DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

# Crowdsourcing mobility data with privacy preservation through decentralized collection and analysis

Simon van Endern

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

# Crowdsourcing mobility data with privacy preservation through decentralized collection and analysis

# Crowdsourcing von Mobilitätsdaten ohne Einschränkung der Privatsphäre durch dezentrales Sammeln und Analysieren

| | |
|---|---|
| Author: | Simon van Endern |
| Supervisor: | Prof. Dr.-Ing. Jörg Ott |
| Advisor: | Trinh Viet Doan |
| Submission Date: | 30.06.2019 |

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 30.06.2019 Simon van Endern

# Acknowledgments

# Abstract

We propose a method to publish location ata without raising privacy concerns.

As still this data could be useful for many stakeholders, we will investigate how on the one hand aggregated data can be published without imposing any privacy risk to the owners of the data and on the other hand develop a prototype of a mobile application through which this location data is aggregated in a decentralized manner so that the raw user data never leaves the users' device.

# Contents

**Acknowledgments** **iii**

**Abstract** **iv**

**1. Introduction** **1**
- 1.1. Motivation . . . . . . . . . . 1
- 1.2. Research Questions . . . . . . . . . . 2
- 1.3. Contributions . . . . . . . . . . 3

**2. Related Work** **4**
- 2.1. Classification of location data usage according to acceptable delay . . . 4
- 2.2. Research has identified the following privacy problems . . . . . . . . . . 4
  - 2.2.1. Central databases itself pose a risk due to possible theft . . . . . 4
- 2.3. Inference attacks on published anonymized data . . . . . . . . . . 5
  - 2.3.1. Inferring home and work location from consecutive data samples 5
  - 2.3.2. Solutions / Countermeasures to prevent inference attacks . . . . 5
  - 2.3.3. Problems still after countermeasures . . . . . . . . . . 6
  - 2.3.4. All methods depend on trust to a third party or the provider itself 6

**3. Methodology** **7**
- 3.1. Aggregation schemes . . . . . . . . . . 9
- 3.2. Narrowing the area of the aggregation request . . . . . . . . . . 10

**4. Design and Implementation** **12**
- 4.1. Technology Stack . . . . . . . . . . 12
- 4.2. API . . . . . . . . . . 13
  - 4.2.1. Data Aggregation Design . . . . . . . . . . 14
- 4.3. Android Application . . . . . . . . . . 16
  - 4.3.1. Separation of concerns regarding Data Collection and Aggregation 18
  - 4.3.2. Data Collection . . . . . . . . . . 18
  - 4.3.3. Local Data Aggregation . . . . . . . . . . 19
  - 4.3.4. Serving Aggregation Requests . . . . . . . . . . 20

4.4. Server Design and Implementation . . . . . . . . . . . . . . . . . . . . . 21
    4.4.1. Data Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
    4.4.2. request-chain . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

**5. Performance and Evaluation** **24**
5.1. Deployment . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
5.2. Data Consumption . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
5.3. Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
5.4. Implications . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
5.5. Scenario Traffic Jam . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31

**6. Conclusion and Discussion** **32**
6.1. Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
6.2. Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
6.3. Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
6.4. Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

**Appendix A. Data Usage Screenshots** **36**

**List of Figures** **47**

**List of Tables** **48**

**Bibliography** **49**

# 1. Introduction

## 1.1. Motivation

"Data is the new oil" is a quote [5, 7] many people agree with. More and more businesses are based not on specific production capacities but on data and especially the ability to process it and the exclusive owenership over it. The success and monopoly of companies like Google, Facebook and Amazon can be attributed to this exclusive ownership at least to a reasonable extend.

While patents that used to power companies' success provide a balance through granting exclusive rights having to make the knowledge public, many companies e.g. Coca cola have decided successfully not to go for a patent and thus not reveal their knowledge [16]. If that approach is not compromised, it guarantees both - non-disclosure and also exclusive rights. Similarly, the non-disclosure of huge data sets collected by Facebook, Google and Amazon circumvent the balance intended by patents. Though, the unavailability of huge amounts of data to the public is an impediment of innovation and increased growth. For example, cities would benefit from aggregated location data in order to optimize traffic scheduling as also highlighted by [9]. Nevertheless, the publication of raw data sets is impossible because it severly intrudes the privacy of the owners of the data.

So, even if companies would agree on a publication, a problem arises. There is a conflict between preserving user privacy and publishing user data.

Nevertheless, user privacy is already compromised even without publication of user data. Already the mere existence of central data sets pose a privacy risk to users, because security issues might allow for theft and unwanted publication of these data. An example is the theft of 14 million users' data from facebook [8].

Some governments and other institutions already publish some of their data sets after anonymizing them e.g. through cloaking of data so that it achieves k-anonymity and there are crowdsourcing and open source approaches to make data available to everybody. Nevertheless, the applied anonymization is often not sufficient or at least critical if the resulting data set should still be useful. Research shows that inferences can be drawn from the published data sets that violate the respecitve users' privacy [4, 22]. Also, most research is based on datasets that span only over a few days or weeks. When collecting location data over longer periods, the chance of successful inference attacks increases in all cases with more data available. So, in addition to the main risk of a central data set, publishing anonymized data poses another risk to users' privacy.

Furthermore, besides the remaining risk of inference attacks on published anonymized data sets, the anonymization through those algorithms always depend on a trusted server to collect the data from all users and then publish the results of any analysis applying privacy-preserving algorithms beforehand. So even if the data is only stored anonymized on the server, besides the remaining risk of inference attacks, this still imposes a high privacy risk to every user, as trust can be misued by the trusted server itself.

## 1.2. Research Questions

In order to solve the problems, we pose the following research questions. (We assume trust to the server)

### RQ1: Is aggregation possible without raw data being accessible to anybody but the owner?

Previous work has shown that in order to overcome the conflicit between privacy intrusion and (public) data availability, a solution is needed that gets along without storing raw data in a central data set. This solution should eliminate the risk of leaking raw user data through theft from a centralized database and also eliminate the remaining risk of inference attacks on published believed-to-be-anonymized raw data. We investigate whether there is a solution and what are the limitations, respectively which assumptions must hold for the solution to be valid.

### RQ2: What types of aggregations (e.g. mean, median) can be published?

We investigate how many details can be published without user privacy being compromised.

### RQ3: What is the risk of inference attacks on aggregated data due to overlappings in the covered timespan?

Previous work has mostly covered inference attacks on anonymized raw data sets. We will investigate, whether the same problems of inference attacks are possible on aggregated data as well and examine the limitations of publishing aggregated data.

## 1.3. Contributions

For our solution, we will focus on the sub-area of location data and location privacy. We investigate the possibility of storing raw location data only decentralized on the collecting devices. On a central server available to the public, only aggregated data is stored, thus the main problem of privacy risk by a central database containing the overall raw data set is solved. We assume the central server to be trusted and also the clients to be trusted. As the aggregation process happens decentrally, the central server will never hold any other than aggregated data and never know about individual raw data.

The structure of our research is organized as follows: First we review related work in the areas of location privacy and anonymisation techniques. Chapter 3 outlines our approach of decentralized data analysis and chapter 4 documents the setup and explains design decisions. While we present the solution and the results of field-testing our setup in 5, chapter 6 highlights limitations and future work. It also topics the limitations as of trust to the server and the clients and gives a short outline how this problem can be solved.

In summary, our approach takes the opposite direction as todays standard. We do not first collect the whole data set and then reduce it to a data set meeting privacy-constraints but we start from the bottom up - first by performing analysis in a decentralized manner so that there never is an overal data set imposing a security risk on all the entries' users, and second by proposing a framework that only releases aggregated data where no interference of any user information is possible. This data will then be available to the public. This gives us maximum possible feedback on eventual privacy problems, creates trust through transparency and fosters innovation through availability of data to everyone.

# 2. Related Work

## 2.1. Classification of location data usage according to acceptable delay

In order to review existing approaches and research, we classify location aware services by the acceptable delay of the location information being available. Similar to the classification made by [9], we define three categories:

1. Almost no delay tolerance: e.g. an application showing a pop-up about a nearby venue e.g. a coffe shop when a pedestrian passes
2. Some delay tolerance e.g. one minute is acceptable: An application e.g. google maps derives the information of congested traffic from devices reporting their GPS data which show lower than usual speed. As congestions worth reporting last longer than one minute, some delay in the device's information reaching the server is acceptable.
3. Significant delay tolerance of hours, days or even weeks is acceptable for historical and statistical use of location data e.g. to find out about popular visiting times

Most research investigates user's privacy in case 3 [14, 4, 6, 22] where the delay of the data being available for processing is not an issue. Some research deals with case one [1, 2] where solutions can be provided e.g. as in [15] where not the exact location is send to a server but a region and the server sends a list of possible matches that is then matched locally with the exact location. Our research targets case 3 in which a significant delay is acceptable.

## 2.2. Research has identified the following privacy problems

### 2.2.1. Central databases itself pose a risk due to possible theft

Centralized databases containing raw location data expose users to a privacy risk (through theft) [19, 10]. [12] proposes the use of P2P over WIFI and Bluetooth to decrease the need of central instances. A decentralized analysis approach and it's implications for data privacy is also investigated by [19] as an alternative to cloud-based IoT. [13] proposes an approach in which raw data is hidden from the central instance but still aggregated data can be obtained by using encryption methods. Raw data is encrypted using a modified approach of public-private key-pair cryptography in which the sum of two encrypted messages can be decrypted to the sum of the encrypted messages. Furthermore, only a number of messages above a certain threshold can be encrypted this way using the different encryption shares. While this approach is close to our work in general, we propose a more general and flexible setup. Another approach by [10] also uses encryption in combination with a middleware. The server storing the data and the participant (e.g. a vehicle) share a symmetric key which is stored securely in the vehicle. The middleware ensures authentication of the participant, and forwards the locaation to the central server. Nevertheless, several research as e.g. [14, 22, 4] have shown that it is easy to infer identity from such datasets. Even though they solve the problem of user authentication, it seems hardly feasable as the symmetric key has to be pre-installed on every participating device.

## 2.3. Inference attacks on published anonymized data

### 2.3.1. Inferring home and work location from consecutive data samples

Research has shown, that from a location data set that is pseudonymous, i.e. the identifiers have been stripped or anonymized from the data, it is possible to infer the home location of single users through so-called inference attacks [14, 4, 6, 10, 22] and also the work location with a slightly lower probability [4, 6]. The same problem arises when using data collected through crowdsourcing [13]. These home locations or home-work location pairs can then be used to look up the corresponding user's identity e.g.by combining it with publicly available information e.g. reverse map coding of coordinates to addresses and then searching for entries in telephone books to infer the users identity from it's home location [14, 6, 10]. Especially in suburban areas this is quite successful as usually one house can be mapped to only one person or family. This identity can then be linked to other sensitive data, e.g. locations visited by the identified user. This problem also arises in the area of IoT [19, 10].

In general, the problem can be summarized as not adhering to k-anonymity.

### 2.3.2. Solutions / Countermeasures to prevent inference attacks

In order to protect users' privacy but still be able to publish data, several approaches have been investigated. Spatial cloaking: Achieving k-anonymity by dropping data points or perturbing them or dropping all data points around a random point around the home location [14]. Also obfuscating locational data close to a home location and mapping GPS points to the next street crossing is possible to increase anonymity. More sophisticated approaches: [11]

### 2.3.3. Problems still after countermeasures

Nevertheless, data suppression algorithms have only limited success and can only reduce, but not eliminate the risk [10].

#### Data is useless if k-anonymity is guaranteed

Insufficient accuracy / the data set becomes useless [14, 4, 17, 21, 20].

#### Countermeasures not effective in sparsely populated areas

Anonymization techniques might score well in densly populated areas or areas with high traffic but poorly in sparsely populated areas especially where a single address can be mapped to a single person or family [11, 1, 10] [location-privacy correct paper or cited wrong paper???] or might not work for individuals whos work and home location are further away than average [6].

#### Still privacy breaches possible

- More advanced privacy breaking algorithms
- Taking other sources into accouont, e.g. history of location data Extending the time period over which data is collected generally increases the risk.
- quasi-identifiers not thought of

### 2.3.4. All methods depend on trust to a third party or the provider itself

Still all approaches depend on first centrally collecting the original raw data and then before querying [21] applying anonymization techniques.

Also [15] proposes a solution (close to our summary) how to enable privacy for instant use of location data.

