Time of Arrival Prediction with Dynamic Route Tracking for Public Transportation Systems

Vitor Borges C. da Silva, Tatiana Sciammarella, Miguel Elias M. Campista, and Luís Henrique M. K. Costa
Federal University of Rio de Janeiro - PEE/COPPE/GTA - DEL/POLI
E-mail:{borges,tatiana,miguel,luish}@gta.ufrj.br

***Abstract*—A great challenge in developing countries is the constant traffic jams in urban areas, which are a consequence of excessive use of private vehicles. In these countries, more people would adopt public transportation, e.g., buses, if the time of arrival for the next vehicle to each bus stop could be predicted. In this direction, we propose WiBus, which is a system to estimate buses arrival time, based on information from opportunistic IEEE 802.11 contacts. These estimates are provided to users via a website and a graphical interface for mobile devices. Unlike previous systems, WiBus explicitly takes into account bus route changes, which occasionally happens in developing countries , adjusting the routes with an algorithm for dynamic route creation and maintenance. WiBus is implemented and analyzed via emulation of a real scenario. Experimental results show that WiBus is scalable and meets the needs of large cities with accumulated error of at most a few minutes.**

I. Introduction

Today, big cities in developing countries suffer from constant traffic jams. In Brazil, in just one year, from December 2013 to 2014, more than 4 million new vehicles started circulating on the streets, causing a significant deterioration of the traffic system [1]. In these countries, the preference for private transportation is a consequence of the lack of confidence on the public alternative, be it for safety, comfort or commitment with timetables. To change this trend, the idea of the Advanced Public Transportation Systems (APTS) has emerged using, among other technologies, communication networks to provide more information to users [2].

This work aims at improving the quality of the service provided by public transportation systems at developing countries. The idea is to offer fresh information regarding the next bus arrival time of every bus line to each possible bus stop. Nevertheless, even considering the Vehicle-to-Infrastructure (V2I) architecture, estimating the arrival time of a bus at a given bus stop is a challenge, because traffic and passengers can influence the time a bus takes on its route (e.g., when an accident results in a traffic jam). Moreover, situations such as an accident, a large event (e.g., World Cup, Olympic Games, etc.) or a roadwork may result in temporary changes in a bus route. If these changes are ignored, the estimates become useless. Therefore, two requirements to calculate the arrival times without significant errors are to track buses position whenever possible, which most systems do; and to be aware of buses route changes in a dynamic fashion, which is a contribution of our work. To deal with the first requirement, many solutions use GPS, due to its greater accuracy. Nevertheless, the GPS utilization requires an additional communication link to inform the vehicle position. As an alternative, there are localization techniques that use network technologies [3], which can simultaneously discover the bus position, signalize it and provide communication services. This type of solution is used in this paper.

Network localization has been explored even in simple proximity-based location systems, where the position of a vehicle is given by the position of the node the vehicle is connected to. Although the precision is smaller than using GPS, techniques to improve the performance of these systems are evolving, providing mean square error of only a few meters [4]. Regarding estimates of buses arrival time, some works use long term historical series collected even for months [5]. With such data, estimates are calculated based on averages from the same period of time in the past. This method does not behave well in case of atypical situations, such as accidents or eventual congestion. To better model these situations, another historical-based solution exists, the real-time approach. In this approach, only short-term information is used, e.g. the information from the last four hours. This method models unforeseen situations more efficiently, but assumes that the time spent by previous vehicles in a given route is equivalent to the time spent by next vehicles [6]. There are also works that use artificial neural networks to predict buses arrival time [7]. Finally, there are some methods that use Kalman filters, which do not perform well when location data are temporally sparse [8].

This paper proposes WiBus, a system to estimate the arrival time of buses using IEEE 802.11 networks. The design goals of WiBus are low cost and low complexity, combining the advantages of previous proposals. To this end, WiBus tracks the position of buses based on their proximity to IEEE 802.11 access points installed alongside streets. This approach allows the utilization of only one type of device to track and communicate, reducing the system cost and complexity. In this work, we use a real-time method to estimate arrival times since localization information is not obtained at the same rate as with GPS and because we do not assume the existence of a database with long-term historical information. Then, based on short-term past information, WiBus computes the arrival time of buses and dynamically adapts these times to route changes. This yields that even if a bus deviates from its route, the system can adapt, keeping users informed. Experiments

978-1-4673-7306-7/15/$31.00

with real datasets show that WiBus is capable of meeting the demands of a big city such as Rio de Janeiro, incurring in errors on the order of a few minutes. In addition, to promote public access to the system, an application for Android smartphones, called `BuZoom`, and a web page, `BuZoom Web`, are developed in this work. Our experimental results demonstrate the applicability of WiBus in real environments.

