Lower bound computation for SecureRide: A cheat-proof ride-hailing service

Concetto Emanuele Bugliarello, Anh Pham, Italo Dacosta
Privacy Protection (CS-622), Mini-project, Fall 2016, EPFL, Switzerland
{emanuele.bugliarello, thivananh.pham, italo.dacosta}@epfl.ch

***Abstract*—Ride-hailing services have become widely used. Such services, however, rise security issues such as the risk of drivers cheating on their trips to increase their fares. In this report, we propose a solution to partially tackle this issue by computing a lower bound on the traveled distance based on geometric algorithms and cryptographic techniques. Combining the lower bound with an upper bound will then provide a reliable way to detect cheating. Our system relies on existing Wi-Fi access point networks deployed in urban areas and we evaluate our solution by using real datasets from the FON community networks and a set of GPS updates from taxis in Rome. We report the accuracy of our solution along with a sensitivity analysis of the density of access points along trips.**

I. Introduction

Over the last decade, the spread of smartphones with increasing capabilities has given rise to new services, aiming for a better life. Location-based services (LBSs) offer very high utility for users, providing entertainment, navigation aid, and activities based on the real-time location of the user. More recently, ride-hailing services, a new type of LBSs, are increasingly used by people. As the name suggests, ride-hailing consists in a person who hails a car and is immediately picked up and driven to their destination for a time and distance-based fare. Basically, this type of services aims to offer an alternative to the traditional taxi services. For example, among the companies offering such services, as of August 2016, Uber Technologies Inc. (shortly, Uber) is the most popular provider in Europe and in the U.S.A. [1], and is available in over 66 countries and 545 cities worldwide. When a person requests a ride, the service provider forwards it to all the drivers that are close to the person's position to reduce their waiting time. In the current form of such systems, the driver's smartphone collects and sends the driver's locations (during the ride) to the service provider (SP). The SP is then responsible for computing the fare due for the ride according to the data received from the driver.

Although ride-hailing services are getting more and more popular, an important issue comes with them: drivers might be tempted to cheat when reporting their positions to over-charge the riders. This would endanger the viability of the system for the service provider and its affiliates, as well as its attractiveness to other users. To cheat about the locations sent to the service provider, the driver can tweak her smartphone to report erroneous GPS information. Such attacks have already been reported for Uber drivers in San Francisco in 2014, where they sent fake locations to pick more clients at the airport [2].

In this report, we propose an infrastructure-based approach to tackle the security issues given by dishonest drivers willing to lie about on their positions, while allowing the service provider to compute accurate approximations of the total distance of the trip without relying on the smartphone of the driver. Our approach relies on existing wireless access point (AP) networks and on street map networks. Our approach consists of two phases: first, drivers obtain secure proofs of locations during their rides, by relying on a lightweight message exchange protocol between a driver's smartphone and Wi-Fi access points encountered in the trip; second, the service provider computes secure and accurate lower bound and upper bound on the total distance covered in the ride. In particular, in this report, we show results related to the lower bound on the traveled distance and leave the computation of the upper bound as a future work.

The contribution of this report can be characterized by three main results. Firstly, we provide a very effective visualization tool that is essential in analyzing and debugging distances traveled by drivers in the real street maps. Secondly, we provide a Python implementation that can be easily extended to the upper bound computation. Finally, we evaluate our lower bound solution on a dataset of real taxi traces collected in Rome (Italy) and available on Crawdad[3]. It consists of GPS locations of 316 taxis in February 2014 with an average interval of 15 seconds between two consecutive GPS updates (for a given taxi). We also extract the actual locations of a network of deployed Wi-Fi APs operated by FON[4]. Experimental results show that our solution achieves a median accuracy of 55.21%. Even though this does not seem a good accuracy, we anticipate here that our solution is evaluated on traces which are not always comparable to real rides. We also conduct a sensitivity analysis to assess the effect of the distribution of access points on the performance of our solution.

The remainder of the report is organized as follows. We

first present the closest studies to our work, and we introduce the system and adversarial models. We then present our solution and report on its evaluation in terms of its performance, and of its security properties. After that, we present in more detail our contributions and discuss some of the challenges related to this work. Finally, we present directions for future work and conclude this report.

II. Related Work

Our report builds on top of SecureRun [5] by Anh Pham, Kévin Huguenin, Igor Bilorevic and Jean-Pierre Hubaux, which proposes a solution for privacy and cheating issues in the context of location-based apps. In particular, we rely on the mechanisms they propose to prevent users from cheating on their location updates to also prevent drivers in ride-hailing services. Specifically, we rely on the same infrastructure of access points that provide Location Proofs to users inside their coverage area, and we make use of their public key cryptography algorithms to generate such location proofs. In the following, we do not describe such algorithms, so the interested reader can refer to this paper.

Our solution is very close to the work presented in the Master's thesis of Louis Magarshack [6]. Even though we follow a very similar approach, we develop our solution de novo. In fact, not only is his code written in Julia, but, most importantly, many relevant steps related to data cleansing and applied approximations are missing. Finally, the performances of his solution are evaluated on traces of $2.4\ km$, though it is not specified how these distances are computed. Consequently, his results are not easily reproducible.

