A new approach to the organization and tracking of Walking School Buses

Duarte Baptista
duarte.baptista@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

May 2017

Abstract

Over the years, public health problems such as obesity have taken epidemic proportions in developed countries, the lack of physical activity being one of the biggest causes.

Part of that lack of physical activity resides in the way children locomote in a daily basis. This is influenced by a growing tendency to develop car-oriented lifestyles, which leads to children usually being transported to school on their parents private cars. This tendency also leads to unseen levels of air pollution that exacerbate multiple health issues. Walking school bus (WSB) is a concept that tries to address those issues by pushing people to organize trips to school on foot.

Multiple applications and systems try to support the organization of WSBs, but they all seem to fail when it comes to one very important point: real-time tracking and monitoring of the bus.

This report presents an overview on existing related work about WSBs generic architecture, position and tracking technologies and approaches, privacy solutions and an analysis to existing applications and systems that support WSBs. It also presents our designed and implemented solution for a complete and efficient system that tackles the issues present in existing systems.

Keywords: Walking school bus, tracking, mobile, ubiquitous

1. Introduction

Over the years, public health problems such as obesity have taken epidemic proportions in developed countries, the lack of physical activity being one of the biggest causes [12].

Part of that lack of physical activity resides in the way children locomote in a daily basis. This behavior is choice limited as some parents may not allow children to travel unaccompanied due to insecurity in their respective neighborhoods or cultural reasons. Thus, with the growing tendency to develop car-oriented lifestyles, children generally rely on their parents to drive them to school on their private car [16].

That growing tendency is contributing to unseen levels of air pollution, which experts believe to be leading to increased risks for those with health problems such as cardio-vascular disease, asthma, chronic obstructive pulmonary disease, lung cancer, and diabetes [20]. Even more for children and elderly people, who have weaker immune systems [20]. There are also other issues associated with over use of cars such as emission of greenhouse gases [20], congestion near schools that leads to dangerous situations (e.g. parking in sidewalks and crosswalks) [14], lack of development of road safety skills and independence on children [1].

Walking-School-bus (WSB), a concept credited to David Engwicht in 1992 [13], is a form of school transportation that addresses those same issues. Its goal is to have children commute on foot, from home to school and vice-versa. This is supported by spreading Walking-bus stops throughout the neighborhood, and having volunteer adults pick up children at a specific time and route through those stops [17]. This concept can also adapted to cycling instead of walking.

There are multiple applications/solutions/systems available that try to support the organization of a Walking Bus, but those seem to commonly fail when it comes to one particular point: real-time tracking of the bus. None of them seems to present features such as:

- Providing real-time information to parents like estimated time until the WSB arrives to the get-in-point and timely warnings on relevant events, like cancellation of the bus next day
- Adults responsible for leading the WSB having the ability to know which children enrolled for the WSB and get-in-point for each child enrolled.

The goal of this thesis is to study the possibility, implement and evaluate a system to support walking school buses' organization fulfilling the requirements proposed above, while decreasing human intervention and increasing security levels. In order to achieve that purpose and help children gain conscience on both environmental and health issues, it is necessary to draw a solution built on a reliable, automated platform so that teachers, parents and volunteers could communicate and organize themselves. This approach presents many challenges that need to be discussed:

- Tracking - As we are dealing with young children it is unlikely that they will own a smartphone. Therefore it is not viable to track their absolute position. Instead, track the adult or adults responsible for the WSB. There are multiple technologies available capable of tracking users' positioning - Global Positioning System (GPS), Assisted GPS (A-GPS), WiFi networks based and cellular network positioning. Besides the need for the device to retrieve current position, it needs to send that information to a remote server in order for it to be available for other authorized users. This represents a major overhead in terms of bandwidth, energy, computation and memory that should be minimized [11].
- Privacy - The information that the users of the system must be able to retrieve, such as the WSB schedule and real-time positioning, is very sensitive. Thus, the need to discuss how to limit who can access that information is crucial.
- Ensure group cohesion - The system should be able to ensure that no child has inadvertently left the WSB without the adult responsible for the WSB noticing it.

The solution described in this document, which we designed and implemented, addresses the challenges referred to previously by:

- Reducing the overhead caused by the real-time tracking of the WSB by compressing the trajectory data using an approach based on the Dead Reckoning algorithm, effectively reducing mobile data, memory and battery consumption.
- Using the Facebook Social Graph and Firebase real-time database security rules to ensure that only people who are supposed to be able to access the data regarding a walking bus can actually do it.
- Using beacons to establish a sort of geofence and assure that no child gets lost from the group, thus ensuring group cohesion.

