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

I would like to thank my supervisor Prof. Dr.-Ing. Jörg Ott for allowing me to write this thesis at his chair. Furthermore I am grateful to my advisor Trinh Viet Doan, for motivating me and for lending me their expertise and knowledge.

Abstract

A large fraction of GPS data sets is not available to the public but proprietory to the companies collecting them e.g. Google. The public might benefit from access to these data sets. For example, cities might better understand traffic problems and come up with innovative solutions. Nevertheless, the publication of these data sets is not wanted in most cases and also not possible without severly intruding the privacy of the people the data was collected from. While research has investigated possibilities to anonymize data sets to enable publication, the overall problem has not been solved yet. Especially does the publication of these data sets depend on the goodwill of companies owning them. To solve this, a crowdsourcing approach for Android smartphones has been developed and tested. This approach decentrally aggregates location data and publishes only aggregated data in order to preserve users' privacy. Due to P2P not generally being possible on mobile devices, the solution depends on the assumption of a trusted server for passing encrypted messages between devices. The devices have to be trusted as well, respectively it has to be trusted that developed source code runs as it is on the server and on the devices. A field test confirmed the feasibility of the proposed solution. Furthermore, the results collected in this field test with 16 participants showed that aggregations like mean value, median value and distribution function seem to be possible to be published without violating users' privacy. It could also be shown that more complex aggregations e.g. identifying which subway stations are most suitable for combining biking and public transport seem possible with the developed setup and the resulting data quality.

Contents

1. Introduction

1.1. Motivation

“Data is the new oil" [37, 39] is a quote 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 file for a patent and thus not having to reveal their knowledge [14]. If that approach is not compromised, it guarantees both, non-disclosure and exclusive rights. Similarly, the non-disclosure of huge data sets collected by Facebook, Google and Amazon circumvent the balance intended by patents. But this 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 Hoh et al. [15]. 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. But in fact, users’ privacy is already compromised even without publication of their 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 [38].

Some governments and other institutions already publish some of their data sets after anonymizing them and there are crowdsourcing and open source approaches to make data available to everybody [CITE!!!]. 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 respective users’ privacy [9, 41]. Also, most research is based on data sets 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 privacy risk imposed on the user by the existence of a central data set, publishing anonymized data poses another risk to users' privacy.

1.2. Research Questions

In order to further investigate the trade-off between publication of location data and preservation of user privacy, we pose the following research questions:

RQ1: How is aggregation of location data possible without raw data being accessible to anybody but the owner?

Previous work has shown that in order to overcome the conflict between privacy intrusion and (public) data availability, a solution is needed that works 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. It should furthermore eliminate the risk of inference attacks on published believed-to-be-anonymized raw data if only aggregated data is stored on a central server. We investigate whether there is a solution and also what limitations this solution has, respectively which assumptions must hold for the solution to be valid.

RQ2: What types of aggregations can be published?

Clearly, a mean value exposes less details about the underlying raw data than the whole distribution from which e.g. the median value and other percentiles can be computed. We will investigate how many details, respectively which statistical values 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 covered inference attacks on anonymized raw data sets. They have shown that inference attacks can be based on overlapping timespans or the same data set being published using different anonymization techniques. We will investigate, whether there is also a risk of inference attacks based on overlapping timespans in aggregated data and examine the limitations of publishing aggregated data.

1.3. Contributions

While most research follows the approach of degrading data in order to provide anonymity, 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 aggregation in a decentralized manner and second by analyzing if an aggregation does not violate users' privacy pior to using this aggregation in a production environment. Step by step, the set of possible aggregations to be published could be extended after analyzing in a test environment if the respective aggregation does not intrude the users' privacy. The data stemming from those aggregations will then be available to the public.

We investigate the possibility of storing raw location data only decentralized on the collecting devices. On a central database available to the public, only aggregated data is stored, thus, the privacy risk arising form a central database containing the overall raw data set is eliminated. Similar to previous research [CITE!!!], 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.

Our research is organized as follows: First, we review related work in the areas of location privacy and anonymization techniques. Chapter 3 proposes our solution and outlines the general architecture of our approach using decentralized data analysis. Chapter 4 documents the implementation of the setup proposed in Chapter 3 and explains design decisions. We present the results of field testing our setup in Chapter 5, Chapter 6 summarizes our work, discusses limitations as e.g. a trusted server, outlines possible future work and concludes with reproducability considerations.

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 implied by Hoh et al. [15], 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 by.

2. **Some delay tolerance e.g. one minute:** An application e.g. google maps derives the information of congested traffic from devices reporting their GPS data which show lower than usual speed on the respective road. 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:** For historical and statistical use of location data e.g. to find out about popular visiting times of a venue, almost any delay is acceptable.

Some research deals with category one where almost no delay tolerance is acceptable [6, 7]. Mokbel et al. [23] proposes a solution where not the exact location is send to a server but the rough region of the user. The server then sends a list of all possible matches e.g. petrol stations in this area to the client. Locally, this list is then matched with the exact location in order to fulfill the aim of the respective application. Most research though investigates users' privacy for category 3 where the delay of the data being available for processing is not an issue [20, 9, 10, 41]. In the following, we will concentrate on this case as well.

2.2. Privacy problems arising from location data

2.2.1. Risk of privacy intrusion through theft of central databases

Centralized databases containing raw location data expose users to a privacy risk (through theft) [34, 16]. Jabbar et al. [18] 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 Stolpe [34] as an alternative to cloud-based IoT. Kajino et al. [19] 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. Another approach by Hoh et al. [16] 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 location to the central server without giving away the vehicle's identity. Nevertheless, several research as e.g. [20, 41, 9] have shown that it is easy to infer identity from such data sets and thereby de-anonymize them.

2.2.2. Inference attacks on published anonymized data

Research has shown, that from a location data set that is pseudonymous, i.e. the identifiers have been stripped off or the data set has been anonymized in another way, it is possible to infer the home location of single users through so-called inference attacks [20, 9, 10, 16, 41] and also the work location with a slightly lower probability [9, 10]. The same problem has been identified by Kajino et al. [19] when using data collected through crowdsourcing. 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. One possibility is to reverse code GPS coordinates to addresses and then e.g. search for entries in telephone books to infer the user's identity from it's home location [20, 10, 16]. 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. The same problem also arises in the area of IoT [34, 16].

2.3. Countermeasures to prevent inference attacks

In general, it has been found by Sweeney et al. [36] that the problem can be solved by providing k-anonymity for the data set. A data set is k-anonymous, if querying for an identifier in this data set always returns a result set of at least k entries. In order to achieve this, several approaches have been investigated. Krumm [20] proposes spatial cloaking. K-anonymity is achieved by dropping data points or perturbing them or dropping all data points around a random point around the home location. 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 as e.g. by Hoh et al. [17] focus on making it less likely to identify GPS points of one trajectory being subsequent and belonging to the same user.

2.4. Limitations of countermeasures

Usually there is a trade-off between the level of anonymity and the usefullness of the data. When k-anonymity is guaranteed, often the resulting data set becomes useless because the data quality is not sufficient anymore [20, 9, 25, 36, 35]. On the one hand, when the data set is tried to be kept usefull, data suppression algorithms have only limited success and can only reduce, but not eliminate the risk as shown by Hoh et al. [16]. Several research [17, 6, 16] finds that 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. Also Golle et al. [10] find that the applied techniques might not achieve the expected results for individuals who's work and home location are a lot further away than average. Furthermore, Sweeney [35] finds that taking other sources and databases into account, k-anonymity might be compromised due to quasi-identifiers e.g. a combination of attributes that do not identify an individual but allow linking two different data sets and by that creating new identifiers.

3. Methodology

3.1. Overall aggregation approach

In order to avoid having a central raw data set 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 initiated by a central server. The aggregation requests have to be defined upfront. Apart from GPS data, we also take other movement data into account - the number of steps and the current activity e.g. walking. These can be used to enrich the pure GPS data. Hence, an example for an aggregation is the determination of the average number of steps per day accross all users participating in the respective aggregation. An examplary aggregation request sent to a smartphone is depicted in Fig. 3.1. It specifies the timespan which should be covered by the aggregation, the type of aggregation and the current data combined in the ongoing aggregation.

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

Figure 3.1.: Exemplary body of an aggregation request to be served.

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 Fig. 3.1.

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

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

Figure 3.2.: Exemplary response to an aggregation request.

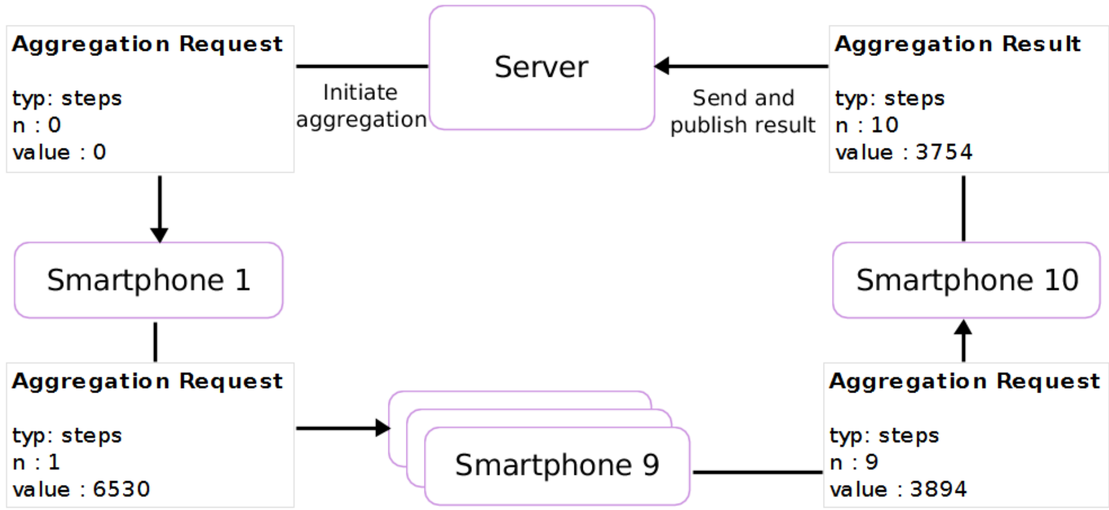

Figure 3.3.: Overview of the decentral unencrypted aggregation process via P2P.