Related to our work is the study conducted by Balash et al. on electronic toll pricing [7]. In this study, they try to eliminate the delay on toll roads by collecting tolls electronically in a privacy-preserving way by using modern cryptographic primitives. The system is based on a tamper-evident black box that collects the billing information when the car transit a toll. Such data is then sent to the service provider to bill the user.

Saroiu and Wolman [8] present different scenarios where users of LBS might have an incentive to engage in location cheating. To ensure the presence of a user in a given region, they propose a protocol for providing location proofs based on beaconing of information over the Wi-Fi SSID of dedicated access points (APs). The proof of presence is based on the fact that only devices in the access point's proximity can receive these beacon signals. Our solution relies on this protocol to certify that at a specific time, the user is at a specific geographical location.

III. System Architecture

In this section, we describe the entities involved in our system: a ride-hailing service provider, a driver, a rider and a Wi-Fi AP network operator.

A. Driver

The driver is similar to a traditional taxi driver but she is not always required to hold a special license, like the ones necessary to drive a taxi. Thus, it is usually easier to join a ride-hailing service as a driver but there are still different requirements to be satisfied, and they vary between service providers and countries. The driver then just needs to login into the service provider's app and wait for ride requests.

As we describe in the following, drivers are supposed to send their locations to the service provider to compute the fare for the ride. We then consider drivers equipped with smartphones having GPS and Wi-Fi enabled. Hence, they can locate themselves and communicate with nearby Wi-Fi access points.

B. Rider

The rider is any person that wants a ride from her location towards a given destination. To request a ride, as long as she has an account with the service provider, she logs into the service provider's app and specifies the pickup location. In the most popular services, specifying the destination is optional.

C. Ride-hailing Service Provider

The principal role of the ride-hailing service provider (SP) is to match riders' requests with drivers' availabilities. A common metric used in these matchings is the time the rider has to wait for a driver to reach her pickup point. The other main role of the SP is to compute the fare of the trip as a combination of a base fare, the distance traveled during the trip and its duration. Each of these quantities is time and location dependent. Typically, fares are computed as follows:

$$Fare = base + x \times distance + y \times duration$$

The distance is measured from the location updates sent by the driver's smartphone to the SP.

D. Wi-Fi AP Network Operator

We rely on one or multiple Wi-Fi network operators controlling a set of fixed Wi-Fi APs deployed in the regions where the ride-hailing service providers operates. Each AP is aware of its geographic position and of its communication radius. For Wi-Fi communications, we assume a unit-disc model where a driver and an AP can communicate only if their distance is less than a given radius R, constant across all drivers and Wi-Fi APs.

Moreover, as in [5], we also assume that APs can compute public-key cryptographic operations to provide location proofs.

IV. Adversarial Model

We now describe the adversarial model in the described scenario.

- **Drivers.** Drivers are considered *malicious*, being able to spoof their locations to increase the computed fare.
- **Riders.** Even riders can behave maliciously by attacking the driver's smartphone to force it to send wrong location updates so that the total distance is less than the true one. This can be achieved by either spoofing or jamming the GPS signal of the driver's smartphone. Hence, riders are also assumed to be *malicious*.
- **Service providers.** Even though the revenue of SPs is based on taking a commission out of every fare, we consider them to be *honest* because it is in their interest to avoid overcharging riders; they would opt for other service providers.
- **Access point operators.** Access point operators are assumed to be *honest*. APs are then assumed to follow the protocol specified in our solution.

V. Our Solution

In this section, we present our approach to provide a driver cheat-proof lower bound on the total distance in a ride-hailing service. First, we give an overview of our solution. Then, we describe the details of the operations involved. .

A. Overview

The following is a general description of our solution. During a trip, the driver communicates with the Wi-Fi access points located along her route to obtain location proofs (LP). A location proof is a digitally signed message, delivered by an access point, that certifies that the user is, at a given time t, in a given range of an access point located at a given position (lat, lon). Finally, the driver sends all the location proofs she has collected in her trip to the service provider, which then computes a lower-bound and an upper-bound of the total distance.

By comparing the location sent by the driver's smartphone with the lower and upper bounds obtained from the location proofs, the service provider can then spot a dishonest driver. Such comparison can be performed either at the end of the trip, so that the rider directly pays an amount determined by the service provider in case the driver cheated; or afterwards and, for instance, the service provider repays the rider the difference between the amount she paid and the one it determines. To obtain tight bounds, the service provider can combine location proofs from different access points in a short interval of time into a more precise location proof by means of intersection techniques, as shown in Figure 1.

B. Location Proofs

At each sampling time (determined by a sampling algorithm), a driver collects location proofs from APs in her communication range. To do so, she periodically broadcasts location proof requests and all the access points in her range reply with digitally signed messages containing a timestamp t indicating when the request is processed by the AP and its coordinates.