The remainder of the paper is organized as follows. Section 2 is a survey of the related work. Section 3 outlines our proposed approach. Section 4 presents how the proposed solution was implemented. Section 5 addresses the evaluation of the work done. Lastly, Section 6 presents conclusions drawn.

2. Background

This section describes existing work related to the generic architecture of a WSB, an analysis to the existing applications that support a WSB tracking and real-time trajectory tracking.

2.1. Generic architecture of a Walking Bus

Quoting Melrose et al. [17] defines the school walking bus as "a group of children that walks to and from school with parents or other adults, in which the children are picked up throughout the neighborhood." There are multiple mobile applications/systems that try to support and simplify the organization of such events. They all have a similar domain model, and functionally they work in a similar way: Firstly, a parent who wishes to lead a WSB logs in the platform and creates a route. In order to do that he/she inserts a starting location, time and to which school will be walking to. Another parent logs into the platform and is able to query the system for existing routes that have as destination his child's school. If any routes are available, the parent will be able to enroll his child on that walking bus. After enrolling, the parent will take his son/daughter to the designated stop, and waits until the WSB arrives. When the bus arrives at the designated stop, the child is handed to the WSB leader which is assigned the responsibility to take him to school. The parent who handed his child to the WSB is now able to proceed with his routine, and is able to watch WSB's position by consulting the information on the WSB through the application. When the WSB arrives to its final destination, the parent is notified by either the platform itself or cellular messaging.

2.2. Related systems'/applications' analysis

In summary all the existing solutions analysed (PetitBus, UDOT WSB app, Crocodile, goWSB) that support WSBs seem to commonly fail when it comes to particular one matter: real-time tracking of the bus. Although some of those systems claim to provide or that will be provided in future versions, none of them provide it adequately.

2.3. Real-time trajectory tracking

To support a WSB, besides the need for the user to retrieve and compute its position, there is also the need for the position to be sent to the system. This way the system can store an updated representation of the user's route taken and make possible for authorized users in the platform to have the ability to access information regarding the WSB in real-time, e.g. a parent who tries to retrieve the current position and route taken of a WSB in which his son is enrolled.

Tracking of users' position brings obstacles to the table, such as the following: the position of the mobile device is retrieved by itself and needs to be transferred to Moving Objects Databases (MOD), which "manage trajectory information"[11] of, in our case WSBs, through wireless networks. For systems that require timely and accurate dissemination of real-time trajectories, this can easily lead to a high communication and storage costs, besides the energy and computation cost for the device to retrieve and compute its position.

2.3.1 Trajectory compression

It would be ideal to record timestamped geographical points for a moving object with a high frequency. However this leads to high costs in terms of battery consumption, communication overhead and data storage [21]. [21] presents two categories of **trajectory compression** strategies to trade-off between those costs and precision needed. Those are:

- Offline compression or batch mode - This approach takes a fully generated trajectory and generates an approximate trajectory from that by discarding points with negligible error [21]. It is similar to the line simplification problem, which means reducing the number of points needed to represent a polyline [11]. Douglas-Peucker is a popular algorithm used to approximate the original trajectory [21]. An example is shown in Figure 1.a).
- Online compression - This approach compresses a trajectory instantly as the object travels [21] - To tackle the fact that many applications need a timely update of the object's position, two categories of online compression techniques were developed to decide if a gathered position is important and should be included in the trajectory[21]:
  - Window-based algorithms such as the Sliding Window algorithm [15] which keeps on increasing the window until the window exceeds a distance threshold, and do the same from that point, as shown in Figure 1 and the Open Window algorithm [18] which uses Douglas-Peucker to choose the point with maximum error in the window.
  - Algorithms based on the speed and direction of the object (e.g. use a concept of safe area to say if a point is redundant or not and can be discarded in the approximated trajectory).

Trajcevski et al. describes a velocity based trajectory compression algorithm named Dead Reckoning algorithm. [19]. This approach tries to reduce communication cost by reducing the amount of updates sent to the server by the client (moving object). The client starts by sending its sensed position, expected speed vector and timestamp, and from then on the server is able to draw an estimated trajectory of the moving object. Periodically the client senses its position, and checks if the distance between this position and the expected position that can be extrapolated from the last update sent to server is bigger than the threshold. If it is, the client updates the server of its current sensed position and expected velocity vector. The server keeps the actual speed, position time, and positions where update occurred. Figure 2 illustrates this approach.

3. Solution

This chapter aims to explain the solution designed (and implemented) in this thesis.