This paper is organized as follows. Section II and III present, respectively, the architecture and the implementation structure of WiBus. Section IV further details the algorithms used to predict arrival times and track route changes, while Section V shows the user interface developed. In Section VI, the evaluation setup is described and the corresponding results are shown. Finally, Section VII concludes this work and presents future directions.

II. WiBus Architecture

WiBus (Figure 1) is composed of four entities: Central, Roadside Unit (RSU), Bus, and Client. The Central plays the most important role in the system. It stores the information needed to compute arrival time estimates of buses, their route, location, and the time spent between consecutive stops. The Central also answers to customers' arrival time requests. Roadside Units (RSUs) are ordinary IEEE 802.11 access points installed at bus stops. They provide wireless access to each router within each bus and run the developed programs for bus tracking. These developed programs locate the bus along its route and send the localization information to the Central through the RSU. A Client is any person requesting estimates to the system. These requests can be made either through the Android `BuZoom` application or through the website.

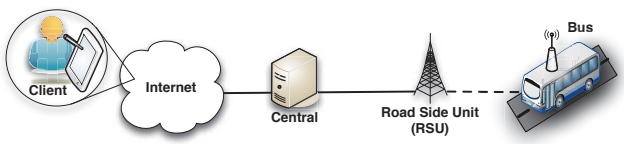


Figure 1. Entities of the WiBus system architecture.

WiBus services: WiBus offers complementary services of localization and arrival time of buses at all stops along their routes. Both services are based on proximity to connected RSUs. Therefore, upon arriving at a bus stop, the router within the bus connects to the network provided by the RSU. Messages are then exchanged between the bus router and the RSU for mutual identification. Then, the bus router sends a message to the Central entity informing the RSU it is connected to. This allows the system to track the bus position. Although this information is fundamental, it is not enough for arrival time computation. The bus route and a short-term history of the times spent by buses between consecutive stops are still needed to estimate arrival times. Although the trivial approach to have the bus route would be to assume that they do not change, we consider the possibility of changes, since in developing countries roadwork and construction sites can happen more often.

III. WiBus Implementation

WiBus is implemented as a set of six programs. Four of them, listed below, are used for data exchange required for buses arrival time estimation. Their names indicate their role in the localization process. The remaining two programs are client interfaces, which are presented later:

- `WiBus - Localization Client`: This program runs every time the bus connects to an RSU, sending a localization request and receiving its answer. The message sent by the bus contains the bus and bus line IDs and allows RSUs to log buses passing by. This message can be seen in Listing 1.

```
<WIBUSLocRequest>
<BusLineID>
Bus line ID, e.g. 455.
</BusLineID>
<BusID>
Bus ID, e.g. BUS125.
</BusID>
</WIBUSLocRequest>
```

Listing 1. Localization request message.

- `WiBus - Localization Server`: This program continuously runs in RSUs waiting messages from buses. Upon a bus message arrival, the RSU sends its geographical position to the corresponding bus. This message allows the bus to localize itself. This message can be seen in Listing 2.

```
<WIBUSLocRSU>
<RSUPosition>
Latitude;Longitude
</RSUPosition>
</WIBUSLocRSU>
```

Listing 2. RSU localization message.

- `WiBus - Communication Client`: This program is used by buses to communicate their current and previous localization to the Central. It runs after the end of `WiBus - Localization Client` and can also receive replies from the Central. The message produced can be seen in Listing 3.

```
<WIBUSLocBus>
<BusLineID>
Bus line ID, e.g. 455.
</BusLineID>
<BusID>
Bus ID, e.g. BUS125.
</BusID>
<PrevRSUPosition>
Latitude;Longitude
</PrevRSUPosition>
<CurrRSUPosition>
Latitude;Longitude
</CurrRSUPosition>
</WIBUSLocBus>
```

Listing 3. Bus localization message.