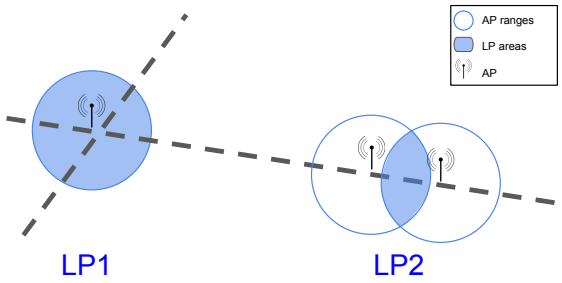

Figure 1: Location proof areas when the driver collects a LP from one AP only (LP1) and when she collects LPs from two APs in a short time interval (LP2).

In the unit-disc communication model, such proof certifies that, at time t, the driver is in a disc of radius R, centered at the coordinates of the access point.

C. Tight Bounds

We now describe how we can obtain accurate bounds between two points in a map (identified by their coordinates). To do so, we rely on street map networks. In these networks, each crossroad is associated with a node and each street is defined by two such consecutive nodes. Additionally, other nodes can be included in the streets to better define them, especially if they are not straight lines (see Figure 2).

Figure 2: Street map representation of a portion of Rome. Green triangles represent the nodes with which OpenStreetMap represents streets in this area.

By considering the underlying street map, we can compute accurate paths that the driver can take between two points, instead of relying on the flight distance between them.

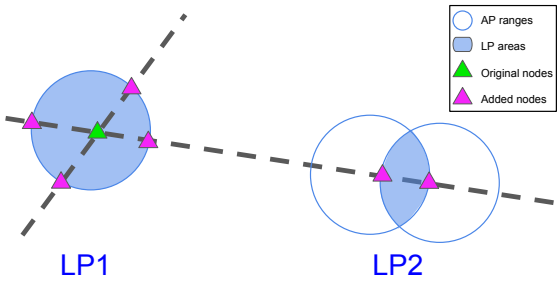

Figure 3: Additional nodes (purple) are introduced at the intersections of the location proof areas and streets.

Furthermore information such as the direction of travel (in case a street is one way only) and the maximum speed allowed in a street can improve the bounds on the total traveled distance. In fact, the first one avoids considering paths that would involve streets taken in opposite directions, while the latter provides an optimal lower bound of the time needed to go down a street, thus also removing possible paths that cannot be taken in a certain amount of time.

In addition, we add nodes on lines between two crossroads every 0.5 seconds. This gives a denser representation of streets in a map, resulting in tighter lower and upper bounds.

Finally, to make sure we can compute precise lower bounds, we add nodes at the intersections of the perimeter of each location proof area with the underlying streets, as shown in Figure 3. In fact, the driver might request a location proof while she is at the perimeter of the location proof area and not having a node there would denote her as a possible cheater while still being honest. Such nodes vary from ride to ride, depending on the location proofs sent by the driver.

D. Lower-bound Computation

Finally, we describe how the service provider can compute accurate lower bounds on the total distances. Given the street map representation described above, the service provider can represent, for instance, each city as a directed graph where vertices are nodes of the street map network (including the ones giving a dense representation) and edges represent the streets. In particular, each edge is weighted by a lower bound on the time needed to reach the next node, computed as the haversine distance[1] between the coordinates of the endpoints divided by the maximum speed allowed in that street. The haversine distance is computed as described in Algorithm 1 where R_E is the Earth's radius in meters, and the returned distance is also in meters. For each ride, the service provider receives the starting location, the destination location, the location updates and the location proofs.

- Firstly, the service provider maps the starting and destination locations to their closest vertices in its graph (based on the haversine distance).

[1]It calculates the great-circle distance between two points – that is, the shortest distance over the Earth's surface.

Algorithm 1 Haversine distance

Inputs: xLat, xLon (source coordinates),
yLat, yLon (destination coordinates)
Output: haversine distance between source and destination

1: $R_E = 6371000$
2: $\theta_1 \leftarrow xLat \times \frac{\pi}{180}$
3: $\theta_2 \leftarrow yLat \times \frac{\pi}{180}$
4: $\delta_\theta \leftarrow (yLat - xLat) \times \frac{\pi}{180}$
5: $\delta_\lambda \leftarrow (yLon - xLon) \times \frac{\pi}{180}$
6: $a \leftarrow \sin^2\left(\frac{\delta_\theta}{2}\right) + \cos(\theta_1) \times \cos(\theta_2) \times \sin^2\left(\frac{\delta_\lambda}{2}\right)$
7: $c \leftarrow 2 \times \arctan\left(\frac{\sqrt{a}}{\sqrt{1-a}}\right)$
8: $distance \leftarrow R_E \times c$
9: **return** $distance$

- Secondly, it defines *location proof areas* as previously shown in Figure 1. To do so, if multiple location proofs differ by a short time interval, the service provider considers a more precise location proof area given by the intersection of the coverage areas of the APs providing these location proofs. Otherwise, it simply considers as location proof area the unit-disc of radius R and centered at the AP signing the location proof.
- Given a location proof area, the SP adds nodes to the graph at the coordinates given by the intersections of this area with the underlying streets. The coordinates of these new nodes can be found as follows:
  - For each street in the trace, the SP defines a line based on the coordinates of the street's endpoints.
  - If the location proof area consists of only one AP, the service provider simply adds nodes at all the intersections of the circumference of the AP coverage disc with these lines.
  - If the location proof area consists of multiple APs, the service provider firstly computes all the intersections of the APs with the lines.
    - If the driver's location is not at a crossroad, the SP adds a node at the first intersection at the top-left of the driver's location, and a node at its first intersection at the bottom-right.
    - If the driver's location is at a crossroad, the SP adds a node at the closest intersection to the driver's location for each street in the intersection.

