Mobile Ticketing System For Public Transport Based On Bluetooth Low Energy

João Guilherme Martins
joao.g.c.s.martins@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2017

Abstract

Ticketing systems for public transport have evolved, according to the latest technologies, to provide a better travel experience for passengers. Most of the ticketing systems use technologies that require a lot of manual interactions from passengers. Meanwhile, smartphones have gained increasing acceptance, and their capabilities can be exploited in the transport area. This brings us to Mobile Ticketing, allowing to use our mobile phones to travel in public transport. We propose an implicit ticketing system based on Bluetooth Low Energy when travelling on public transport. Our system detects the presence of passengers inside a public transport vehicle, using BLE technology available on mobile phones. The accuracy in passenger detection is crucial to the acceptance by the public transport companies and passengers. This work addresses mobile computing issues like performance, battery consumption, security and connectivity. It also concerns about mobile ticketing topics as the fare calculation method and the validation process. A prototype with the discussed implementation was developed and tested, with results that prove the potential of the solution.

Keywords: Mobile Ticketing, Ticketing System, Smartphone, Bluetooth Low Energy, Public Transport

1. Introduction

Public transport systems have gone through many changes, providing better conditions to passengers in order to meet their needs. Nowadays it is possible to verify seat availability, as well as reserve and purchase a ticket online. The time-consuming task of acquiring a ticket is becoming easier and more intuitive, therefore less and less a barrier to public transport.

The adoption of mobile phones has become so widespread, and mobile Internet access affordable to the masses. This enables public transport providers to explore new ways to purchase and carry electronic tickets using smartphones, introducing a mobile ticketing approach. Mobile ticketing is the process whereby customers can reserve, purchase and/or validate tickets using smartphones or other mobile handsets. It is becoming the future of ticketing in public transport, allowing passengers to buy tickets in advance and from anywhere. From the providers side, it allows them to gain insight into the behaviour of the system, providing improvements according to usage. All check-in/check-out actions taken in just one trip are time-consuming tasks. However, these actions are very relevant to the companies, once they provide information about the service occupation. This way, it is essential not only to reduce the check-in/check-out actions but also to provide the information required for the operators to study possible improvements in the system.

Therefore, we propose an alternative to the traditional methods, to implicitly interact with a technical system to make a check-in/check-out, simplify the validation process. Passengers may travel without the need to present the ticket. It only needs to carry a daily device - the smartphone - which, communicating with the vehicle devices, automatically takes care of billing for the journey. Bluetooth Low Energy (BLE) is available in every modern smartphone, and its capabilities can be explored to develop a passenger detection system that can be used in a new mobile ticketing system. This project stands out for its deploying simplicity because it is not necessary to have an On-Board System (OBS) to process the check-in/out information. It only needs to deploy a set of BLE beacons inside the vehicle and a client-side smartphone application. This application performs all the necessary processing, including the communication to a central server.

1.1. Objectives

One of the main goals is to achieve a good accuracy when detecting a passenger on board. It
is imperative to avoid charging people that are not on board, or not billing passengers that completed a journey. The reliability of the detection is crucial to the acceptance of the service by the stakeholders since the system must not lead to a monetary loss for customers neither for public transport companies.

The system should facilitate the public transportation experience, providing the passenger with an easier way of travel in a public transport vehicle. It should be as simple as buying a ticket using the smartphone, wait for the vehicle, enter it, travel, and exit at another station. The system automatically calculates the fare of the journey and charges it from the user account.

It is crucial that the system achieves a low energy consumption. Communication is the main source of battery drain and should be controlled. Connection intervals can be tuned specifically to develop a system with low energy consumption without losing accuracy.

Since wireless communication happens at open air, everyone can eavesdrop the communication or impersonate a legit device. Issues like these mean that security and privacy measures have to be taken to prevent abuse of the system.

In order to provide a great solution to public transport companies, the system should consist in just a few changes in the infrastructure.