3.1. Main concepts

We consider three actors interacting in our system. These are Coordinator (the user that leads a WSB), Parent (the user that enrolls his/her child in a WSB) and Child (the child that attends the WSB).

It all starts when a parent thinks about taking his child to school on foot. Another parent, who was also thinking about taking his child to school through a similar route, could save his time and effort by sending his child to school with the other parent. For that to be achievable, the parent who is going to locomote to school with his child and is available to also take other children logs into our platform and creates a WSB route.

Other authorized parents (explained later in this chapter) may now query the system, find the route by inserting the range of desirable dates and selecting it on the screen. A map indicating the route and the corresponding distance will be presented. The user may straight away enroll his child in the route. After enrolling his son in the WSB, the parent may propose a waypoint to the existing route. If the deviation caused by this new waypoint is below the tolerance threshold defined by the parent that created the route, the waypoint is added to the route. Otherwise, the user is warned that the selected waypoint is not valid and the waypoint is discarded.

The user that enrolled his child in a WSB is

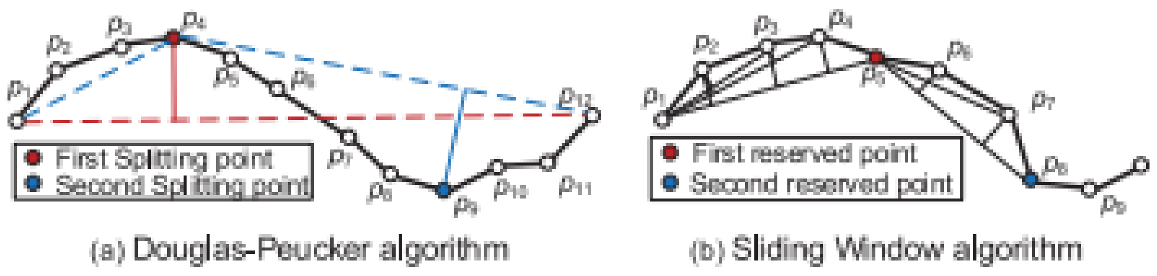

Figure 1: Figure showing an illustration of Douglas-Peucker algorithm (a) and Sliding Window Algorithm (b), taken from [21]

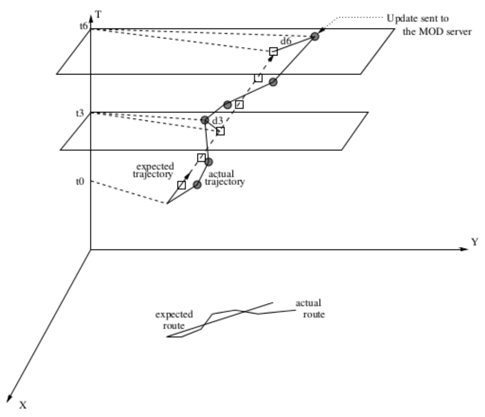

Figure 2: Dead reckoning approach. The dashed line corresponds to the velocity vector sent by the client, the squares on the dashed line are the expected position in six time instants (t1 to t6), the black color filled squares represent the position sensed by client. At each instant, the client measures the distance between the position sensed and the position that the server expects (extrapolated from the last update). From t1 to t5, the distance is smaller than the threshold, but on t6 the distance is bigger than the threshold, triggering an update to server. Taken from [19].

now able to find this route on the WSBs currently enrolled menu. As the name suggests, this menu lists the routes in which a user has enrolled his child. Selecting a route, the user is able to see information about the route: the route drawn on the map, start time, the last time which his son/daughter presence was sensed, and the status of the route (Not started, On route, Finished). If the WSB is on route the map will also show a marker that corresponds to a real-time estimate of the location of the WSB at the time.

When the coordinator is going to start the trip, he selects the route in the My WSBs menu. Firstly he hits the Start tracking button which will show him where he is and the distance to the starting location of the route. When he is 15 meters or nearer to that point, the Start trip button will become available. Start trip should be hit/pressed approximately when he starts to cover the path of the route. At this time, compressed location information is sent to the server so that it can be broadcasted to the other parents which enrolled their children in the route. This information will be updated when necessary (later explained in detail in this section). Throughout the path, enrolled children will join the route at different locations, and their presence will be acknowledged by the coordinators device (addressed later in this chapter, subsection Prevent children getting lost from the group). Throughout the journey, the platform informs the coordinator about which children have been detected to be with the WSB and the estimated distance from the coordinators device to them. As told above, parents which handed their children to the Walking School Bus will be able to see if the childrens attendance in the WSB has been acknowledged and at what time.

