Taxi Application over Peer-to-Peer Mobile Networks

Ştefania Bucur[1], Radu-Ioan Ciobanu[1(✉)], Gabriela Răducan[2], and Ciprian Dobre[1]

[1] Faculty of Automatic Control and Computers, National University of Science and Technology POLITEHNICA Bucharest, Bucharest, Romania
stefania.bucur0406@stud.acs.pub.ro, {radu.ciobanu,ciprian.dobre}@upb.ro
[2] Faculty of Law, Titu Maiorescu University, Bucharest, Romania
gabriela.raducan@rlga.ro

Abstract. Nowadays, the use of mobile data is becoming more and more difficult because there are many people who own a smartphone, and thus the signal is fragmented. Therefore, in quite urgent situations, if we need Internet access, there is a possibility that we cannot access it because there are many devices already connected. A service that would benefit from a new solution that does not require Internet access is the taxi service. In situations such as leaving from a concert or visiting a foreign country, a taxi application over the peer-to-peer mobile networks would be a very good idea in meeting the mobility needs. To develop such an application, we need to understand the concept of opportunistic networks and other features that come with them such as routing, trusting mechanisms, etc. This paper presents a possible solution of a peer-to-peer mobile taxi application, the motivation behind the implementation of such an application, as well as all the technical details needed to understand and develop the concept. Finally, we also turn our attention to some potential legal implications of the proposed solution.

Keywords: Mobile networks · Opportunistic networks · Taxi

1 Introduction

This paper presents the attempt to find a solution in the context of a peer-to-peer network of a taxi application that offers the same facilities as an application that requires Internet access. A mobile application that would come with an implementation over Drop Computing [3] would be very helpful when it comes to finding a taxi in certain situations. Through opportunistic device-to-device communication, the user would connect with the driver, by pressing a search button, and so the request would be sent from node to node to the nearest available taxi.

The biggest challenge in implementing such an application will be ensuring that the application will reach a reliable driver. However, this can also happen

T. Zlateva and G. Tuparov (Eds.): CSECS 2024, LNICST 609, pp. 3–28, 2025.
https://doi.org/10.1007/978-3-031-84312-9_1

with a normal taxi application, so existing strategies such as giving rating stars and feedback can be a reliable solution.

From a technical standpoint, this paper presents two approaches of showing the concept of peer-to-peer mobile application. The first part is the consolidation of routing algorithms in an existing framework, simulating interactions between devices in this context. The second part is a proof of concept of a taxi application over peer-to-peer mobile networks, to demonstrate the real utility, giving the current context.

There are situations in which a user cannot or prefers not to use their mobile data, or they have no access to a Wi-Fi access point, but they need certain services that could only be provided by a mobile application.

Related to this paper topic, here are several scenarios in real life where individuals need a taxi, but they cannot order one via an application that requires Internet access. For example, when people go in vacation and they do not have the roaming option enabled or maybe they do not want to pay extra costs when accessing the Internet. So, they arrive at the airport late at night and have to reach the accommodation, which is quite far from where they are, but there is no taxi around. This situation is a good example of the need for an application that does not have to have Internet access to order a taxi.

Another example would be the participation in an event with many people (of the order of tens of thousands), which ends late at night and there is not public transportation available. Of course, at the end of the event, most people will want to order a taxi to get home. Because there are many people in the same area accessing a taxi application that needs Internet access, the connection will be fragmented and the service will not be provided in a timely manner. This is the moment when opportunistic networks [5,12] can be useful, as a subset of mobile ad hoc networks (MANETs) and vehicular networks. Through the transmission of information from node to node, messages can reach the other end of the request (i.e., the taxi driver), who will be able to accept the order and come to the client to offer his service. Also, all the people waiting to go home have the same purpose, so the information can be propagated with the same context.

The main purpose of this paper is to come up with a solution to implement a taxi application over peer-to-peer mobile networks. Although there are existing taxi applications, they require Internet access. The working flow will be similar to a normal taxi application. The user will press the search button and their request will be sent to the nearest taxi available. After the request reaches the driver, he will be able to offer the mobility service.

There are several important issues that need to be solved in this topic. Once the request is sent, it will be forwarded from node to node from source to destination. First of all, we need to take into consideration the context propagation between devices. Each node must keep the information required for the respective request so that the driver can see all the details regarding the future client. In the next sections, we will present the main idea underlying this paper, namely opportunistic networks.

Another goal that needs to be achieved is the trust of devices. It is very important to be sure that at the destination node will be who it should be, and there must be a way in which we must assure the users that using the application, they will receive a high-quality and safe service. Furthermore, an important goal is to implement a taxi application as a proof of concept, that will work based on the opportunistic networks concept. A very important aspect is the user experience. The application must be easy to use, suggestive and the process behind it must be hidden.

The rest of the paper is structured as follows. Section 2 presents a brief overview of opportunistic networks and what are their advantages regarding the paper's subject. Also, we present various methods that will help fulfill our goals, such as reputation mechanisms or data forwarding algorithms to avoid malicious nodes, respectively to transmit the context message from the source to the destination. In Sect. 3, we present the method of this paper, representing the use cases, architecture and employed technologies. We also present a deep dive in MobEmu [1] and in the implemented taxi application. Section 4 contains the evaluation of the simulations and implementations, followed by interpreting the results. In Sect. 5, we present a brief legal analysis of aspects related to personal data protection and applicable jurisdiction, depending on the implications that the proposed application in this study might generate. Finally, Sect. 6 presents the conclusions drawn after running the simulations and implementing the application, as well as ideas for future work.

2 State of the Art

2.1 Opportunistic Networks

In this section, some details about opportunistic networks will be presented. Opportunistic networks [7,8] are a considerable evolution of MANETs. The most important feature of these networks is that the nodes in the network don't need to know any information about the topology in which they are located, the nodes being able to communicate with each other even if a route is not created. Also, between the source and the destination, there may be several nodes such as next hop, which opportunistically can retrieve information and forward the message. This can lead to a delay in retrieving the message, as the messages are buffered in the network, waiting to be forwarded to the destination. In this context, a taxi application can support this delay.

Opportunistic networks (ONs) transmit messages through a method called store-carry-forward [9,10]. This means that the nodes that are in the vicinity of the node to be transmitted must add over the stack protocol a layer, called a bundle layer. Through this bundle layer, the node can store, carry and forward the entire bundles or fragments of them with its neighbors from the same area, but also between different regions.

The nodes are dynamic within the network, they transmit the information they have to the other nodes. Therefore, the concept of altruism [4] is introduced, which means that the nodes not only send their own information, but also

exchange information with the other nodes so that the message transmitted from the source reaches the destination in the best time possible.

Opportunistic networks are based on Delay Tolerant Networks (DTN) [7], but there are some important differences. In short, the DTN architecture is represented by a network with Internets that are independent and each of them is characterized by an Internet-like connectivity within it. They only have occasional communication opportunities, sometimes scheduled over time. There is a DTN gateway system that is responsible for connecting the Internets. Therefore, the points of possible disconnections are known and isolated at the gateways.