And finally, the system should be available whatever smartphone capabilities, except BLE and Global Positioning System (GPS). Although our prototype being developed in Android, similar solutions for other platforms can be implemented.

2. Related Work

Each public transport operator has a tariff method according to its business model. The ticket fare can be a fixed amount per journey, independently of the source or destination. Regarding the location awareness, it can be calculated based the distance travelled; or on the number of stations within entrance and exit stations; or aggregating several stations in zones and calculate using the number of zones crossed by the passenger [7].

Concerning about ticket validation mechanisms, one of the most basic ways is the check-in only, in which the user has to present the ticket at the entrance. Check-in/Check-out (CICO) method, needs the user to explicitly presents the ticket at a reader, when entering and exiting [9]. There are more technological approaches like Check-in/Be-out (CIBO), where the check-in is explicitly made but the system automatically detects that the user exited the vehicle, and finally Be-in/Be-out (BIBO), a hands-free scheme that detects the presence of a device inside/outside a vehicle automatically [6, 10, 8].

When using electronic fare management, the communication between tickets and readers is crucial. BLE is a wireless technology that exchanges data over a short distance using radio transmissions, using low bandwidth and low latency, providing a very energy efficient communication [4]. It follows active RFID approach, although coupled with battery on the tag side. Active RFID systems are suffering a gradual replacement by BLE [2]. Nowadays every recent released mobile phone has BLE support.

BLE beacons consist of small devices that periodically broadcasting Bluetooth signals containing programmable information [5].

ALLFA was the first big innovative fare management system tested in a real environment. It detects passengers inside vehicles using special electronic cards or mobile phones. These devices contain a radio frequency communication interface, which communicates with the antennas installed inside the vehicles. There are two types of antennas: “wake-up” antennas, that sends an initial wake-up signal to the ALLFA ticket, and an access antenna to communicate with the OBS [6, 3]. All data transmitted from the ticket to the OBS is also transmitted to a remote system. The fare is calculated centrally and offline in this back-office system [3].

“Esprit” is a BIBO solution based on an electronic user device, with a radio-based communication interface. The vehicle is equipped with an OBS that controls the data that is sent to the user devices through an antenna. This computer also sends all the information to a remote back-office server [6]. When the passenger enters in a public transport vehicle, the computer broadcasts a signal to all users inside the vehicle to activate the user devices from the sleep mode. When inside the vehicle, the user device is periodically receiving information with enough information to calculate the fare [6].

Johannes Kepler University’s investigation team in Austria aims to develop a BIBO system, which includes a user side BLE beacon or a BLE-enabled smartphone, an OBS controlling the communication with the user device and with a remote server. The user device broadcasts its unique ID when entering the vehicle, which is received by the OBS. In order to accurately detect the presence of the passengers, OBS performs scheduled BLE scans to remove ambiguosity. Capturing a BLE device at all scans indicates the presence of a passenger inside. When the scan does not detect the user anymore, then his trip is over.

Link Consulting started a project with aim on the automatic detection of user position relative to the bus, using BLE and location services. This project uses BLE beacons, deployed inside the public transport vehicles, to interact with the mobile user de-
vice. When the user is outside the vehicle, the application is scanning for beacons. When it detects a set of beacons from the same bus, it turns on the location and starts recognizing if the passenger is moving, using the smartphone GPS. When app does not detect enough beacons from the same bus, the user should have exited the bus, and it is checked-out. Since this master thesis is supported by Link Consulting, some of the ideas used in this project were adopted in the development of our system.

3. Implementation

We developed a mobile ticketing platform for public transport, proving the viability of using BLE beacons in this domain. In our solution, the user owns a BLE-enabled smartphone with the developed application installed. A set of BLE beacons are deployed inside public transport vehicles. Beacons have unique identifiers that indicate which vehicle they belong. The app detects user presence interacting with the beacons, through BLE. GPS and smartphone sensors are also used to detect movement and location, helping on the fare calculation.