When the WSB is very close to the established destination, the coordinator is asked to confirm that the route is at its finish and the parents who enrolled their children in that route will be notified that the WSB has arrived to its destination, and at what time the children was last acknowledged by the system. This ends the lifecycle of a Walking School Bus route.

Figure 3 represents a graphical illustration of the interaction between the actors involved in the system.

3.2. Trajectory compression

To optimize energy and data consumption, instead of having the coordinators device sending raw positioning coordinates to the server we designed an algorithm based on the Dead Reckoning approach referenced in Section 2 to reduce the trajectory data that is sent. This way the server may show a real-time simulation of the trajectory without being constantly updated by the coordinators device.

Since the path that the WSB is going to take was defined beforehand, **both the coordinators de-**

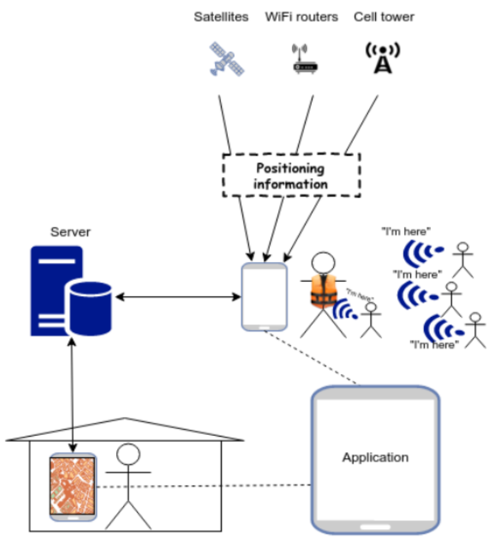

Figure 3: Graphical illustration of the interaction between the actors involved in a Walking school bus

vice and server know the route to be taken by the WSB. Taking this assumption into consideration we can simplify the approach referenced above and use the scalar speed instead of the vectorial because theres no need to transmit the direction, reducing the amount of data sent - one double (timestamp), and two floats (distance and scalar speed) instead of three doubles (timestamp, latitude, longitude) and two floats (x and y components of the speed vector). The algorithm consists in:

The coordinators device:

- After starting the route, upon receiving the first position update from the position providers, the coordinators device sends its current speed (scalar),position (not absolute, but the distance to the initial position in meters) and current timestamp to the server.
- Coordinators device periodically senses its position. As it knows the prediction the server has, when it senses that that prediction has an error bigger than a certain distance threshold, it updates the server with a new prediction (new current speed + new current position + new timestamp).

The clients device (the one that retrieves real-time information about the WSB from the server):

- Whenever the coordinators device sends a new prediction to the server, the server forwards that new prediction to the clients device. As this device knows the route that is being taken, and with the information that the server provided, it can present a simulation of the real-time trajectory of the coordinators device, in a dead reckoning fashion - the pin moves through the route drawn on the map at the speed provided by the server.

The algorithm's pseudocode is presented in Algorithm 1. We will now explain the algorithm, and the usage of the algorithm with the system.

```
Coordinator starts the WSB;
FirstPrediction = CurrentPrediction =
  Current speed, distanceCovered, Timestamp;
Send(FirstPrediction);
while not reached path destination do
    Predicted position = Extrapolated position
      from CurrentPrediction, Timestamp;
    if Distance between Predicted Position and
      CurrentPosition > errorBound then
        NewPrediction = Current speed,
          distanceCovered, Timestamp;
        Send(NewPrediction);
    else
        Do nothing, CurrentPrediction is still
          valid;
    end
end
```

Algorithm 1: Compression algorithm

3.3. Privacy

How to ensure that sensible information about the WSBs will not end up in the wrong hands? By using a social graph to define relationships between the users (in a Facebook-like fashion), like the ones presented in Figure 4.

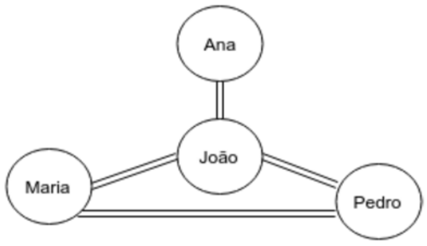

Figure 4: Example of a social graph - Pedro is a friend of Joao (directly related) and Joao is a friend of Maria.

The route creator/coordinator defines a privacy option - Who can see and enroll in a scheduled route:

- Friends - If this option is chosen, only users directly related to the coordinator in the social graph are able to see information and enroll in a WSB route. E.g. for the social graph presented in Figure 4 if Maria is the coordinator for a WSB route, only Joao and Pedro will be provided access to that route. Only users enrolled in a WSB route are able to consult real time information about the positioning of the WSB.