TODO: Relate to [21]

# 3. Methodology

In order to avoid having a central raw dataset and to eliminate the risk of inference attacks on anonymized data, we propose a framework which collects and locally aggregates location data on end user devices (smartphones) through an application designed for this purpose. The raw data will stay on each device and will only be used to serve aggregation requests send from the central server. The aggregation requests have to be defined upfront. An example is the determination of the average number steps per day accross all users participating in the respective aggregation. An examplary incoming aggregation request is depicted in figure 3.1. It specifies the timespan which should be covered by the aggregation and the type of aggregation.

```
{
        "start": "2019-05-30",
        "end": "2019-06-02",
        "type": "steps",
        "n" :3,
        "value": 2000
}
```

Figure 3.1.: Incoming aggregation request

The response send back to the server after processing the request only needs to contain the data itself, as the type and timespan can be inferred by the server from the request identifier. Figure 3.2 shows an examplary response to the server that matches the request in figure 3.1.

```
{
        "n" :4,
        "value": 2500
}
```

Figure 3.2.: Outgoing aggregation response

Figure 3.3.: Decentral unencrypted aggregation process via P2P

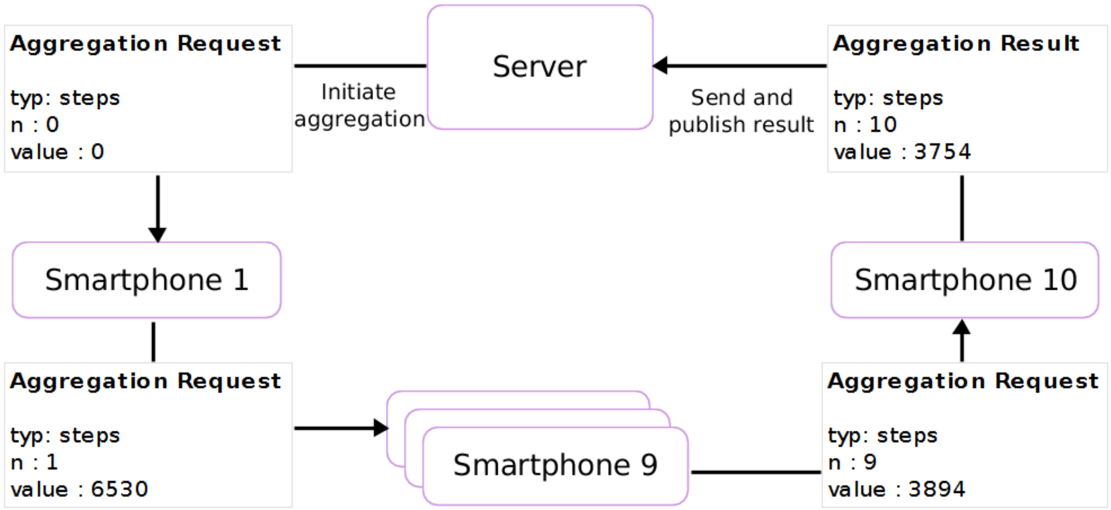
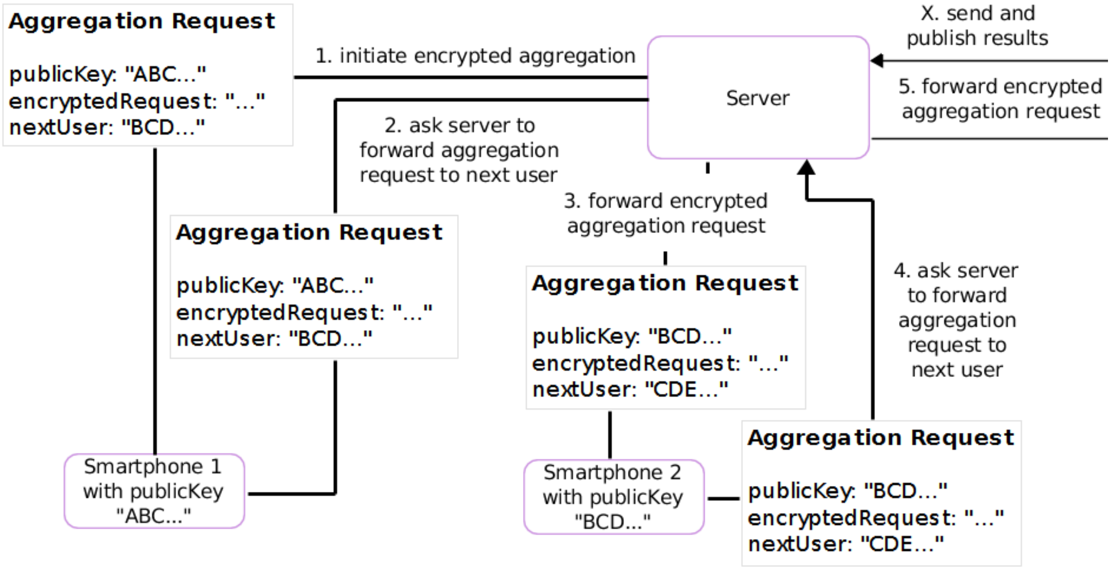

Aggregation Request
typ: steps
n : 0
value : 0
Initiate aggregation
Server
Send and publish result
Aggregation Result
typ: steps
n : 10
value : 3754
Smartphone 1
Smartphone 10
Aggregation Request
typ: steps
n : 1
value : 6530
Smartphone 9
Aggregation Request
typ: steps
n : 9
value : 3894

Figure 3.3 depicts the overal process of such a decentral aggregation request using an example of 10 participating devices. In order to protect the user's privacy and completely shield the raw data from the server, it would be necessary to pass the request via P2P from one device to another until the last device finally sends the results to the server. P2P on mobile phones though is hardly possible, if the devices are not locally close to each other [12]. On the other hand, if the server is used to pass an aggreation request from one device to another, it could read the data and compute the respective user's input from the difference. We propose to use encryption in order to hinder the server from reading the data and assume the server to be trusted in the first place. Chapter 6 discusses this limitation. The modified aggregation process is depicted in figure 3.4. On installation of the application on a smartphone, a public-private key-pair is generated and every installed application registers at the server with this public key. The corresponding private key is stored locally. On start of an aggregation request, not only the first user but also the following user who should deal with the aggregation request is determined and the public key of the next user is passed along with the aggregation request. When one end user device needs to send the processed aggregation request to the next phone, it encrypts the data using the provided public key of the next user in the standard hybrid encryption[1] approach leveraging the benefits of synchronous keys. This way the next phone in the aggregation chain will be able to decrypt the request and process the data while the server is unable to read the data until the aggregation request is finally send in plain text for publishing to the server. This process currently assumes a trusted server and trusted devices. Chapter 6 discusses these limitations.

---

[1] In hybrid encryption as used in SSL, the message itself is encrypted with a synchronous key while this key itself is encrypted using the public key

Figure 3.4.: Decentral encrypted aggregation process


Aggregation Request
publicKey: "ABC..."
encryptedRequest: "..."
nextUser: "BCD..."
1. initiate encrypted aggregation
Server
X. send and publish results
5. forward encrypted aggregation request
2. ask server to forward aggregation request to next user
3. forward encrypted aggregation request
Aggregation Request
publicKey: "ABC..."
encryptedRequest: "..."
nextUser: "BCD..."
Aggregation Request
publicKey: "BCD..."
encryptedRequest: "..."
nextUser: "CDE..."
4. ask server to forward aggregation request to next user
Smartphone 1 with publicKey "ABC..."
Smartphone 2 with publicKey "BCD..."
Aggregation Request
publicKey: "BCD..."
encryptedRequest: "..."
nextUser: "CDE..."

The aggregated results are planned to be available through a public route on the server. Nevertheless, we restricted access in out test run to our research team in order to protect the research participants' privacy in case there is a privacy risk we have not thought of. The aggregation requests themselves are initiated by our research team but can be initiated automatically on a regular basis in the future.

## 3.1. Aggregation schemes

We found two types of aggregation requests to be of interest. First, the evalulation of mean values and second, the evaluation of more advanced statistical values such as median or other percentiles and distribution. The latter includes the possibility to calculate the former. Nevertheless, we want to test all research questions independently. We found the following aggregations to be especially of interest:

1. Computing the average number of steps walked across all users participating in

the request. (e.g. to calculate how many people reach the 10.000 steps per day[2].)

2. Computing the average time spent walking, running, in a vehicle or on a bicycle[3].
3. Computing how many people respectively which share of people combine using a bicycle with using a vehicle such as public transport or a car in one trajectory.
4. How much time do people stay at work.
5. How long do people travel to work.
6. Where did many participants spend a significant amount of time on a certain day? (Event recognition)
7. What percentage of whole travelling time do people spend on their bike, car, ...
8. What is the average speed on roads.
9. Collecting a list of the average number of steps walked by each participant during the timespan.
10. Collecting a list of the duration of all registered activities.
11. collecting a list of all trajectories registered by the users' phones.

For all these and more aggregations, always both, the mean value and if possible, a complete list of values to compute other statistical figures are of interest.

## 3.2. Narrowing the area of the aggregation request

The aggregation requests outlined in the former subsection only provide useful data if the area of the aggregation can be limited e.g. to the scope of a city. Otherwise the resulting data would not allow for comparison and the scope of each aggregation would either be the whole user base which especially in the case of listing values would result in a huge amount of data passed around. Or, the limited number of participants in each aggregation would not be locally close which would 1. not result in useful data and 2. make it impossible to do aggregations as the average speed on roads. In order to limit the area of aggregation and avoid sending the aggregation request to each user and leaving him with determining whether he is inside the area and participates in the request, the initiator of those requests has to now the location of users, respectively the location where the user is active the most. We do not see this as a violation of the user's privacy due to the following reasons:

---

[2] It has to be evaluated, which percentage of steps are registered because the phone will not always be on the person

[3] https://developers.google.com/android/reference/com/google/android/gms/location/DetectedActivity

1. The exact location of the user e.g. the home or work location is not of interest at all. Rather the area for which the user can provide data is of interest. We propose to cluster location areas in a hierarchical structure similar to [15] and determine the granularity of the location published to the server as follows:
   a) Each user sends the most coarse locational area as possible to the server - e.g. the continent.
   b) If more than the required anonymity threshold of e.g. 10 active users are already registered with this location at the server, the server not only links this location to the user but also requests the user to send a less coarse location.
   c) The user step by step sends a less coarse location e.g country, district, etc. until the server denies the location because not enough active users have registered with the location on this granularity level. It nevertheless increases the counter of users who requested access to this location. Once the counter exceeds a certain number, the server sends an aggregation request to participants in the more coarse area encompassing the requested less coarse area to determine the number of active users. If this number is above the required threshold, the granularity level for this location is made available and users can now register with this area and aggregation requests targeting this area can be send.
   d) The fullfilment of the threshold has to be checked on a regular basis in order to close areas once the user base sinks below the anonymity threshold.

   In a more advanced setting, it should be possible to register with more than one area on the same level to avoid that a user e.g. living close to a city provides only data about the city or the bordering area. Nevertheless, 1. this should be limited to areas bordering each other to avoid user identification similar as in XX and 2. it has to be investigated whether this indeed not poses a risk or whether also the combination of areas needs to meet a certain anonymity threshold.

# 4. Design and Implementation

## 4.1. Technology Stack

In order to implement the architecture proposed in chapter 3 we choose Android as end user device platform. Android has the highest market share [18] among mobile devices and offers a healthy ecosystem of frameworks and libraries that simplify development. Furthermore we opted for an implementation in Kotlin instead of Java to reduce boilerplate code and improve readability.

As server side technology we choose node.js in conjunction with a mongoDB object store database. A NoSQL database like mongoDB provides flexibility and easy adaption of data schemes without much overhead and thus perfectly fits our prototyping purpose. We choose node.js out of the same reasons. In contrast to a statically typed language, javascript provides more flexibility and ease of change. Furthermore, node.js is often used in combination with mongoDB and provides seamingless integration.

The results can be made available to the public either via a dedicated route of the server or directly through granting access to the respective collection[1] of the database.

The following sections describe the implementation of the Android application and the server and explain design decisions. We start with an outline of the API shared by both, the Android application and the server.

---

[1] A collection in an object store is the equivalent to a table in an SQL database.

