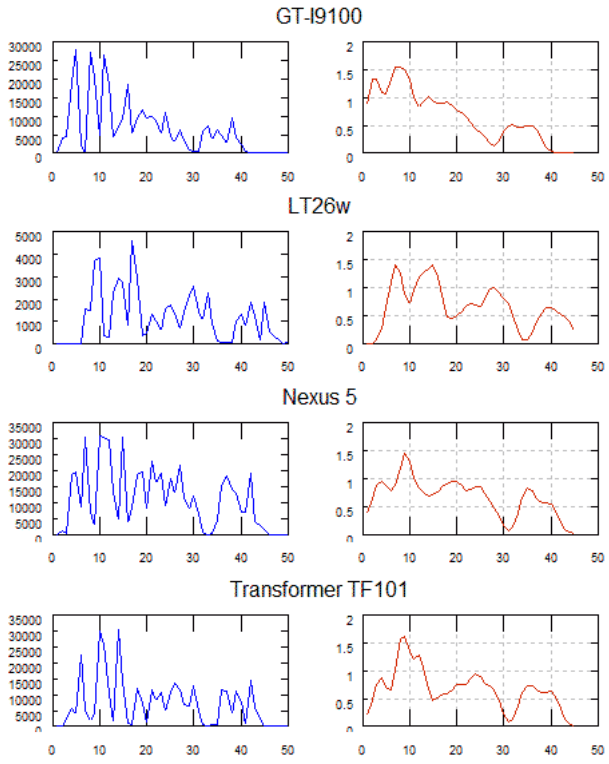


Figure 6: Left column: original data, right column: Smoothed signal

3.6 Creating Pairing Instructions

Each time a clustering finishes, the resulting clusters are processed to generate pairing instructions for the client devices. These instructions must support the forming of the Bluetooth network described in 3.2.5.

This is done by iterating through each device in each cluster, and comparing a given device against other devices in the same cluster. The aim is to detect if a given device was in that cluster during the previous clustering. If so, this means that the algorithm already generated instructions earlier and doesn't need to create new ones. If, however, the algorithm determines that a device is appearing in it's cluster for the first time, it then creates a new set of instructions for it. The contents of the instructions are brought out in figure 2.

The algorithm can determine if a device was previously in a given cluster or not by keeping a state of the links in the network in memory. If a device has 0 links to other devices in the same cluster according to that state, we can conclude that this device is appearing in the cluster for the first time.

If a device doesn't appear in any cluster for two concurrent clusterings, it's links are removed from the stored state of links. This allows for clients to dynamically leave the network without isolating other clients.

4 Results

To measure the effectiveness of this approach, the following tests were conceived.

The tests were done using Android devices of different types, namely several smartphones of different manufacturers and one tablet device. The exact device models have been brought out in table 1 .

Device	Amount
LG Nexus 5	5
Sony Ericsson Xperia Acro S	1
Samsung Galaxy S2	1
Asus Eee Pad TF101	1

Table 1: Devices used for testing

The test scenario constituted of splitting devices into two groups and positioning both groups in separate rooms. Each room had an audio source of a person reading a book. In both rooms, the devices were positioned close to the audio source (1-2 meters). The client application was started on all the phones and more than 60 iterations of clustering live audio time series were then run concurrently.

4.1 Measures Used

To measure the precision of the approach the following method is used. Per a clustering instance, for every device which is placed in the wrong cluster or not placed in a cluster at all, 1 is added to the **clustering error rate**. Then, we get the **error rate per device** by dividing the clustering error rate by the total number of devices clustered. For example, if we had devices A, B, C in room 1 and devices D, E, F in room 2; and the system creates the following clusters: $[A, C, D]$ and $[E, F]$, then the error rate per device would be 0.33.

In table 2, the average *error rate per device* is shown for each test round.

4.2 Conducting the Tests

4.2.1 Round One

In the first round, five devices were used, thus creating groups of 2 and 3. One room had an actual human reading a book out loud, in the other room a speaker was playing back an audiobook. 65 clusterings were run, the average error rate per device was 0.178. A noteworthy phenomena in the first round, however, was the

fact that when looking at instances where just one device was missing or misplaced, in 8 cases out of these 9, the device causing the error was the *Samsung Galaxy S2*.