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. On the one hand, P2P on mobile phones though is still hardly possible, if the devices are not locally close to each other. Otherwise, P2P could be established using WIFI and Bluetooth connections [18]. 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 using encryption is depicted in Fig. 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. In order to provide more anonymity, this public-private key-pair could also be exchanged regularly similar to the approach of mix-nodes by [CITE!!!]. The corresponding private key is stored only locally. On start of an aggregation request, not only the first user but also the subsequent user in the chain of users who should deal with the aggregation request is determined and the public key of this 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 leveraging the benefits of synchronous keys using the standard hybrid encryption approach[1]. 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.

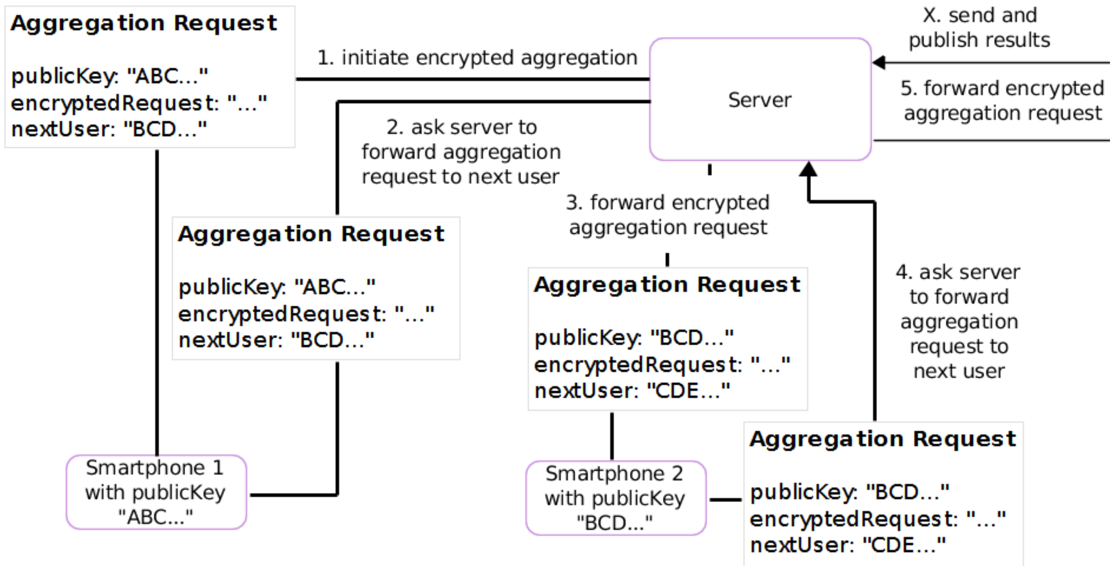

Figure 3.4.: Excerpt of the decentral encrypted aggregation process using a central server for message passing.

The aggregated results are planned to be available through a public route on the server. Nevertheless, we restrict access in our 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. The next section will detail the planned aggregations.

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

3.2. Aggregation schemes

per Two types of aggregation requests are of special interest for our research. First, the aggregation of mean values and second, the aggregation of more advanced statistical values such as median or distribution function. The latter includes the possibility to calculate the former but requires the simultaneous availability of each single user's response. The following list is an excerpt of aggregations that would be of interest and can be computed based on GPS location data in combination with the number of steps and the current activity of the respective person:

1. The average number of steps walked across all users participating in the request. (e.g. to calculate how many people reach e.g. 10.000 steps per day[2].)
2. The average time spent walking, running, in a vehicle or on a bicycle.
3. The share of people who combine using a bicycle and using a vehicle within one trip.
4. The time spent at work and the time it takes people to travel to work.
5. Locations where many participants spend a significant amount of time on a certain day. (i.e. a posteriori event recognition.)
6. The percentage of their overall travelling time that people spend on their bike, car, etc.
7. The average speed on roads.
8. The share of people who go to work by car, bike, etc. or a mixture of these.

For all aggregations, always both, the mean value and if possible, a complete list of single users' mean values in order to compute other statistical figures, are of interest.

3.3. Limiting the spatial area of an 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. Furthermore, the scope of each aggregation would either be the whole user base or the limited number of participants in each aggregation would not necessarily be locally close to each other. The former

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

might result in a huge amount of data being passed around, especially in the case of listing values, the latter would often make aggregations impossible because some aggregations like the average speed on roads require a certain number of participants providing data about the respective area. In order to limit the area of an aggregation and avoid sending the aggregation request to every single user, leaving him with determining whether he can provide data about the respective, the initiator of those requests has to now the area, for which a user can provide data. We do not see this as a violation of the user's privacy due to the following reason: 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 Mokbel et al. [23] and determine the granularity of the location published to the server as follows:

1. Each user sends the most coarse locational area possible to the server - e.g. the continent.

2. If more than the required anonymity threshold of e.g. 10 active users are already registered with this area at the server, the server not only links this location to the user but also requests the user to send a less coarse location.

3. The user step by step sends a less coarse location e.g country, district, etc. until the server denies to link the user to the area because not enough active users have registered with the location on this granularity level. The server nevertheless increases a counter of users who requested access to this area. Once the counter exceeds a certain number, the server sends an aggregation request to all participants in the more coarse area that encompasses the requested less coarse area to verify 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.

4. 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 data only about the city or the bordering area but not both. Nevertheless, this should be limited to areas bordering each other to avoid user identification similar as in Golle et al. [10]. Also, it has to be investigated whether this indeed does not pose a privacy risk to the user or whether also the combination of areas and not only each of the combined 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 among mobile devices [33] 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 aggregation results are made available to the public via a dedicated route of the server but might also be made available directly through granting access to the respective collection[1] of the database.

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

4.2. API

4.2.1. Overall API Architecture

The API is designed as a REST API based on JSON data exchange format and consists of the endpoints listed in Table 4.1.

The routes used for interaction with the application (except the one for creating a new user) are secured and can only be accessed if the user can be authenticated successfully. The routes for initiating new aggregations and retrieving all results require

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

Data Consumption from 30.05.2019 - 04.06.2019	
HTTP call	**Description**
POST /user HTTP/1.1	Creates a new user and corresponding password (see Fig.4.1 and 4.2)
POST /forward HTTP/1.1	Sends a processed aggregation request back to the server (see Fig.4.4)
POST /admin/sampleRequest HTTP/1.1	Saves and initiates a new aggregation (see Fig.4.6)
GET /requests?pk=... HTTP/1.1	Retrieves aggregation requests for a user (see Fig.4.3)
GET /aggregations HTTP/1.1	Retrieves all aggregation results (see Fig.4.5)

Table 4.1.: Description and usage of API endpoints shared by the Android application and the server.

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

Figure 4.1.: Body of an HTTP request for creating a new user.

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

Figure 4.2.: Body of an HTTP response upon successful creation of a new user.

administrator authentication (see Fig. 4.6). We used Postman [TODO: Link] to initiate aggregations. The collections [ref!!!] are here. The route for the results is designed to be available to the public without authentication. Nevertheless we restrict access in this test setup to fully protect users' privacy.
On creation of a new user, the server response contains a password randomly generated for the user to be used for authentication in all future server communication as e.g. in Fig. 4.4. All aggregations waiting to be processed by the user and also the respective responses (see Fig. 4.4 and 4.3) contain the public key of the user to identify which user should handle the request. Furthermore, the request contains a field *encryptionKey*

```
[{
  "publicKey": "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG...",
  "serverId": "acfbdxxfceoela0dkfeecIf",
  "nextuser" :"-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhBms...",
  "encryptionKey":"HXporXstqrgkfsCpezlqZcpLb...",
  "iv":"pMxQznHqbzeZuzVLfiFHYQ==",
  "encryptedRequest":"xtgPteNAeAKEhfPXQeZt..."
}]
```

Figure 4.3.: Body of an HTTP response listing all aggregation requests for the specified user.

```
{
  "publicKey": "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG...",
  "password": "LSFDfzduSFSozfwf",
  "serverId": "acfbdxxfceoela0dkfeecIf",
  "encryptionKey":"HXporXstqrgkfsCpezlqZcpLb...",
  "iv":"pMxQznHqbzeZuzVLfiFHYQ==",
  "encryptedRequest":"xtgPteNAeAKEhfPXQeZt..."
}
```

Figure 4.4.: Body of an HTTP request sending the processed aggregation request back to the server.

which is a synchronous key, encrypted with the user's public key. This synchronous key in combination with the initialization vector *iv* has been used to encrypt the actual aggregation request described in Chapter 3 and send along in it's encrypted version in *encryptedRequest*. The *serverId* is the identifier of the request used by the server's database.