This system combines BIBO and CIBO. It uses a "check-in" perspective because the user has to manually confirm their presence inside the vehicle pressing a button in the app. However, this check-in can be made at the passenger seat. The need of a check-in action is detected when the passenger is near the vehicle, so the system can detect that the user is inside the vehicle, introducing the "be-in" concept. The check-in action is necessary to eliminate doubts about the user presence inside the vehicle. The system is also able to detect that a user is outside the vehicle, experiencing the "be-out" idea. To perform these detections, we use BLE technology, due to its advantages, as referred in section 2.

With our architecture is the mobile application that detects the beacons and communicates the information, through mobile operator internet or using the vehicle Wi-Fi, to a back-end server that takes care of all processing. Therefore it is not necessary an OBS to control the passenger detection or to communicate with the server. This is an advantage to public transport operators since it avoids a huge modification in the public transport infrastructure.

3.1. Architecture

In Fig. 1 are represented the modules that compose the system and the relationships between them. In the mobile app the user can check the current state, i.e. if is far or near a vehicle, or if a trip is currently in course or it is already finished. The app can also show the past trips and generate validation codes. It does not store sensible information, such as detailed user information or vehicles routes. All this data is controlled by the back-end that also performs the sensible tasks like fare calculation, charging money and management of every trip information.

![Figure 1: Components and their relationships of the proposed solution](image)

Inside the vehicle, we have a set of BLE beacons. As we are using as example the case of Carris (Lisbon) public transport, we made a study of their bus dimensions, in order to conclude which would be the best configuration for beacons interval time and broadcasting power, as well as the configuration of the beacon set inside the vehicle [1].

We used Estimote® Proximity Beacons (hardware version G1.8 and firmware version 4.9.1). These beacons can either operate in iBeacon or Eddystone protocols, but we used iBeacon due to its simplicity and documentation abundance. To achieve a good balance between detection reliability and battery consumption, we decided to use a Broadcasting Power of -20dBm (approximate ranging of 4 meters) and an Advertising Interval of 500ms, which allows a battery lifetime of approximately 18 months.

With this values well defined, we found the best way to deploy a set of beacons inside a typical vehicle of Carris: 12m length and 2.5m width for standard models. The configuration example for a standard model can be observed in the scale Figure 2. The blue dots represent the beacon, and the blue circles represent the range of the beacons. The circles become darker when they intersect each other. We ensure that the disposition of beacons must be such that within the vehicle area it is always possible to detect at least two beacons with the same identifier.

Using iBeacon protocol we have three different values to define: Universally Unique Identifier (UUID), major and minor. This way we can set different values for each service. We use UUID to define all the beacons that are used in the application. Then, we have major associating each beacon to a different public transport operator. Finally, minor values differentiate the vehicles from the same...
operator.

Smartphone’s GPS is responsible for detecting the source and destination stations of the trip. The "check-in" button pressed inside a vehicle triggers GPS to register the current location as the origin station, to use in fare calculation. Then GPS is turned off to save battery. When the beacons are not detected anymore, the GPS is turned on again to store the destination station, and the trip is over, and GPS is turned off.

The traveller is responsible if the ticket is not checked-in because the smartphone is out of battery, no GPS signal or without internet connection, or also if the check-in button was not pressed when entering the vehicle. When an inspector enters the vehicle to check the validity of the passenger tickets, the passenger has to generate a Quick Response (QR) code. To do this, the user clicks on "inspection" button, that automatically shows a code containing the user information and current trip data. Then, the inspector can read the code and confirm the validity of the traveller’s journey.

The back-end is responsible for storing all the system information, as well as perform the more compute-intensive tasks. It contains information of all users; a journey history with all the trip events (check-in, updates and check-out); route and stations information; and is responsible for calculating the fare of each trip performed. Its features will be described in Section 5.