In the first round, out of 65 clusterings, 43 had no mistakes, meaning 66.15% of the clusterings were entirely accurate.

4.2.2 Round Two

In the second round, the *Samsung Galaxy S2* was removed and 3 additional *LG Nexus 5*-s were added. This time, a live person was reading a book in both rooms.

Out of 70 clusterings, 53 had no mistakes, thus 75.71% of the clusterings were entirely accurate.

However, 11 of the erroneous results were from clustering instances where the clustering failed to work at all, that is, all devices were put in a single cluster. When discarding these instances where the clustering failed entirely, the accuracy is considerably higher, 89.8%.

Detailed data of the 70 clustering instances from round two have been brought out in Appendix C.

Test round	Average error rate per device
1	0.178
2	0.104

Table 2: Test results from different rounds of testing

5 Related Works

The potential of using microphones in mobile applications has been explored less than other sensors such as GPS receivers, cameras or gyroscopes.

Several works exist which aim to derive context of a single client via sensors. This works by gathering sensor data from the client, and then running that data through a classifier, which usually runs on a server.

Hewlett-Packard Labs presented an implementation which handled the same goal that this work is about. Their project uses silence signatures to match similar audio signals. To create these silence signatures, the audio signal is quantized into silence and non-silence through the use of an adaptive silence threshold [2].

Spartacus [22] uses an acoustic technique based on the Doppler effect to enable users to accurately initiate an interaction with a particular target device in their proximity through a pointing gesture. The application runs continuously in the background, removing need for manual user pre-configuration for interactions. Within a 3 meter distance, *Spartacus* achieves 90% device selection accuracy on average. Their solution also uses Android and passive, periodic audio sensing.

However, in their work, the communication itself is *initiated* by the user. After the software detects which device the user wishes to connect to, the connection is established automatically.

5.1 SurroundSense

SurroundSense [23], for example, uses the microphone in conjunction with other sensors to form ambience fingerprints which describe the location or context the user is in. The client-side of the SurroundSense framework records sensed values from sensors, pre-processes them at the client-side and then sends the preprocessed data to a server.

The server side segregates the different types of data and deals with each one according to a module assigned to it. For example, the server might perform color clustering on image data. After the server has processed the different types of sensor data, an "ambience fingerprint" is formed, which is then forwarded to a matching module, which matches it to already known fingerprints for localization. The authors note that audio information provides benefits such as recognizing walls, which, in addition to working as sound barriers, are something that humans often associate with barriers of context.

5.2 Mobile Sensing

[24] The CenceMe application is a system which uses various sensors of the Nokia N95 phone to detect activities and context of users (e.g. walking, having a conver-

sation, sitting in a vehicle). Because analyzing continuous streams of sensor data can be computationally expensive, their solution "splits" some of this work. Some classification is done by the client, some by the server. More complex classifications, such as ones that involve sensor data from multiple clients, are done on the back-end server. This so-called *split-level design* offers benefits such as allowing users to create custom markers for activities or contexts that their phone has classified only locally. This allows for users to create classified states beyond the ones that the framework initially proposes. In addition, because some classification is done in-phone, the data being sent to the server is more light-weight.

6 Conclusions & Future Research Directions

This work presented an approach for using mobile device microphones and cluster analysis to create a collaborative work network on-the-go. The goal was to create a dynamic Bluetooth network of persons in close proximity without requiring the users to setup the connection.

A complete implementation for this approach was described, involving an Android client application equipped with file sharing capabilities and a *Java Servlets*-based web server. The server-side of the implementation uses *Dynamic Time Warping* and *Density-Based Spatial Clustering For Applications with Noise* to analyze audio data. The presented implementation automatically creates Bluetooth connections between devices in proximity without additional steps required from the user other than initiating the client application.

To support this implementation, a number of similar and related works were researched, involving sensor usage and mobile wireless connectivity. Different categories of clustering algorithms were examined and density-based clustering was chosen as a fitting method for the given goal.

The accuracy of the clustering, which is the key factor in this approach, was tested. When grouping up devices based on microphone data, test results showed an accuracy of up to 75.71% and in certain cases up to 89.8%.