## 4.2. API

The API is designed as a pure REST API based on JSON and consists of the following endpoints:

- POST /user HTTP/1.1
  for creating a new user (see figure 4.1 and 4.2).
- GET /requests?pk=... HTTP/1.1
  for retrieving aggregation requests for a user (see figure 4.3).
- POST /forward HTTP/1.1
  for sending a processed aggregation request back to the server (see figure 4.4).
- GET /aggregations HTTP/1.1
  for retrieving all completed aggregation results (see figure 4.5).
- POST /admin/sampleRequest HTTP/1.1
  for starting a new aggregation request (see figure 4.6).

```
{
  "publicKey": "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG..."
}
```

Figure 4.1.: POST /user

```
{
  "publicKey": "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG...",
  "password": "LSFDfzduSFSozfwf"
}
```

Figure 4.2.: Response to POST /user

The routes used for interaction with the application (except the one for creating a new user are secured and can only be used if the user can successfully be authenticated. The routes for starting new aggregation requests and retrieving all results require administrator authentication. The route for the results is designed to be available to the

```
[{
  "publicKey": "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG...",
  "serverId": "5cfbd6734e733a0079ee5c0f",
  "nextuser" :"-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhBms...",
  "encryptionKey":"HXporX7tqrgkf6Cp8OlqOcpLb...",
  "iv":"pMxQ2nHqbz1Zu24Lfi5HYQ==",
  "encryptedRequest":"xtgP2e9AeAK8hfPXQe03..."
}]
```

Figure 4.3.: Response to GET /requests

```
{
  "publicKey": "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG...",
  "password": "LSFDfzduSFSozfwf",
  "serverId": "5cfbd6734e733a0079ee5c0f",
  "encryptionKey":"HXporX7tqrgkf6Cp8OlqOcpLb...",
  "iv":"pMxQ2nHqbz1Zu24Lfi5HYQ==",
  "encryptedRequest":"xtgP2e9AeAK8hfPXQe03..."
}
```

Figure 4.4.: POSt /forward

public without authentication. Nevertheless we restrict access only to result types that have been examined to fully protect users' privacy. So, for example, our experimental trajectory request type is not available to the public.

The encryptedRequest field e.g. in figure 4.3 contains the encrypted version of the request described in chapter 3.

### 4.2.1. Data Aggregation Design

All aggregations proposed in section 3.1 can be implemented using only 4 fields: An Integer $n$ specifying the current number of participants, a Floating Point Number *value* e.g. containing the current mean value of the aggregation, a list *valueList* of Floating Point Numbers that can be reused for different purposes accross requests and a String *type* that specifies the request and the used scheme how the other fields are populated e.g. the encoding of the *valueList*. In addition, all aggregation requests have a start day and an end day. Days' time is always treated as 00:00 o'clock. Thus, the timespan for an aggregation is always a multiple of 24 hours.

```
[{
  "timestamp": 1559216514521,
  "startedAt": 1559216514685,
  "start": "2019-05-28",
  "end": "2019-05-31",
  "type": "steps",
  "n": 6,
  "value": 4271,
  "valueList": []
},
{
  "timestamp": 1559216514521,
  "startedAt": 1559216514685,
  "start": "2019-05-28",
  "end": "2019-05-31",
  "type": "stepsListing",
  "n": 5,
  "value": 0,
  "valueList": [1661,1246,3195,7714,7842]
 }]
```

Figure 4.5.: Response to GET /aggregations

```
[{
  "password": "adminPassword",
  "start": "2019-05-31",
  "end": "2019-06-05",
  "type": "activity_2",
}]
```

Figure 4.6.: POST /admin/insertDample

Due to limited scope of this thesis, only the following out of the aggregation requests defined in section 3.1 were implemented:

1. Computing the average number of steps walked across all users participating in the request.
2. Computing the average time spent walking, running, in a vehicle or on a bicycle[2].
3. Collecting a list of the average number of steps walked by each participant during the timespan.
4. collecting a list of all trajectories registered by the users' phones.

All aggregations work as a prove of concept of the proposed setup and hence serve answering research question 1. Aggregation 1, 2 and furthermore help to investigate which types of aggregations are possible to be published and allow for an investigation whether overlapping timeperiods allow for inference attacks. Thus they help to answer research question 2 and 3. Aggregation 3 was implemented especially in order to test research question 2. Aggregation 4 imposes a privacy risk discussed in [22, 4, 14] on the user and was implemented in order to get an overview of the data quality of our setup and validate the feasability of the other aggregation requests proposed in section 3.1.

## 4.3. Android Application

The Android application targets Android Oreo (API level 27) and requires a minimum API level of 19. Approximately 96.8% of devices run on this or a higher version of Android [3] which allows our application to be installed on the majority of Android devices. Furthermore, the application leverages Goolge Play Services to obtain GPS and activity data. Without Google Play Services installed, the application will not work. In order to ease future adaptability, we chose to use the Dagger2[3] framework for dependency injection in order to decouple classes as far as possible. Further use of frameworks and libraries will be explained in the following respective sections.

---

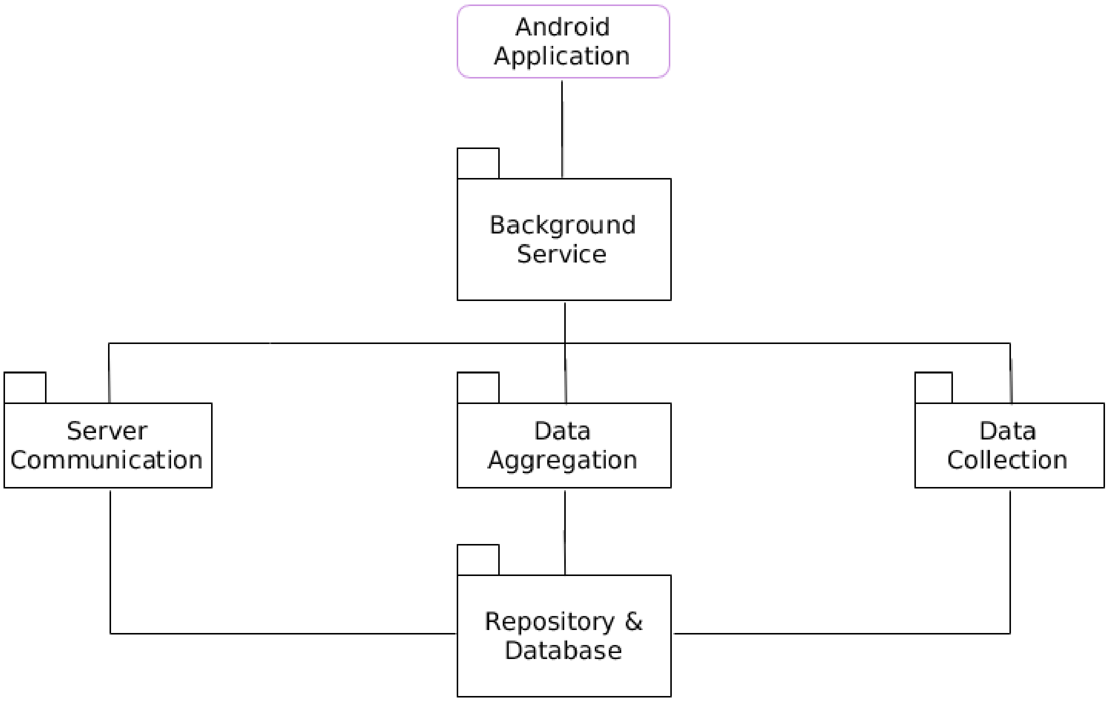
[2] See https://developers.google.com/android/reference/com/google/android/gms/location/DetectedActivity

[3] See https://dagger.dev/

Th application is aimed to collect GPS data, detect the user's current activity and count the user's steps. The data collection process happens in the background without any user interaction needed. Aggregation requests to aggregate data accross devices are also served without any user interaction needed. The android application can be grouped into three main parts of loosely coupled modules:

- A module responsible for collecting and saving raw data
- A module responsible for locally aggregating raw data
- A module responsible for communicating with the server and handling aggregation requests

Figure 4.7.: Android architecture



The control flow as depicted in figure 4.7 is as follows: The application has only one Main Activity in order to ask the user to allow location access and start the background services. Apart from that the only Activity does not serve any specific purpose. The local aggregation as well as the polling of new requests from the server happens in the background on a 15 minute interval. The Android Workmanager controls this periodic work without any user interaction being required.

For the App in order to have maximum possibilities collecting esspecially GPS data and preventing the Android operating system from shutting down when not interacted with by the user (which is usually never the case), a non-dismissible status notification is displayed at all time[4]. Also the application is registered to be automatically restarted upon boot[5] and also when the application is closed by the user (e.g. via the task manager) so that once installed, no further user interaction is necessary. Furthermore, the application, respectively each module is heavily unit tested in order to guarantee functionality and facilitate further development by other research teams. Unit and integration tests are based on AndroidJUnit4[6] and the expresso[7] framework.

### 4.3.1. Separation of concerns regarding Data Collection and Aggregation

We choose to separate the aggregation and collection of location data in order to decouple the modules and provide the possibility to extend the model of aggregated data in the future without the need to change the raw data model. Vice versa the data collection process can be modified without impacting the aggregation process.

### 4.3.2. Data Collection

We use the Android Room Persistence library[8] to locally store data. The library provides a layer over the standard LiteSQL database commonly used in many Android applications. We collect three types of data:

- Steps: If available, the phone's internal step sensor provides updates on a regular basis. The step sensor always informs about the total number of steps since the last reboot. Upon each time we receive data from the step sensor, this data is stored directly in the *step_counter_table*.
- User's activities: The Google Play Services activity recognition API leverages different data and sensors available on the phone in order to inform about the most probable current activity of the user as one of *still, walking, running, in a*

---

[4] Compare to the non-dismissible status notification displayed by Google Maps when the navigation system is active

[5] From Android 6.0 on (API level 23), restricts apps' behaviour in order to reduce battery consumption. E.g. all apps are automatically managed by the battery manager which restricts background launches. The user has to switch this option to manual management in order to allow the app to function in the way it was designed. See https://developer.android.com/guide/background/ and also https://developer.android.com/training/monitoring-device-state/doze-standby

[6] See https://developer.android.com/reference/android/support/test/runner/AndroidJUnit4

[7] See https://developer.android.com/training/testing/espresso

[8] See https://developer.android.com/topic/libraries/architecture/room

*vehicle, on bicycle*. Whenever there is a change detected, two events are fired - one for exiting the former and one for entering the current activity. The events might not be dispatched instantly but contain the timestamp of the exact occurence. Upon each received event, this data is stored directly in the *activity_transition_table*.

- GPS positions: GPS data is retrieved through the *FusedLocationProviderClient* which leverages cellphone-tower and WIFI data apart from GPS to determine the position. In order to limit battery consumption, GPS data is only requested every minute if the device's detected activity is *still*[9]. If the current detected activity is *walking*, the interval is set to 5 seconds and in any other state, the interval is set to every second. The data is stored in the linked tables *gps_data_table* and *gps_location_table*. We choosse to separate the GPS point itself from the timestamp having in mind that future aggregations might need or leverage the separation of spatial data and time and more than one event might be attached to the same GPS point.

### 4.3.3. Local Data Aggregation

From the received values of the step counter since last reboot saved in *step_counter_table* the daily steps are computed and stored in *steps_table*. The exit and enter events received via the activity recognition framework and stored in the *activity_transition_table* are matched in order to compute activities with start and duration. Those activities are then saved in the *activity_table*. GPS data is used to compute trajectories through the following algorithm:

1. When there are more than 10 minutes between two subsequent GPS points in the sequence of all GPS points to be processed, the sequence is separated into two separate sequences and each is processed separately as a possible trajectory in the next step.
2. First, we identify still moments - periods of no movement - as follows:
    a) For each GPS point, we identify a subsequent GPS point that was registered at least two minutes after the first one.
    b) If the average speed between those two points was below 0.6 m/s, the pair is added to a list to be processed in the next step.
    c) The list of pairs of GPS points resulting from the last step is fused into sequences of GPS points as long as possible: Whenever two pairs overlap

---

[9] Neertheless, if other applications request a GPS position, our application also receives this data, even if it occurs on a faster interval

in their timestamp, they are fused to a new pair covering the combined timespan.

3. The resultig GPS pairs of still moments are used to exclude still moments from the original sequence and divide it into subsequences marking trajectories.

Of each trajectory, the start and end location as well as the respective timestamp are then saved in *trajectory_table*. We tested 0.5 m/s, 0.6 m/s and 0.7 m/s as threshold in step 2b) and found 0.6 m/s to be best fit the tested sample. On the one hand the threshold must be low enough to still include slow walking which might be below 1 m/s. On the other hand, the threshold should not be too low because inaccuracy in GPS data might otherwise induce trajectories where the device has actually not moved at all.