3.2. Assumptions

As this project has a main focus on the accurate detection of passengers, therefore some of the capabilities of the system are not going to be explored, allowing greater flexibility in future when continuing the development of the project. To simplify the system and to give priority to the most crucial points, the following assumptions are made: we assume internet availability in all smartphones and inside every public transport vehicles, whether via on-board vehicle Wi-Fi or through mobile operator data; although there are different possibilities to calculate the fare due to system flexibility, we assume a station-based fare, where the price is determined according to the number of stations crossed by the passenger; in this implementation due to time constraints the payment system integration will not be addressed.

4. Mobile Application

We developed an Android application to test and prove that the project is feasible.

4.1. Features

To start using the application, the user needs to perform a registration. To identify the user, we use the phone number, since it is unique, so there is no possibility of ambiguity. Besides, is required a password to protect the account.

If when the user submits the information, it is accepted by the back-end, then returns a random 4-digit code. This code ideally is sent by a Short Message Service (SMS). However, due to the difficulty of access to free SMS system, this message is simulated as a pop-up message in the app with the generated code. If the user enters the received code and it matches the generated user, then can charge the account with money and start travelling.

If the current balance is not higher than the minimum fare, the application does not allow to start a new trip. Therefore, at any time the user can charging money, in a quick task where only has to choose the amount to charge, fill the tax information (if required), and submit.

The main feature of this mobile application is the state visibility of the user at any time. The states have different behaviour and representation, with distinct colours and figures that made the state easy to distinguish. Here we present the five states in which the application can operate:

- **Out**: No beacons are found. Neither check-in nor check-out can be made, so the user cannot interact with the state. GPS is turned off, in order to reduce energy consumption. This state is represented in red, with a figure of a user far from a vehicle.
- **Near**: Minimum number of beacons from the same vehicle is found. GPS and activity recognition is turned on, so it is possible to detect movement and register the current location. In this state, we can see which vehicles are being identified, as we can observe in Figure 3. The user can choose in which vehicle are going to enter. Otherwise, the application selects the nearest (using RSSI). Pressing the "Check-in button" triggers the entrance transition. The "Near" state is represented in dark blue, with a figure of a user at a bus stop, as shown in Figure 3.
- **Checking in**: Not also the minimum number of beacons of the same vehicle are detected but...
also movement, through the GPS or Activity Recognition module. Therefore the user must press the "Check-in button" to confirm that is inside the vehicle. This state is represented in orange, with a user entering a bus.

- **In**: When the user press the "Check-in button", the application enters in this state. The location is stored and also a list of every possible bus to check-in, in case the bus automatically was chosen was not the right one. While the application is detecting the minimum number of beacons, GPS is turned off to reduce battery consumption because it is not necessary during the journey. The "In" state is represented in green, with a user inside a bus.

- **Checking out**: When the application stops detecting the minimum number of beacons of the current vehicle, it enters in this state. Here the application checks if any alternative buses have been detected since the entrance in which it was also possible to check in. If yes, the application updates the current bus and enters again into the "In" state. Otherwise, the journey is finished, and a summary is shown, with the entrance and exit information, as well as the cost. After showing the summary (whether the user presses the button or the timeout of 10 seconds expires), the application transits to "Out" state, trying to detect new beacons. If the application is running in background, the summary is not shown, and the application transits fast.

At any time it is possible to check the trip history with all the travels that were made by the currently logged user, as we can see in Figure 4. In this window, we can observe dates of check-in and check-out, the travelled bus, and the entrance and exit stations. When the user clicks on a trip of this list, a map is shown with marks in all the stations within that journey. We can observe an example in Figure 5. A green marker indicates the entrance station; the red marker shows the exit station and all the yellow markers are the stops performed by the vehicle during the journey. It is also possible to generate inspection codes for each trip, at the specific inspection button.

App has an **inspection mode** that generates a validation code for a travel. This is need when an inspector enters the vehicle to check if all passengers have a valid ticket. When within a journey, there is an "Inspection button", that shows a Quick Response code. This code contains the user identification token, the beacon identification that is detected by application, and the identification token of the relevant trip. To implement this feature, we used ZXing, an open-source barcode image processing library implemented in Java.