6.1 Future Developments

To improve this approach, I propose several possible future developments. Firstly, a survey of audio preprocessing methods could be carried out to acquire better knowledge about methods used to extract more features out of audio data. The preprocessing in this thesis' approach is relatively light.

Secondly, the Bluetooth network structure used in this approach is rather limited. The current structure requires a message to traverse to all nodes in the network in the worst case, meaning long delays. Furthermore, the current structure potentially isolates some nodes for a time when one member of the network leaves. Use a different structure for the bluetooth network, as the current linked list one forces a message to hop through each device in the worst case.

Thirdly, it would be desirable to receive instructions from the server as soon as they are created, instead of receiving them when making future requests.

Automaatne mobiilseadmete ühendumine integreeritud mikrofonide abil

Bakalaureusetöö (6 EAP)

Jakob Mass

Resümee

Enamus tänapäevaseid nutiseadmed omavad mitmeid integreeritud sensoreid nagu kaamera, mikrofon, GPS, liikumissensorid jt. Tegemist on mitmekülgsete andmeallikatega, milledelt pärineva info rakendamist on võimalik võimendada pilveteenustega. Viimastel on suutlikkus koguda infot korraga eri seadmetelt ja viia läbi andmeanalüüsi suuremal skaalal kui üksikseadmed seda suudaks.

Teiseks on enamus nutiseadmed varustatud ka tehnoloogiatega, mis võimaldavad otsest seadmetevahelist andmesidet (Bluetooth, Wi-Fi Direct). Paraku nõuab nende kasutamine mõningaid eeltegevusi kasutajalt.

Kõigepealt tuleb otsida lähedalasuvaid seadmeid ning seejärel valida välja seade, millele andmeid saata. Lisaks peavad nii saatja kui vastuvõtja enne ühenduse loomist kinnitama, et nad on nõus ühenduse loomisega. See kõik mõjub kasutajale koormavalt ning tihti pöördutakse vahendite poole, millega ollakse juba harjunud nagu seda on näiteks e-mail või *Facebook*.

Käesolev töö uurib võimalust lahendada ülalkirjeldatud probleem nutiseadmetes olemasolevate mikrofonide ning välise serveri abil. Selleks on loodud *Android*-rakendus, mis kogub pidevalt andmeid seadme mikrofonist ning edastab neid serverile. Serveris aga võrreldakse kõigi andmeid saatnud seadmete helisignaale ning otsustakse, millised neist on sarnased. Sarnased signaalid grupeeritakse kasutades klasteranalüüsi. Helisignaalide sarnasust võimaldab mõõta algoritm *Dynamic Time Warping*. Klasteranalüüsiks kasutatakse algoritmi *DBSCAN*.

Kui seadmed on grupeeritud, saadab server igale seadmele infot temaga samas grupis olevate seadmete kohta. Seda teavet kasutades loovad seadmed omavahelise *Bluetooth*-võrgustiku, sealjuures ei nõuta kasutajalt lisategevusi.

Kui *Bluetooth*-võrgustik on loodud, võimaldab rakendus jagada võrgustiku sees faile ning saata tekstisõnumeid. Võimalik on jagada infot nii kogu võrgustikule korraga kui ka üksikutele liikmetele.

References

[1] C. Chang, S. N. Srirama, and S. Ling, "Towards an adaptive mediation framework for mobile social network in proximity," *Pervasive and Mobile Computing*, 2013.

[2] W.-T. Tan, M. Baker, B. Lee, and R. Samadani, "Sensing device co-location through patterns of silence," in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, (New York, NY, USA), pp. 473–474, ACM, 2013.

[3] N. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. Campbell, "A survey of mobile phone sensing," *Communications Magazine, IEEE*, vol. 48, pp. 140–150, Sept 2010.

[4] D. Johson, "Mobile support in CSCW applications and groupware development frameworks," *International Journal of Interactive Mobile Technologies*, vol. 7, no. 2, 2013.

[5] V. Herskovic, S. F. Ochoa, J. A. Pino, and H. A. Neyem, "The iceberg effect: Behind the user interface of mobile collaborative systems," *J. UCS*, vol. 17, no. 2, pp. 183–201, 2011.