It could say that opportunistic networks are more general and include DTNs. The difference would be that for opportunistic networks it is not mandatory to know the network topology, whereas at DTN it is necessary. As for routes, there are some differences between DTN and ON. The unavailability of the link is interpreted at DTN as another component of the link cost, routes in DTNs being typically determined via legacy-Internet techniques. On the other hand, opportunistic networks use a lot of knowledge saved in the nodes. When a node receives a message for the destination, it looks at the information and it determines among its current neighbors which node is the best for a next-hop. Routes are calculated for each data in each next-hop. However, there is the possibility that there are no nodes in the vicinity or nodes that are not meeting the forwarding requirements. Then the node keeps the message and waits for other connection opportunities.

2.2 Trust and Reputation in Opportunistic Networks

As was presented previously, one of the main goals in using opportunistic networks is to provide trust in devices. In an opportunistic network where the topology is not known, there can be many malicious nodes, which could spam messages, change information or drop the messages of interests to other nodes. Ciobanu et al. [2] proposed an opportunistic trust and reputation mechanism entitled SAROS, which could detect and prevent malicious nodes[1]. SAROS is a framework for opportunistic networks and it uses context information (interests, location, social connections) in order to disseminate data to interested nodes. Given that the nodes communicate only if they are near to each other, handling trust and reputation is a very complex subject. Because there is no global authority that can say if a node is malicious or not, and nodes gather information from other nodes, there is no guarantee that the information received is reliable. If more malicious nodes collaborate and send false information, a node can be tricked and believe the information received is good. A good advantage of opportunistic networks is device mobility. It is much harder for a malicious node to follow a good node in order to give it false information continuously. In addition, if the mobile devices are part of a social group, it is from the beginning an implicit trust and they can detect the malicious node. Therefore, SAROS

[1] We also refer the reader to this paper for an in-depth analysis of trust and reputation mechanisms in the context of mobile networks.

is a solution that takes advantage of the opportunistic network characteristics, detecting false messages and pre-establishing trust using social information.

2.3 Routing Without Infrastructure

Routing without infrastructure [7] is designed for flat ad hoc networks and it can be classified into dissemination-based routing and context-based routing, based on the flooding in the network. For this context, routing without infrastructure may be a good solution.

Dissemination-Based Routing. The most important feature of this type of routing is that when a message has to be sent from the source to the destination, it is disseminated throughout the network. This technique is applied when the networks are very mobile, and the probability of two nodes meeting is very high. A disadvantage is that it delays the transmission of messages and consumes a lot of resources. Because there are many transmissions, there is the possibility of high contention which leads to network congestion. One solution to increase network capacity would be to limit the spread of the message by reducing the number of relay hops to a maximum number. It can also limit the number of copies of messages present at the same time in the network. For example, the Epidemic Routing protocol [11] associates the transmission of messages with the spread of viruses. A node is infected when it creates its message or is received from another node for forwarding. The message is stored in a local buffer. Nodes that are not yet infected are susceptible and they become infected when they come in contact with another infected node. The node heals when the message reaches its destination, at which point the node becomes immune if it receives the same message from other nodes. This whole process is similar to the dissemination process because each message has a hop count limit, which means the number of hop the node can do. When hop limit is 1, the message can be transmitted directly to the destination node.

Context-Based Routing. Context-based routing focuses on using context to figure out which are next-hops to the destination. For example, information such as the user's address can be valuable in deciding the next hop for a message. A great advantage of context-based routing is that it can reduce duplicate messages, but can lead to a delay in sending messages because errors may occur by making wrong assumptions for the next hop. The cost is higher because the nodes must also maintain a state to track the context values associated with all the other nodes in the network. A good example is Context-Aware Routing (CAR) protocol [6], where each node calculates its own probability of delivery to each destination it knows. The probabilities are changed at a time interval so that each node can generate the best carrier for each destination node. Based on the context of the nodes, the best carriers are generated, using attributes such as the residual battery level, the rate of change of connectivity, the degree of mobility etc. When a carrier receives a message to forward it, it will store it in

a local buffer and send it to the destination node when it gets close to it or give it to a nearby node that is more likely to reach its destination faster. Compared to epidemic routing, CAR is more scalable because the overhead stays constant regardless of node buffer size.

3 Proposed Method

3.1 Use Cases

There are multiple use cases that can be taken into consideration. One first example is when people try to leave from a concert and there is no public transport at the late hour in the day. So, they try to call a cab, but the network is down because there are so many connections from other people. A taxi application over peer-to-peer network would help find a taxi in a nearby area without using the Internet.

A second use-case would be when people visit a foreign country, but they don't have roaming on their phones. They don't know exactly the address where they need to arrive. It is a very complicated situation, because, nowadays, people use Internet for everything, but if they are not able to access it, they could use the taxi application over peer-to-peer network and find the nearest taxi that would bring them to the accommodation and to Wi-Fi.

3.2 Architecture

First of all, Fig. 1 represents a minimal architecture where 2 types of nodes are defined: the node that is the mobile phone of a person and the node that is the taxi driver's mobile phone in the car and its speed is higher than the speed of a man walking. The person node can be of two types: one person that wants a taxi and one normal person. The taxi node has several characteristics that distinguish it from a normal node such as "isBusy" and trust variables. Also, all the other nodes (Taxi/Person) that are not involved in the call taxi flow will be able to behave as a simple node that can transmit the message. The resource through which a person will communicate with a taxi is a smartphone application that has built-in Bluetooth. Both the taxi driver and the customer will have the application installed and each smartphone will represent a node in the opportunistic network.

3.3 Integrated Technologies

3.3.1 MobEmu

Simulating the behavior of mobile devices is essential before implementing an application, especially in scenarios like concerts where hundreds of devices may interact. MobEmu [1], a Java-based framework, is used for this purpose. It can simulate traces and run various routing algorithms while considering multiple parameters, such as the simulation area, node numbers, communities, node speed, runtime, memory size, and battery levels. During simulations, MobEmu

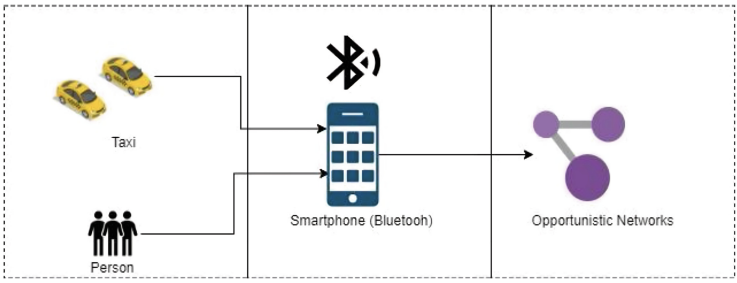

Fig. 1. Opportunistic taxi application architecture.

evaluates node interactions and applies routing or dissemination algorithms, printing metrics like hop count and exchanged messages at the end. Key components of MobEmu include:

- **Trace**: Implements and tests algorithms in real-life network scenarios using data from experiments. It tracks node interactions and contact duration, considering different types of connections (e.g., Bluetooth, Wi-Fi).
- **Node**: Represents mobile devices with unique IDs. Nodes handle two types of messages: received messages (which may be forwarded) and self-generated messages (stored in separate memories). The store-carry-and-forward paradigm ensures message delivery across nodes.
- **Context**: Holds node IDs and topics of interest, aiding in message dissemination based on node interests.
- **Exchange Stats and History**: Record data exchange times and message history during node contacts.
- **k-Clique Algorithm**: Identifies node communities and calculates their centrality within and across communities.
- **Battery and Altruism**: Track battery levels and altruism, influencing whether nodes forward messages. Nodes may refrain from forwarding due to low battery or lack of interest, impacting message delivery efficiency.