- `WiBus - Communication Server`: This program communicates with buses receiving their localization. From messages received, the Central can dynamically update the bus line route, be aware of buses location, and estimate times of arrival at every bus stop. This program also responds to clients' requests for bus information.

The localization process is illustrated in Figure 2. When a bus connects to an RSU, it sends a localization request message with the `WiBus - Localization Client` program (Figure 2(a)). In the following, the RSU replies the message received from the bus with its own position using the `WiBus - Localization Server` program (Figure 2(b)). Finally, the bus sends its localization message to the Central with the `WiBus - Communication Client` program (Figure 2(c)). The bus localization message contains information about the RSU where the bus is connected, the previous RSU it has been connected, and the bus and bus line identifiers. Upon receiving this message, the `WiBus - Communication Server` program handles the information. Besides the four programs, there are also two others operating as Client interfaces:

- `BuZoom`: An interface for Clients with Android smartphones request buses arrival times to the Central entity.
- `BuZoom Web`: A platform-independent web interface for Clients to also request buses arrival times.

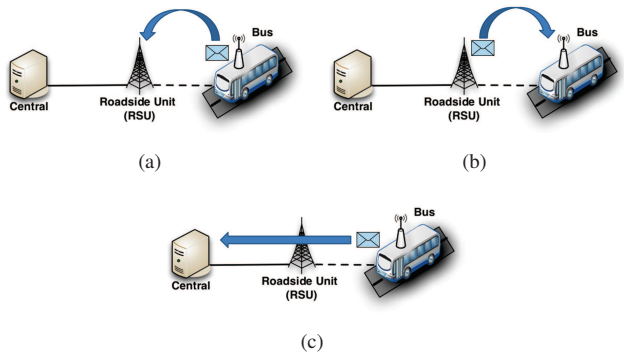


Figure 2. Localization process: (a) Bus requests RSU position. (b) RSU replies the bus with its position. (c) Bus sends localization message to the Central.

In the next section, we detail the Central operation.

IV. Central Entity Operation

A. Bus Line Route Model

In WiBus, a bus line is defined by its number and direction (e.g., 913 South). As a consequence, bus routes are not circular for the system. Additionally, each bus route consists of a list of segments, where each segment starts in an RSU and terminates in the consecutive one. Since we consider that each bus stop has an RSU installed, a segment also starts in a bus stop and ends in the consecutive bus stop of a bus line. Note that consecutive RSUs or bus stops can be known according to the movement of the buses. It is worth noting that the bus departure from its initial bus stop is considered known, as announced by bus companies.

A bus line route is modeled as a sequence of RSUs, where each RSU has directed edges to the next and previous RSUs of the route. To prevent premature changes to bus line routes, a weighted strategy is employed: non-negative weights are assigned to edges and also to RSUs. Weights assigned to edges govern the bus line route dynamics, capturing how fast a change in the route will be considered permanent. On the other hand, weights assigned to RSUs indicate if they are still used in the bus line route. In Figure 3 we model a fictitious bus line route from RSU-1 to RSU-5 to show these concepts. Circles represent RSUs and rectangles represent RSU weights. Arrows represent the directed edges linking one RSU to its successor or predecessor in the bus line route, where the number near the edge is its corresponding weight.

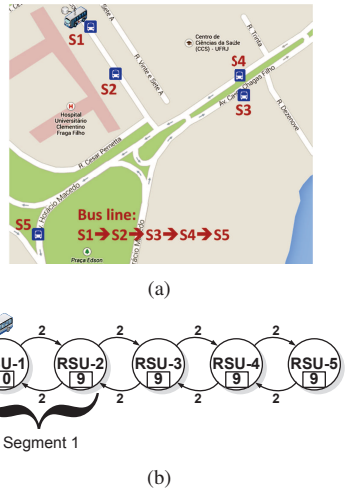


Figure 3. (a) Fictitious bus line route. (b) Bus line route model.