[6] E. Ferro and F. Potorti, "Bluetooth and wi-fi wireless protocols: a survey and a comparison," *Wireless Communications, IEEE*, vol. 12, pp. 12–26, Feb 2005.

[7] J.-S. Lee, Y.-W. Su, and C.-C. Shen, "A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi," in *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*, pp. 46–51, Nov 2007.

[8] G. Inc., "BluetoothSocket. Android API." `https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html`. Accessed: 31.03.2014.

[9] G. Inc., "BluetoothAdapter. Android API." `https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html\#listenUsingInsecureRfcommWithServiceRecord(java.lang.String, java.util.UUID)`. Accessed: 31.03.2014.

[10] J. Han and M. Kamber, *Data Mining: Concepts and Technique. Second Edition.* Morgan Kaufmann, 2006.

[11] A. Amini, T. Wah, and H. Saboohi, "On density-based data streams clustering algorithms: A survey," *Journal of Computer Science and Technology*, vol. 29, no. 1, pp. 116–141, 2014.

[12] M. Ester, H. peter Kriegel, J. S, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," pp. 226–231, AAAI Press, 1996.

[13] I. H. Witten, E. Frank, and M. A. Hall, "Data mining: Practical machine learning tools and techniques. third edition," 2011.

[14] S. Salvador and P. Chan, "FastDTW: Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[15] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, pp. 10–18, Nov. 2009.

[16] T. Abeel, Y. V. de Peer, and Y. Saeys, "Java-ML: A machine learning library," *Journal of Machine Learning Research*, vol. 10, pp. 931–934, 2009.

[17] G. Lu, "Indexing and retrieval of audio: A survey," *Multimedia Tools and Applications*, vol. 15, no. 3, pp. 269–290, 2001.

[18] "Android Developers." http://developer.android.com/develop/index.html, 2014. Accessed: 13.05.2014.

[19] G. Milette and A. Stroud, *Professional Android Sensor Programming*. John Wiley & Sons, Inc., 2012.

[20] G. D. Poli and L. Mion, *Algorithms for Sound and Music Computing*. 2006. Chapter 5: From audio to content.

[21] G. Inc., "Intents and intent filters." `http://developer.android.com/guide/components/intents-filters.html`. Accessed: 13.05.2014.

[22] Z. Sun, A. Purohit, R. Bose, and P. Zhang, "Spartacus: Spatially-aware interaction for mobile devices through energy-efficient audio sensing," in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, (New York, NY, USA), pp. 263–276, ACM, 2013.

[23] M. Azizyan, I. Constandache, and R. Roy Choudhury, "Surroundsense: Mobile phone localization via ambience fingerprinting," in *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking*, MobiCom '09, (New York, NY, USA), pp. 261–272, ACM, 2009.

[24] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell, "Sensing meets mobile social networks: The design, implementation and evaluation of the cenceme application," in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08, (New York, NY, USA), pp. 337–350, ACM, 2008.