The framework supports several routing algorithms, with Epidemic routing being studied and simulated in this specific context.

3.4 Taxi Android Application

In this section, we present two components used for developing the proof-of-concept application: Google Nearby and Mapbox.

Google Nearby. The most important Google feature employed while building the application is Google Nearby[2], that demonstrates the peer-to-peer connection between two devices, without using Internet. This service provides low-latency, high-bandwidth and exchanging data securely. It provides an abstraction

[2] https://developers.google.com/nearby.

layer over Bluetooth, BLE and Wi-Fi hotspots and the developer doesn't need to know all the implementation details for them. The developer only needs to call the APIs and Nearby Connections takes care of versions, hardware and what protocol should use. The only focus is providing the best user experience and make the life of developers easier. The API splits into two phases: pre-connection and post-connection. There are two types of entities: Advertisers and Discoverers. As the name implies, Advertisers "advertise themselves" and Discoverers will then discover the Advertisers that are in the close range, in the pre-connection phase. After a connection is established between Advertiser and Discoverer, they can exchange data. This step happens in the post-connection phase. After that, it is the developer's decision based on APIs, what should happen after a connection is established, what type of data should be exchanged and so on. Google Nearby is very powerful and very useful, especially for the current context.

Mapbox. In order to have a taxi application, it is a must to have a map showing the user's location and to have access to other addresses and locations. Mapbox[3] provides such maps with many other features, for example, routing, navigation and styling the map. Also, the map can be configured with attributes like camera position, current location and setting the pins anywhere. The main purpose of this paper is integrating Mapbox with Nearby Connections, by exchanging positions data between devices and displaying them on the map.

3.5 MobEmu Simulations

Context Details. In the MobEmu simulation using the HCMM model, nodes and movements are generated based on human interactions. Existing traces were unsuitable due to their lack of modifiable node behavior. The HCMM model defines several parameters, including 20 nodes interacting over one hour, with normal node speeds ranging from 0.25 to 1 m/s and higher speeds for taxi nodes. The simulation area is 200×200 meters, divided into 20×20 meter cells, with nodes forming communities and four traveler nodes moving between them, increasing interaction chances. The transmission range is 10 m, suitable for Bluetooth[4]. The simulation tracks message transmission, highlighting relevant information for timely interception by the destination. Metrics are displayed at the end, and an additional simulation showed halved message exchanges with a specific context for even nodes. The goal is to ensure taxis intercept messages timely, considering factors like intermediate nodes and transmission range. Altruism, influenced by battery levels, affects message forwarding; nodes with less than 30% battery drop non-intended messages. Taxis, distinct from regular nodes, include extra parameters like trust and busy status, with trust affecting reliability perceptions. The trust field adjusts based on taxi performance, while the busy field indicates if a taxi is occupied, allowing message forwarding like a regular node.

[3] https://docs.mapbox.com/help/getting-started/.

[4] We considered Bluetooth over Wi-Fi because it is cheaper in terms of energy consumption, but we are also planning to investigate Wi-Fi in the future.

Epidemic Routing Improvements The goal of this paper is to find a routing algorithm in the context of a taxi application, in which the response to the customer must be fast, the message must reach the nearest reliable taxi driver and communication must be easy. Initially, the project is based on a simple algorithm such as Epidemic Routing [11]. The simulation in MobEmu is based on epidemic routing, which will be improved step by step.

At a conceptual level, there are some key points where the algorithm will have to make certain decisions depending on the context of current research. When a node meets another node, it is first checked what type of node it is. If the node has a taxi type, the busy parameter is verified for the first time in order to know from the beginning if the ride can be accepted. If the busy parameter is true, the taxi node will behave like a normal node and will forward the message, if the battery level allows it. If the busy parameter is false, it means that a taxi has been found that could accept the order. The taxi node sees that the message that was received is from a future customer and generates an acceptance message that it sends back. When the message reaches the client, there will be a time interval between which the taxi and the person who requested a taxi must meet, for example 5 min.

When the client gets in the taxi, several parameters change: the busy parameter becomes true, which means that the taxi is in a course, the number of trips parameter increases by 1 and the trust can also increase by depending on the client. The battery level must be verified when the node is waiting for a response from a taxi, which means that there is a person waiting for a car. If a message comes from a node, but is not intended for it and the battery level is below 30%, the node will drop the received message and will not use the remaining power to forward the message. The context can be useful when we can find out if a node went in a taxi area, so there is a chance that a taxi will be available and take the ride. Thus, based on the context, messages can be sent faster and with a precise purpose.

However, as specific context components are defined, such as node types, the algorithm becomes more complex and more precise in transmitting messages. There are many variables that we must take into account: the level of trust, how we are sure that our messages are not lost, how nodes react when the mobile device does not have enough power to transmit messages to other nodes. We also have to test the algorithms on different situations, to see how the algorithm reacts when there are many nodes or there are very few travelers, we must take into account the number of communities in the perimeter. Because there are two types of nodes with different speeds, the algorithm will have to handle each situation differently.

The metrics that will be considered when implementing a taxi application over peer-to-peer networks are very much related to the messages transmitted between nodes. First of all, a metric represents how many messages arrived at the destination and how many were lost. What is also interesting to analyze is the way in which nodes with different speeds communicate and how many times they manage to meet until the message reaches its destination. These higher speed

nodes are taxis that are busy, but can send messages to other nodes. Therefore, the hit rate will also be analyzed.

Because in opportunistic networks nothing is certain, a fixed topology does not exist, the nodes are not always the same, the messages can be lost very easily, seeing the problem from another more formal perspective can influence making some decisions and can help in the further development of the algorithm. The behavior of the nodes within the network, the way they interpret the information and how long it will take for the messages to reach the source to the destination will be analyzed.

There are many conditions to be taken into account and many parameters influence the final implementation of the routing algorithm. The analysis of the level of trusting will draw some real limits in which we have to think if it is safe to trust some answers coming from any mobile device. If this level of reliability is satisfying, this paper can lead to a future real implementation of the application.