4.2.2. Data Aggregation Design

All aggregations proposed in section 3.2 can be implemented using only 4 fields: An Integer *n* specifying the current number of participants, a Floating Point Number *value* containing e.g. 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 type. The request type specifies the used scheme, i.e. how the other fields are populated e.g. the encoding of the *valueList*. In addition, all

```
[{
  "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.: Body of an HTTP response listing all completed aggregations.

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

Figure 4.6.: Body of an HTTP request for initiating a new aggregation.

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. Fig. 4.5 shows two examples of completed aggregations which highlight the reusability of the 4 fields. They furthermore contain a *timestamp* when the aggregation has been completed and when it had been started (*startedAt*).

Due to limited scope of this thesis, only the following out of the aggregation requests defined in section 3.2 were implemented:

1. Computing the average number of steps walked across all users participating in the aggregation.
2. Computing the average time spent walking, running, in a vehicle or on a bicycle [12].
3. Collecting a list of the average number of steps walked by each participant during the timespan.

In addition, we implemented another aggregation in order to test our setup and the data quality:

4. Collecting a list of all trajectories registered by the users' phones.

The implementation of additional aggregations is straight forward and requires only few changes and additions using the provided setup. All aggregations work as a prove of concept of the proposed setup and hence serve answering research question 1. Aggregation 1, 2 and 3 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. Especially aggregation 3 in comparison with aggregation 1 and 2 allow for answering research question 2. Aggregation 4 imposes a privacy risk on the user as discussed in [41, 9, 20] and was only 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 3.2.

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 [13] 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 framework [11] 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.

The application is aimed at collecting GPS data, detecting the user's current activity and counting the user's steps. The data collection process happens in the background without any user interaction required. Aggregation requests to aggregate data accross devices are also served without the need of any user interaction. 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.

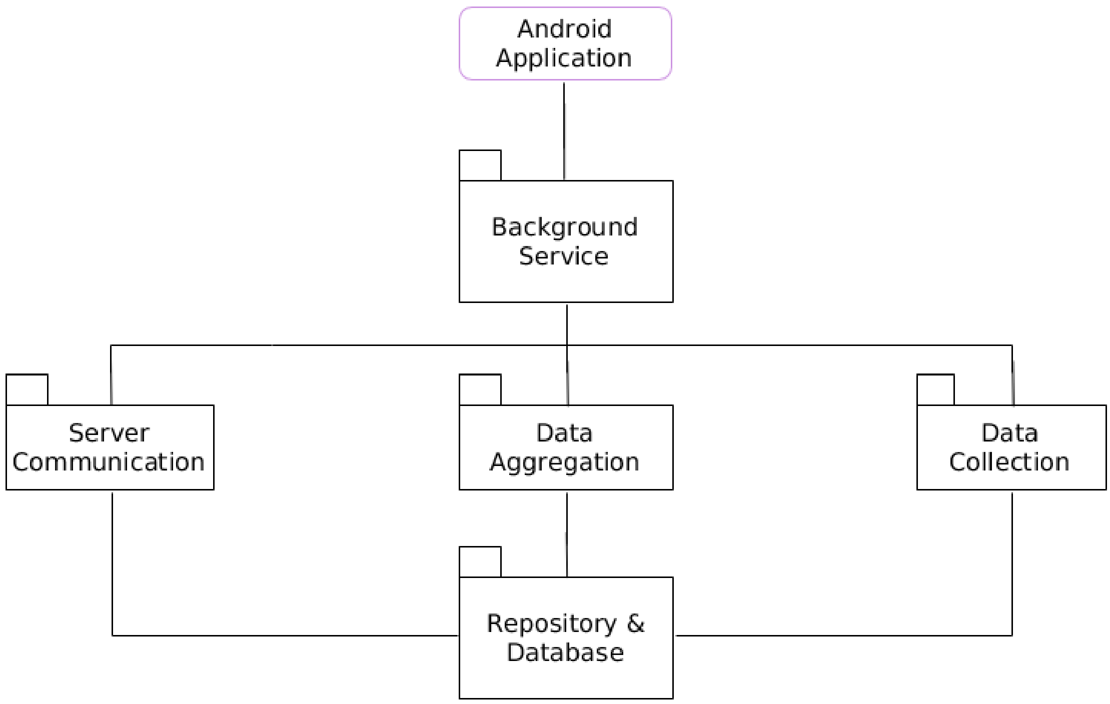

Figure 4.7.: Layered Android architecture with three decoupled main modules controlled by the background service.

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

For the App in order to have maximum possibilities collecting especially GPS data and preventing the Android operating system from entering doze mode when not interacted with by the user, a non-dismissible status notification is displayed at all time[2]. Also, the application is registered to be automatically restarted upon boot[3] 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 [1] and the espresso [2] framework.

4.3.1. Data Collection

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.

We use the Android Room Persistence library [5] to store data locally. The library provides a layer over the standard SQLite database commonly used by 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 vehicle, on bicycle* [12]. Whenever there is a change detected, two events are fired - one for exiting the previous and one for entering the current activity. The events might not be dispatched instantly but contain the timestamp of the

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

[3]From Android 6.0 on (API level 23), apps' behaviour is restriceted by the operating system in order to reduce battery consumption. E.g. all apps are automatically managed by the battery manager which restricts background launches [3, 4]. The user has to switch this option to manual management in order to allow the app to be started in the backgorund after boot.

Sample GPS data set			
Id	**Latitude**	**Longitude**	**Timestamp**
1	44	11	10:11:03
2	44.5	11.1	10:11:15
3	44.4	11.05	10:12:12
4	44.3	11.07	10:33:00
5	44	11.2	10:34:00
6	44.00001	11.2	10:36:10
7	44	11.2	10:38:23

Table 4.2.: Sample GPS data input for calculating trajectories.

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*[4]. 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.

4.3.2. Local Data Aggregation

From the received values of the step counter since last reboot saved in *step_counter_table* the number of daily steps is 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 properties. Those activities are then saved in the *activity_table*. GPS data is used to compute trajectories through the following algorithm examplarily applied to the sample GPS data in Table 4.2:

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

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

separate sequences and each is processed separately as a possible trajectory in the next step. Regarding the sample data, the GPS points with the ids 1, 2 and the GPS points with the ids 3, 4, 5 would be treated as separate sequence.

2. Furthermore, 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. For example, regarding the GPS point with id 1, the GPS point with id 4 is the first one with at least two minutes in between.
   b) If the average speed between those two points calculated from the time and the distance between the points is below 0.6 m/s, the pair is added to a list to be processed in the next step. This is the case e.g. for the pairs with the ids 5, 6 as well as the pair of 6, 7.
   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 in their timespan, they are fused to a new pair covering the combined timespan and consisting of the first pair's first GPS point and the last pair's last GPS point. An example here is the sequence from id 5 to 7.
3. The GPS pairs of still moments resulting from the last step are used to exclude still moments from the original sequence of GPS points and hence divide it into subsequences each defining a single trajectory. In the example the resulting sequences are 1,2,3 and 4,5.

Of each trajectory, the start and end location as well as the respective timestamps are then saved in the *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 fit the tested sample the best. 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.