Example of algorithm (TODO):

```
{
Original dataset:
Latitude Longitude Time
44  11    10:11:03
44.5  11.1    10:11:15
44.4  11.05    10:12:12
44.3  11.07    10:33:00
44      11.2    10:34:00
}
```

### 4.3.4. Serving Aggregation Requests

We use the retrofit2 framework[10] based on OKHTTP[11] to handle communication with our REST server dexribed in 4.4. An HTTP Interceptor is used to modify incoming and outgoing requests. The interceptor decrypts the request body of incoming messages using the private key of the installation before the body is parsed into Java Objects. On outgoing messages the inteerceptor adds authentication before sending them to the server. The app polls for new aggregation requests every 15 minutes. New aggregation requests are first stored locally in the database. Those requests are then processed and the results are again stored locally as pending outgoing requests until they are finally send to the server. Figure XX illustrates this process. This separation of concerns is useful especially in case of an interrupted communication during processing the aggregation request. When the results cannnot be send to the server, the app automatically retries the next time that the communication module is invoked. The aggregation itself takes the type parameter of the request to specify which actions to take on the three fields (n:int, value:Float, valueList: List<Float>) shared across all aggregation requests. In case of the tyes "steps" and "activity_X" the field value contains the current mean of the data and the field n is the number of participants so far. In case of "stepsListing" only the field "valueList" is used. Each user's mean value is added to the list. In case of "trajectories", only the field "valueList" is used. Four subsequent elements of the list always represent one trajectory as of latitude of start, longitude of start, latitude of end and longitude of end.

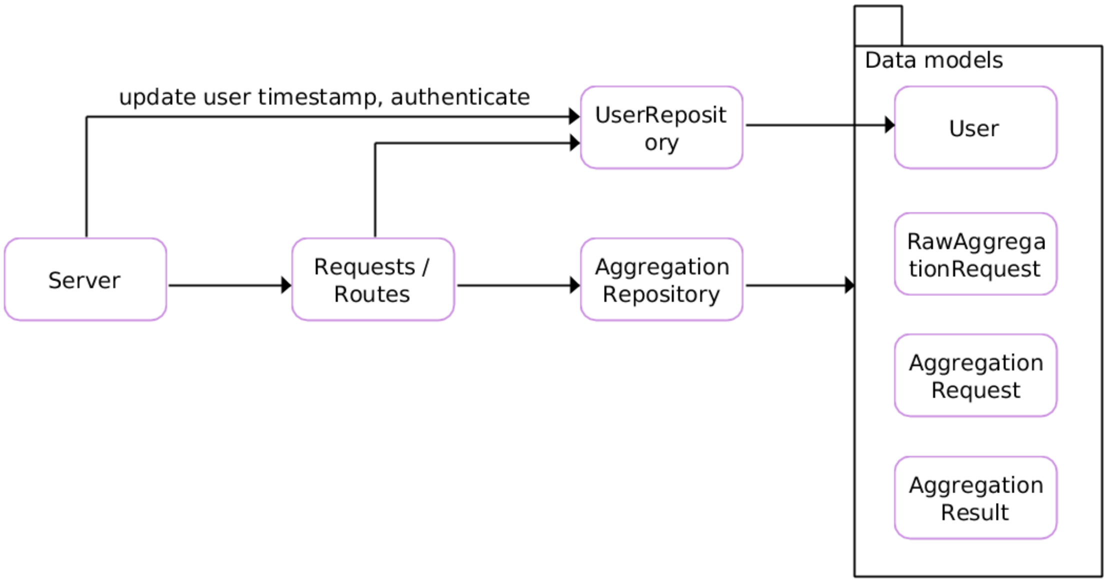
---

[10] https://square.github.io/retrofit/

[11] https://square.github.io/okhttp/

In case that the aggregation should be changed to actually work over P2P e.g. using local WIFI networks this module only has to be adapted to the new routing of requests. No further changes to the application are necessary.

Figure 4.8.: Server architecture


Data models
update user timestamp, authenticate
UserReposit ory
User
Server
Requests / Routes
Aggregation Repository
RawAggrega tionRequest
Aggregation Request
Aggregation Result

## 4.4. Server Design and Implementation

The server is build using the event-driven node.js verion 10.15.3 leveraging the express[12] web-server framework and using the mocha[13] testing framework in combination with the chai[14] assertion library for unit and integration testing. The server is designed using a layered architecture as described in figure 4.8. On the lowest level are the data models which define and verify the data schemes defined in subsection 4.4.1. The *commonRepository* and the *userRepository* are build on top of those models and persists data in a mongoDB object store. They also handle e.g. transactions where several objects are modified depending on the result of the previous modification (TODO: implement transactions?). The third level provides the logic to be executed for each endpoint defined in *routs.js* while *server.js* on the top level starts the server on the port specified in TODO.environment.json-TODO. It also registers the routes described in section 4.2, adds authentication and interacts directly with the *userRepository* in order to update the respective users lastSeen property. Furthermore, it starts a scheduled repeating task to keep the request chain running. This process is described in subsection 4.4.2

---

[12] See https://expressjs.com/

[13] See https://mochajs.org/

### 4.4.1. Data Model

We organize the data in four collections. The user collection stores the user data which is the public key, the hashed password and "lastSeen" - the timestamp of the last interaction of the user with the server. Aggregation requests are split into two collections. The collection "rawAggregationRequests" stores the initial aggregation request inserted through the admin interface containing the fields start, end, type - the type of the request, the three fields n, value, valueList reused across all aggregations to pass data, the timestamp when the request was filed to the server and a flag indicating whether this request has been started yet[15]. Upon start of the aggregation request, a list of the 10 most recently active users is retrieved in order to serve this request. The request body is then encrypted with the first users public key and stored in the collection "aggregationRequests". Each time, a user requests an aggregationRequest, proceeds with it and sends the results back to the server, the result is inserted into the database as a new aggregationRequest. The fields of this collection are

- rawRequestId - The id of the related rawRequest. This field is not available through the API.
- started_at - The timestamp, when the request has been started
- publicKey - The public key of the user that should proceed this request
- nextUser - The public key of the user that will receive the request afterwards. This is necessary so that the user that should proceed this request can encrypt the

---

[14] See https://www.chaijs.com/

[15] E.g. when the end data of a newly inserted aggregation request is in the future, the request will be started only when this day has passed

processed request with the public key of the next user.

- previousRequest - The id of the previous request. This is null, if it is the first request in the chain. This is used for the mechanism taking care if a request is not processed by the user it is pending for.
- users - the list of public keys of the following users that will proceed with this request. This field is not available through the API.
- encryptionKey - A synchronous key, encrypted with the public key of the user the request is aimed at.
- iv - the initialization vector used for synchronous encryption and decryption of the actual aggregation request.
- encryptedRequest - The actual aggregation request encrypted with the synchronous key.
- timestamp - The timestamp when this object has been created.
- commpleted - A flag indicating whether this aggregation request has already been proceeded by the respective user and the resulting aggregationRequest has been received by the server.

The last collection called "aggregationResults" is used to store the results of an aggregation request. Once there are no more users to serve an aggregationRequest, the last user sends the final data unencrypted to the server where it is stored as an aggregationResult. It contains the same fields as the rawAggregationRequest except the started flag and additionally a field started_at and timestamp - indicating when the aggregation request referenced through rawRequestId was started and when it was completed.

### 4.4.2. request-chain

//TODO server.js also invokes a scheduled task which re-routes stale requests where the user has not proceeded with the pending request either due to being offline or due to a problem handling the request. When new aggregation requests are started, the lastSeen timestamp of users is taken into account to exclude users that have not connected for a certain time. Furthermore, the list of users who are selected to serve the new request is ordered by the time the user was last seen.

# 5. Performance and Evaluation

## 5.1. Deployment

The proposed Android application and server have been tested on 16 devices for six day from 30.05.2019 until 04.06.2019. However, evaluating the lastSeen timestamp of the users showed that only 13 installations were active after 31.05.2019. The server was deployed[1] in the IBM Cloud as a 128 MB node.js instance. The database was hosted as a free version at mongodb.com.

We ran each of 7 requests on a daily basis and also for each timespan of several days within this period. The results can be found in the following tables[2]. The results of aggregation 8 were modified in order to protect user priacy. We used this testing period also to improve the performance of the server as well as the Android application and to find and remove bugs.

## 5.2. Data Consumption

The application used very few data. We are happy that some research participants provided us information about the app's data usage. Table XX shows the collected results of this rather qualitative analysis. Screenshots are attached in the appendix. On most phones the data consumption for 6 days was below 20 MB. The highest data consumption was 376 and is attributable to an error that occurred on the last day of the testing period. Requests were sent multiple up to unlimited times. This explains the high data consumption reported also by two other participants. This error was present during the whole testing period but interacted with another error that occurred only on the last day. This suggests, that the actual data consumption of the application would be far lower in practise (with the errors being fixed). Also the few available reports about battery consumption indicate a rather moderate battery consumption.

---

[1] The version deployed during field testing can be found at https://github.com/SimonVanEndern/location-server/releases/tag/v1.0 The final version provides the same functionality but includes improvements (e.g. bug-fixes, comments, ...) over the deployed version.

[2] The results are also available in JSON format at https://github.com/SimonVanEndern/tum-thesis-latex/tree/master/published/results

| **Average time spent running** | | |
|---|---|---|
| **Time period / Day** | **N** | **Value [min]** |
| 30.09.2019 | 10 | 1.43 |
| 31.05.2019 | 10 | 1.21 |
| 30.05.2019 - 31.05.2019 | 10 | 1.96 |
| 30.05.2019 - 31.05.2019 | 10 | 1.89 |
| 30.05.2019 - 01.06.2019 | 10 | 1.73 |
| 30.05.2019 - 03.06.2019 | 9 | 1.79 |
| 03.06.2019 | 9 | 1.95 |
| 04.06.2019 | 8 | 0 |
| 30.05.2019 - 04.06.2019 | 8 | 1.55 |
| 01.06.2019 - 04.06.2019 | 7 | 0.99 |
| 31.06.2019 - 04.06.2019 | 7 | 0.83 |
| 02.06.2019 - 04.06.2019 | 7 | 1.05 |
| 03.06.2019 - 04.06.2019 | 7 | 0.16 |

Table 5.1.: Average time spent running

| **Average time spent walking** | | |
|---|---|---|
| **Time period / Day** | **N** | **Value [min]** |
| 30.09.2019 | 10 | 17.63 |
| 31.05.2019 | 10 | 97.25 |
| 30.05.2019 - 31.05.2019 | 10 | 63.20 |
| 30.05.2019 - 31.05.2019 | 10 | 78.41 |
| 30.05.2019 - 01.06.2019 | 10 | 75.14 |
| 01.06.2019 | 4 | 72.85 |
| 30.05.2019 - 03.06.2019 | 9 | 81.94 |
| 03.06.2019 | 9 | 146.49 |
| 04.06.2019 | 8 | 86.77 |
| 30.05.2019 - 04.06.2019 | 8 | 61.86 |
| 01.06.2019 - 04.06.2019 | 7 | 57.44 |
| 31.06.2019 - 04.06.2019 | 7 | 67.63 |
| 02.06.2019 - 04.06.2019 | 7 | 51.75 |
| 03.06.2019 - 04.06.2019 | 7 | 64.04 |

Table 5.2.: Average time spent walking

| Average time spent biking | | |
|---|---|---|
| **Time period / Day** | **N** | **Value [min]** |
| 30.05.2019 | 10 | 2.29 |
| 30.05.2019 - 31.05.2019 | 10 | 11.10 |
| 01.06.2019 | 4 | 14.99 |
| 03.06.2019 | 9 | 31.03 |
| 04.06.2019 | 8 | 23.81 |

Table 5.3.: Average time spent biking

| Average time spent in a vehicle | | |
|---|---|---|
| **Time period / Day** | **N** | **Value [min]** |
| 30.05.2019 | 10 | 8.24 |
| 31.05.2019 | 10 | 119.92 |
| 30.05.2019 - 31.05.2019 | 10 | 68.40 |
| 30.05.2019 - 31.05.2019 | 10 | 90.49 |
| 30.05.2019 - 01.06.2019 | 10 | 82.19 |
| 01.06.2019 | 4 | 90.40 |
| 30.05.2019 - 03.06.2019 | 9 | 62.16 |
| 03.06.2019 | 9 | 71.74 |
| 04.06.2019 | 8 | 35.51 |
| 30.05.2019 - 04.06.2019 | 8 | 59.34 |
| 01.06.2019 - 04.06.2019 | 7 | 51.07 |
| 31.05.2019 - 04.06.2019 | 7 | 68.20 |
| 02.04.2019 - 04.06.2019 | 7 | 48.89 |
| 03.04.2019 - 04.06.2019 | 7 | 45.96 |