### 4.2. Solution architecture

A region can be described as the entire area reachable by beacons that match the region specification. These values are the same as those used by beacons: UUID, major and minor. A region can be defined using only UUID, UUID + major or UUID + major + minor. We defined a UUID to relate beacons to our application and a major value associated with all Carris vehicles.

There are two distinct methods for apps to interact with regions and beacons within a region: monitoring, which triggers actions when a device is entering/exiting the range of beacons defined by the region; and ranging, where actions are triggered according to the proximity to a region, indicating if the device is immediate, near or far from that region. It is important to know that once the beacon is out of range, it still takes to the mobile operative system up to 30 seconds to truly recognise that fact. This built-in, non-adjustable delay is there to prevent false "exit" events, like when a crowd temporarily obstructs the beacon, and it stops being detectable for a few seconds.

The monitoring task is not always detecting entering and exiting occurrences. It is defined based on two values: the scan period and the interval between scans. We set the values to perform 2 seconds scanning with 10 seconds of interval, which gives us
a good balance between battery consumption and expected results, based on tests conducted. After monitoring task detect the presence inside the region, ranging task count the number of occurrences for each minor. Ranging is also defined based on the scan period value and the interval between scans. We set the values to perform ranging scans with 2 seconds duration and with intervals of 2 seconds too.

The app shows a notification to check-in if is detecting the required number of beacons and identifies movement. To identify movement a service tracks the user movement, considering that the user is moving aboard if three conditions are simultaneously reunited: the beacons of that vehicle are being detected for more than the minimum detection time (15 seconds); if movement is being tracked for more than the minimum threshold (10 seconds); and if the average speed of the user matches a vehicle speed (assuming that a minimum of a vehicle is about 21.6 km/h).

Notice that activity recognition and movement detection is only used to identify that a check-in is required to notify the user to do it. If the user is inside the vehicle and the application does not detect activity/movement, it is always possible to perform a check-in, and the user should do it.

Every sensible information is computed and stored in back-end side, to guarantee that there are no inconsistencies or fraud. This includes information about the user, travels or vehicle transport routes; and tasks like the translation between beacon identification to its respective vehicle, fare calculation or balance charging.

The back-end is implemented in order to provide its services through a REST (Representational State Transfer) architecture style. To consume the services provided by the back-end, we use Retrofit, a REST client for Android (and Java).

4.3. State machine

Travel processing is controlled by a state machine dependent on several factors, that can be observed in Figure 6.

1. The application starts in the "Out" state. When it starts detecting the minimum number of beacons (2) with the same identification, it transits to "Near" state. Otherwise, it stays in "Out" state.

2. While it is detecting the minimum number of beacons it stays in "Near". If at any point, the application detects that the user is moving, using the activity recognition, the current location is stored. Then, if the user has a positive balance, the application changes to "Checking In" state. Otherwise, the application does not allow to check-in and warns the user to charge it. Clicking on "check-in button" transits directly to "In" state. Although it only changes to this state after storing the current location. If the minimum number of beacons are not being detected the application goes to "Out" state.

3. In "Checking In" state, the current location is always getting stored with four seconds of interval. This state is waiting for the user to press the check-in button, in order to change to "In" state, indicating the beginning of a journey. When the button is clicked, the "check-In" process begins, in which the application communicates to the back-end not only the current location but also the beacons being detected. The back-end returns the identification number of that journey. Again, if no beacons are detected transits to "Out" state.

4. When "In" it is only possible to change to state "Checking Out" when the minimum beacons of the bus in which the check-in was made, and there are no alternatives to update the check-in. If the application stopped detecting the current bus beacons but continues to detect the beacons from an alternative bus, it starts the "Update Check-in" task, communicating it to the back-end. Notice that it does not indicate that the user changed from one bus to another, but that the initial check-in was made in the wrong bus.