4.3.3. Serving Aggregation Requests

We use the retrofit2 framework [31] based on OkHttp [30] to handle communication with our REST server described in Section 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 interceptor 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. 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 process 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 types *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 position, longitude of start position, latitude of end position and longitude of end position.

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

4.4. Server Design and Implementation

The server is build using the event-driven node.js verion 10.15.3 leveraging the express web-server framework [24] and using the mocha testing framework [22] in combination with the chai assertion library [8] for unit and integration testing. The server is designed using a layered architecture as described in Fig. 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 these models and persists data in a mongoDB object store. They also handle transactions where several objects are modified subsequently. The third level provides the logic to be executed for each endpoint defined in *routes.js*. On the top level, *server.js* starts the server on the specified port, registers the routes described in Section 4.2, respectively adds authentication and interacts directly with the *userRepository* in order to update the respective users *lastSeen* property. Furthermore, it starts a scheduled repeating task in order to bypass users in the aggregation chain that have not responded to their aggregation request after a certain time. This process is described in Section 4.4.2.

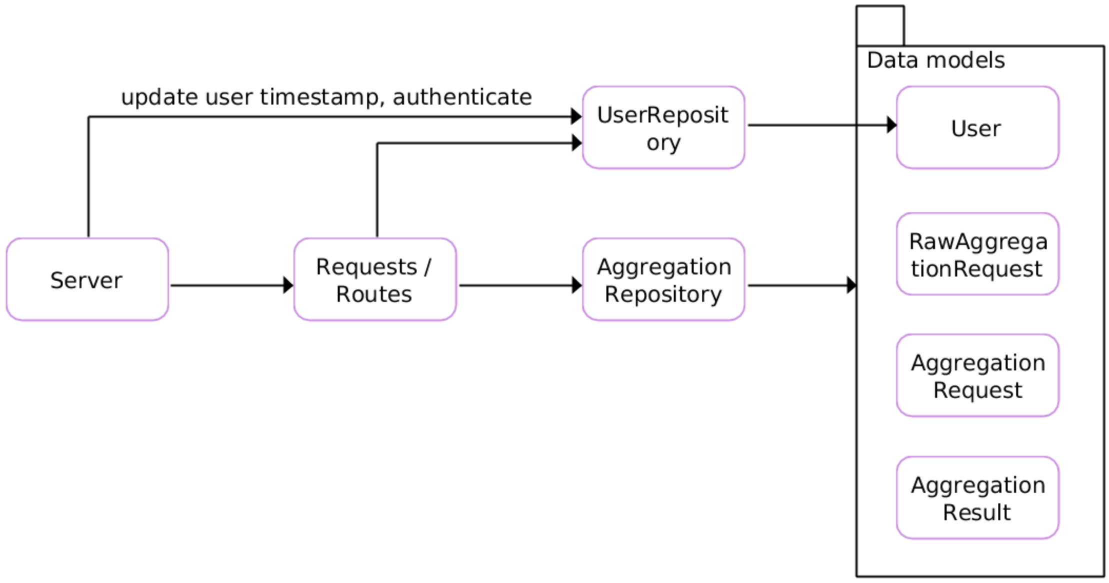

Figure 4.8.: Layered Server architecture with repositories and data models on a separate layer.

4.4.1. Data Model

We organize the data in four collections. The user collection stores the user data which is the corresponding 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, and type. Furthermore, the three fields n, value, valueList reused across all aggregations, the timestamp when the request was filed to the server and a flag indicating whether this request has already been started [5] are stored. Upon start of an 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 user's public key and stored in the collection *aggregationRequests*". Each time, a user requests an *aggregationRequest*, proceeds with it and sends the result back to the server, the result is inserted into the database as a new *aggregationRequest*. The fields of this collection are described in Table 4.3.

[5]In case of a newly inserted aggregation where the end date is in the future, the request will be started only when this day has passed.

Fields of *aggregationRequest*	
Name	**Purpose**
rawRequestId	The id of the related *rawAggregationRequest* . This field is not available through the API.
started_at	The timestamp, when the request was started.
publicKey	The public key of the next user that should process this request.
nextUser	The public key of the user that will receive the request afterwards. This is necessary so that the user that should process this request can encrypt the processed request with the public key of the next user.
previousRequest	The id of the previous request which is null, if it is the first request in the chain. This field is used by the process described in Section 4.4.2 in order to bypass a not responding user in the chain.
users	A 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 used to encrypt the request itself. The synchronous key in turn is 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.
completed	A flag indicating whether this aggregation request has already been processed by the respective user.

Table 4.3.: Fields of the collection *aggregationRequest*.

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. Additionally, it contains the fields *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 days from 30.05.2019 until 04.06.2019. The version of the Android application deployed during field testing can be found on GitHub [27]. However, evaluating the *lastSeen* timestamp of the users showed that only 13 installations were active after 31.05.2019. The server was deployed in the IBM Cloud as a 128 MB node.js instance. The version deployed during field testing is hosted on GitHub [28]. The latest version of the same repository provides the same functionality but includes improvements (e.g. bug-fixes, comments, ...) over the deployed version. The database was hosted as a free version at mongodb.com.

We ran 7 different aggregation requests defined in section 4.2.2 covering different timespans from single days to the whole time period. Also, we limited the number of participants in each aggregation to 10. The results can be found in the tables in this section and are also available in JSON format on GitHub [29]. The results will be discussed in Chapter 5.3. The results of the collection of trajectories were modified in order to protect the priacy of all research participants. 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. All improvements are incorporated in the most recent commits of the respective GitHub repositories.

5.2. Data Consumption

As expected, the data consumption of the Android application was low. Some research participants provided information about the app's data usage. Table 5.1 shows the 12 collected results of this rather qualitative than quantitative analysis. Screenshots of the provided information are attached in the Appendix. On 75% of the devices the data consumption for 6 days was below 21 MB. The highest data consumption was 376 MB and is attributable to an error that occurred on the last day of the testing period. Requests were sent multiple times during the whole period and up to unlimited times on the last day due to this error in combination with another error that was present during the whole timespan. This explains the high data consumption reported also by two other participants as the error occurred at least on three devices. The data suggests that the average data consumption in a production environment would be lower and regarding the size of our setup below 100 MB per month. Furthermore, a mobile connection is not required. In the future, the setup can be changed to use only or mostly WIFI connections. Some few available reports about battery consumption also indicate a rather moderate battery consumption.

Combined Data Consumption from 30.05.2019 - 04.06.2019	