Proposed Solution. There are 2 entities that are involved in the routing algorithm. First, there is a Person entity that can be a client (a person that wants a taxi) and a simple person that will participate in the simulation and will be the carrier. These two types of persons are differentiated by a parameter called "wantsTaxi" and also by the message they generate. The person who wants a taxi will generate a message with "I want taxi" and this message will be forwarded to the other nodes until a taxi will receive it. The taxi entity has also some special parameters such as "isBusy" and "trustLevel" that will be discussed later.

The first step in tuning the algorithm is to introduce altruism analysis, which is very important in terms of transmitting messages between nodes. It transmits messages coming from other nodes if a node is altruistic and thus facilitates the process of transmission from source to destination. But the degree of altruism will be influenced according to the battery the individual has in the current context. If, for example, the battery stays below 30%, the node will definitely not be altruistic and will drop any message that comes and is not intended for it. It is an understood scenario in which you do not want to run out of battery, and the primary objective will be to find a taxi and have a good battery level unit. The altruism analysis is implemented only when the person entity is also the client because the main goal is to find a taxi without waiting too much. When 2 nodes meet and one of them wants a taxi, it is very important to know that the waiting client will help other nodes only if they are also altruist. If not, the client node becomes selfish for the encountered node. Also, the other types of nodes will save the messages of the encountered node, without checking the altruism.

The second step in tuning the algorithm is introducing the "isBusy" field in the logic for the Taxi entity. This parameter shows if a taxi is already busy with a client. In other words, the taxi already received a message for a potential client and saved the answer that it accepted the ride. In this case, it could forward the message and act like a normal node. When a taxi node meets another node, the "isBusy" parameter is verified for the first time in order to know from the

beginning if the ride can be accepted. If the busy parameter is true, the taxi node will behave like a normal node and will save the messages, if the battery level allows it. If the busy parameter is false, it means that a taxi has been found that could accept the order. At this moment, the "isBusy" parameter is set to "true" and stays that way in the simulation process.

The third step in tuning the algorithm is the message flow that was implemented in order to see if a message is transmitted from the client to the taxi and vice versa. This step is important for the result evaluation. First, the client that wants a taxi will generate a message "I want taxi" and forward it to some random nodes. When a taxi encounters a node, it first checks if there is a message with "I want taxi" string and if it is not busy, the taxi will save the message, changing the string. That means it saves an acceptance message that it sends it back in the network. Also, it implemented a logic at the client level node, that also checks if there is an acceptance message for it. It checks if the message it received has the Id of itself and that will mean that the taxi accepted the ride for it. After that, it can save the message with the text "Waiting for taxi", which means the client is not searching for a taxi and he/she is waiting for the taxi that accepted the ride.

3.6 Mobile Application TaxiApp

This section presents a proof of concept on how a taxi application over peer-to-peer mobile networks would work. Because this application is a proof of concept, the communication between devices will happen without accessing the Internet, but there will be used other APIs that will require Internet access. The sign in process or using dynamic map will require Internet access, but the connection establishment between a client and a taxi driver is made via Google Nearby.

Figure 2 presents an overview of the application. There are two devices that communicate with each other: a client and a taxi driver. Both have a smartphone that uses the implemented application. The client will request a taxi and will thus be the Advertiser. A taxi driver will always listen for new connections and if the taxi is in proximity, it will respond and a connection will be established, only if both the devices accept the connection with each other. After accepting the connection, the application uses message exchange between devices. Using Google Nearby, a connection is established between two devices and at this point, they can communicate through messages, thus they can send their location and other information in order to meet at a specified meeting point.

The first message that will be sent is "Hello Client!", which means that a connection has been already made and the client will respond "Hello, Taxi!". The final process is that the client will send the meeting location and the taxi's current position will be displayed on the map.

The implementation of the Android application consists of several critical components organized into four main classes: *Login*, *AccountChooser*, *Client*, and *TaxiDriver*. The *Login* class is responsible for handling user authentication, managing credentials, and maintaining the user session. If the authentication fails, an error screen is displayed to the user. The *AccountChooser* class presents the

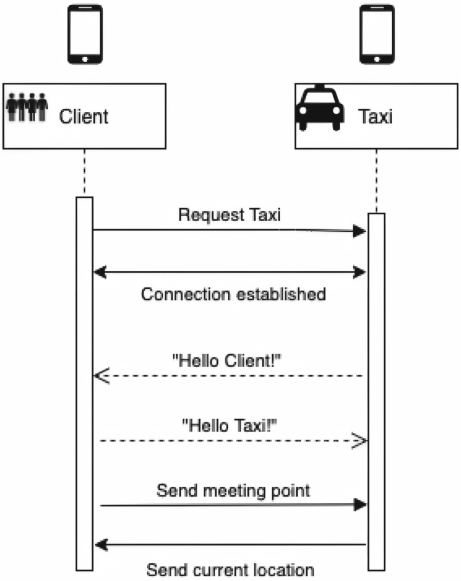

Fig. 2. Client-taxi interaction sequence diagram.

user with the option to choose between a driver or client account, displaying the user's first and last name after a successful Google sign-in. This class is dependent on the *Login* class and only appears upon successful login. The *Client* and *TaxiDriver* classes are crucial as they contain the core logic for real-time map display and information transmission between devices using Google Nearby. The *TaxiPayloadInfo* entity class is designed to encapsulate all the data exchanged between devices, including latitude, longitude, timestamp, and messages. This class implements the *Serializable* interface to facilitate the conversion of objects into byte arrays for transmission, as Google Nearby APIs support payloads in the form of bytes, files, and streams.

The application requires various permissions, such as location and Bluetooth access, specified in the *AndroidManifest.xml* file. These permissions are sensitive and must be explicitly granted by the user, as indicated in the application's user interface. For example, the user is prompted to allow location access, with options to grant it always, only once, or deny it, which affects the application's functionality. One of the key methods, *sendPayload*, is used to send data from the current device to a connected device. This method serializes the *TaxiPayloadInfo* object into bytes before sending it. Success and failure listeners are attached to log the outcome of the transmission. On the receiving end, the *onPayloadReceived* method deserializes the payload back into a *TaxiPayloadInfo* object, allowing the taxi driver to access the necessary information to pick up the client. This comprehensive approach ensures efficient and reliable communication between clients and taxi drivers within the application.

The application is minimalistic, it has a simple UI interface and a few buttons that can be easily accessed by the user. What is important to mention is that this application also uses Internet access, but the data exchange is done without using Internet, which can be a very viable starting point of implementing a full Internet-less application.

When the user enters in the application, he first sees the logo and the sign in button. The first step is that the user needs to have a Google account in order to login in the application. For the future implementations, there will be added other ways of sign in, but it wasn't the main focus on implementing the taxi application. This login phase can be made only once because the account will be saved until the user wants to sign out.

This step requires Internet access, but in order to have a login session without accessing the Internet, we have taken into account other possibilities like having an internal memory that stores the username and password, but this will be done in the future. Also, this step can be easily omitted and the application could start with the account chooser activity, without having an account.