  This now gives the final graph for the considered ride.
- Finally, the service provider defines *location proof nodes* for a location proof area as the nodes inside it.

To compute the lower bound on the traveled distance, the service provider computes:

1) The shortest path between the node representing the starting location and the set of nodes defying the first[2]

[2]Recall that location proofs include timestamps.

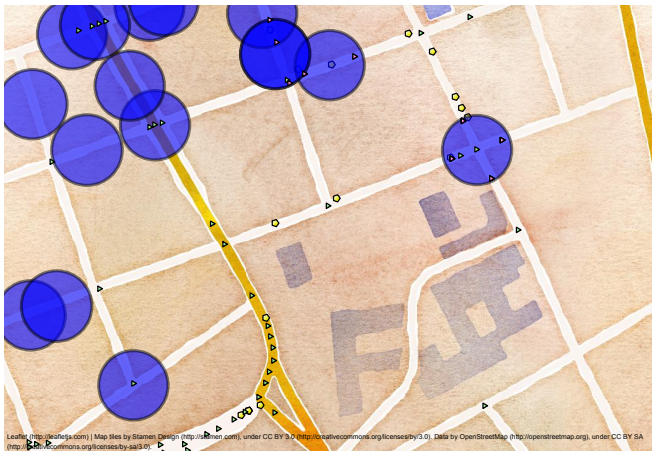

Figure 4: Trace visualization, using Stamen Watercolor tiles.

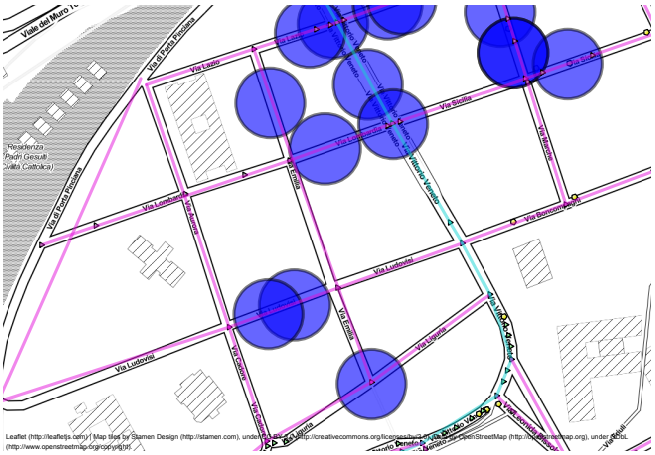

Figure 5: Trace visualization, using Stamen Toner tiles.

location proof area.

2) The shortest path between all the nodes in any two consecutive location proof areas.
3) The shortest path between the nodes belonging to the last location proof area and the node representing the final destination.

However, if any of these shortest paths takes more time than the time the driver actually spent to drive between them, we consider instead the flight distance between the nodes representing the endpoints of this path. In principle, this provides a lower accuracy but it is actually fundamental to obtain a robust computation when the GPS location reported by the driver's smartphone is not very precise (in the order of a few meters).

The lower bound is then given by the sum of the distances covered in each of these shortest paths.

This is indeed a lower bound because (i) the distances covered inside the location proof areas are not counted, (ii) the path between any two consecutive location proof areas is the shortest one between them, and (iii) the edges of the graph are weighted by the minimum time needed to travel from one node to its successor. It is, though, a tight bound because (i) we consider the underlying street map (instead of considering location proofs in straight lines), and (ii) we consider additional information associated to each street, such as its direction and the maximum allowed speed.

VI. Visualization Tools

In devising our solution, it is of primary importance to properly visualize different traces to make sure the algorithm is correct. In fact, visualization tools help us in fine-tuning the algorithm and handle corner case, obtaining a final solution that is reliable and robust.

To satisfy this need, we have developed a function, based on *folium* [9], which plots an interactive map in the format of a HTML page. An example taken from a trace is shown in Figure 4.

Here, green triangles represent the original OSM nodes (without nodes every 0.5 seconds), purple ones instead represent the new nodes that we add at the intersections of APs' areas (blue circles) with streets, yellow pentagons represent the location updates from the taxi driver, white streets are one-way only, while gold ones are two-ways. Another representation of the same map is shown in Figure 5, which uses different tiles. The only difference here is that one-way roads are in pink, while two-way roads are in cyan.

These maps are interactive in the sense that they provide additional information by clicking on the elements we add. It is possible to associate to each element any desired textual value. In particular, for each AP, our function shows the AP id in the database and its coordinates; for each taxi GPS coordinate, it shows its ordinal value (0 for the first one in the trace, etc.) and its coordinates; and for each OSM node, it shows its id in the database and its coordinates.

VII. Performance Evaluation

To evaluate the accuracy of the proposed solution, we use traces of taxi drivers in Rome (assuming each of them represents a driver of a ride-hailing service) and the network of access points of the FON operator. The scenario we analyze has drivers equipped with Wi-Fi and GPS enabled smartphones, to collect location proofs, which are then sent along with location updates to the service provider.