Data Consumption	**Comment**
0 MB	mobile data only, WIFI unknown
2.6 MB	
6.19 MB	mobile data only, WIFI unknown
6.29 MB	mobile data only, WIFI unknown
8.18 MB	
8.5 MB	From 01.04.2019 - 04.06.2019 only
11.98 MB	
18.53 MB	16.5 MB mobile, 2.03 MB WIFI
20.5 MB	
75.73 MB	From 01.04.2019 - 04.06.2019 only, possible error candidate
49.39 MB	mobile data only, WIFI unknown, possible error candidate
376 MB	Known and fixed error

Table 5.1.: Overall data consumption of the Android application during the testing period.

5.3. Aggregation Results

We computed the aggregation over several different timespans. We aggregated the average number of steps across all participants, the average time spent walking, running, in a vehicle or on a bicycle and collected a list of each individual's average number of steps. The results can be seen in Table 5.5 - 5.4. The results from the experimental aggregation of trajectories will be discussed in Chapter 5.3.4.

5.3.1. Validity of results

Excluding the value of 30.05.2019 where the collection of results started and the data of the fraction of the day before installation is not included, the average time spent walking

Average number of steps per day		
Time period / Day	**N**	**Value**
30.05.2019	5	4332
31.05.2019	5	3440
01.06.2019	1	17
03.06.2019	3	455411
04.06.2019	3	1123
30.05.2019 - 31.05.2019	6	3279
30.05.2019 - 31.05.2019	5	4030
30.05.2019 - 01.06.2019	6	4271
30.05.2019 - 03.06.2019	5	643959
30.05.2019 - 04.06.2019	5	3623
31.05.2019 - 04.06.2019	3	4681
01.06.2019 - 04.06.2019	2	2089
02.04.2019 - 04.06.2019	3	326363
03.04.2019 - 04.06.2019	3	455904

Table 5.2.: Average number of steps per day aggregated across all participating users for different timespans.

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

Table 5.3.: Average time spent walking (in minutes) aggregated across all paricipating users for different timespans.

computed in the aggregations ranges from 52 to 146 minutes per day. The average time spent walking computed for the whole timespan where data from 8 devices is included amounts to 62 minutes. At the same time, reported number of steps per day ranges from 1123 to 4682 per day, excluding outliers (see Table 5.2). The average number of steps computed for the whole timespan with data from 5 participating devices is 3622. Taking an average speed of roughly 100 steps per minute into account [40], the results from average time spent walking and average number of steps diverge by a factor of 1.7. Considering that apparently only 6 devices had a step sensor on their phone and the sample size of the steps aggregation is only about half of the sample size of the other aggregations, this deviation does not seem out of range. Furthermore there was an error computing the number of steps on at least one phone which caused some of the results to be off. Regarding Table 5.4 it is clear, that the error occurred in the local aggregation process and it seems likely that only on one device this error occurred, as in each aggregation only one value is errorneous.

The average time spent running ranges up to only 2 minutes per day with one day having a 0 value across 8 participating devices. Most likely, only a fraction of the

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
01.06.2019	4	17
03.06.2019	9	914, 1042, 1364278
04.06.2019	8	1103, 13332, 934
30.05.2019 - 31.05.2019	10	550, 2905, 3195, 775, 8828, 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
30.05.2019 - 03.06.2019	9	598, 3195, 1669, 8828, 3205505
30.05.2019 - 04.06.2019	8	757, 1719, 8828, 3614, 3195
31.05.2019 - 04.06.2019	7	757, 9203, 4084
01.06.2019 - 04.06.2019	7	826, 33511
02.04.2019 - 04.06.2019	7	974691, 1231, 3168
03.04.2019 - 04.06.2019	7	1364278, 1231, 2202

Table 5.4.: List of the average number of steps per day for each user aggregated for various timespans.

research participants goes running on a regular basis and probably even less take their smartphone along. This assumption reflects the low values for the average time spent running. Especially as attributing the whole average value to one device only would mean that one person conducted a run of less than 20 minutes seems unlikely, while it seems likely that quite some persons run for a short amount of time e.g. to catch a train or metro. Nevertheless, we cannot prove this assumption. It furthermore highlights, that for thorough analysis the aggregation of the mean value is not sufficient. Similarly, the average number of steps across all participants of an aggregation is not very robust and does not provide much information. Table 5.4 provides the average number of steps of each individual participating in the request. With the median ranging from 3195 to 3565 in all aggregations with more than 3 valid values and a mean number of 5205 steps in Germany [32] we have no doubt about the correctness of the values except the values resulting from (probably only one) errorneous device.

The average time spent in a vehicle (including means of public transport) ranges from 36 to 120 minutes (excluding the aggregation only covering the first day). The average time spent in a vehicle computed by the aggregation covering the whole timespan is 59 minutes. The average time spent biking ranges from 11 to 31 minutes. The aggregation across the whole timespan was errorneous and had to be discarded. Both aggregations

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

Table 5.5.: Average time spent running (in minutes) aggregated across all paricipating users for different timespans.

Average time spent in a vehicle [min]		
Time period / Day	**N**	**Value**
30.05.2019	10	8.24
31.05.2019	10	119.92
01.06.2019	4	90.40
03.06.2019	9	71.74
04.06.2019	8	35.51
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
30.05.2019 - 03.06.2019	9	62.16
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 5.6.: Average time spent in a vehicle (in minutes) aggregated across all paricipating users for different timespans.

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

Table 5.7.: Average time spent biking (in minutes) aggregated across all paricipating users for different timespans.

- the time spent biking and the time spent in a vehicle seem reasonable to us, especially as all participants were active in the Munich area which is also a quite bike friendly city. Furthermore, the collected trajectories in section 5.3.4 show two rather long trajectories which probably contributed to the high average time spent in a vehicle in one of the aggregations.

5.3.2. Implications