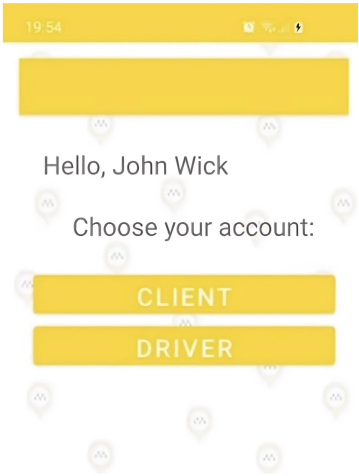

Fig. 3. Taxi application account type chooser.

After the user logs in, the application displays the account chooser activity, where the user can choose if they want to be a client that calls a taxi, or they can be a taxi driver that will accept the requests. Figure 3 shows the application user interface, where the user has to choose between "CLIENT" and "DRIVER".

As it can be seen in Fig. 4, both of devices will have this screen where a connection needs to be accepted before and after clicking the "Accept" button, the two devices can share the information. The "TaxiDriver" and "Client" are the endpoint IDs, meaning the devices' names that are going to be connected. On the left side of the image, we can see the client's screen being asked if they accept

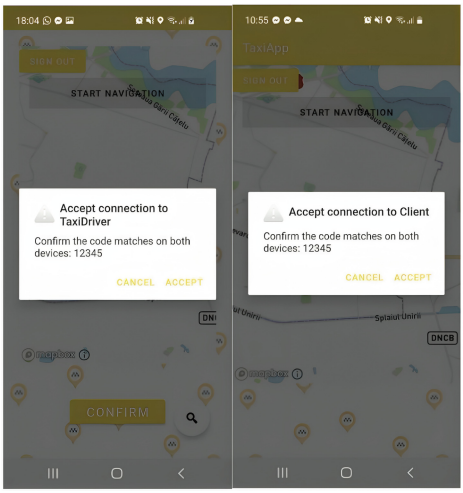

Fig. 4. Accepting the connection in the taxi application.

the connection from TaxiDriver. Both of the devices need to have the same code in order to establish a connection and both need to accept the connection. If one of them clicks "Cancel", the connection won't succeed.

After accepting the connection, on the taxi side will appear a map with the current position and a pin with the client's location, as shown in Fig. 5. The map can be zoomed in or out, depending on what the user wants to see. The pin with the client location is sent via Nearby, without using the Internet.

The "GO TO CLIENT" button starts the navigation process and the driver is able to follow the path and arrive at the destination, which is the client position. As it can be seen in Fig. 6, the map is displayed on a very close zoom level. The navigation process is made using MapBox, which renders the map and computes a route that has a starting point and the destination point. The starting point is the driver's current location and the destination point is taken from the TaxiPayloadInfo entity that is received after the connection between the two devices is established.

Figure 6 represents the moment when the driver arrived at the destination. The blue arrow is the driver's current location and the pin is the destination. On top of the screen there are details regarding the distance that the driver has left until arrival. MapBox is a very efficient tool in implementing a taxi application because it has all the facilities to compute a route and see the source and the destination positions in real time. To have a pleasant user experience, the map is dynamically generated, which means that the application requires Internet access. For the next implementation, integrating a static map that does not require Internet access may be a valid option.

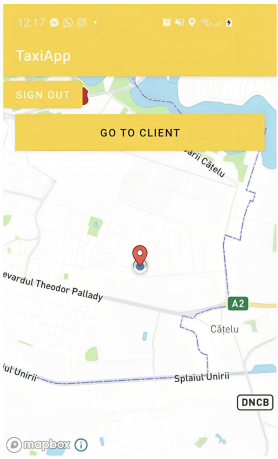

Fig. 5. Client pin in the taxi application.

4 Evaluation

4.1 MobEmu Simulation

The simulation was done in MobEmu, with the HCMM model, giving the different usage scenarios: number of nodes, number of communities, number of travelers, the duration in which the nodes will interact etc. Several relevant parameters are defined for HCMM that describe the usage scenario: the number of nodes will be 20, the duration in which the nodes interact will be 3 h. Also, the perimeter in which the search will take place will be 200×200 meters, split into cells with 10×10. The nodes will be divided into communities, i.e., in a real example there will be groups of friends who want to call a cab. Also, there will be 4 travelers that means each traveler can go to one community to another and that increases the chance that 2 nodes will interact and exchange messages. The transmission range will be 10 m (maximum distance for Bluetooth).

At the end of the simulation, there are several metrics that are given in the console output, adding a metric "messagesThatClientReceivedWithResponse". This metric represents the messages that were forwarded from the client initially to all the other nodes, after that they were processed by the taxi node and saved them with another message and then they came back to the client, which were changed in the waiting state. Also, there are other metrics that are used in the opportunistic networks. Hit rate is the percentage of messages that have reached their destination successfully. The delivery cost is the ratio between the total number of messages exchanged in the network and the total number of generated messages. The delivery latency computes how long it takes a message to reach its destination since it was generated and the hop count is the number of nodes that carry a message until it reaches its destination on the smallest route. It

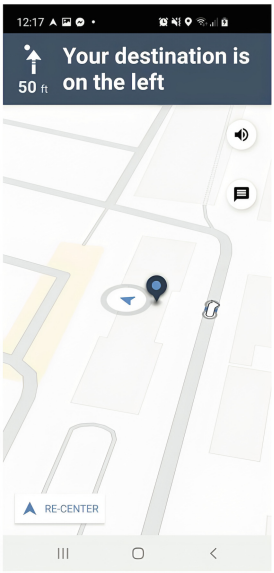

Fig. 6. Navigation in the taxi application.

is important that the number of hops be as small as possible in order to have a good efficiency.

In Table 1 we can observe all the steps, starting with the Epidemic algorithm and adding the new logic presented in Sect. 3. The final logic is adding the message flow and the "isBusy" parameter. The simulation was made with 20 nodes, 2 communities, 4 travelers. In the first simulation there were only a taxi and only a client, rest of the nodes were common person nodes. The "isBusy" parameter changes the results significantly because the taxi will not save any other messages if it is already busy. That means that the taxi driver has already accepted a client and will meet them in the near future. The row which represents the final logic implementation has the best latency and the best hop count because the nodes that were involved in the taxi call flow saved only the messages they were interested in depending also on the "isBusy" field.

Table 1. First simulation.

Algorithm	Hit rate	Delivery cost	Delivery latency	Hop count	Response
Epidemic	1.0	119.51	459.12 s	7.00	
Add Altruism	1.0	190.76	326.22 s	6.71	
Final logic	1.0	179.62	317.67 s	6.14	1

The second simulation was made with 20 nodes, 4 communities and 4 travelers. In this case, there was only a taxi and only a client, while the rest of the nodes were common person nodes. As shown in Table 2, compared to the first scenario, the delivery cost is smaller, but the delivery latency increases.

Table 2. Second simulation.