A. Datasets

To assess the performance of our solution, we use a subset of data that has been previously collected at LCA 1, as described in the following. In particular, we remove, from the full data sets, elements that fall outside a region approximating the city of Rome (Italy). This region is shown

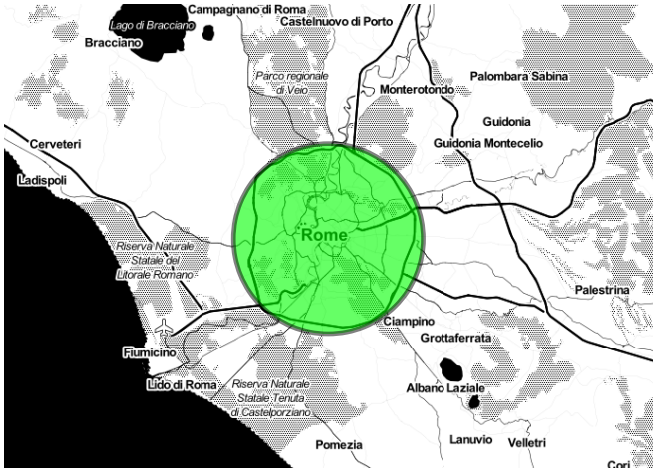

Figure 6: Approximation of the city of Rome as the circle having radius of $11.5\ km$ and centered at $(41.89, 12.49)$.

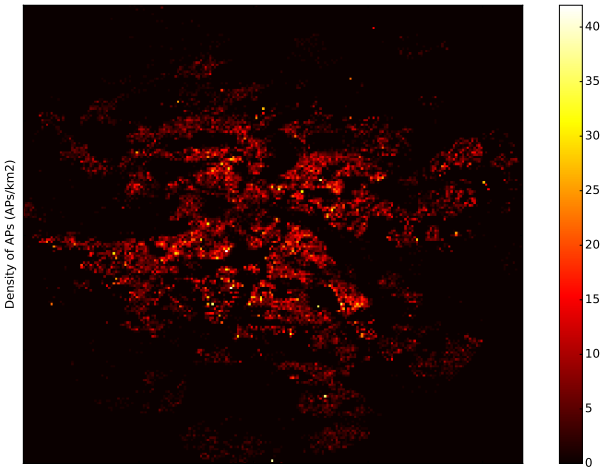

Figure 7: Heat-map of the density of FON access points in Rome.

in Figure 6 and is defined as the circle having radius of $11.5\ km$ and centered at $(41.89, 12.49)$.

1) Wi-Fi access points: We have available the coordinates of the Wi-Fi access points from the FON operator in 2016. FON is a large community network with more than 20 million hotspots worldwide, most of them located in Europe. FON establishes strategic partnerships with local ISPs so that routers of the ISPs act as FON hotspots. Since ISPs control the routers through firmware updates, they could easily implement our solution.
By applying the geographic filter for Rome specified above, we obtain $44,243$ APs in Rome. Their density can be observed in the heat-map in Figure 7.

2) Drivers traces: As mentioned in the Introduction, we use a data set of taxi traces in Rome available on Crawdad. They refer to 316 taxis and consist of GPS coordinates relative to their positions in February 2014.
This data just consists of GPS coordinates, and so we firstly split them into traces. By looking at when the position of a taxi driver does not change in consecutive GPS updates, it could be possible to identify when a client is dropped off. However, the time to drop a person off has usually the same duration as the time a taxi driver waits at traffic lights.
Thus, we decide to split these GPS updates according to the distribution of average trip distances by yellow taxis in New York City in 2014 [10]. This is a skewed normal distribution, which can be approximated by the one shown in Figure 8. Distances here are computed as the sum of the flight distances between any two consecutive GPS updates. Instead of just applying this distribution to our data set, we also check that no two consecutive GPS points have timestamps that differ by more than 1 hour. In fact, it is very reasonable to assume that the same trip does not have two consecutive GPS updates in that interval. In case we notice such gap, we split the trace at that point and we keep it only if its total distance is at least as long as the minimum average trip ($0.75\ km$ in our case).

Lastly, since some GPS coordinates in the dataset also fall outside Rome, we remove traces containing any point falling outside the Rome region defined above.

The resulting number of traces is finally equal to $176,603$.

3) Street map network: To represent the underlying street map, we rely on OpenStreetMap [11]. OpenStreetMap is a collaborative project to build a free map of the world. Map data is collected from scratch by volunteers using tools such as a GPS unit, a notebook, or a digital camera. The data is then entered into the OpenStreetMap database.
OpenStreetMap uses a topological data structure, with four core elements:

- *Nodes* are points with a geographic position, stored as WGS 84 [12] coordinates (the same used by the GPS).
- *Ways* are ordered lists of nodes, representing a polyline, or possibly a polygon if they form a closed loop. They are used both for representing linear features such as streets and rivers, and areas, like forests and parks.
- *Relations* are ordered lists of nodes, ways and relations. Relations are used for representing the relationship of existing nodes and ways. Examples include turn restrictions on roads and areas with holes.
- *Tags* are key-value pairs. They are used to store metadata about the map objects (such as their type, their name and their physical properties).