- Everyone - In case this option is chosen, there will be no restrictions to who can access, enroll and consult real time positioning information about that route.

3.4. Tolerating unexpected situations

Throughout the WSB route, if the coordinator decides to take a detour, for reasons such as roadworks taking place in the WSB route, the predictions computed by the coordinator's device will no longer be valid, as the predictions computed are only valid if the route is the one previously stipulated, and known by both parties (coordinator and client). In the case a detour is made relatively to the originally agreed route, the system system detects that situation and switches to a no-compression mode. This no-compression mode consists in the coordinator device stop computing and sending predictions, and start sending the absolute position coordinates (latitude, longitude, timestamp). The system considers that the WSB is out of route if the coordinators device senses three consecutive bad predictions. Although we propose this value for the amount of consecutive bad predictions to be considered a detour as it seems a reasonable number, it is customizable. This algorithm is shown in Figure 5.

3.5. Prevent children from getting lost from the group

There may be several children in a WSB but one or few adults coordinating it. How does the coordinator ensures group cohesion, I.e. no child is lost from the group. This is achieved by the use of beacons:

- Each parent owns a beacon.
- The child carries the beacon while walking with the WSB.
- The beacon periodically broadcasts packets. Throughout the journey the application estimates distance from the device to the beacon using the Received Signal Strength (RSSI).
- When the distance between the coordinators smartphone and the beacons is bigger than a certain threshold, the application warns the coordinator so that he could act on it.
- The device notifies the coordinator showing the distance to each beacon and corresponding timestamp (how much time ago the beacon was detected).

4. Implementation

This chapter will address matters such as choice of Operating System/Platform/Language, database server, and technologies used.

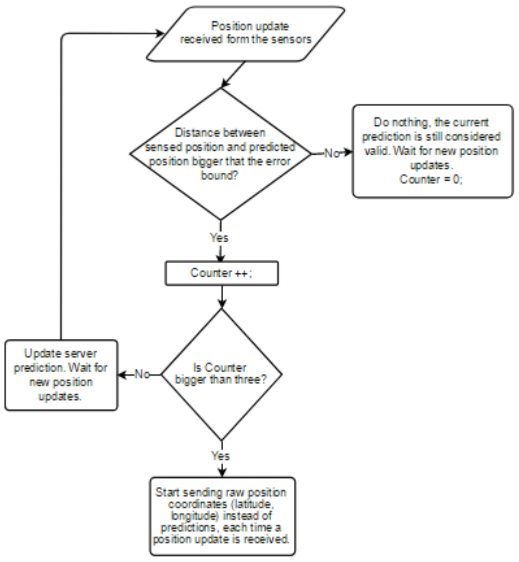
Position update received form the sensors
Distance between sensed position and predicted position bigger that the error bound?
No
Do nothing, the current prediction is still considered valid. Wait for new position updates.
Counter = 0;
Yes
Counter ++;
Is Counter bigger than three?
No
Update server prediction. Wait for new position updates.
Yes
Start sending raw position coordinates (latitude, longitude) instead of predictions, each time a position update is received.

Figure 5: Hybrid algorithm. Each time the coordinator's device receives a position update it checks if the distance between the sensed position and the prediction position is larger than the error bound. If not, does nothing. If so, sends a new prediction to the server, and it checks if it was the third consecutive update that lead to a new prediction. If so, it enters in what we name "no-compression mode" and starts sending raw position coordinates.

4.1. Operating System/Platform/Language

The first step to start the implementation of the system was choosing if we could use an already developed real-time tracking solution or if we had to develop a new one from the ground up, and the platform and language to program it. In order to facilitate that decision, we developed two sample real-time tracking applications and compared them to some existing tracking applications in terms of mobile data usage and accuracy. The applications analyzed are our own sample tracking Android application, our own sample tracking web application (AngularJS), Glympse, MapMyWalk, Strava.

In order to compare these applications, we have drawn a path through which we could walk multiple times with minimal detours. To achieve that we used sidewalk edges and crosswalk edges to define the path. That path was approximately 278 meters. The route, on purpose, was situated in an urban environment, surrounded by buildings.

The device used was a Samsung I9100 Galaxy SII smartphone (Android OS, v2.3.4, Dual-core 1.2 GHz Cortex-A9 CPU, 1GB RAM, A-GPS).

Table 1 shows the amount of data uploaded and downloaded, in Kilobytes.