Algorithm	Hit rate	Delivery cost	Delivery latency	Hop count	Response
Epidemic	1.0	119.51	459.12 s	7.00	
Add Altruism	1.0	155.43	476.53 s	6.41	
Final logic	1.0	146.42	493.26 s	6.26	1

In the third simulation, shown in Table 3, the number of taxis became 2 and the number of clients became also 2 in order to see if each client received a response from a single taxi. The delivery latency increases significantly, but the delivery cost and hop count are decreasing. Although the algorithm manages to have the maximum hit rate, the results are not quite satisfying because the latency increases and the taxi application should have smaller latency and the client should not wait too much. The best results are seen in the delivery cost column, which means there was a smaller number of messages. However, the results are understandable because the more we add conditions for saving messages in a taxi context application, the more we avoid network congestion, but it will take longer until the messages reach the right nodes.

Table 3. Third simulation.

Algorithm	Hit rate	Delivery cost	Delivery latency	Hop count	Response
Epidemic	1.0	119.51	459.12 s	7.00	
Add Altruism	1.0	155.42	476.52 s	6.43	
Final logic	1.0	139.78	513.25 s	5.83	2

The fourth simulation was made with 3 taxis, and the results (shown in Table 4) are similar with the previous simulation, with a slightly decrease of delivery cost, latency and hop count.

4.2 The Android Taxi Application

The evaluation of the taxi application was made between two devices that were both connected to Android Studio on the same implementation, and writing logs in the application, we were able to see and analyse the behavior between them.

After connection two devices, the logging showed that the connection was established. The client received a message from the taxi and it can send back

Table 4. Fourth simulation.

Algorithm	Hit rate	Delivery cost	Delivery latency	Hop count	Response
Epidemic	1.0	119.51	459.12 s	7.00	
Add Altruism	1.0	156.2	469.55 s	6.41	
Final logic	1.0	135.8	493.2 s	4.98	2

a message to confirm to the taxi that the message was received. On the client side, a "Hello, Client!" message was received from the taxi. On the taxi side, the TaxiPayloadInfo object was received, with all the needed information from the client. Thus, the taxi will know where it should meet with the client.

```
V/NEARBY_TAXI Wed Jun 16 11:43:45 GMT+03:00 2021: addOnSuccessListenerFromDiscovery

V/NEARBY_TAXI Wed Jun 16 11:43:47 GMT+03:00 2021: onEndpointFound
V/NEARBY_TAXI Wed Jun 16 11:43:49 GMT+03:00 2021: We successfully requested a connection
V/NEARBY_TAXI Wed Jun 16 11:43:49 GMT+03:00 2021: onConnectionInitiated


V/NEARBY_CLIENT Wed Jun 16 11:43:45 GMT+03:00 2021: addOnSuccessListener
V/NEARBY_CLIENT Wed Jun 16 11:43:56 GMT+03:00 2021:  We're connected! Can now start sending and receiving data.
```

Fig. 7. Time spent until a connection is established.

In Fig. 7, we have extracted some logs from the application, with the current time when they are produced. The logs that have the tag "NEARBY_TAXI" are from taxi side and the others that have "NEARBY_CLIENT" are from the client side. Both listeners of discovery and advertising start at the same time at 11:43:45. The endpoint is found after 2 s and the request for connection is sent after another 2 s. This is the point where both client and taxi driver will receive on the screen the accept connection pop-up. The last log with "We're connected! Can now start sending and receiving data" is displayed after both devices accepted the connection and they now can communicate. There are thus 7 s delay between "onConnectionInitiated" log and the last log because it is the moment when the user presses on the accept button. This delay may vary depending on the user's reaction speed. It can be concluded that two devices next to each other need 4 s to find and connect, and depending on the user's response time there can added a few seconds. It is a very good time, considering the current context.

5 Possible Legal Implications of the Application from the Perspective of Personal Data Protection

The application proposed by the research topic presented above, besides having the potential to present high practical interest, appears in a highly contemporary context where digital technologies are increasingly advancing to support society.

However, these advancements necessarily involve the analysis of the protection of human rights and freedoms in the digital sphere.

In general, online platforms and digital applications enable the development of relationships and communities among users, through which information and content are exchanged for various purposes. Most often, these platforms align with customer preferences and market demands. However, they should always consider digital rights, which are essentially an extension of human rights and freedoms in the digital environment. The implementation of digital applications and platforms, along with new technologies, inevitably collides with some fundamental human rights (e.g., the right to privacy, the right to data protection, freedom of expression, the right to information, etc.). The main regulation in this area is Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, known as the General Data Protection Regulation GDPR)[5].

The protection of personal data represents a guarantee for each individual regarding their data, and its use and purpose necessitated the establishment of a protective framework in the digital world, thus giving rise to the digital rights mentioned earlier. Since digital security and the safety of citizens are very important, and citizens need to be made responsible and protected by authorities, benefiting from a fair online or digital environment, being protected from illegal and harmful content, and having access to the necessary means to interact with new and evolving technologies, it becomes necessary to examine the extent to which the digital rights of its users might be affected by the emergence of a new application, such as the one proposed in this study.

From the outset, one of the highlighted goals of the proposed application in this study (which implements the specific case of taxis, ridesharing) is to increase the level of safety and data protection, of course, as long as its functionality can be maintained. The description of the research indicates that the main challenge is the safety component, and it is emphasized that the proposed application has the potential to be safer than other relatively similar products already operational on the market. This is because, in a system where the network is dynamic and constantly changing (e.g., users move as they please, direct connections occur when at least two of them are a few meters apart), it is much harder to carry out an organized attack. Practically, for such an attack to occur (in a broad sense), it would require synchronization and collaboration, as well as the physical co-location of a large number of attackers. The fact that the proposed network has a certain level of unpredictability (randomness) makes a potential attack much more difficult.

Nevertheless, we aim to analyze aspects that may create sensitivities regarding the protection of personal data and possibly misinformation. One of the objectives to be achieved is the trustworthiness of devices, since it is very important to ensure that the intended recipient at the destination node is who they should be and that the optimal method is being researched to assure users that,

[5] https://eur-lex.europa.eu/eli/reg/2016/679/oj.

by using the application, they will receive a quality and safe service, thereby ensuring that the application will reach a trusted driver. This entails not only trusting that the application user receiving the message via the opportunistic network will forward it but also ensuring that they will not read its content for other purposes (potentially sensitive data) and/or alter it.

This is because, for the application to function, each node must retain the necessary information for the respective request so that the taxi driver can see all the necessary details about the future client, which may include: last name and first name, profile picture, rating, destination, etc. In this regard, we believe that mechanisms for obtaining and managing the consent of individuals whose personal data will be used, as defined by the GDPR, must be considered.