At LCA 1, we have available a PostgreSQL [13] database containing OSM data structures relative to streets only. However, they do not exclusively refer to Rome and so we filter out data outside our Rome region. In particular, we only keep ways that have at least one node inside the region.

Moreover, when building the graph for Rome, we notice that it is not fully connected. This is because these maps

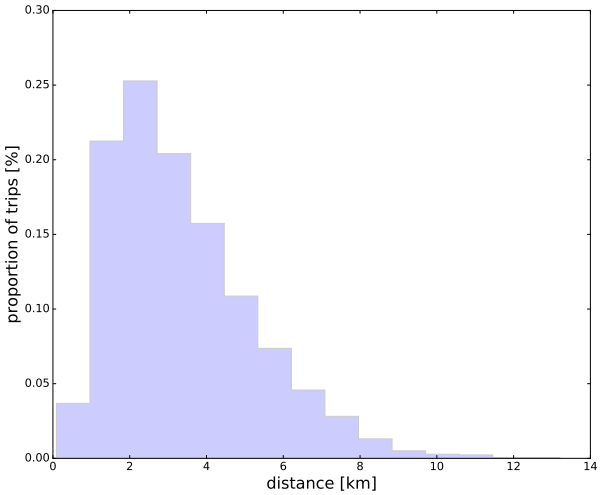

Figure 8: Distance distribution of the average trip distances by yellow taxis in NYC in 2014.

are based on data collected from volunteers, who can forget to associate a node in an intersection to all its streets; thus possibly creating a network partition. Our solution then uses the giant component of the graph for Rome, containing $793,433$ nodes instead of the original $811,306$ ones.

B. Methodology

We use Jupyter notebooks for developing our solution. Such documents contain both computer code (Python in our case) and rich text elements (paragraphs, equations, figures, links, etc...). They are both human-readable, containing the analysis description and the results (figures, tables, etc..), as well as executable, which can be run to perform data analysis. We have written several notebooks, each tackling a different aspect of our project. To name a few, we have notebooks showing step by step how we build the graph, how we split the GPS updates into traces, how we add nodes every 0.5 seconds to the original graph, and many more.

The notebooks are written in Python and we also have self-contained Python scripts that can be launched to obtain the final results we are interested in and useful intermediate structures. Moreover, we have written two libraries containing all the functions that can be helpful in addressing lower bound computations in our system. Specifically, we have functions that interact with the databases available at LCA 1. For instance, it is possible to retrieve all the OSM nodes, APs or taxi GPS updates lying inside a box of specified coordinates; to retrieve all the streets and their metadata containing a specific set of OSM nodes, or to retrieve the APs which have a taxi GPS coordinate in their unit-disc coverage area. Given a taxi trace, there are methods to compute densities of APs inside that trace, both in numbers of APs/km^2 and APs/km. We have functions that define the location proof areas and the nodes lying inside each of them; another function builds the graph, and we can also modify it by adding additional nodes at their correct street positions with respect to the already existing ones. Data cleaning and handling of missing values is handled by a single method, as it is the computation of the shortest path between consecutive location proofs.

Several of these functions also come with a dual representation which is database-free. In fact, we also store the filtered data used by our final solution in the form of `pickle` files. These are compact binary encodings of Python data structures. By using the database-free representation of such functions and these binary files, we can run our code in any machine, without locally installing the full database.

The libraries alone contain more than $1,700$ lines of code (LOC), while the script iterating through all the taxi traces computing their accuracies is around 250 LOC.

Finally, we expect our implementation to be easily extended to the upper bound computation in this scenario.

Hence, for each taxi, we split its collected GPS updates according to the distribution in Figure 8 and simulate the execution of our solution. For each resulting trace, we compute an approximation of the real traveled distance, its tight lower bound and densities of APs. We consider a 25-meter radius unit-disc communication model for the FON access points. The performance of our solution is measured in terms of accuracy of the lower bound with respect to the estimated real traveled distance. The accuracy is then a value between 0 and 100%.

C. Results

To assess the performance of our solution on $1,000$ randomly selected traces from the set of splitted traces. The cumulative distribution function of the accuracy for all the traces is shown in Figure 9 (left). To analyze the sensitivity of our solution with the density of APs, we plot the experimental density functions of the accuracy for three different ranges of number of APs per km. By looking at Figure 9 (right), it is clear that the performance is much better for traces having high density of APs. In fact, even though the number of traces having more than 20 APs per km are few, their accuracy is of at least 79%. Moreover, by comparing the two plots, it is possible to see that most of the traces contain less than 6 APs/km. Having such small density of access points is one of the main reasons that give low accuracy on the lower bound computation.

D. Poorly performing scenarios

By means of our visualization tools, we have manually investigated some cases where the accuracy of the lower bound is below 50%.