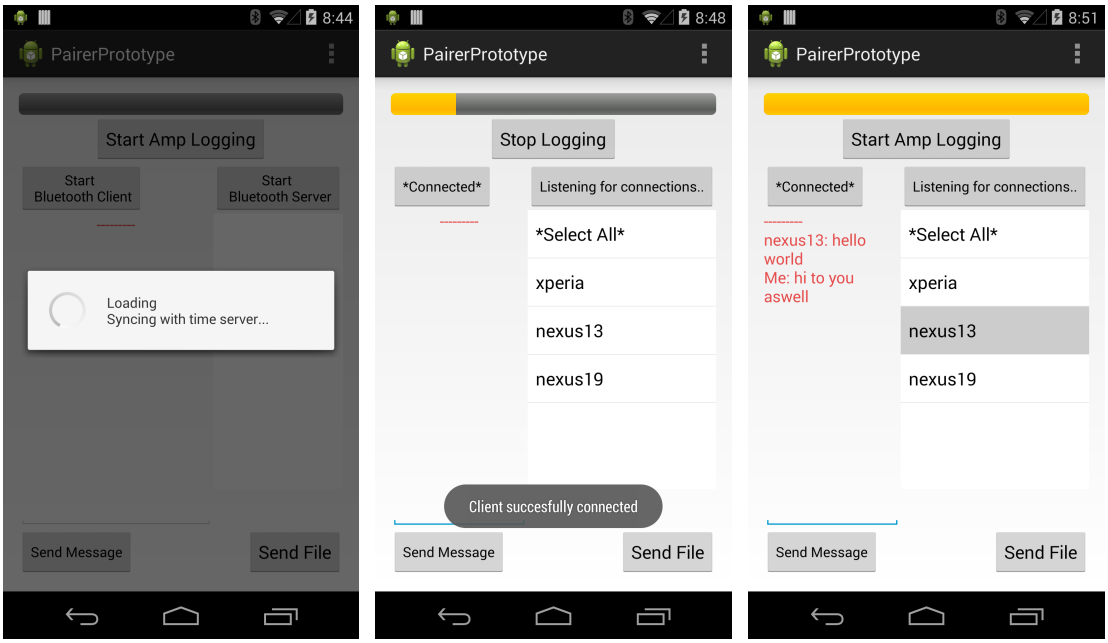
Appendices

A Github Repositories

The source code of the approach proposed in this thesis is available on GitHub.

- The server-side `https://github.com/jaks6/PairerPrototypeServer`
- The client-side `https://github.com/jaks6/PairerPrototype`

B Client Application Screenshots



(a) (b) (c)

Figure 7: Various states of the client application.

C Test Round Two Data

This table shows 70 instances of clusterings run in testing round 2. The highlighted row shows the actual device groupings, rest of the rows show clustered results.

#	Cluster1	Cluster 2	Cluster 3	Errors
1	Nexus13;Nexus11;Nexus17	Nexus19;Nexus14;Asus;Xperia		0
2	Nexus11;Nexus17;Nexus13	Nexus14;Asus;Xperia;Nexus19		0
3	Nexus19;Xperia;Nexus14	Nexus11;Nexus13;Nexus17		1
4	Xperia;Asus;Nexus14;Nexus19	Nexus17;Nexus11;Nexus13		0
5	Nexus17;Nexus13;Nexus11	Nexus14;Nexus19;Asus;Xperia		0
6	Xperia;Asus	Nexus19;Nexus14	Nexus17;Nexus11;Nexus13	2
7	All devices			4
8	Nexus13;Nexus17;Nexus11	Nexus19;Asus;Nexus14;Xperia		0
9	Nexus19;Asus;Nexus14	Nexus11;Nexus13;Nexus17		1
10	All devices			4
11	Nexus19;Xperia;Nexus14	Nexus17;Nexus11;Nexus13		1
12	Xperia;Nexus14;Nexus19;Asus	Nexus17;Nexus11;Nexus13		0
13	Xperia;Nexus14;Asus;Nexus19	Nexus17;Nexus11;Nexus13		0
14	Nexus17;Nexus13;Nexus11	Xperia;Asus;Nexus14;Nexus19		0
15	Xperia;Nexus19;Asus;Nexus14	Nexus13;Nexus11;Nexus17		0
16	All devices			4
17	Nexus14;Xperia;Asus;Nexus19	Nexus17;Nexus13;Nexus11		0
18	Nexus19;Asus;Xperia;Nexus14	Nexus11;Nexus17;Nexus13		0
19	All devices			4
20	Nexus17;Nexus13;Nexus11	Asus;Nexus14;Xperia;Nexus19		0
21	Nexus14;Xperia;Nexus19;Asus	Nexus13;Nexus17;Nexus11		0
22	Nexus13;Nexus11;Nexus17	Xperia;Nexus19;Asus;Nexus14		0
23	Xperia;Nexus14;Asus;Nexus19	Nexus17;Nexus13;Nexus11		0
24	All devices			4
25	Nexus17;Nexus11;Nexus13	Nexus19;Asus;Nexus14;Xperia		0
26	Nexus19;Xperia;Asus;Nexus14	Nexus13;Nexus17;Nexus11		0
27	Nexus13;Nexus17;Nexus11	Nexus19;Xperia;Nexus14;Asus		0
28	Nexus13;Nexus11;Nexus17	Xperia;Nexus19;Asus;Nexus14		0
29	All devices			4
30	Xperia;Nexus19;Nexus14;Asus	Nexus11;Nexus13;Nexus17		0
31	Nexus19;Nexus14;Xperia;Asus	Nexus11;Nexus17;Nexus13		0
32	Nexus17;Nexus11;Nexus13	Nexus19;Nexus14;Asus		1
33	Asus;Nexus19;Nexus14;Xperia	Nexus17;Nexus13;Nexus11		0
34	All devices			4
35	Nexus19;Xperia;Asus;Nexus14	Nexus13;Nexus17;Nexus11		0
36	All devices			4
37	Nexus11;Nexus13;Nexus17	Xperia;Nexus19;Nexus14;Asus		0