5. When in "Checking Out" state, the current location is stored in order to communicate it to the back-end, along with journey and user identifications, to perform the "Check-out" and change to "Out" state. This is an automatic transition that has not related conditions. In back-end side, the amount to pay for the complete journey is calculated, and it is deducted from the user balance.

![Figure 6: Implementation State Machine diagram.](image-url)
5. Back-end server

We developed a back-end that should be managed by the public transport operator. This server can control all information about the system, such as journeys and users. It is possible to the operator to make adjustments to its services based on the information held by the back-end, like vehicle route modifications or even changing the fare calculation. To do this, it is connected to a transport buses and route database that has the information about all the vehicles in transit and what their possible routes.

5.1. Features

Each user has an account, in which is stored the personal information, trip history and balance. All users are identified by their phone number, which is unique and personal. A password protects each account, to prove that the person who is trying to use the application is who claims to be. When the user is making the registration in the application, is necessary to fill a form with the personal information and send to the back-end. If all data is valid, the back-end generates a random 4-digit code that returns to the application. When the user correctly inserts this code, will obtain a unique token that identifies the user in each future communication.

One of the main focus of the back-end server is to calculate how much to pay for a journey. The fare calculation is based on the following steps:

1. Identify the bus in which the passenger is travelling. Then, get that bus route, which includes all the stations it stops from the origin to the destination.

2. Identify the entrance location stored in the check-in information. Then, compare each station coordinates with the user location when the check-in was performed, to select the nearest station. Comparing all bus route stations with check-in location, the minimum distance obtained refers the nearest station from where the check-in was performed, therefore the entrance station to use in the calculation.

3. After that, we need to know the exit station of the travel. Now we just have to compare the check-out location with the stations in the route that follows the already selected entrance station.

4. Finally, as we have already obtained the stations of entry and exit, we can now count the number of stations between them and calculate the price to pay for the trip.

Although it is possible to choose any fare calculation method, in this prototype we are using a station-based method. In this calculation, we use a fixed price for starting a trip plus a fixed price for each number of intermediate stations.

Back-end server is responsible for storing every trip that is made using the system. It is important to record trip logs with every relevant information. During a journey, there are three important moments in which is important to record the application context: when the application starts detecting a minimum number of beacons from the same bus, it is necessary to translate the beacon identification to the correspondent bus number.

1. When the check-in button is pressed, the application sends the user token, to identify which user is starting the journey, the identification of the beacons in which is checking-in, a list of beacon IDs of all other possibilities to check-in, and finally the current latitude and longitude. The back-end creates a new journey entry with this received information, generates an identification of this journey entry and returns it to the application.

2. When the application stops detecting the minimum number of beacons of the given beacon ID in the check-in, it will verify if any of the alternatives are being identified. If yes, an “Update check-in” is performed, in which the application sends the journey identification to say which trip is being updated, as well as the coordinates (latitude and longitude) of the point in which this update is performed. The back-end returns the journey identification to confirm that the update was done.

3. Finally, when no more beacons are being detected, the applications performs a “Check-out”, and sends the journey identification that is being finished, the user token to identify the passenger and the current location (latitude and longitude) where the check-out is done. The back-end calculates the price to pay for the travel and returns the new user balance, and a journey summary with the check-in and check-out dates, the cost of the finished trip, the bus number and the list of stations crossed within the travel.

It is essential for the back-end to have full knowledge about all the stations, buses and routes of the public transport operator. Regarding this issue, we implemented a transport route and stops database with information of several stations, buses and routes of Carris, our use case for this prototype. This database have three main entities: Stop, represented by its coordinates and name, which is identified by a unique code; Line, which has a unique name and indicates the correspondent bus, the destination station and direction; and fi-
nally LineStop, which connects the other two entities, indicating the stops that constitute a line.