We can conclude that our Android application spent much less than the other ones, especially the third party ones, without compromising its posi-

Application	Sent (KBytes)	Received (KBytes)	Total (KBytes)
Strava	105.8	652.7	758.5
MapMyWalk	43.4	223.2	266.6
Glympse	88.8	82.3	171.1
Our Android app	24.6	30	54.6
Our AngularJS 1 app	85.7	58.1	143.9

Table 1: Amount of data uploaded and downloaded by each application tested, in Kilobytes.

tioning accuracy. This might happen due to the fact that these other applications have unwanted features such as burnt calories meter and other statistics. Thus, it seemed better to develop a solution from scratch.

4.2. Database/Server

In this section, we address how the devices involved in the system interact with each other, and how the information is stored.

In order to choose the database that suits our system the best, it is fundamental to know what type of features and guarantees that need to be provided. We can define those as the following:

1. **Availability** - This means that the database is available and answering requests when asked [9].
2. **Response quickness over query complexity** - Our purpose is to provide realtime tracking. Thus the need to have to have quick response times, so that the predictions that are sent and retrieved do not become erratic. Even more, the nature of the system does not require highly complex queries.
3. **Security mechanisms** - Information stored and exchanged is quite sensible, thus the need for appropriate security mechanisms.

Firebase [8] by Google is a NoSQL database hosted on the cloud. It is known as a BAAS (backend as a service) and facilitates/speeds up the developing of an application by eliminating the need for server side code. Firebase realtime database offers features such as:

- Authorization with OAuth social providers such as Facebook and Google.
- Security rules to restrict who can read and/or write each data instance. Work as well for verification and indexing purposes.
- Data is stored in JSON. Ability to send raw Java classes (String, Long, Double, Boolean, Map<String, Object>, List<Object>) and custom Java classes.
- Capabilities such as CRUD (create, read, update, and delete) without the need to write server side code.
- APIs for several platforms such as iOS, Web, and Android. REST also available. This facilitates migrating an application to other platforms.
- Every time there is an update on the database, the connected devices receive that update within milliseconds. The Realtime Database API only receives requests that can quickly be solved, allowing to build a great realtime experience that can serve millions of users without compromising on responsiveness [8].

4.3. Feature implementation

4.3.1 Authorization

The user logs in the application with his/her Facebook account. In order to achieve that we use Facebook SDK for Android to retrieve the access token. Then that access token is used to gain access to Firebase by using the Firebase SDK for Android [2].

4.3.2 Privacy

As it has already been mentioned in this document, the information that is stored and exchanged in the system is highly sensible, therefore the need to control who is able to access it. To achieve that, we apply two mechanisms:

1. Facebook Graph API v.2.8 to implement the privacy option referenced in Section 3, section Privacy is accomplished.
2. Firebase security rules [5] to make sure that, for example only a user that is enrolled in a WSB may retrieve retrieve realtime information about that route, and that only the coordinator is able to write realtime information.

4.3.3 Realtime propagation of data

For users to receive updates about changes in the database in realtime, Firebase implements listeners with a callback mechanism to trigger when changes are done.

4.3.4 Ensure group cohesion

To ensure group cohesion as proposed in Section 3, subsection Ensure group cohesion we have chosen to use Estimote Proximity beacons.

Each child holds a beacon with a different ID. The server keeps records of which beacon corresponds to each user. Each of those IDs defines a region, and we range each of those regions. Throughout a WSB, anytime the coordinators device detects a beacon of a user enrolled in that WSB, the distance to the beacon is estimated from the RSSI and is shown on the coordinators application, along with the corresponding timestamp e.g., Users beacon was detected 3 seconds ago, 3 meters away. These beacons are not originally intended to measure the distance (due to the fact that RSSI readings vary on the environment, obstacles), but we are more interested in checking for the presence of the beacon near the coordinator's smartphone (if the beacon is in detection range), and not so much for having a very accurate estimate of the distance.

You might be asking yourself: Since the beacon broadcasts information and the child owns the same beacon all the time, someone could capture that information and use it in a malicious way, right? Well, to deal with that problem, we use the Secure UUID mechanism from Estimote. When in Secure UUID mode, a beacon broadcasts using an alias UUID that periodically changes. For the application to decipher the real UUID of the beacon, it must be authenticated to Estimote Cloud using the Estimote SDK [7]. This way, a possible malicious person will not be able to detect that that same beacon has passed through that place multiple times, thus not being able to establish a pattern and infer that a beacon belongs to one child.