Article 7 of the GDPR document states the following (emphasis ours): "1. Where processing is based on consent, the controller shall be able **to demonstrate that the data subject has consented to processing of his or her personal data**. 2. If the data subject's consent is given in the context of a written declaration which also concerns other matters, the request for consent shall be presented in a manner which is clearly distinguishable from the other matters, in an intelligible and easily accessible form, using clear and plain language. Any part of such a declaration which constitutes an infringement of this Regulation shall not be binding. 3. **The data subject shall have the right to withdraw his or her consent at any time**. The withdrawal of consent shall not affect the lawfulness of processing based on consent before its withdrawal. Prior to giving consent, the data subject shall be informed thereof. It shall be as easy to withdraw as to give consent. 4. When assessing whether consent is freely given, utmost account shall be taken of whether, inter alia, the performance of a contract, including the **provision of a service, is conditional on consent to the processing of personal data that is not necessary for the performance** of that contract".

The GDPR requirements for consent being significantly more robust and established for specific circumstances, it is mandatory to obtain Informed or Affirmative Consent for Data Processing and Explicit Consent for Processing a Special Category of Data.

From the outset, the study of this application seeks to identify protective methods, including for sensitive personal data, such as certain reputation mechanisms or data transmission algorithms that protect users from malicious nodes, especially since - as detailed in the technical part of the study - there can be multiple nodes (such as the next hop) between the source and destination that opportunistically collect information and then redirect the message.

We started from the premise that, generally, in an opportunistic network where the topology is unknown, many malicious nodes could exist, sending spam messages, altering information, or discarding messages of interest (containing sensitive personal data). For this reason, we presented an opportunistic trust and reputation mechanism (SAROS) that could detect and prevent such malicious nodes by detecting false messages and pre-establishing trust using social information.

Simultaneously, the study reflects concern for ensuring that only users who want to take a taxi and those providing taxi-like services receive these messages, with the node's interests tied to services like "taxi," "ridesharing," etc., serving as a way to filter messages in the opportunistic network.

In the technical research, we showed that there are two types of messages: messages received from other nodes that need to be redirected because they have not yet reached their destination, and messages generated by the node itself, with each message type having a separate memory location. For messages from other nodes, there is a data memory that, when it reaches its maximum capacity, allows for the deletion of messages under certain conditions.

Legally, the issue arises whether storing messages would allow them to be used/delivered to third parties interested by a taxi driver, who would be a malicious node in this regard. Since the possibility cannot be excluded that a malicious node might try to "peek" at the messages passing through it, rather than just forwarding them, or even attempt to alter or discard them, and since there is no centralized authority or system to verify trust levels, the we proposed implementing reputation mechanisms or similar solutions.

The legal status of the node (or its holder) must be analyzed, specifically whether it becomes a data processor or controller. A data processor is an entity that processes data on behalf of the data controller (e.g., a company providing a SaaS CRM platform storing data for its client, a large bank, would be a data processor). A CRM (Customer Relationship Management) system manages sales information, contacts, and the entire interaction process with a customer long before the actual sale occurs. The aim of a company is not only to access new customers but also to retain existing ones by exploiting new sales opportunities. This mechanism can be applicable to taxi and ridesharing services.

We previously showed that nodes receive messages from other nodes and store them in memory to forward them. The node's own messages are kept in a separate memory and are never deleted except under special conditions when new messages need space.

Regarding data deletion - beyond the technical aspect of storage capacity and progressive data elimination to make room for new data, which should somehow happen automatically from the system - the user (Client) should have the option to request the deletion of their own data when their interests prevail, and the application must allow for such a request, as required by personal data protection laws.

We also indicated that exchange statistics and exchange history contain information about data exchanges when contact occurs. The main information includes the time and duration of exchanges and the history of exchanged messages. To clarify, the application is designed to retain/store, in principle, only metadata (information about the message, not its content, such as sender and destination), although even from such metadata, some conclusions about the message sender can be drawn.

Therefore, the node's function must be considered, whether it only stores or also processes data. Here, GDPR rules apply, allowing data storage/processing

only with express consent, meaning the data usage must be legitimate (conventional). Additionally, the method for exercising the right to request the deletion of personal data after use (the right to "be forgotten") must be considered.

Given that with standard Google authentication, the user, once authenticated in the application, does not have to re-enter the username and password each time, as they are remembered from the first login, it is proposed to use Google Nearby, which allows a peer-to-peer connection between two devices without using the Internet, providing secure data exchange. Traffic from Google Nearby is encrypted, so an intermediate node can decrypt this data to forward the message.

Moreover, Google's well-known right to ask for and obtain user consent for using personal data for its platform's authentication and operation needs to be evaluated in terms of GDPR compliance (applicable to EU citizens and non-EU residents in the EU, with stricter rules in the USA).

Since the proposed platform involves collecting, storing, and processing personal data, its effective operation will also require appointing a Data Protection Officer (DPO).

According to Article 37 of the GDPR, a DPO must be designated if the controller's or processor's main activities involve "regular and systematic monitoring of data subjects on a large scale" or if the entity processes large-scale special categories of personal data (e.g., race/ethnicity, political beliefs as defined in Article 9 of the GDPR). A DPO can be an employee or a third-party service provider with extensive knowledge of national and European legislation and proven IT expertise (e.g., a consultancy firm or law firm), who must report directly to the highest management level and operate independently (the GDPR explicitly prohibits dismissing or penalizing the DPO for performing their duties and does not limit the term of office).

The study also shows that after a connection is established between the Advertiser and Discoverer, they can exchange data in the post-connection phase (usually including the client's rating, a profile picture, etc., similar to an Uber application). Subsequently, it will be up to the developer to decide based on APIs what should happen after a connection is established and what type of data should be exchanged.

Beyond the altruism of the system, from a legal perspective, the GDPR obligations (see Articles 15, 24, 30, and 32) for conducting a complete data mapping analysis must also be considered.

We showed that, after accepting the connection, the application uses message exchange between devices; using Google Nearby, a connection is established between two devices, allowing them to communicate through messages, sharing their location and other information to meet at a specified point. The application can develop a chat feature (like Bolt, Uber, etc.), and generally, exchanged messages can be automatically (or manually) deleted by the Client or Taxi driver. However, these messages often still need to transit through intermediate nodes, requiring an evaluation of how to ensure data protection for all application users.

Although in this phase, the application also uses Internet access, it is important that data exchange happens without using the Internet. We consider this a very viable starting point for implementing a complete application without Internet use, which would further enhance the safety component, the main challenge of the project.

While it is premature to analyze, if the application proposed in this paper is implemented and operationalized, and disputes arise regarding liability for non-compliance with personal data protection laws or misuse of such data broadly, a special issue will be the territorial jurisdiction of the courts resolving such claims. In the absence of conventional norms established and accepted by the user upon installing the application, for national disputes, the jurisdiction rules from the Civil Procedure Code will apply. For disputes with extraterritorial elements from EU member countries, the provisions of Regulation (EU) No 1215/2012 of the European Parliament and of the Council of 12 December 2012 on jurisdiction and the recognition and enforcement of judgments in civil and commercial matters will apply. For non-EU states, international private law norms will apply.