6. Evaluation
The evaluation of this work focuses on measuring the times of the system critical tasks, such as entrance and exit detection times; the differences between the internal state transitions; and the visibility of the new state to the user. We found this values indicative of the system performance since it allows to verify some failure cases, related to the detection reliability. Notice that the values in this evaluation were obtained in a simulation environment and not in a real public transport vehicle. Therefore, the times collected are approximate to the real-time operation of the system.

We decided to perform tests in a room large enough to include the standard bus dimensions and a margin for the user to move inside/outside the beacon range. In this test case, the bus remains still and is the user who moves away from the "vehicle" (beacon range). We used the standard bus dimensions, and the beacons were placed with the same configuration as inside a standard bus. Three different users were invited to participate in the test set. The smartphone used to perform the tests was an LG L90 D405 (Android 5.0.2 Lollipop, API 21) with a 2540mA battery.

Users could perform travels in two modes: clicking in "check-in button" when in "Near" state; or simulating movement using a third-party application to simulate the navigation between two points on the map, and finally clicking on "check-in button" when our application detected movement. We used an app called "Mock Locations (fake GPS path)" developed by Dvaoru, to simulate the positions.

An auxiliary button ("Simulation button") in the application was created for the user to warn when left the room (passing through the door), and therefore move out from the beacon range, to calculate the delay compared to the time detected by the application. Each user was asked to simulate a journey, depending on the mode and according to the following steps:

- Not simulating movement (Experiment 1):
  1. Move away from the beacon range (wait for state "Out")
  2. Move inside the beacon range (wait for state "Near")
  3. Click on "check-in button" (wait for state "In")
  4. Move out from room and click the "simulation button" when passing the door
  5. Wait for the end of the journey (state "Checking-out") and confirm the summary shown or wait for the timeout.

- Simulating movement (Experiment 2):
  1. Move away from the beacon range (wait for state "Out")
  2. Move inside the beacon range (wait for state "Near")
  3. Open auxiliary app ("Mock Locations") and initiate stored simulation. Return to our application and click "simulation button" to indicate that the movement has initiated (wait for state "Checking-in")
  4. Click on "check-in button" (wait for state "In")
  5. Move out from room and click the "simulation button" when passing the door
  6. Wait for the end of the journey (state "Checking-out") and confirm the summary shown or wait for the timeout.

6.1. Obtained results
After setting up the test environment and providing the experiment instructions to users, 72 travels were simulated: 56 using the process of Experiment 1, in which is not detected movement before the check-in, and 16 following the instructions of Experiment 2, in which the movement is mocked using the auxiliary application. We decided to test this difference to verify if there is a delay between checking-in when entering the bus or checking-in when already inside the bus and moving, waiting for application check-in warning due to motion detection.

First, we calculated the average duration of every transition for each experiment and elaborated the graph shown in 7.

![Figure 7: State transitions graph with average durations.](image)

As we can observe, transitions "Out to Near", "Near to Checking-out" and "Checking-out to Out/Near" have similar values in both experiments. The difference between the two testing processes is at the check-in moment, so only the transitions that include the "Checking-in" state are affected. Therefore, as we can see, in Experiment 1, transition "Near to Checking-in" takes much less time than in Experiment 2. In contrast, transition "Checking-in to In" takes almost twice (180%) as long as in Experiment 2. This difference is due to the delay caused by the motion detection in Experiment 2.

The graph in Figure 8 specifically shows the time spent on each task of transition "Checking-in to In", 

8
we can see that in Experiment 1, the duration is due to the location delay. This value is the time spent between the last position saved and the moment when the check-in information is sent to the back-end. This delay may be caused by the interval between each collection being set for 4 seconds, which is a reasonable value that does not compromise the fare calculation.