4.3.5 Retrieve the walking path between origin and destination

Whenever a user creates a WSB route, or enrolls in one and proposes a new waypoint, the application needs to retrieve the walking path between the origin and the destination. This is done by requesting the Google Directions API [4]. This API supports waypoints and mode of travel (Walking, driving, etc.) as parameters and returns the route in JSON which is then parsed.

4.3.6 Graphical presentation and geometrical computations

We used Google Android Maps v2 [3] to present graphical representations of maps, routes, markers, and position estimates to the user. The class Spherical Util (com.google.maps.android.SphericalUtil) allowed us to do geometrical computations and develop functions necessary to compute predictions, extrapolate the estimated position form one prediciton, and checking if a prediction is wrong and consequently a new one needs to be sent to the system.

5. Evaluation

In this Section, we present the summary of some of the experiments made to evaluate the system, and correspondent methodologies. With these experiments, we try to answer the following questions:

- How accurate are the predictions that are sent to the server and how do different parameters given to the location listener of Android affect the amount of predictions sent, the accuracy of these predictions, and mobile data consumption?
- How does our solution compares with a few selected tracking applications in terms of data consumption?
- Does the application produce false positive detections of the beacon, i.e. is the beacon detected at a larger distance that its supposed range?

For these experiments, we used an LG Nexus 5x smartphone (Android OS v7.0, Snapdragon 808 CPU, 2 GB RAM, A-GPS and Bluetooth Low Energy support).

5.1. Data consumption and accuracy - No stops or detours

The first experiment consists in doing a WSB route and measure the impact on data consumption and accuracy of the predictions sent to the server, by varying the two following parameters:

- **Minimum time between positioning updates (minT)** - asking the location providers to return updates less frequently lowers energy consumption. However, at the cost of accuracy. [6].
- **Error bound (eb)** - The distance between the position update received from the providers and the position extrapolated from the current prediction that leads to a new prediction. Theoretically, the smaller the value, the more accurate the predictions, but more predictions sent to the server, thus more data used and more battery drained.

The results from the tests in 5.1. are summarized in Table 2. As expected, as the error bound allowed increased, the accuracy of the predicted position decreased. However, this did not lead to a

minT-eb	# Predictions sent	Average distance (m)	Data sent (KB)	Data received (KB)	Total data (KB)
5-20	2	8.337	5.6	9.5	15.1
10-20	3	7.868	5.5	9.5	15
5-30	3	12.978	8.6	10.7	19.3
10-30	2	11.764	9.5	11.3	20.8
5-40	2	17.306	6.0	10.4	16.4
10-40	2	15.427	5.6	9.7	15.3

Table 2: This table summarizes the results of the test 5.1 - Number of predictions sent, average distance between the position retrieved and the position given by the prediction at the time, data sent, data received, and the sum of the last two.

	Data sent (KB)	Data received (KB)	Total data usage (KB)
Strava	62.3	95.9	158.2
MapMyWalk	128.0	295.7	423.7
Glympse	63.7	73.6	157.3
Our solution without compression minT=5 minD=7	30.2	19.0	49.2
Our solution without compression minT=10 minD=14	13.2	12.9	26.1
Our solution with compression minT=5 eb=20	5.6	9.5	15.1
Our solution with compression minT=10 eb=20	5.5	9.5	15.0

Table 3: This table summarizes the data usage results for the analyzed applications.

smaller amount of predictions being sent. In our opinion, this might be happening due to inaccuracies retrieving the position and speed, and also due to the fact that was hard to keep a constant pace throughout the path. There was a slight increase in data consumption noticeable in the cases where eb is equal to 30 meters. We cannot find any logical for that rise, except for some possible overhead from the Firebase real-time database doing some synchronization operation. Analyzing the results, we can also notice that the sets of parameters that lead to the fewer bytes sent (minT=5, eb=20 and minT=10, eb=20), also presented a better accuracy. Therefore, we proceeded with the evaluation by using these two sets of values.

5.2. Comparison with the other tracking applications

In this test, we compare our solution in terms of data usage with other selected tracking applications, as well as with a adapted version of our solution configured to work in no-compression mode only. Although no-compression is applied, we introduced a new customizable parameter to reduce the number of staypoints: minD, which is the minimum distance estimated between position updates. This means that the Android location listener only informs of a position update if the new position update distances the previous by minD or more.

We do not apply this policy with the compression because if, for some reason, the WSB is still in a position for a very large period of time, no position update will be received from the position provider. The predictions are verified whenever a new position update is received, therefore if no position update is received the prediction cannot be verified, and the estimate the client has would still be considered valid and moving. To compute the minD values we used the average walking speed in group presented in [10] (4.22 feet per second) and multiplied it by the minT. Strava, MapMyWalk and Glympse versions used were the ones available in the Google App store in April, 2017.