Figure 10 shows the final part of a trace giving around 40% accuracy. It is clear from the map why it is the case. In fact, the driver reaches a train station and spends most

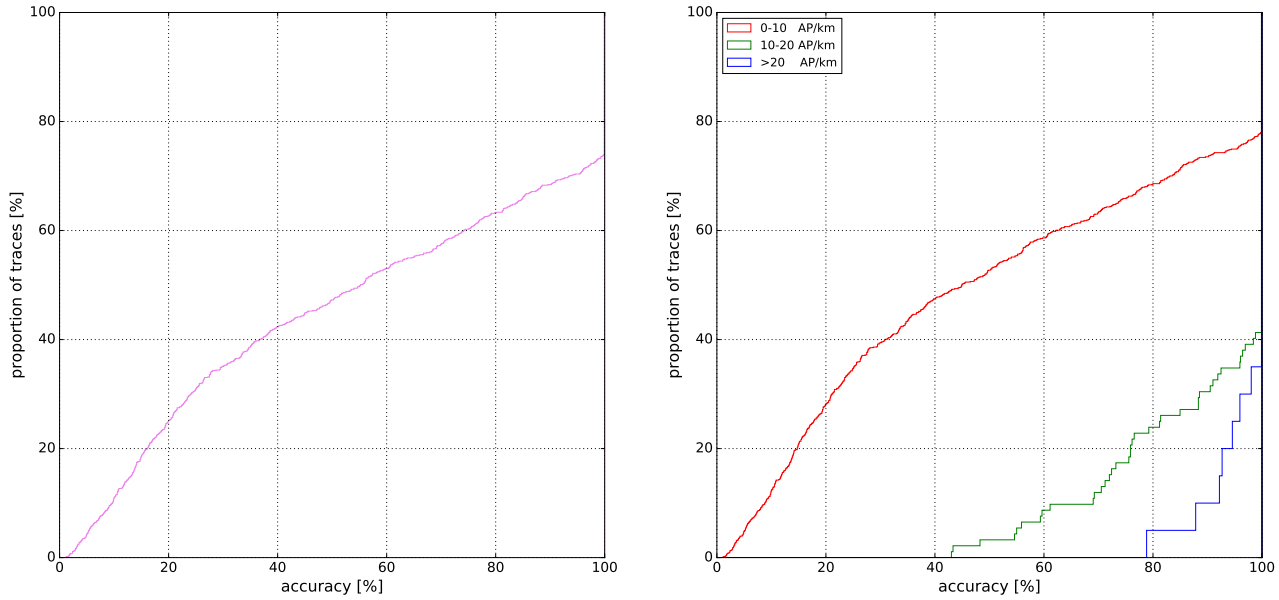

Figure 9: Cumulative distribution functions of the accuracy for all the traces (left), and of the traces grouped by their density of access points per km (right).

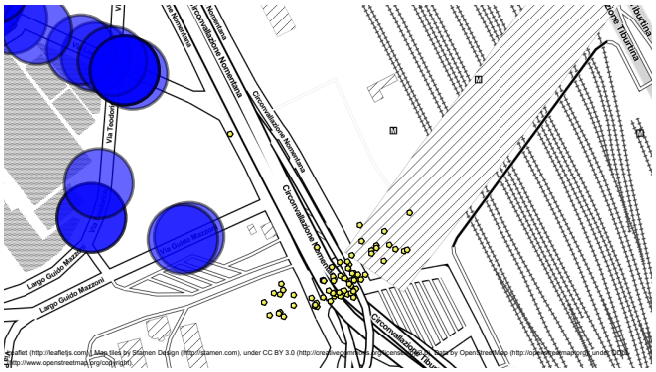

Figure 10: Visualization of a trace not linked to a ride.

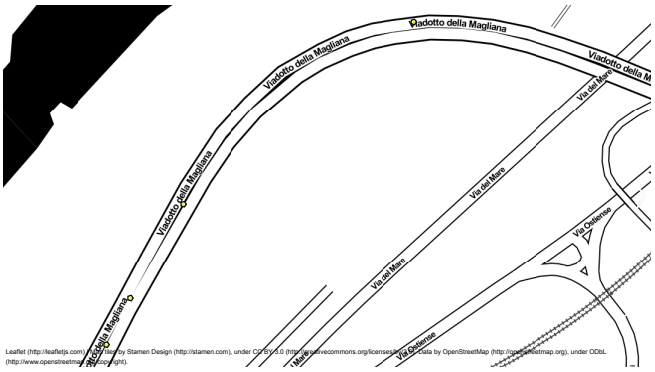

Figure 11: Visualization of a trace having GPS updates mapped into a different street.

of the time driving around the station. Here, there is no access point and thus all those GPS updates (the majority in that trace) would be discarded and the approximation would consist of the path between the last available location proof and the last GPS update in the trace.

Hence, this gives us a reasonable explanation of why our results are not very accurate: given that we have GPS updates from taxi drivers, many of them are not linked to actual rides but to drivers moving around the city, waiting for a client. We can then say that many of the traces we have created by splitting GPS updates as in Section VII-A2 are noisy: they do not correspond to real rides and thus are not relevant in assessing the performance of our solution.