Table 5.4.: Average time spent in a vehicle

| **Average number of steps per day** | | |
|---|---|---|
| **Time period / Day** | **N** | **Value** |
| 30.05.2019 | 5 | 4331.6 |
| 31.05.2019 | 5 | 3439.8 |
| 30.05.2019 - 31.05.2019 | 6 | 3278.7 |
| 30.05.2019 - 31.05.2019 | 5 | 4030.2 |
| 30.05.2019 - 01.06.2019 | 6 | 4271 |
| 01.06.2019 | 1 | 17 |
| 30.05.2019 - 03.06.2019 | 5 | 643959 |
| 03.06.2019 | 3 | 455411.3 |
| 04.06.2019 | 3 | 1123 |
| 30.05.2019 - 04.06.2019 | 5 | 3622.6 |
| 01.06.2019 - 04.06.2019 | 2 | 2088.5 |
| 31.05.2019 - 04.06.2019 | 3 | 4681.3 |
| 02.04.2019 - 04.06.2019 | 3 | 326363.3 |
| 03.04.2019 - 04.06.2019 | 3 | 455903.7 |

Table 5.5.: Average number of steps per day

| **Listing of average number of steps per day per of each participant** | | |
|---|---|---|
| **Time period / Day** | **N** | **Values** |
| 30.05.2019 | 10 | 1661, 1246, 3195, 7714, 7842 |
| 31.05.2019 | 10 | 2747, 775, 3924, 9203 |
| 30.05.2019 - 31.05.2019 | 10 | 550, 2905, 3195, 775, 88828, 3419 |
| 30.05.2019 - 31.05.2019 | 10 | 550, 8828, 5145, 3195, 2433 |
| 30.05.2019 - 01.06.2019 | 10 | 8126, 3195, 283, 8828, 1629, 3565 |
| 01.06.2019 | 4 | 17 |
| 30.05.2019 - 03.06.2019 | 9 | 598, 3195, 1669, 8828, 3205505 |
| 03.06.2019 | 9 | 914, 1042, 1364278 |
| 04.06.2019 | 8 | 1103, 13332, 934 |
| 30.05.2019 - 04.06.2019 | 8 | 757, 1719, 8828, 3614, 3195 |
| 01.06.2019 - 04.06.2019 | 7 | 826, 33511 |
| 31.05.2019 - 04.06.2019 | 7 | 757, 9203,, 4084 |
| 02.04.2019 - 04.06.2019 | 7 | 974691, 1231, 3168 |
| 03.04.2019 - 04.06.2019 | 7 | 1364278, 1231, 2202 |

Table 5.6.: Average number of steps per day of each user

## 5.3. Results

We computed the aggregated results for each of the 6 days of the testing period separately and for 5 timespans from each of those days until the last day. The results can be seen in figure XXX. On most of the days the average time spent walking is roughly around one hour per day. Nevertheless, on the 3rd of June the value is clearly higher and 31st as well. We do not see any reason for this spike – 03.06. is a regular work day - nevertheless, the value is not that high as it would suggest errors. The value on the first day being below the other values is definitely attributable to the fact that we started to roll out the application on this day. The average time running is around 1-2 minutes per day. This is not surprising. The only scenarios usually are when somebody has to catch some transport or actively is running (and might probably not carry his or her phone). Due to an error in the setup, we only have the daily average data for the time spent biking which ranges from 11 to 31 minutes per day and sounds realistic regarding that the participants where all in Munich. The time spent in a vicle ranges from half an hour to 90 minutes on average per day and has one spike of almost 2 hours on 31.06. This can be explained as one of our research team had a very long car ride on this day. Comparing the data with the registered trajectories, there is another pretty long car ride which might have attributed to the high average of 90 minutes per day. The average steps per day range in the lower range of some thousands. Though, there are some values which clearly expose an error in our aggregation process. Excluding these definitely incorrect values from the statistics, the values all are in an expected range. Investigating the list of average steps, it is clear that the erro did not occur in the aggregation process but in the process of local aggregation of data. There is always maximum one value off, all other values are withinn an expected range. The zero values are due to non step sensor being present on the respective phones (which are excluded in the average aggregation). Using the algorithm described in chapter 4.3.3 a total of 406 trajectories where computed out of the raw GPS data. A part of the trajectories are shown in figure 5.1 and 5.2. The complete results can be found in the appendix. [3]

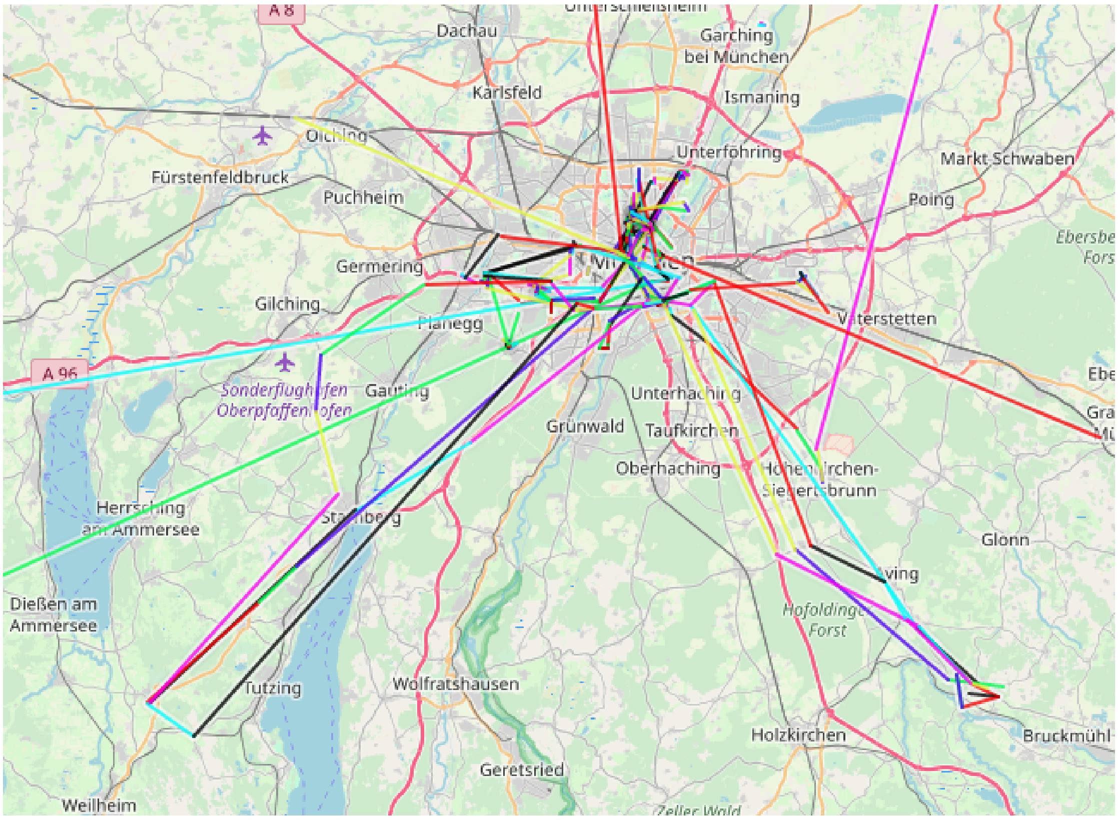
## 5.4. Implications

Our results show clearly, that there is no privacy risk imposed on the user upon publication of the aggregated average data. Especially it was shown that when repeating the request, the user base changes and the value accordingly. So, publishing pure

---

[3] Whenever the trajectory started or ended clearly in a precise private location e.g. housing, we modified this location. So the results actually do not represent real trajectories anymore

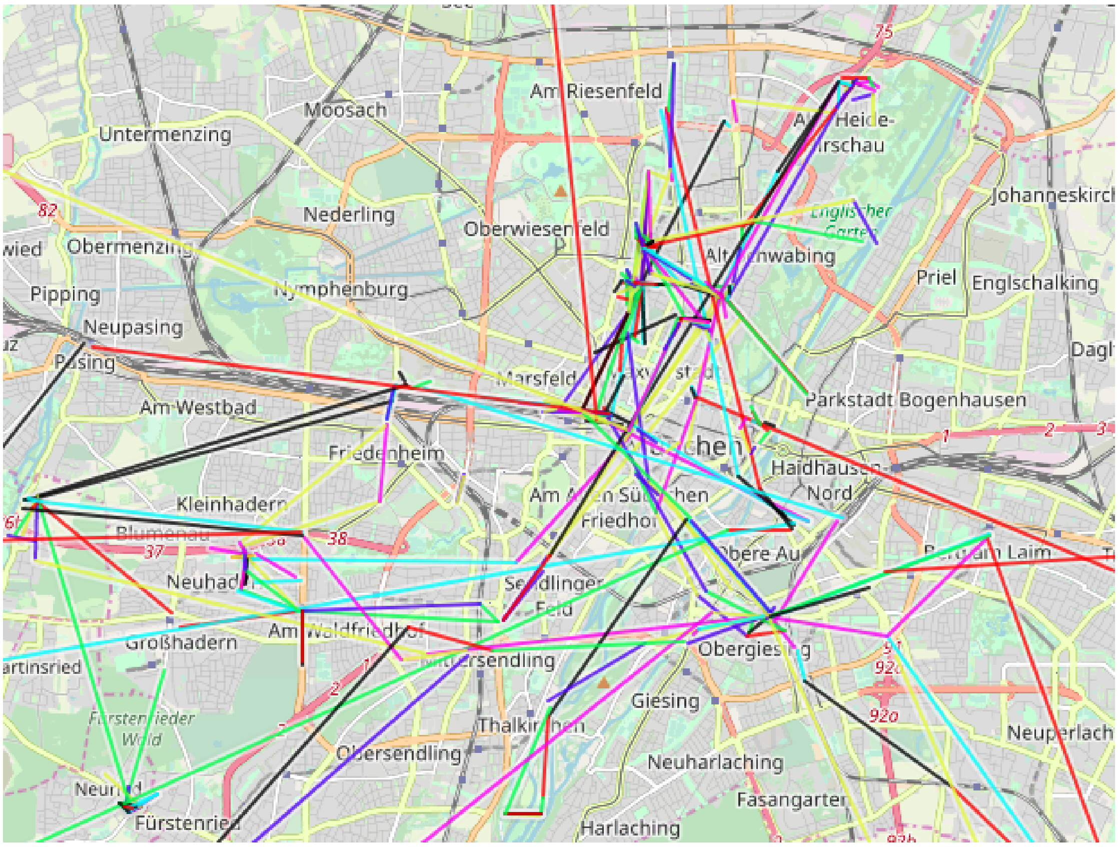
Figure 5.1.: Trajectories


A 8
Dachau
Garching
bei München
Karlsfeld
Ismaning
Olching
Unterföhring
Fürstenfeldbruck
Puchheim
Markt Schwaben
Poing
Ebersbe
Fors
Germering
Gilching
Planegg
Vaterstetten
A 96
Ebe
Sonderflughafen
Oberpfaffenhofen
Gauting
Unterhaching
Gra
Mü
Grünwald
Taufkirchen
Oberhaching
Hohenkirchen-
Siegertsbrunn
Herrsching
am Ammersee
Starnberg
Glonn
Aying
Dießen am
Ammersee
Hofoldinger
Forst
Tutzing
Wolfratshausen
Holzkirchen
Bruckmühl
Geretsried
Weilheim
Zeller Wald

mean values of analysis does not impose any privacy risk to the user. Furthermore we showed exemplary with the steps aggregation, that also the collection of the users' average values is possible without posing a privacy risk to the user. This implies, that not only the aggregation of mean values but also the collection of locally aggregated data is possible without privacy concerns. We do not see any possibility to infer that the same user took part in different aggregations. There might be a chance in case of a very high steps value and a very high mean time spent walking or running, but this would only allow to assume the same user being present in both aggregations which allows for no further inference.

Nevetheless when collecting the average data, through overlapping requests e.g. one from 30.05 - 02.06 and one from 01.06. - 04.06 it might be possible to infer the exact date of some of the data present in both requests. Though, we do not see any risk aggregating on a daily basis or even hourly if there is any reason for this granularity level. Thus, the inference does no harm.