B. Arrival Time Computation

The notion of segment is important since at the end of each one, the bus must inform the Central its position with a bus localization message. This information is used to compute the time spent between consecutive RSUs, which is calculated by the Central subtracting the time of delivery of the current position and previous position messages. Then, when the Central receives a message from a bus, it updates the current and previous bus positions. Also, it updates the average time needed to traverse that segment. With the average value of the next segments and the bus departure time, the system can estimate the time required for a bus to reach the following RSUs on its line. This is done by summing up the average time needed by buses to traverse each of the segments from the current RSU until the target RSU. The problem, therefore, can be reduced to a problem of estimating the time required for a bus to traverse a given segment. This can be calculated as the average time spent by the K previous buses at the same segment, i.e. a moving average with window size K as maintained by the Central. The use of a simple moving average provides reduced complexity in arrival time estimation.

Upon receiving a message from a bus, the Central entity re-estimates the time needed for every bus on the same line to reach all the next RSUs in their route. This is done by getting the buses current RSU and summing up the estimates of the segments from the current RSU to the last one in the bus line. To show only the estimate to the next bus arrival time at each RSU, only the lowest estimate time is stored. This process is formally described in Algorithm 1, where $estimate_b$ stores the estimated time needed for a bus b from a given bus line l to reach an RSU from its initial position. Note that this algorithm is used to possibly update the time needed to reach all the RSUs of the same bus line. In addition, Algorithm 1 uses the function `segmentEstimate`. This function implements the aforementioned moving average to estimate the time needed for a bus to traverse one segment. Next, we detail the update and maintenance algorithms of the bus lines routes.

Algorithm 1: Estimated arrival time for the next bus to reach each RSU in the bus line.

Input: list of segments S of the bus line l.
Output: time estimate for the next bus to reach each RSU in the bus line.

```
forall the b of bus line l do
    estimate_b = 0; segment = current segment from S where b is;
    while segment exists do
        estimate_b += segmentEstimate( segment);
        if estimate_b < current RSU estimate then
            current RSU estimate = estimate_b
        end
        segment = next segment in S;
    end
end
```

C. Route Update Algorithm

The route update algorithm is responsible for updating RSUs weights. The basic idea is to increment the weight of RSUs currently used by buses along a bus line. In opposition, RSUs currently not used have their weight decremented. The update process of RSU weights consists of decrementing by one the weight of all RSUs in the bus line. Next, the weight of the current RSU, the one informed in the message received from the bus, is incremented by an integer (specified later).

The update algorithm can culminate on RSU creation or deletion. RSUs are deleted if their weights reach zero, meaning that the RSU is no longer part of the bus line. In case of RSU creation, the assigned weight must be large enough so that the RSU will not be mistakenly removed. So, the initial value assigned to an RSU is considered equal to twice the product of the number of RSUs in the bus line by the amount of bus circulating on that bus line. This value is also the maximum value allowed to the RSU weight. The initial value of the RSU weight depends on the number of RSUs in the bus line, every time an RSU is added, to be fair, the weights of all RSUs in the line are set to the new maximum value. Similarly, the increment given when an RSU is used is equal to twice the number of RSUs in the bus line. Hence, the initial weight assigned to an RSU guarantees that all buses of a given line must travel over a route without a given RSU at least twice, in order to delete this RSU.

For instance, consider a bus doing the route in Figure 3(a) leaves S1 and goes to S2. When arriving at S2, its router exchanges messages with the RSU-2 at S2. After this, the bus informs the Central its position. On receiving the bus position, the Central updates the weights of the RSUs decrementing all of them and incrementing the weight of RSU-2, where the bus is. In addition, the Central checks that the bus is coming from the RSU-1 and is at RSU-2, exactly following the bus line route expected by the Central. The updated route at the Central can be seen in Figure 4. Next, we describe the route maintenance algorithm, responsible for dealing with route changes.

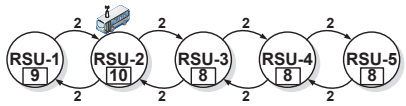


Figure 4. Node weight updates after a bus movement to the following RSU.

D. Route Maintenance Algorithm

The need for dynamic representation of bus routes comes from situations where unforeseen events change the sequence of segments usually followed by buses. Such changes can be permanent or temporary. One way of checking if the bus route has really changed is by observing other buses from the same line. The change can be confirmed if they continuously repeat the same new route. The number of times a change must be repeated until it is considered valid is captured by the WiBus route maintenance algorithm. The algorithm aims at avoiding errors on arrival time estimates either because of premature conclusions or lack of responsiveness to bus route changes.