#	Cluster1	Cluster 2	Cluster 3	Errors
38	Nexus11;Nexus13;Nexus17	Xperia;Nexus19;Asus;Nexus14		0
39	Xperia;Asus;Nexus14;Nexus19	Nexus13;Nexus11;Nexus17		0
40	Xperia;Nexus14;Asus;Nexus19	Nexus17;Nexus11;Nexus13		0
41	Nexus19;Xperia;Nexus14;Asus	Nexus13;Nexus17;Nexus11		0
42	Nexus17;Nexus13;Nexus11	Xperia;Nexus14;Nexus19;Asus		0
43	Nexus14;Nexus19;Asus;Xperia	Nexus13;Nexus11;Nexus17		0
44	Nexus11;Nexus17;Nexus13	Nexus19;Xperia;Asus;Nexus14		0
45	Nexus19;Xperia;Nexus14;Asus	Nexus11;Nexus17;Nexus13		0
46	Nexus13;Nexus17;Nexus11	Xperia;Nexus14;Nexus19;Asus		0
47	Xperia;Nexus19;Nexus14;Asus	Nexus13;Nexus17;Nexus11		0
48	Nexus11;Nexus13;Nexus17	Xperia;Asus;Nexus19;Nexus14		0
49	Nexus14;Xperia;Asus;Nexus19	Nexus13;Nexus11;Nexus17		0
50	Nexus11;Nexus17;Nexus13	Nexus19;Xperia;Asus;Nexus14		0
51	Nexus13;Nexus11;Nexus17	Nexus14;Nexus19;Xperia;Asus		0
52	Nexus13;Nexus11;Nexus17	Asus;Nexus19;Nexus14;Xperia		0
53	All devices			4
54	Asus;Nexus14;Nexus19;Xperia	Nexus13;Nexus17;Nexus11		0
55	Nexus11;Nexus17;Nexus13	Xperia;Nexus14;Nexus19		1
56	Nexus14;Asus;Nexus19;Xperia	Nexus13;Nexus17;Nexus11		0
57	Nexus11;Nexus13;Nexus17	Nexus19;Asus;Xperia;Nexus14		0
58	Nexus19;Xperia;Asus;Nexus14	Nexus13;Nexus11;Nexus17		0
59	Xperia;Asus;Nexus19;Nexus14	Nexus13;Nexus11;Nexus17		0
60	Nexus17;Nexus13;Nexus11	Asus;Xperia;Nexus19;Nexus14		0
61	All devices			4
62	Nexus19;Nexus14;Xperia;Asus	Nexus13;Nexus11;Nexus17		0
63	Nexus19;Nexus14;Xperia;Asus	Nexus11;Nexus13;Nexus17		0
64	Nexus19;Xperia;Asus;Nexus14	Nexus11;Nexus17;Nexus13		0
65	Asus;Nexus19;Xperia;Nexus14	Nexus17;Nexus13;Nexus11		0
66	Nexus19;Nexus14;Asus;Xperia	Nexus13;Nexus11;Nexus17		0
67	Nexus11;Nexus17;Nexus13	Asus;Nexus14;Nexus19;Xperia		0
68	Nexus11;Nexus13;Nexus17	Nexus19;Asus;Nexus14;Xperia		0
69	All devices			4
70	Nexus17;Nexus13;Nexus11	Nexus19;Xperia;Nexus14;Asus		0

Non-exclusive licence to reproduce thesis and make thesis public

I, Jakob Mass (date of birth: 22.01.1992),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

"Automatic Mobile Device Pairing via Integrated Microphones"

supervised by Satish Narayana Srirama

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 14.05.2014