Figure 5.2.: Trajectories


Untermenzing
Moosach
Am Riesenfeld
Alte Heide-Hirschau
Nederling
Oberwiesenfeld
Englischer Garten
Johanneskirchen
Obermenzing
Altschwabing
Pipping
Nymphenburg
Priel
Englschalking
Neupasing
Pasing
Marsfeld
Parkstadt Bogenhausen
Am Westbad
Friedenheim
Haidhausen-Nord
Kleinhadern
Am Alten Südfriedhof
Blumenau
Neuhadern
Sendlinger Feld
Obere Au
Großhadern
Am Waldfriedhof
Obergiesing
Giesing
Thalkirchen
Obersendling
Neuharlaching
Neuperlach
Fasangarten
Harlaching
Fürstenried
Forstenrieder Wald
Neuried
Martinsried

In more than 90% of cases, local trajectories could match an activity for more than 90% of the time. The experimental collection of trajectories clearly shows privacy risks as pointed out in XX. Nevertheless, the data suggests that those trajectories can easily be linked locally and identify changing of transport system or e.g. metro line. For example at the locations Giselastraße, Odeonsplatz and central station and other locations, many trajectories end respectively start. Mapping the current activity to those trajectories enables aggregations as mentioned in chapter 3.1. Noting that locally all data points of the trajectories are available, it is also easy to compute the distance travelled. Also it is possible to compute how many people combine e.g. bike and car or public transport and furthermore identify, which station is most likely (in case of public transport) to be combined with bike.

Also it would be possible to create a "travelled road" or "travelled public transport" map by aggregating the trajectories data so that there would not be multiple trajectories but each road either marked used or not used. This way it would be possible to identify that a user lives in a certain area but could not link to the work location due to obfuscation with other routes joining. This map would on the other hand allow to have a feeling of which areas are covered by the data / app. This map could also be extended with the average speed on the respective road depending on vehicle or bike and also compare whether bike is faster.

Computing time to work and average time at work.

It also allows for computing how many people go by car, bike, or mixed to work.

We also started an aggregation a second time - walking from 30.05 - 01.06. which shows as expected a different result than the original aggregation due to the 10 users being not the same users as in the first request. Nevertheless, the value is not far off as expected.

TODO: The trajectories also show, that in the not anonymized data set we had at hands, often trajectories of which the endpoints did not clearly indicate an ongoing, the numeration of the trajectory indicated that they were connected. This 1. highlights the importance of list order randomization and 2. shows that our algorithm of finding trajectories still has possibilities for improvement. (TODO: Randomize order in published data set)

## 5.5. Scenario Traffic Jam

A typical scenario of google maps is o notify users about traffic jams and suggest alternate routes. The calculation of alternate routes taking traffic jams into account can clearly happen locally with the maps data. Google maps works when offline. The data about all current traffic jams can also be made publicly available through a server. Generating the data can also happen without exposing raw data: The user downloads a map containing data of the usual speeds at each street. While the user is driving, the app registers the speed and compares it in the background to the normal speed. If the speed is significantly lower, the user chooses a random list of known users and sends the signal as a request for those users to the server. They randomly according to a fixed percentage choose to inform the server about the traffic jam or forward the signal another time. The signal contains a unique id thus that the server even when receiving it muliply times knows it is from one user. If more than a threshold of signals is received, a traffic jam is "created". Also the request is not forwarded anymore after a certain time to stop it from spreading unlimited.

# 6. Conclusion and Discussion

We have shown that our hypothesis holds but only for aggregated data. This is fine because except in one experimental setting as with trajectories, there is no need for (anonymized) raw data. Also the experimental setting could be replaced by directly implementing the aggregations and testing with a greater user base. In theory, the anonymity and preserved privacy that hold for the tested aggregations hohlds also for the other proposed aggregations and aggregations we have not evaluated here. In order to follow with this research, we provide the setup to easily implement and test those and further aggregations in future research in order to further support our hypothesis.

## 6.1. Limitations

- As mentioned in XX, our system is based on trust. In case of user data being compromised, this significantly impacts some of the results. Nevertheless, collecting the list of mean values is far less error prone as the outlier could also be identified.

## 6.2. Future Work

- While XX has found that inference attacks can be based on the same dataset being published two times with different anonymization techniques applied and XX shows that anonymized datasets that overlap poses a risk, it still has to be investigated whether overlapping aggregated data as in our case can pose a risk.
- Some of the techniques identified as useful, such as spatial cloaking, ... should be applied in our setting.
- Our framework would also allow to pre-populate (simulated) smartphones with artificially generated or otherhow collected data in order to test and verify the functionality.
- Another use of our framework is the area of decentralized computation. Problems might be solved locally and collected by the server afterwards and the pieces put together still with anonymity for the users.

- Ask the user for his / her home and work location or infer it from the data in order to process aggregation requests as mentioned in section XX.
- Store the users location on the server (granularity level approach) in order to allow for aggregation requests targeted at specific areas and not the overall user base. (levels and level database and unlocked levels collection necessary)
- Android: Only send a final aggregation back to the server when criteria like minimum n, ... are met.
- Server: Apply that the request has a counter how many times it was actually retrieved, so that after e.g. 10 times it was retrieved but never answered, the request is rolled back because apparently the user somehow cannot process the request.
- Have a nice activity informing the user / displaying some information to the user.
- (Put somewhere else!! TODO) The app can send traffic alerts to the server if it is on a route where usually traffic is far faster. (This also via other nodes in order to not letting the server know who is on this route.)
- a scheduler which automatically creates aggregation requests on a regular basis so that not as now Postman has to be used to start requests. The Postman collections used during field testing can be found here : XXX
- Generate userList of aggregation request dynamically.
- Pre-populate (raw)aggregation requests so that the first users cannot infer data from the users before them with high probabilities. Upon receiving the final result, the server can look up the used initialization values from the rawAggregationRequest and calculate the actual correct result and insert it into the database.
- Generate some use for the user of the application e.g. through showing locally aggregated data and comparing it to aggregated data publicly available e.g. in order to show how many percent walked more this day than the respective user. This is also an approach to motivate app installations in case of a broader user base necessary.
- delete local data after some time.
- Evaluate how many people have to participate in an aggreation request to be representative / how to chose users participating in it in order to not be biased

(e.g. when taking always the most recent active users, the users only online a few times are day are discriminated against)

- Investigate overlapping
- When returning list data to the server, the list order should be randomised by the user before sending.
- Verify public key e.g. through telephone number passing and verifying this way and then building a network of verifications.
- Adhere to standards of schema.org
- In the future it should be implemented that devices change the public / private key pair from time to time
- It could also be implemented to verify another public key but requesting an SMS, .... (harder with changing public-private key-pairs)
- Possible future aggregation: List of all activity times e.g. for walking to find out mean, medium, ...
- When dynamically adding users to the aggregation request, we need another field for setting the limit so that when the limit is reached, the user sends back the result.
- Pre-fill the aggregation request so that the second user cannot read the first user's share.

We suggest to do especially two things in any further research project: Implement error logging and sending those errors to the server in order to find and remove bugs. Secondly bring the application to the playstore so that updates are possible (even without user interaction in case of automatic updates) and be able to activate a broader user base while still working on the final version to be tested.

Also, the approach could easily integrated into existing projects like open maps or be build modular in order to allow using it as a library with other applications.

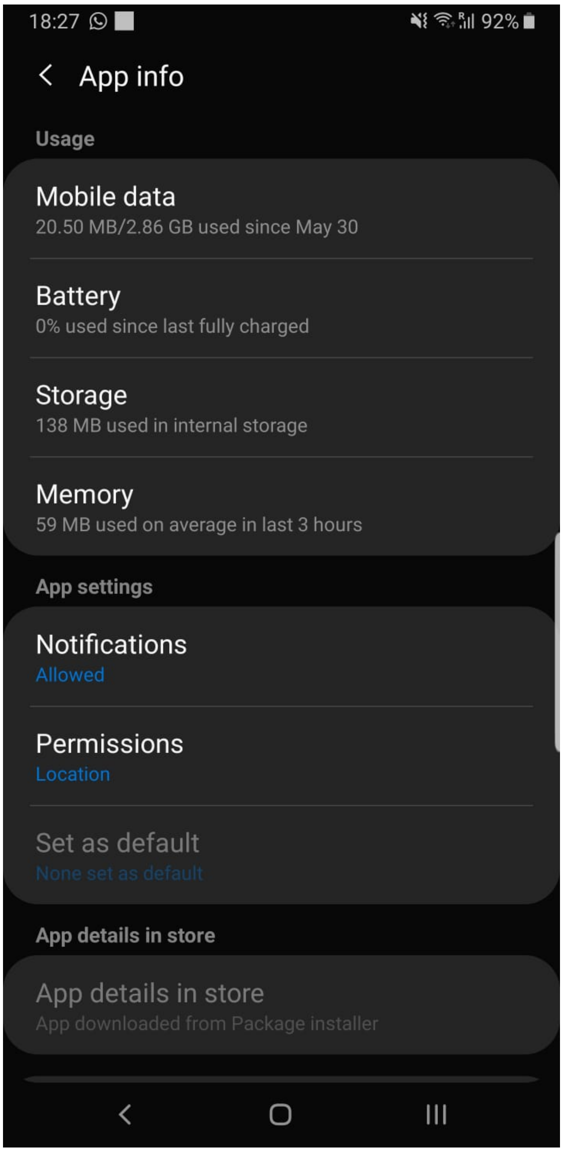
## 6.3. Evaluation

The results support our hypothesis that data can be analyzed decentrally and that aggregated data can be published without any privacy concerns. The aggregations of mean values clearly leave no doubt about full privacy protection as there even is no personal data involved anymore. The listing of mean values of average number of steps per participant allows for more advanced statistical analysis while at the same time the values cannot be mapped to persons. Even when conducting the same request twice, due to users being chosen dynamically, one could most probably see if the same user participated in the second request but nothing else. When aggregating over another time period (that might have an intersection with the other one), there is not even the chance to identify whether the same user participated in both aggregations. As requests are started not at the same time (TODO!!!), and users are in a future setting allocated dynamically to the request, there is also no chance to link the data from different aggregations. From the number of steps one could infer the time somebody spent walking, but as it is not given whether the steps where conducting walking, running or both, this linking would result in a very poor performance and also only reveal that to a very low probability, the user with X steps in one aggregatioin is the same user spending X minutes walking the same day. The listing of trajectories created a dataset that clearly shows the vulnerability highlighted in XX and XX. Nevertheless, this is no aggregation but just a collection of raw data with stripped of identifiers and timestamps anonymized to a daily basis. The results show clearly that our setup is sufficiently accurate to field test the other aggregations proposed in XX and further prove our thesis. The data shows, that e.g. A change of transportation system can clearly be identified.

## 6.4. Limitations

As stated in the introduction, our approach is based on trust among the clients and the server. Nevertheless, while e.g. [13] face the same problems in case of a compromised server - namely that the server can create artificial participants and thus obtain the raw values from each user, our setup is more general, allows for more complex aggregations and can be adapted to work via P2P or to establish trust through something like XXX

# A. Data Usage Screenshots



Usage

Mobile data
20.50 MB/2.86 GB used since May 30

Battery
0% used since last fully charged

Storage
138 MB used in internal storage

Memory
59 MB used on average in last 3 hours

App settings

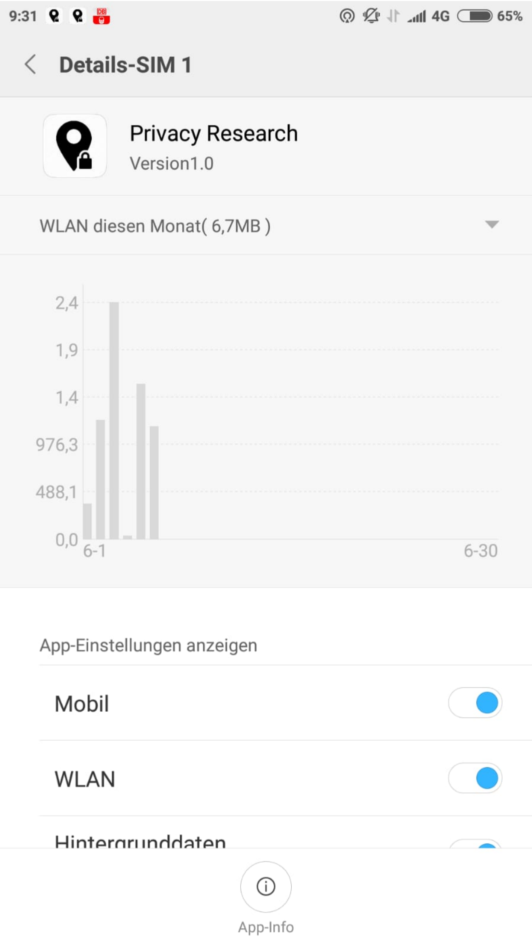
Notifications
Allowed

Permissions
Location

Set as default
None set as default

App details in store

App details in store
App downloaded from Package installer



9:31 4G 65%

Details-SIM 1

Privacy Research
Version1.0

WLAN diesen Monat( 6,7MB )