Summarizing the potential legal implications, we conclude that the application proposed in this study may offer a higher level of personal data protection than other functional applications on the relevant market. Some issues can be technically implemented as far as possible without hindering functionality, while other appropriate data protection policies will remain the responsibility of the developer and especially the application operator. The operator will need to align with the obligations under the GDPR, as per Article 24(1): "Taking into account the nature, scope, context and purposes of processing as well as the risks of varying likelihood and severity for the rights and freedoms of natural persons, the controller shall implement appropriate technical and organisational measures to ensure and to be able to demonstrate that processing is performed in accordance with this Regulation. Those measures shall be reviewed and updated where necessary".

6 Conclusions and Future Work

This paper presented simulations in MobEmu after implementing a routing algorithm in a peer-to-peer taxi application. Routing is one of the most challenging problems in opportunistic networks because of the intermittence of the network connection. A routing algorithm in an opportunistic network was implemented, taking into account the fact that there are 2 types of nodes: taxi and person, with different behavior and different parameters, different messages generated. Of course, the concrete implementation of the routing algorithm in the given context has more challenges and decisions that will need to be made.

Several simulations were performed for the new routing algorithm, moving from the simplest epidemic routing algorithm, to an algorithm that takes into account altruism, message content and new parameters added. This algorithm will be improved depending on the real context and will be performed on several types of simulations to discover new challenges and improve metrics. Opportunistic networks are a vast topic that can be approached by many methods

and there are many things to learn and study. We cannot obtain perfect results for all metrics, they depend on what is desired for the algorithm, e.g., if we want a low latency or low cost or low hop count, one of the metrics will have to be sacrificed. There will always be compromises that we will need to make, but the general idea is to find the best solutions in the context of a taxi application, i.e., a fairly fast response, a high level of trust and not to congest the network.

Several simulations were performed for the new routing algorithm, moving from the simplest Epidemic routing algorithm, to an algorithm that takes into account altruism, message content and new parameters added. This algorithm will be improved depending on the real context and will be performed on several types of simulations to discover new challenges and improve metrics.

On the Android application side, this paper presents a proof of concept regarding a taxi application over peer-to-peer networks. Being a simplistic implementation, it can be a starting point in the implementation of an application with full functionalities, which does not access the Internet. It was possible to send messages between two mobile devices using Google Nearby, calculating the time in which the two devices connect. This field is still one to explore, but we believe that the steps that have been taken in this endeavor are very important. This paper comes to support the existing contributions and to demonstrate that there are well-defined starting points.

The next steps will be to find a better solution for describing the problem and improve the routing algorithm before implementing the application. The biggest issue for now is the latency that needs to be smaller because in a taxi application context, the client should not wait too long. Also, adding the waiting time for a node after it receives the response could avoid waiting too much for a taxi and gives the client the possibility to send a new request.

A very important aspect for the future implementation is the moment when the taxi accepts the client, which implies the transmission of all messages between nodes. A specific waiting time will be required so that the customer can go to the taxi meeting point. The taxi must be at the meeting point at that time because the degree of trust depends on it. There are many factors to be considered and the final implementation of the routing algorithm is affected by many parameters. The level of trust analysis will draw some real limits that we have to think about whether it is safe to trust any responses from any mobile device.

Also, another fact that needs to be implemented is the speed difference between a taxi and a person. Some nodes will move faster than others and this aspect could influence the final results. The latency may depend on the speed at which the nodes are moving and may decrease significantly.

Furthermore, a simulation is needed where to compute how many malicious nodes have been discovered and how many have compromised the messages. There will be certain malicious nodes whose number will increase, and the nodes must figure out if the message received from the malicious node is reliable or not.

At the application level, there can certainly be improvements in terms of communication between devices. The application must also be tested in a real

context, with several devices involved to see if the information is sent. Of course, the logic of communication must be improved, because now the scenario has been implemented with a taxi and a client, but in real context, there can be used other devices to act as normal nodes.

Acknowledgments. This work was supported by the NetZeRoCities Competence Center, funded by the European Union - NextGenerationEU and Romanian Government, under National Recovery and Resilience Plan for Romania, contract no. 760007 / 30.12.2022 with Romanian Ministry Of Research, Innovation and Digitalisation, through specific research project P5-Smart City and Digital Twins, and by project "DIGITAL4Security - European Masters Programme in Cybersecurity Management & Data Sovereignty" (ID 101123430), funded under the Digital Europe Programme.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Ciobanu, R.I., Marin, R.C., Dobre, C.: Mobemu: a framework to support decentralized ad-hoc networking. Modeling and Simulation in HPC and Cloud Systems pp. 87–119 (2018)
2. Ciobanu, R.I., Marin, R.C., Dobre, C., Cristea, V.: Trust and reputation management for opportunistic dissemination. Pervasive Mob. Comput. **36**, 44–56 (2017)
3. Ciobanu, R.I., Negru, C., Pop, F., Dobre, C., Mavromoustakis, C.X., Mastorakis, G.: Drop computing: Ad-hoc dynamic collaborative computing. Futur. Gener. Comput. Syst. **92**, 889–899 (2019)
4. Conti, M., Giordano, S., May, M., Passarella, A.: From opportunistic networks to opportunistic computing. IEEE Commun. Mag. **48**(9), 126–139 (2010)
5. Huang, C.M., Lan, K.c., Tsai, C.Z.: A survey of opportunistic networks. In: 22nd International conference on advanced information networking and applications-workshops (aina workshops 2008). pp. 1672–1677. IEEE (2008)
6. Musolesi, M., Hailes, S., Mascolo, C.: Adaptive routing for intermittently connected mobile ad hoc networks. In: Sixth IEEE International Symposium on a World of wireless mobile and multimedia networks. pp. 183–189. IEEE (2005)
7. Pelusi, L., Passarella, A., Conti, M.: Opportunistic networking: data forwarding in disconnected mobile ad hoc networks. IEEE Commun. Mag. **44**(11), 134–141 (2006)
8. Sharma, D.K., Dhurandher, S.K., Agarwal, D., Arora, K.: krop: k-means clustering based routing protocol for opportunistic networks. J. Ambient. Intell. Humaniz. Comput. **10**, 1289–1306 (2019)
9. Socievole, A., Caputo, A., De Rango, F., Fazio, P.: Routing in mobile opportunistic social networks with selfish nodes. Wirel. Commun. Mob. Comput. **2019**(1), 6359806 (2019)
10. Trifunovic, S., Kouyoumdjieva, S.T., Distl, B., Pajevic, L., Karlsson, G., Plattner, B.: A decade of research in opportunistic networks: challenges, relevance, and future directions. IEEE Commun. Mag. **55**(1), 168–173 (2017)

11. Vahdat, A., Becker, D., et al.: Epidemic routing for partially connected ad hoc networks (2000)
12. Wu, J., Chen, Z., Zhao, M.: An efficient data packet iteration and transmission algorithm in opportunistic social networks. J. Ambient. Intell. Humaniz. Comput. **11**, 3141–3153 (2020)