The same aggregation covering the timespan from 30.05.2019 to 31.05.2019 has been computed twice for 4 of the aggregations. The results show that not only the number of participants varies but also the value varies from 4% to 18%. For each aggregation, the selected participants might be differnt. With a total number of 16 participants in our research, though there has to be an intersection. Nevertheless, the deviation in the same aggregation is within an acceptable range and will be more robust, when the number of participants is increased. Furthermore, the median value will be even more robust to computing the same aggregation twice. We see the same phenomena e.g. regarding the aggregations for the time spent running on 30.05.2019, 31.05.2019 and the aggregation covering both days. The aggregation covering both days yields a higher value than each single aggregation which is due to the fact of a differing underlying research population.

TODO: Compute average of steps and compare

5.3.3. Inference on aggregated data

Regarding the aggregations computing mean values across all participants, we do not see any possibility to infer information about the users who participated. Especially as not even the identifier of the user used by the server is published alongside with the aggregation. In addition, we showed that when repeating the same aggregation request, the user base changes and the value accordingly. Nevertheless, from the two different values for the same aggregation request, we cannot draw any inference. Also an overlapping in the aggregations e.g. 30.05.2019, 31.05.2019 and an aggregation covering both days as well as the aggregations covering the first two days and the last five days does not allow for inference in the case of mean values aggregated across all participants. The collection of individual mean values in Table 5.4 shows that with a high probability the same participant with an average value of 8828 steps per day who participated in the request covering the first two days also participated in the request covering the whole timespan. Based on the information that days with a zero value of the step counter are excluded also locally from the aggregation, we can infer that this person's phone did not register any steps after the first two days, otherwise they value in the second aggregation would have changed. Similarly, we can infer that the user with 757 steps on average per day who participated in the requests covering the whole 6 day timespan and the aggregation covering only the last 5 days probably did not register steps on the first day. In summary, we can only infer information about the fact of some users who participated in one aggregation, did or did not participate in another aggregation. As this information is endogenous to our data-set, it can in no case be used to be linked to other data-sets in order to infer further information. Assuming that we would also have the collected averages of all aggregations and not only steps. Furthermore, linking different aggregations e.g. the number of steps and the time spent walking is only possible with a limited likelihood due to the following reasons. First, there is no exact match of e.g. time spent walking for a number of steps, so that always a range of values of another aggregation would fit. Next, we cannot even assure that the same user participated in both different aggregations. Assuming nevertheless that there is another data-set where the combination of walked steps, time spent walking, etc. is a quasi-identifier as defined by [35], the likelihood of obtaining a correct link is low due to the afore mentioned reasons and a good level of anonymity regarding k-anonymity would always be supplied naturally. And still, those aggregations do not expose raw location data / GPS data points which are usually the risk.

5.3.4. Trajectories

Using the algorithm described in Chapter 4.3.2 a total of 406 trajectories where computed from the raw GPS data. The experimental collection of trajectories clearly shows privacy risks as pointed out e.g. by [9], why we only publish a modified results set[1]. Most of the trajectories are shown in Fig. 5.1 and 5.2. The complete results can also be found in the Appendix.

In more than 90% of all cases, local trajectories obtained from one of the researchers participating devices could match an activity for more than 90% of the time. This data data suggests that these trajectories can easily be linked locally and a change of transport system e.g. metro to bike can be identified. For example at the locations Giselastraße, Odeonsplatz and central station, many trajectories end or start. Mapping the current activity to those trajectories enables aggregations as mentioned in Chapter 3.2. Noting that locally all data points of the trajectories are available, it is also easy to compute the distance travelled or infer a more granular transport definition e.g. train, bus, car, etc. 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

[1]Whenever the trajectory started or ended clearly in a precise private location e.g. housing or work-place, we slightly modified this trajectory. So the results do not represent actual trajectories. Nevertheless, the meaning of the results should not have changed.

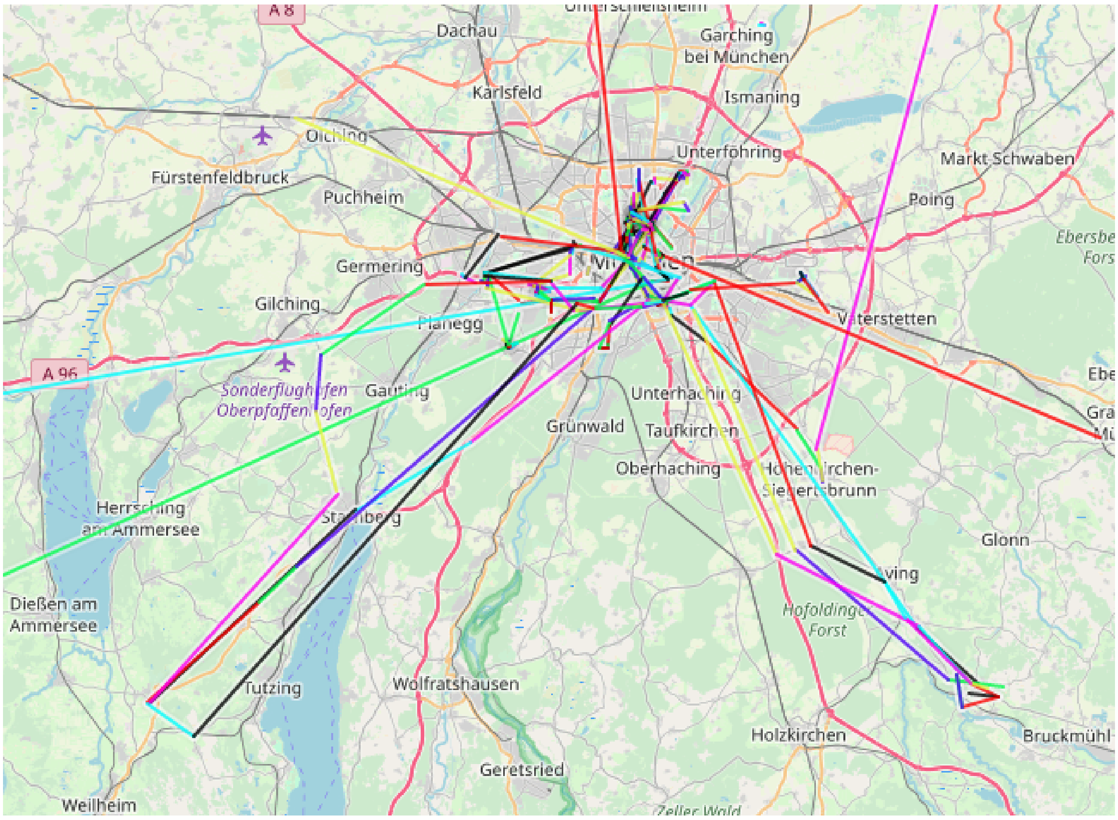

Figure 5.1.: Excerpt of all trajectories in the greater Munich area.

public transport) to be combined with bike.

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)

5.4. Further scenarios e.g. Scenario Traffic Jam

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

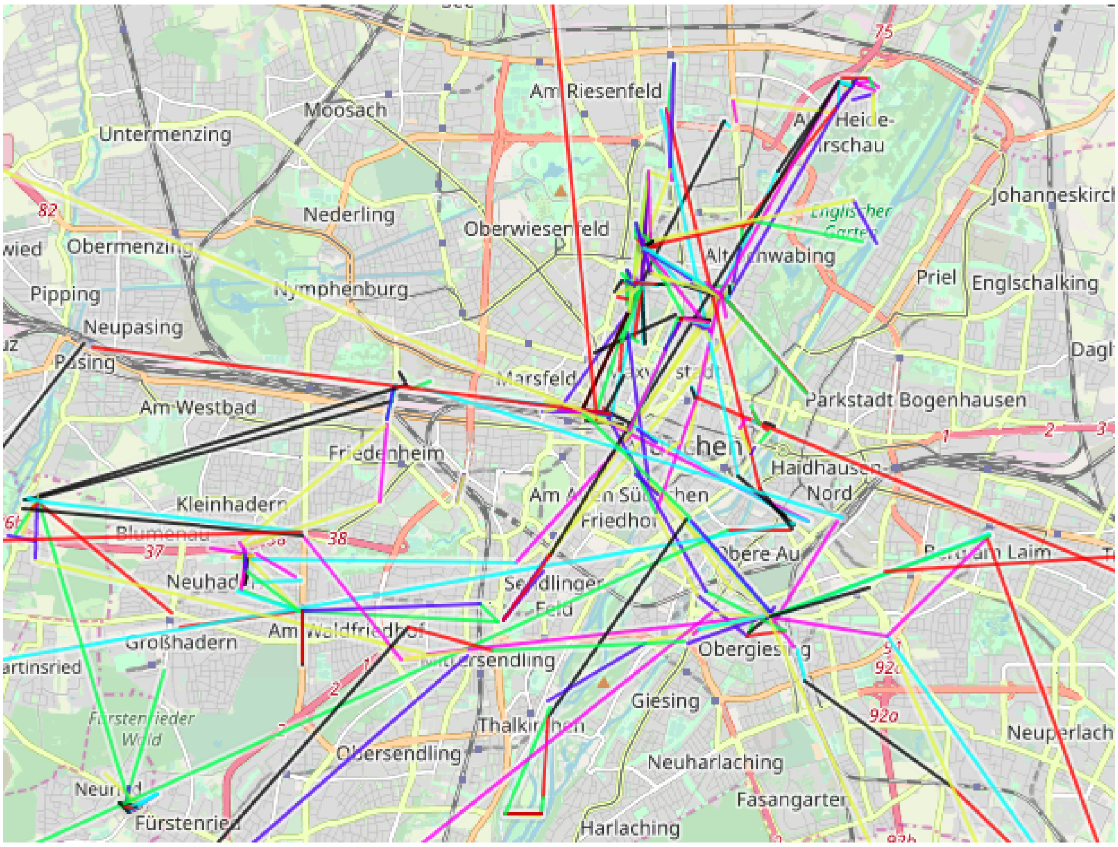

Figure 5.2.: Excerpt of all trajectories inside Munich.

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.

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.

(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.)

6. Conclusion and Discussion

6.1. Conclusion

RQ1: How is aggregation of location data possible without raw data being accessible to anybody but the owner?

We developed and field tested an approach that allows for decentral aggregation of data that preserves the privacy of users. Due to fast-changing IP-addresses, stable P2P on mobile devices is currently limited to WIFI and Bluetooth connections. Hence, we proposed a solution based on the assumption of a trusted server that forwards messages between the different Android mobile phones. Nevertheless, the messages are encrypted and only the target device can encrypt the message, as long as the server is not compromised. On the mobile devices, raw data i.e. GPS data, steps and detected activities is collected and locally aggregated through an Android application. This locally aggregated data is then used to serve aggregation requests initiated and forwarded by a central server but the raw data and the locally aggregated data itself is never send away from the device. Thus, considering the assumption of a trusted server and trusted devices, we propose a solution that indeed manages to aggregate locational data without the raw data being exposed from the collecting devices.