2,4
1,9
1,4
976,3
488,1
0,0
6-1
6-30

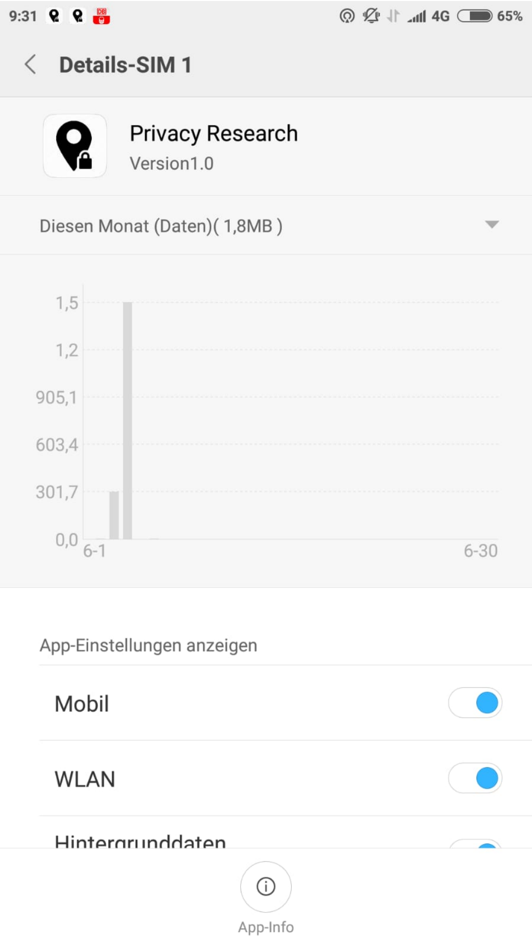
App-Einstellungen anzeigen

Mobil ●

WLAN ●

Hintergrunddaten

App-Info



9:31
4G 65%
Details-SIM 1

Privacy Research

Version1.0

Diesen Monat (Daten)( 1,8MB )

1,5
1,2
905,1
603,4
301,7
0,0
6-1
6-30

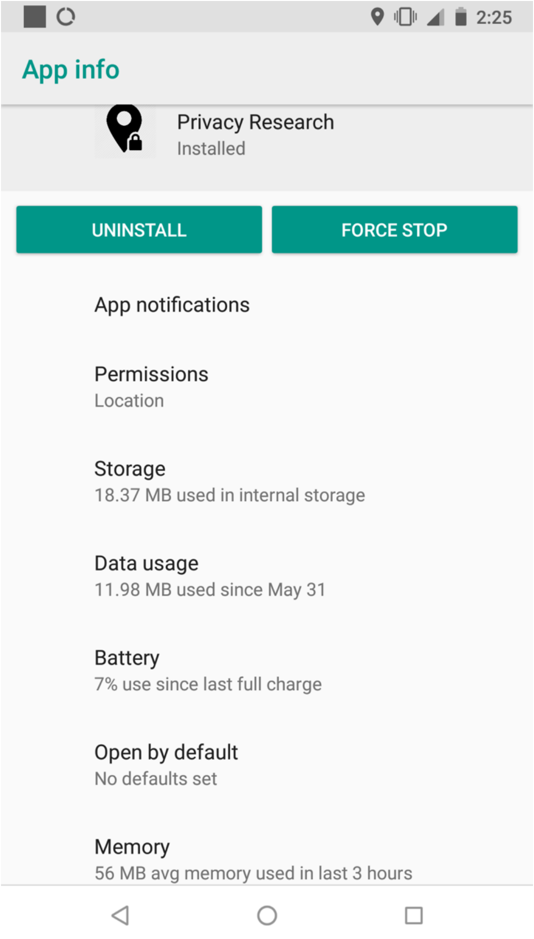
App-Einstellungen anzeigen

Mobil ●

WLAN ●

Hintergrunddaten

App-Info



2:25
App info

Privacy Research
Installed

UNINSTALL
FORCE STOP

App notifications

Permissions
Location

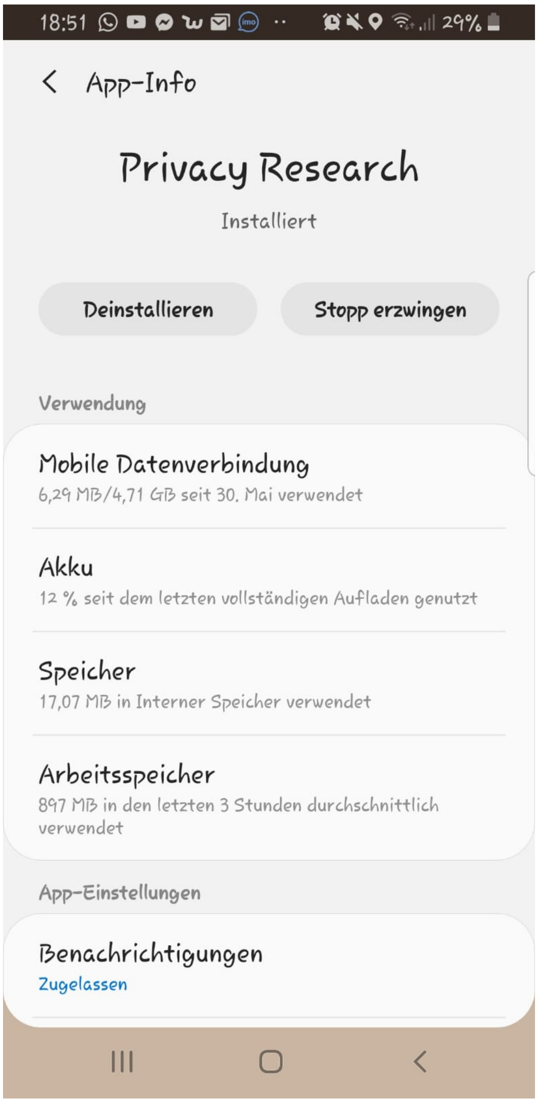
Storage
18.37 MB used in internal storage

Data usage
11.98 MB used since May 31

Battery
7% use since last full charge

Open by default
No defaults set

Memory
56 MB avg memory used in last 3 hours



18:51
29%
App-Info

Privacy Research

Installiert

Deinstallieren
Stopp erzwingen

Verwendung

Mobile Datenverbindung
6,29 MB/4,71 GB seit 30. Mai verwendet

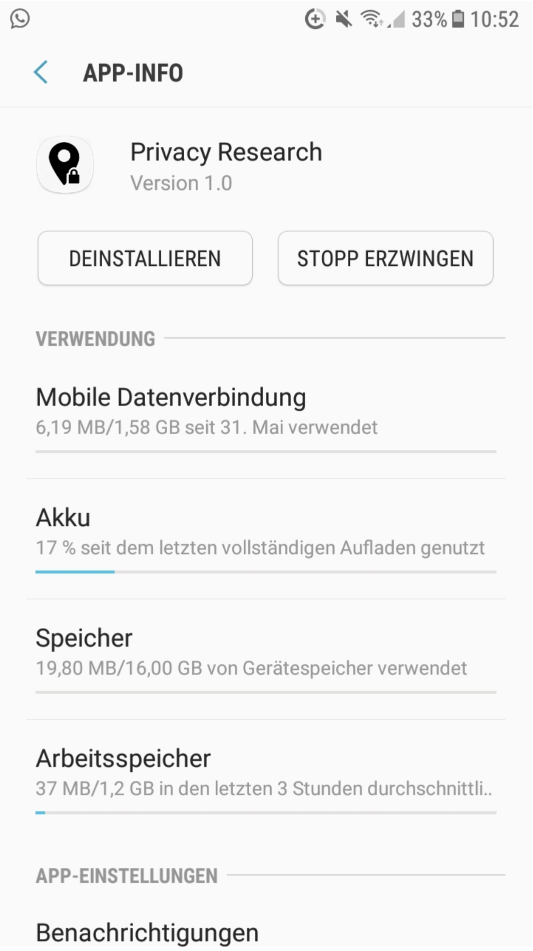
Akku
12 % seit dem letzten vollständigen Aufladen genutzt

Speicher
17,07 MB in Interner Speicher verwendet

Arbeitsspeicher
897 MB in den letzten 3 Stunden durchschnittlich verwendet

App-Einstellungen

Benachrichtigungen
Zugelassen


33% 10:52
APP-INFO
DEINSTALLIEREN
STOPP ERZWINGEN
VERWENDUNG
APP-EINSTELLUNGEN
Benachrichtigungen

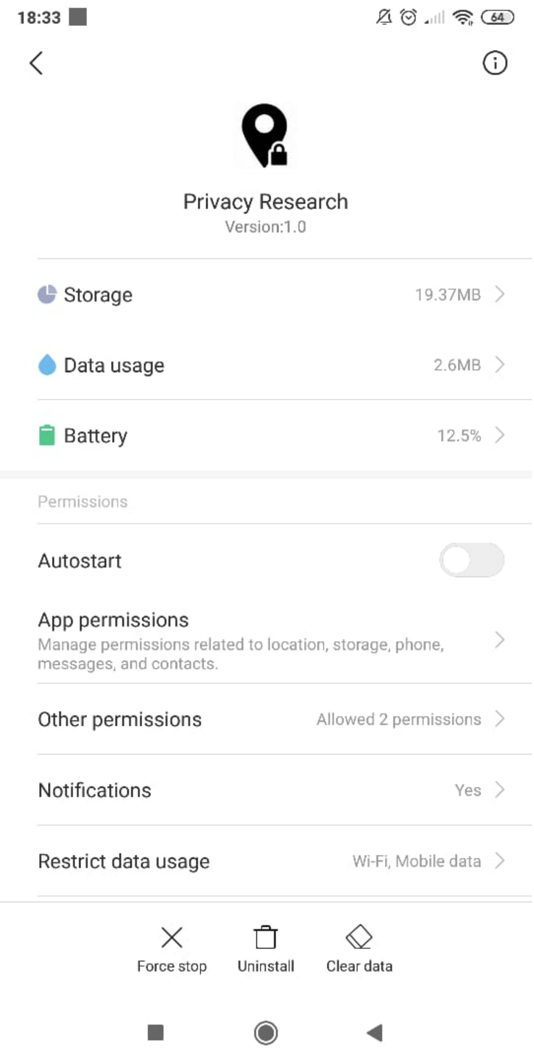
Privacy Research
Version 1.0

Mobile Datenverbindung
6,19 MB/1,58 GB seit 31. Mai verwendet

Akku
17 % seit dem letzten vollständigen Aufladen genutzt

Speicher
19,80 MB/16,00 GB von Gerätespeicher verwendet

Arbeitsspeicher
37 MB/1,2 GB in den letzten 3 Stunden durchschnittli..



18:33

Privacy Research
Version:1.0

Storage 19.37MB

Data usage 2.6MB

Battery 12.5%

Permissions

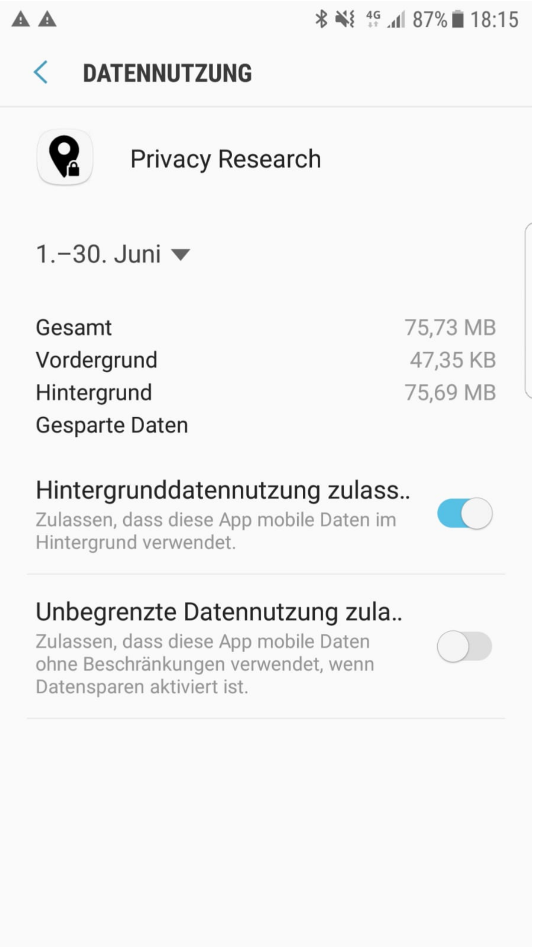
Autostart

App permissions
Manage permissions related to location, storage, phone, messages, and contacts.