The table 3 summarizes the data usage results for the selected applications.

As expected, both cases using our solution without compression consumed more mobile data than both cases using our solution with compression. All third party application tested used the mobile data connection more than our solutions (both compression and no compression). We think that this increase can be explained by at least two factors:

1. The fact that these applications preform operations other than the tracking itself, especially Strava and MapMyWalk which throughout the path record statistics related to health, consuming more data.

2. We tried to reduce as much as possible the overhead introduced by moving the camera (map perspective shown to the user) and consequently having to download other map parts. We acknowledged that this had a big effect on data consumption during the first tests to our application.

6. Conclusions

In this thesis, we approached matters related to the organization and tracking of Walking School Buses, a concept which tries to change the way children locomote to school on a daily basis.

We designed and implemented an Android application that tries to address the issues found in the other apps with the same goal while fulfilling the essential nonfunctional requirements stated above.

In order to achieve energy and mobile data consumption efficiency, we implemented an algorithm to compress the trajectory, based on the distance dead reckoning approach. To tackle privacy concerns, i.e. ensure that the information is only accessible to whom is authorized, we used two mechanisms: Firebase security real-time database security rules and the Facebook social graph API. In addition, to ensure that the children in the WSB stay within a safe area close to the WSB coordinator we used Estimote Proximity beacons.

References

[1] Backseat Children. (May), 2008.

[2] Add Firebase to Your Android Project . https://firebase.google.com/docs/android/setup, 2016. [Online].

[3] Developers - Documentation - Google Maps Android API. https://developers.google.com/maps/documentation/android-api/, 2016. [Online].

[4] Developers - Documentation - Google Maps Directions API. https://developers.google.com/maps/documentation/directions/, 2016. [Online].

[5] Understand Firebase Realtime Database Rules . https://firebase.google.com/docs/database/security/, 2016. [Online].

[6] Android location Strategies . https://developer.android.com/guide/topics-/location/strategies.html, 2016. [Online].

[7] Estimote Android SDK . https://github.com/Estimote/Android-SDK, 2016. [Online].

[8] Documentation - Firebase Realtime Database. https://firebase.google.com/docs/database/, 2017. [Online; accessed 16-April-2017].

[9] D. Bartholomew. Sql vs. nosql. *Linux J.*, 2010(195), July 2010.

[10] N. Carey. DRAFT RESULTS Establishing Pedestrian Walking Speeds. (503), 2005.

[11] R. L. F. D, K. Rothermel, and K. Rothermel. Efficient Real-Time Trajectory Tracking. *The VLDB Journal, Volume 20, Number 5*, 20(5):641–642, 2011.

[12] M. Dehghan, N. Akhtar-Danesh, and A. T. Merchant. Childhood obesity, prevalence and prevention. *Nutrition journal*, 4(Table 1):24, 2005.

[13] D. Engwicht. *Reclaiming our cities and towns: better living with less traffic.* New Society Publishing, 1993.

[14] I. Issues and D. Policies. Strategies to Improve Traffic Operations and Safety.

[15] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. *Proceedings 2001 IEEE International Conference on Data Mining*, pages 289–296, 2001.

[16] R. L. Mackett. Children's travel behaviour and its health implications. *Transport Policy*, 26:66–72, 2013.

[17] J. A. Mendoza, K. Watson, T. A. Chen, T. Baranowski, T. A. Nicklas, D. K. Uscanga, and M. J. Hanfling. Impact of a pilot walking school bus intervention on children's pedestrian safety behaviors: A pilot study. *Health and Place*, 18(1):24–30, 2012.

[18] N. Meratnia, R. A. D. By, and R. a. de By. Spatiotemporal Compression Techniques. *Proceedings of the 9th International Conference on Extending Database Technology (EDBT '04)*, pages 765–782, 2004.

[19] G. Trajcevski, H. Cao, P. Scheuermanny, O. Wolfsonz, and D. Vaccaro. On-line data reduction and the quality of history in moving objects databases. *ACM international Workshop on Data Engineering for Wireless and Mobile Access*, (January 2016):19–26, 2006.

[20] J. Wargo, L. Wargo, and N. Alderman. The Harmful Effects of Vehicle Exhaust. page 64, 2006.

[21] Y. U. Zheng. Trajectory Data Mining : An Overview. *ACM Trans. Intell. Syst. Technol.*, 6(3):1–41, 2015.