In experiment 2, there is a movement detection delay of 05,313 seconds that affects the total transition duration. This delay causes an increase of 04,942 seconds in relation to the first experiment. Notice that the movement detection only verifies if it is necessary to warn the user that a check-in must be performed, so it is possible to check-in as soon as the user is near the vehicle (as in experiment 1). On the other hand, when checking-in, as soon as we obtain the location, the more accurate is the station selection, therefore the fare calculation. We found this value acceptable, as long as it provides a more precise calculation in exchange of an average delay of less than 5 seconds presenting the state to the user that is not forced to wait since it is possible to check-in when in "Near" state.

Communication delay is the difference between sending the check-in information and the response from the back-end. This value depends on the internet bandwidth, so the value is similar in both cases with an average value of 00,841 seconds.

Observing again Figure 7 we can conclude that the transition that takes most of the time is "In to Checking-out". We can observe each task duration of this transition in Figure 9.

![Figure 8: Checking-in to In times graph.](image)

6.2. Battery consumption

During the test battery, we used an application to monitor the battery consumption. Two hours of an intensive usage running on a smartphone with 2540 mA battery, resulted in a battery usage of 138,4 mAh, which represents 5.45%/h. During 4 hours detecting beacons but without performing any travel, we observed a battery usage of 12,5 mAh, which indicates less than 0.05%/h. These values indicate that the application has a low battery consumption, allowing an intensive usage of approximately 18 hours on a regular smartphone.

7. Conclusions

One of the main problems when using public transport is the time-consuming tasks of acquiring tickets, checking-in/out tickets or inspector validation. This led us to look for a solution that allows to reduce these waits, simplifying all the travel process. Also, the usage information of the public transport services is essential for the operator and nowadays is not easy to monitor their traffic based on the current statistics. Our solution suggests a global visualisation of the public transport service, providing information about the past and current travels.

After a deep analysis of the current market and the related projects being matured in this area, we identified some flaws that could be improved. Therefore, we found our opportunity to embrace the change and help provide a mobile ticketing solution, location-aware and based on BLE beacon technology. One of the main innovations of our approach...
is that the mobile application is responsible for coordinating the beacon detection and sending the travel information to the back-end, without needing any other device. It only needs beacons, which are cheap, have their own power supply (battery) and are very easy to deploy.

The work presented had to deal with mobile computing issues, such as memory requirement, battery consumption, interactivity, connectivity, and security. As we are faced with a mobile application for a smartphone, it is essential to ensure its interoperability, i.e. that the application is able to run on any smartphone. Although the prototype has been developed for Android, all of the functionalities used are also available on any other platform development kit.

To evaluate the performance of our system we measured the time consumed by each process performed by the app, to verify if the results were acceptable in this context. We found most of the values satisfactory, with acceptable delays that do not compromise the correct system behaviour. However, the checking-out duration caused by the beacon protocol monitoring should be reduced to detect with more accuracy the exit station of the travel.

The developed system allows to reduce user interaction when travelling inside public transport vehicles, providing a new and simpler travel experience using a daily device such as the smartphone. Besides that, operators can obtain real-time statistics of the system behaviour, allowing them to adapt their services to the user needs. Our solution meets the initially proposed goals, with some innovation and a progression margin for improvements.

8. Future Work
The present work consists of a prototype that is susceptible to improvement since there is still some issues that can be deeply developed in this context.

It would be profitable to provide an external API to access some services of the back-end, allowing users to manage their account, for example. This way, it would be possible to develop a portal in which users could check their trip history or charge their account. Also, it would allow to build applications to monitor operator resources (vehicles occupation, users currently travelling, etc.)

To confirm the benefits of the system, it would be better to perform more tests to validate the solution, especially to the location accuracy, battery consumption. But most importantly, the tests should be made in a real environment, inside a real vehicle with some users generating interferences, to give a better overview of system operation.

Although some of the leading security concerns being addressed in this work, like the crucial information decoupling from the mobile application, it is important to issues need more attention. In a real deployment of a system like the described, should be necessary to have more security concerns like the authenticity of users, preventing an intruder to impersonate other user and perform travels without paying, using the other user’s identification.

References