Other permissions: Allowed 2 permissions

Notifications: Yes

Restrict data usage: Wi-Fi, Mobile data



87% 18:15

DATENNUTZUNG

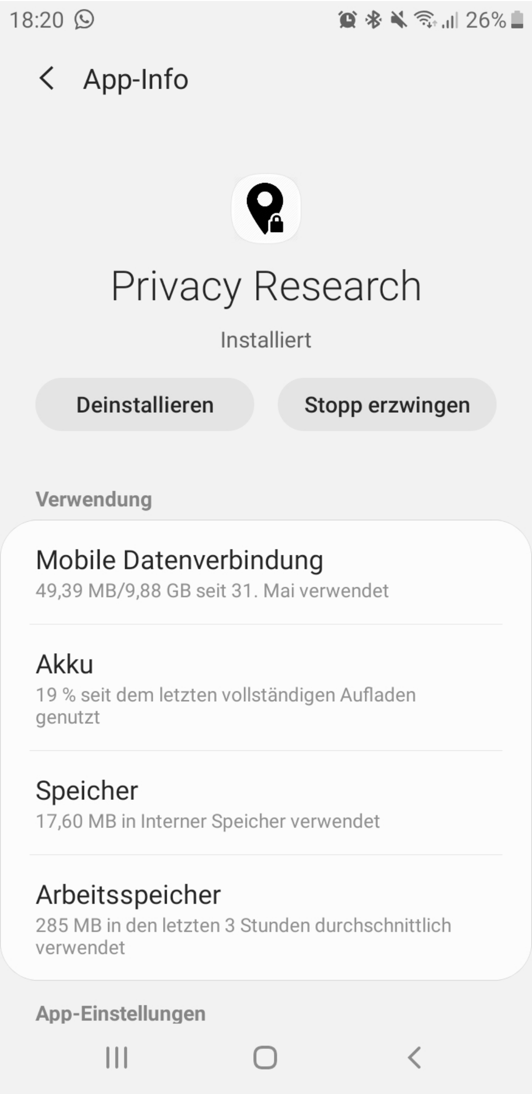
Privacy Research

1.–30. Juni

| | |
|---|---|
| Gesamt | 75,73 MB |
| Vordergrund | 47,35 KB |
| Hintergrund | 75,69 MB |
| Gesparte Daten | |

Hintergrunddatennutzung zulass..
Zulassen, dass diese App mobile Daten im Hintergrund verwendet. ●

Unbegrenzte Datennutzung zula..
Zulassen, dass diese App mobile Daten ohne Beschränkungen verwendet, wenn Datensparen aktiviert ist. ○



18:20
26%
App-Info

Privacy Research

Installiert

Deinstallieren
Stopp erzwingen

Verwendung

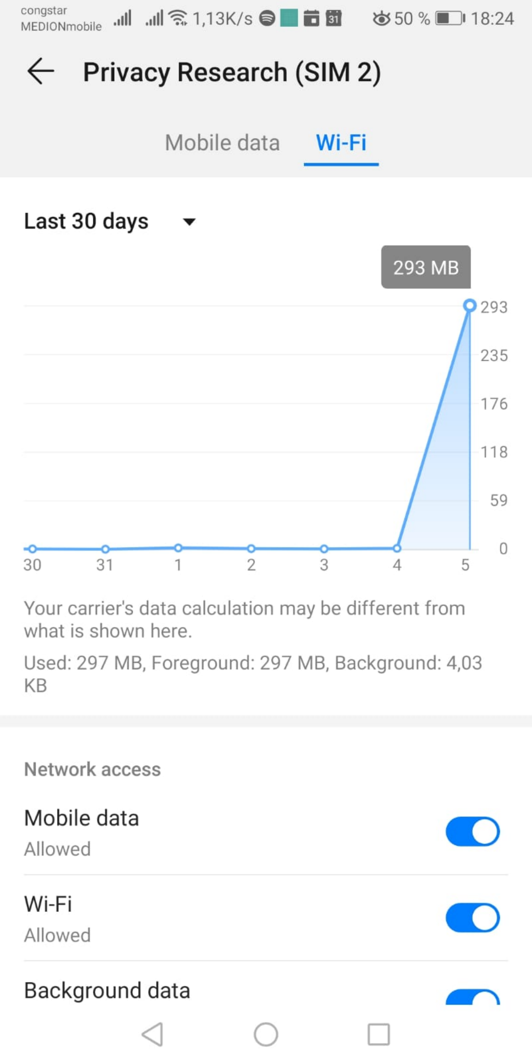
Mobile Datenverbindung
49,39 MB/9,88 GB seit 31. Mai verwendet

Akku
19 % seit dem letzten vollständigen Aufladen genutzt

Speicher
17,60 MB in Interner Speicher verwendet

Arbeitsspeicher
285 MB in den letzten 3 Stunden durchschnittlich verwendet

App-Einstellungen



congstar
MEDIONmobile
1,13K/s
50 %
18:24
Privacy Research (SIM 2)
Mobile data
Wi-Fi
Last 30 days
293 MB
293
235
176
118
59
0
30
31
1
2
3
4
5

Your carrier's data calculation may be different from what is shown here.

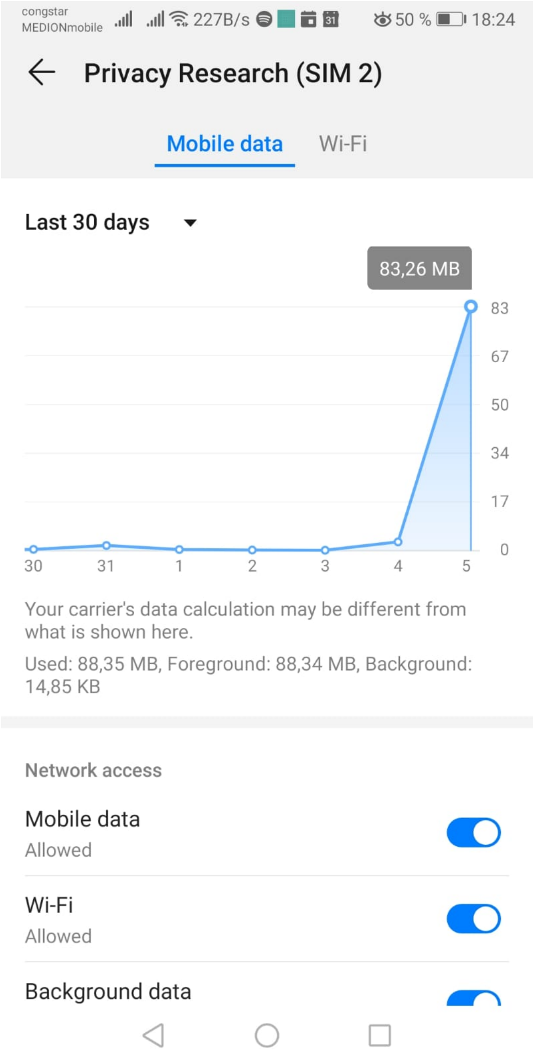
Used: 297 MB, Foreground: 297 MB, Background: 4,03 KB

Network access

Mobile data
Allowed

Wi-Fi
Allowed

Background data


congstar
MEDIONmobile
227B/s
50 %
18:24
Mobile data
Wi-Fi
Last 30 days
83,26 MB
83
67
50
34
17
0
30
31
1
2
3
4
5

Privacy Research (SIM 2)

Your carrier's data calculation may be different from what is shown here.

Used: 88,35 MB, Foreground: 88,34 MB, Background: 14,85 KB

Network access

Mobile data
Allowed

Wi-Fi
Allowed

Background data

# List of Figures

3.1. Incoming aggregation request . . . . . . . . . . . . . . . . . . . . . . . . . 7
3.2. Outgoing aggregation response . . . . . . . . . . . . . . . . . . . . . . . . 7
3.3. Decentral unencrypted aggregation process via P2P . . . . . . . . . . . . 8
3.4. Decentral encrypted aggregation process . . . . . . . . . . . . . . . . . . . 9

4.1. POST /user . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
4.2. Response to POST /user . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
4.3. Response to GET /requests . . . . . . . . . . . . . . . . . . . . . . . . . 14
4.4. POSt /forward . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
4.5. Response to GET /aggregations . . . . . . . . . . . . . . . . . . . . . . . 15
4.6. POST /admin/insertDample . . . . . . . . . . . . . . . . . . . . . . . . . 15
4.7. Android architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
4.8. Server architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

5.1. Trajectories . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
5.2. Trajectories . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30

# List of Tables

5.1. Average time spent running . . . . . . . . . . . . . . . . . . . . . . . . . . 25
5.2. Average time spent walking . . . . . . . . . . . . . . . . . . . . . . . . . . 25
5.3. Average time spent biking . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
5.4. Average time spent in a vehicle . . . . . . . . . . . . . . . . . . . . . . . . 26
5.5. Average number of steps per day . . . . . . . . . . . . . . . . . . . . . . . 27
5.6. Average number of steps per day of each user . . . . . . . . . . . . . . . . 27

# Bibliography

[1] A. R. Beresford and F. Stajano. “Location privacy in pervasive computing.” In: *IEEE Pervasive computing* 1 (2003), pp. 46–55.

[2] A. R. Beresford and F. Stajano. “Mix zones: User privacy in location-aware services.” In: *IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second*. IEEE. 2004, pp. 127–131.

[3] G. Developers. Accessed: 2019-06-07.

[4] K. Drakonakis, P. Ilia, S. Ioannidis, and J. Polakis. “Please Forget Where I Was Last Summer: The Privacy Risks of Public Location (Meta)Data.” In: *CoRR* abs/1901.00897 (2019). arXiv: `1901.00897`.

[5] T. Economist. Accessed: 2019-06-20.

[6] P. Golle and K. Partridge. “On the Anonymity of Home/Work Location Pairs.” In: *Proceedings of the 7th International Conference on Pervasive Computing*. Pervasive ’09. Nara, Japan: Springer-Verlag, 2009, pp. 390–397. ISBN: 978-3-642-01515-1. DOI: `10.1007/978-3-642-01516-8_26`.

[7] T. Guardian. Accessed: 2019-06-20.

[8] T. Guardian. Accessed: 2019-04-10.

[9] B. Hoh and M. Gruteser. “Protecting location privacy through path confusion.” In: *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM’05)*. IEEE. 2005, pp. 194–205.

[10] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady. “Enhancing security and privacy in traffic-monitoring systems.” In: *IEEE Pervasive Computing* 5.4 (2006), pp. 38–46.

[11] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady. “Preserving Privacy in Gps Traces via Uncertainty-aware Path Cloaking.” In: *Proceedings of the 14th ACM Conference on Computer and Communications Security*. CCS ’07. Alexandria, Virginia, USA: ACM, 2007, pp. 161–171. ISBN: 978-1-59593-703-2. DOI: `10.1145/1315245.1315266`.

[12] W. A. Jabbar, M. Ismail, and R. Nordin. “Peer-to-peer communication on android-based mobile devices: Middleware and protocols.” In: *2013 5th International Conference on Modeling, Simulation and Applied Optimization (ICMSAO)*. IEEE. 2013, pp. 1–6.

[13] H. Kajino, H. Arai, and H. Kashima. “Preserving worker privacy in crowdsourcing.” In: *Data Mining and Knowledge Discovery* 28.5-6 (2014), pp. 1314–1335.

[14] J. Krumm. “Inference Attacks on Location Tracks.” In: *Pervasive Computing*. Ed. by A. LaMarca, M. Langheinrich, and K. N. Truong. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 127–143. ISBN: 978-3-540-72037-9.

[15] M. F. Mokbel, C.-Y. Chow, and W. G. Aref. “The new casper: Query processing for location services without compromising privacy.” In: *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment. 2006, pp. 763–774.

[16] H. B. Review. Accessed: 2019-06-20.

[17] P. Samarati and L. Sweeney. “Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and specialization.” In: *Proceedings of the IEEE Symposium on*. 1998.

[18] Statista. Accessed: 2019-06-07.

[19] M. Stolpe. “The internet of things: Opportunities and challenges for distributed data analysis.” In: *ACM SIGKDD Explorations Newsletter* 18.1 (2016), pp. 15–34.

[20] L. Sweeney. “Achieving k-anonymity privacy protection using generalization and suppression.” In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.05 (2002), pp. 571–588.

[21] L. Sweeney. “k-anonymity: A model for protecting privacy.” In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.05 (2002), pp. 557–570.

[22] H. Zang and J. Bolot. “Anonymization of location data does not work: A large-scale measurement study.” In: *Proceedings of the 17th annual international conference on Mobile computing and networking*. ACM. 2011, pp. 145–156.