RQ2: What types of aggregations can be published?

We used the proposed setup to compute mean values for the number of steps and the time spent in the activities walking, in a vehicle, biking and running across all aggregation participants in our field testing. In Chapter 5 we presented and discussed the results. We found that we had no chance to infer any information about the participating users from the mean values. During field testing we also collected lists of individual mean values for the average number of steps per day. These lists enable the computation of e.g. median value or distribution function. We have not found any trivial possibility to infer any information about the identity of the user. Furthermore, it does not seem feasible to link values from two subsequent aggregations to the same user. We further showed in Section 5.3.3 that it seems unlikely to infer that the same user participated in two different types of aggregations e.g. the computation of the average number of steps and the computation of the average time spent walking.

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

We found in Chapter 5 that it is possible to infer that the same user participated in different aggregations that overlap regarding the covered timespan. Nevertheless, this is on the one hand based on the fact that the user did not provide data for the days that were only covered by one of both aggregations. On the other hand, we do not see a possibility to infer further information e.g. the identity of the user from this inference.

6.2. Limitations

As stated in Chapter 3, our approach is based on trust among the clients and the server. Nevertheless, while e.g. Kajino et al. [19] 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 if devices are locally close to each other or P2P generally becomes possible on mobile devices. Trust can also be established through various means e.g. Weng et al. [21] propose a solution through blockchain technology.

In addition, the field testing should be repeated with more users and more aggregations in order to obtain a larger data set. We do not expect the findings to be any different though, because the key to publication without privacy concern seems to be that never a GPS location is included in the publication. Only in the question of the aggregation itself a GPS location might be referenced e.g. by limiting the area of the aggregation as proposed in Chapter 3.

6.3. Future Work

In order to continue this research, the provided setup was carefully designed to allow for very flexible adaptability and extensibility. Some parts proposed in Chapter 3 were not implemented in this research due to scope limitations. The implementation of these features e.g. the proposed but not yet implemented aggregations, limiting the area of an aggregation and the incorporation of the findings of our field test are a good starting point for future work. In addition, the following Subsections show options of improvement and further research in the respective areas of the Android application and the server itself as well as more conceptual advancements.

6.3.1. Android application

There is a list of promising rather technical improvements to the Android application. They are listed as open issues at the public GitHub repository and should be resolved or implemented but were out of the scope of our research. Furthermore, our experience showed that some essentials as e.g. providing a nice information screen to the user when opening the application or showing the own locally aggregated data would boost participation. The aggregations publicly available on the server could also be used to show a comparison of e.g. the personal number of steps to the average or the median. Also making the application available through the play store would be a great relief for the user and simplify the installation process a lot and in addition enable the roll-out of updates and client-side fixes during field testing. Also the option to delete local data after some time or export it might increase user adoption. Furthermore, the user could be asked to specify home and work location which are then stored locally. This spares having to locally infer these locations while enabling better aggregations and also even more guaranteed privacy protection.

6.3.2. Server

The main possibilities for improvement concerning the server center around dealing with the problem of not available users blocking the aggregation chain. Dynamically selecting the subsequent user at every step and not selecting the whole list of users upfront would allow for benefits from taking the probability of the user being active into account. Nevertheless, no matter when the users are determined, it has to be investigated that the selection of users for an aggregation is does not lead to a bias in the aggregation result. Furthermore, encrypting an aggregation not only for one but for several subsequent users[1] would generate some redundancy overhead but yield fewer problems due to not responding users. Some other rather technical improvements like e.g. pre-populating aggregation requests in order to hinder the first users in the chain from inferring the data of the previous user with a high probability and cleaning the aggregation result from this pre-filling upon completion of the aggregation can be found in a list of issues at the respective GitHub repository as described in Section 6.3.1.

6.3.3. Related Research Areas

Our research is very close to some related research areas e.g. decentral computation. Our setup can be used as well as a base for research in this area where problems might

[1]Once one of these users processes the aggregation, the aggregation requests to the other users become disabled

be solved locally and collected by the server afterwards and the pieces put together still with anonymity provided for the users. Similarly, our setup would also allow for pre-populating device databases with artificially generated or elsewhere obtained data e.g. in order to verify whether aggregated results provide the same insights as one could obtain by analyzing a raw data set. The application can also be modified to be a framework that can easily be incorporated into other apps especially in order to allow for a broader user base in research projects. Trust is a basic assumption of our framework. A whole area of research e.g. blockchain technology or credit systems or use of third parties deal with the question of how to establish or verify trust.