By looking at another trace, we notice that the GPS units used by the taxi drivers may not be extremely precise and this could cause very poor approximations. The trace we now discuss is shown below, in Figure 11. In this trace, several GPS updates are located across two one-way streets. What happens here is that some of them are mapped into the wrong street. Thus, if the driver is going towards north east along the road on the left and the following update is mapped onto the road on the right, the algorithm has to travel all the road until it is possible to turn and go back to that point. Now, if the following point is in the road on the left, the algorithm has to do the same thing. However, if it is in the same road on the right, drivers cannot simply go in that direction since the street is one-way only and so the algorithm has to find a way to turn and reach the next location. And so on.

It is clear then how the lower bound distance would explode in such situations. This is why we have introduced the flight distance in our algorithm. That is, if the time needed to go between two points by taking the shortest path in the street map network is larger than the time the driver actually needed, then we do not have a lower bound anymore and so we compute instead the flight distance between these two points. Such change improved our accuracy by 20% on the traces of one taxi that we tested with both versions.

VIII. Challenges

During the development of our solution, we have faced several challenges.

One of the most important is related to discovering that the graph for the city of Rome is not fully connected. This means that is not possible to go from any given point to any other given point in the same city. By analyzing one of the traces, we detect that some nodes at street intersections are not mapped to all the streets, hence causing some partition of the network if the unconnected streets are one-way only. Our solution then just considers the giant component of the graph of Rome, which contains 97.8% of the original OSM nodes.

Working with coordinates and geometrical structures is intrinsically difficult from a programming point of view. In fact, many concepts that are intuitive and easy to visualize on a blackboard are actually all but trivial to code. For instance, one of the most challenging part of our project is to add artificial nodes at the intersections between a location proof area and the underlying streets. In fact, the concept of area and street are not defined in the binary world. What we have, instead, is just a collection of points, expressed in terms of latitudes and longitudes. So, we firstly need to define what a location proof area is, especially when multiple APs have their coverage areas intersecting. Then, we need to define the concept of street and compute the intersection of each possible street with the perimeter defined by location proof area. More details about this algorithm are given in Section V-C describing how we achieve tight bounds. The function for just finding such intersections requires 150 LOC.

IX. Conclusion and Future Work

Ride-hailing services have become increasingly popular over the last few years. In their current form, such systems rely on the drivers mobile devices to collect and to report the locations during the trip. This provides no security guarantee against cheaters. In this report, we propose a solution for providing secure rides. Our solution relies on the existing wireless access point networks (at the cost of only a software upgrade, hence alleviating the need for deploying ad-hoc infrastructures), and it provides protection for both riders and service providers. Our experimental evaluation, conducted using real datasets of deployed wireless access points and GPS updates from taxi drivers, shows that our solution does not achieve high accuracy. However, several traces do not refer to actual rides, thus making our performance questionable. From a practical perspective, we envision our scheme to be of possible interest for strategic partnerships between ride-hailing service providers and access point network operators.

As part of future work, we plan to (i) assess the performance of our solution on the entire dataset of taxi traces, (ii) further improve the accuracy of our solution by optimizing lower bound distances by computing the longest possible shortest path between two points, (iii) compute the upper bound of the traveled distance, and (iv) evaluate our solution on a dataset of traces corresponding to real rides.

References

[1] O. Zaleski and A. Tartar. Uber is now the most popular taxi app in 108 countries, data show. [Online]. Available: https://www.bloomberg.com/news/articles/2016-08-23/uber-is-the-most-popular-ride-hailing-app-in-108-countries

[2] W. Phaneuf. Uber drivers 'cheating' the app to gain fares at sfo. [Online]. Available: http://sfist.com/2014/07/30/uber_still_illegally_working_sfo_al.php

[3] L. Bracciale, M. Bonola, P. Loreti, G. Bianchi, R. Amici, and A. Rabuffi. Dataset of mobility traces of taxi cabs in rome, italy. [Online]. Available: http://crawdad.org/roma/taxi/20140717/

[4] Fon Wireless Ltd. Fon. [Online]. Available: https://fon.com/

[5] A. Pham, K. Huguenin, I. Bilogrevic, and J.-P. Hubaux, "Secure and private proofs for location-based activity summaries in urban areas," *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 751–762, 2014.

[6] L. Magarshack, "Secureride: A cheat-proof ride-hailing service," *Master's thesis*, 2016.

[7] J. Balasch, A. Rial, C. Troncoso, and C. Geuens, "Pretp: Privacy-preserving electronic toll pricing," *USENIX Security*, 2010.

[8] S. Saroiu and A. Wolman, "Enabling new mobile applications with location proofs," *Proceedings of the 10th ACM workshop on Mobile Computing Systems and Applications*, 2009.

[9] Python-visualization. Folium. [Online]. Available: https://github.com/python-visualization/folium

[10] N. T. . L. Commission. 2014 taxicab fact book. [Online]. Available: http://www.nyc.gov/html/tlc/downloads/pdf/2014_taxicab_fact_book.pdf

[11] O. Foundation. Openstreetmap. [Online]. Available: https://www.openstreetmap.org

[12] Wikipedia. World geodetic system. [Online]. Available: https://en.wikipedia.org/wiki/World_Geodetic_System

[13] ——. Postgresql. [Online]. Available: https://en.wikipedia.org/wiki/PostgreSQL