6.4. Reproducability Considerations

The ReadMe file of the server [26] provides detailed instructions of how to install and run the server proposed in our work and where in the code the url of the database has to be provided. Section 5.1 gives an example of a deployment option but the software can be installed on any server. The Android application can be installed on any compatible device[2] using the *.apk* file included in the realease [27]. Once the server is running, the application automatically registers with the server upon successful installation and granting of location access. Aggregations can be initiated using the API endpoint detailed in Section 4.2 and Fig. 4.6. The results obtained should resemble the results obtained in our work. Nevertheless, the actual data will vary accordingly to the activity of the participants.

[2] As stated in Section 4.3, the minimum required API level is 19 and Google Play Services has to be installed on the device.

A. Data Usage Screenshots

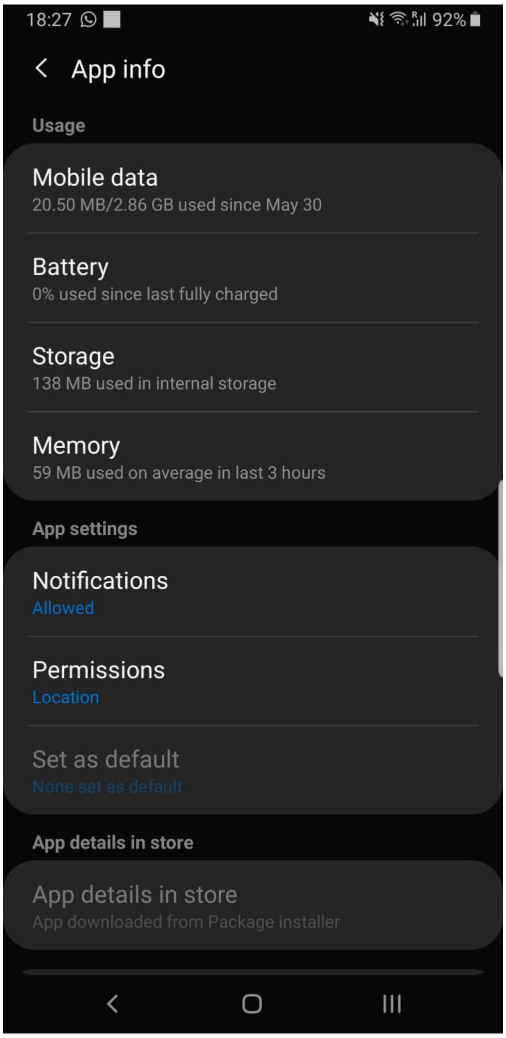

Figure A.1.: Data consumption screenshot 1.

Figure A.2.: Data consumption screenshot 2.

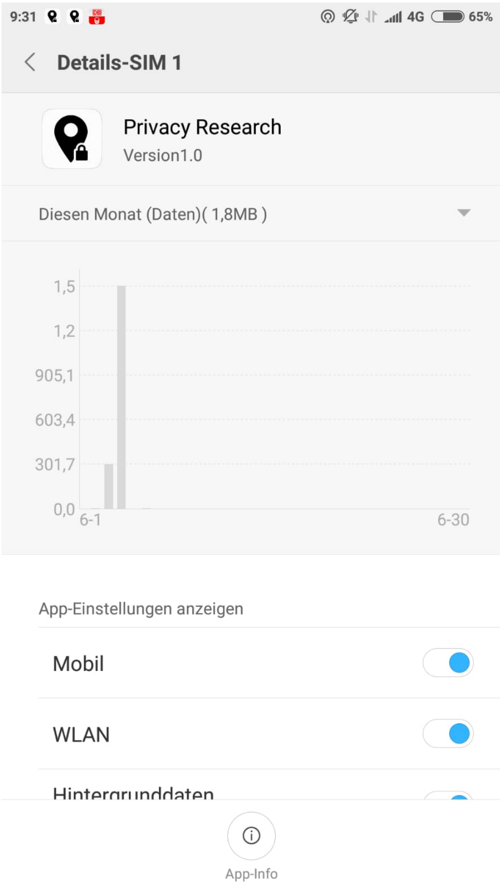

Figure A.3.: Data consumption screenshot 3.

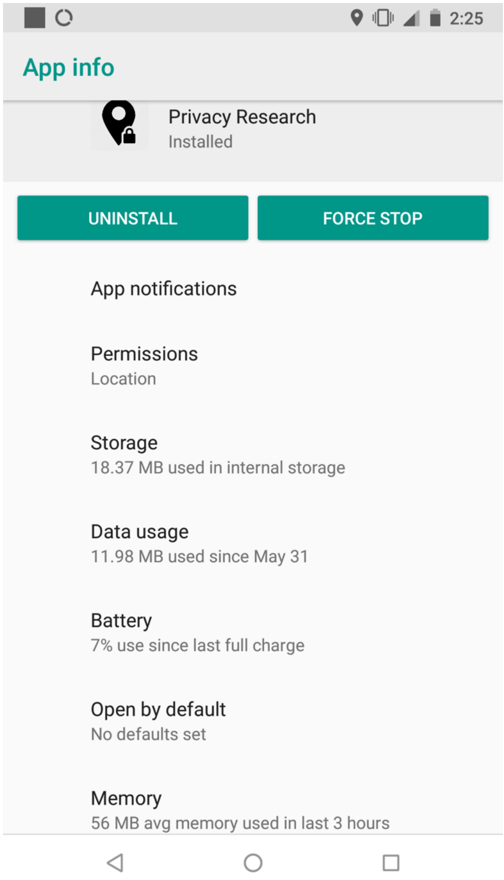

Figure A.4.: Data consumption screenshot 4.

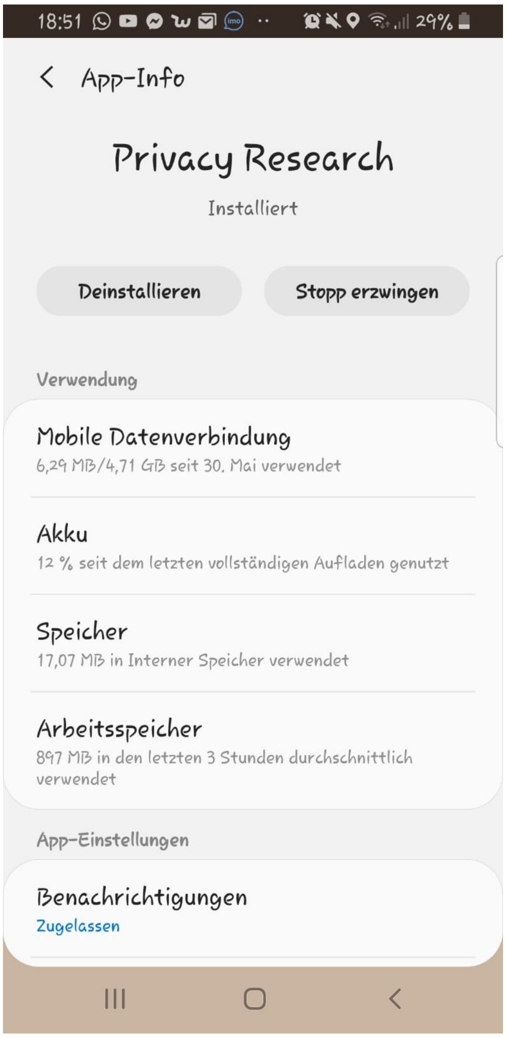

Figure A.5.: Data consumption screenshot 5.

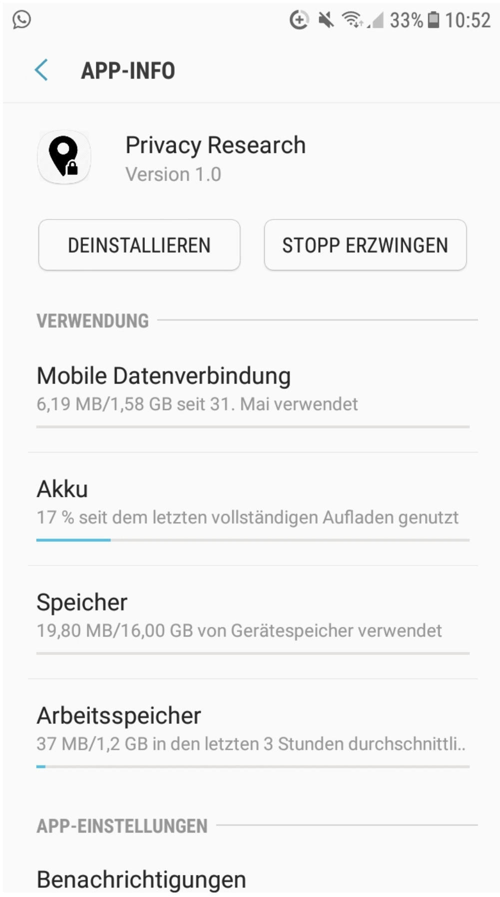

Figure A.6.: Data consumption screenshot 6.

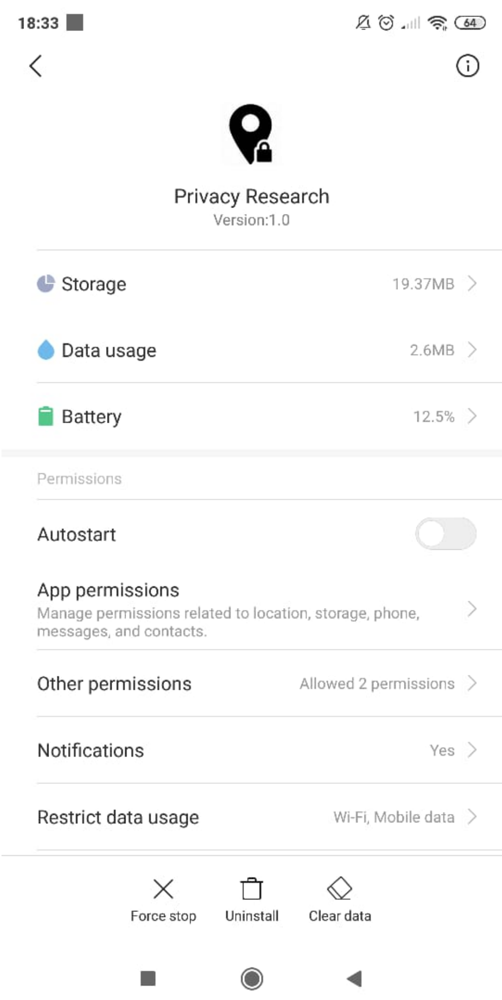

Figure A.7.: Data consumption screenshot 7.

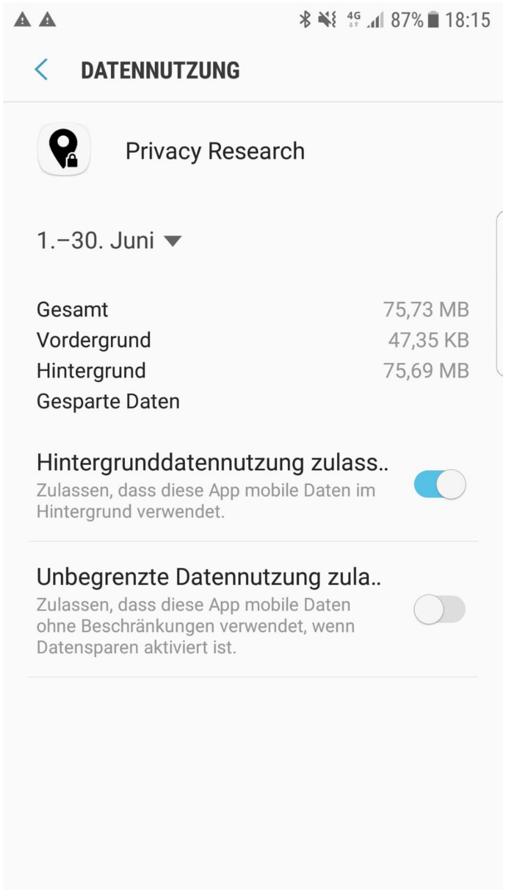

Figure A.8.: Data consumption screenshot 8.

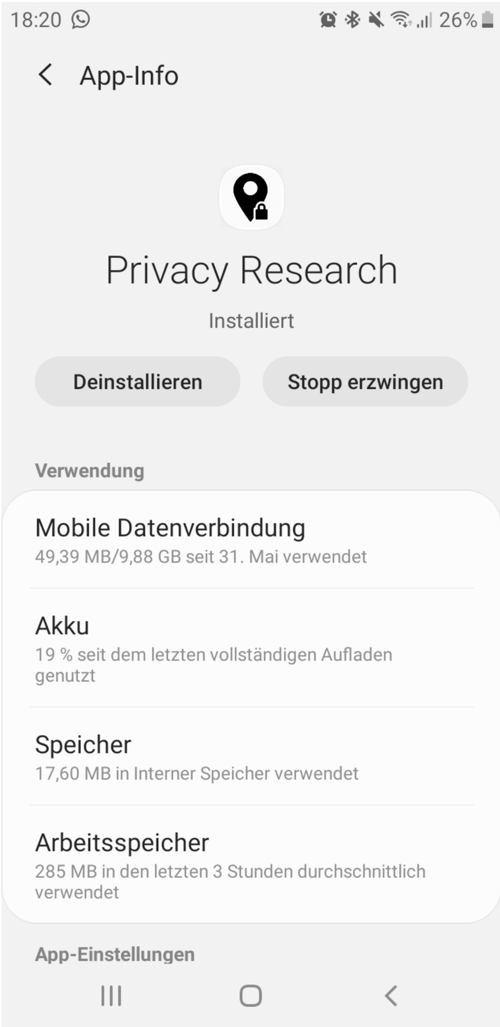

Figure A.9.: Data consumption screenshot 9.

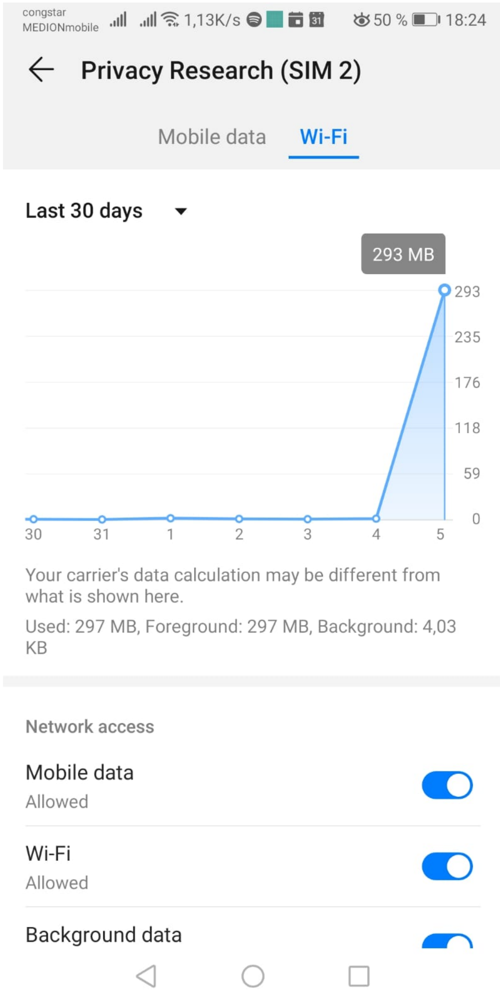

Figure A.10.: Data consumption screenshot 10.

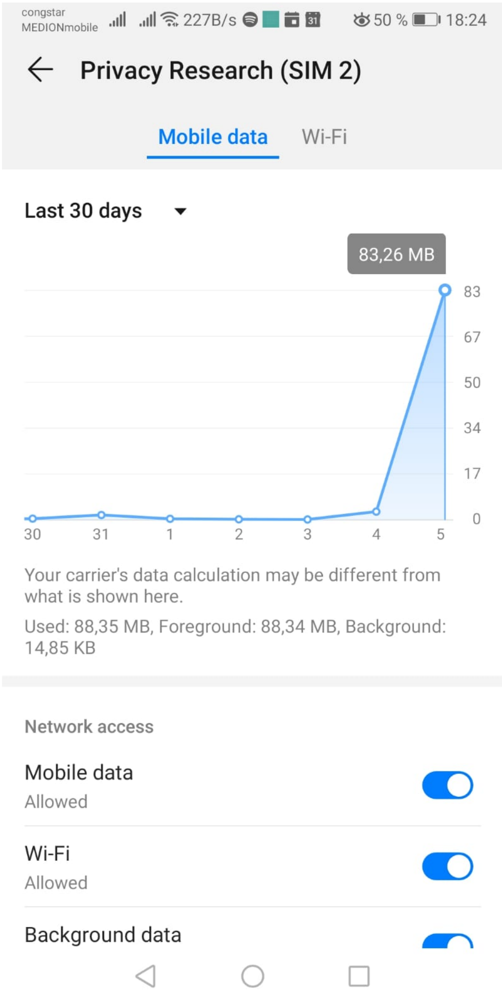

Figure A.11.: Data consumption screenshot 11.

List of Figures

List of Tables

Bibliography

[1] Android Developers. *AndroidJUnit4*. https://developer.android.com/reference/android/support/test/runner/AndroidJUnit4 [Accessed: 2019-06-24].

[2] Android Developers. *Espresso*. https://developer.android.com/training/testing/espresso [Accessed: 2019-06-24].

[3] Android Developers. *Guide to background processing*. https://developer.android.com/guide/background/ [Accessed: 2019-06-24].

[4] Android Developers. *Optimize for Doze and App Standby*. https://developer.android.com/training/monitoring-device-state/doze-standby [Accessed: 2019-06-24].

[5] Android Developers. *Room Persistence Library*. https://developer.android.com/topic/libraries/architecture/room [Accessed: 2019-06-24].

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

[7] 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.

[8] Chai. *Chai Assertion Library*. https://www.chaijs.com/ [Accessed: 2019-06-24].

[9] 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: (2019).

[10] 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.

[11] Google. *Google*. https://dagger.dev/ [Accessed: 2019-06-24].

[12] Google Developers. *DetectedActivity | Google APIs for Android*. https://developers.google.com/android/reference/com/google/android/gms/location/DetectedActivity [Accessed: 2019-06-24].

[13] Google Developers. *Distribution dashboard*. https://developer.android.com/about/dashboards [Accessed: 2019-06-07].

[14] Harvard Business Review. *Filing for a Patent Versus Keeping Your Invention a Trade Secret*. https://hbr.org/2013/11/filing-for-a-patent-versus-keeping-your-invention-a-trade-secret [Accessed: 2019-06-20].

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J.-N. Liu, Y. Xiang, and R. Deng. "Crowdbc: A blockchain-based decentralized framework for crowdsourcing." In: *IEEE Transactions on Parallel and Distributed Systems* (2018).

[22] Mocha. *Mocha - the fun, simple, flexible Javascript test framework*. https://mochajs.org/ [Accessed: 2019-06-24].

[23] 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.

[24] Node.js Foundation. *Express - Node.js web application framework*. https://expressjs.com/ [Accessed: 2019-06-24].

[25] 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.

[26] Simon van Endern. *ReadMe with setup instructions for the server*. https://github.com/SimonVanEndern/location-server/blob/v1.0/README.md [Accessed: 2019-06-24].

[27] Simon van Endern. *Release Final Android app version used during field testing*. https://github.com/SimonVanEndern/location-app/releases/tag/v1.0 [Accessed: 2019-06-24].

[28] Simon van Endern. *Release Final server version used during field testing*. https://github.com/SimonVanEndern/location-server/blob/v1.0 [Accessed: 2019-06-26].

[29] Simon van Endern. *Results of field testing*. https://github.com/SimonVanEndern/tum-thesis-latex/tree/master/published/results [Accessed: 2019-06-24].

[30] Square Inc. *OkHttp*. https://square.github.io/okhttp/ [Accessed: 2019-06-24].

[31] Square Inc. *Retrofit*. https://square.github.io/retrofit/ [Accessed: 2019-06-24].

[32] Statista. *Daily step averages in select countries worldwide 2017*. https://www.statista.com/statistics/747380/daily-step-averages-select-countries-worldwide/ [Accessed: 2019-06-23].

[33] Statista. *Mobile OS market share 2018*. https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/ [Accessed: 2019-06-07].

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

[35] 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.

[36] 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.

[37] The Economist. *The world's most valuable resource is no longer oil, but data - Regulating the Internet giants*. https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data [Accessed: 2019-06-20].

[38] The Guardian. *Facebook says 14m accounts had personal data stolen in recent breach*. https://www.theguardian.com/technology/2018/oct/12/facebook-data-breach-personal-information-hackers [Accessed: 2019-04-10].

[39] The Guardian. *Tech giants may be huge, but nothing matches big data*. https://www.theguardian.com/technology/2013/aug/23/tech-giants-data [Accessed: 2019-06-20].

[40] The New York Times. *Walk Briskly for Your Health. About 100 Steps a Minute.* https://www.nytimes.com/2018/06/27/well/walk-health-exercise-steps.html [Accessed: 2019-06-23].

[41] 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.