Whereas the weights of the RSUs are updated every time a new message arrives to the Central, the weights of the edges are modified only when a bus line route changes. During a route change, an RSU may be connected to more than one next or previous RSU. In such cases, an RSU has a list of edges indicating the next or previous RSU, rather than just a single edge. Thus, the update of the edges to the previous or next RSU is performed as follows: the weight of all edges from the list of previous and next RSUs is decremented by one, and two units are added to the weight of the edges used, i.e., previous and next.

During the route maintenance, edges can be created or deleted. Edges are deleted if their weights reach zero, and are created if buses inform a new segment in a bus line route. In the case of edge creation, the initial value of the edge weights can be adjusted, taking into account that its value must be at most two times the number of buses in the same bus line, thus it is possible to define how fast a change in the route will be considered permanent. The initial value chosen for the edge weight is also the maximum value allowed.

For ease of understanding we present an example of a route change. Suppose there is a bus line in which buses normally

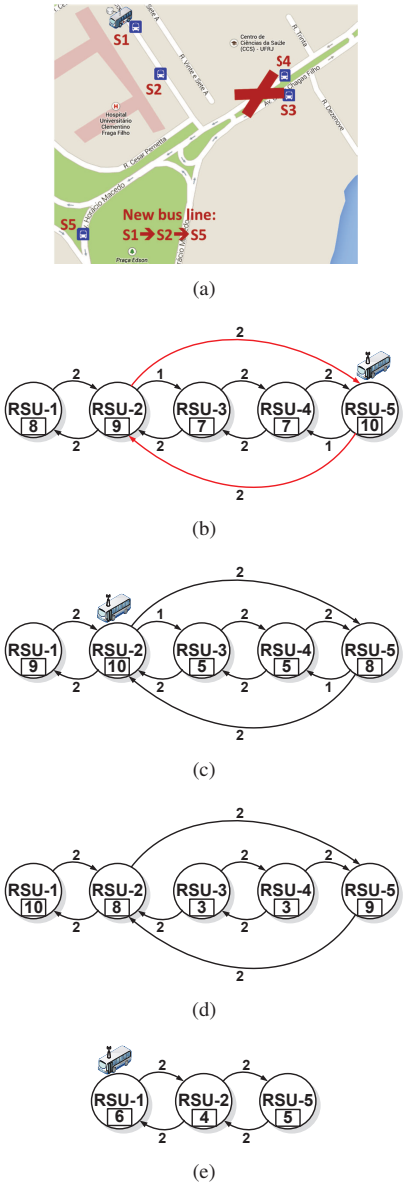


Figure 5. (a) Bus line route with changes due to roadwork. (b) The first bus going from RSU-2 directly to RSU-5 signs a route change. Hence, edges connecting RSU-2 and RSU-5 are created. (c) Nodes weight update after another bus from the same line moves to RSU-2. (d) Route after edges removal. RSU-3 and RSU-4 are no longer reachable from RSU-1, confirming the bus route change. (e) Final bus line route, after unused RSUs removal.

goes from the bus stop S1 to the bus stop S5 passing by S2, S3, and S4, as illustrated in Figure 3(a). Due to a roadwork, the streets connecting S2 to S3 or S4 are now blocked, forcing all the buses to change their route as shown in Figure 5(a). The new route skips the bus stops S3 and S4 because they became unreachable. Since the Central uses information from buses, the route is initially outdated before receiving any updates.

Consider the first bus to do the route after the road block is already at RSU-2 as in Figure 4. In the original route, the next bus position would be RSU-3. However, the Central will hear again from the bus only at RSU-5, when the bus informs its current and previous position. After updating the weights of the RSUs, the Central compares the current and the previous bus positions to the registered bus route. According to the registered route, a bus arriving at RSU-5 must come from RSU-4. Nevertheless, as informed by the bus, the previous stop was at RSU-2. Thereby, the Central registers this new possibility adding an edge from RSU-2 to RSU-5 and vice versa. After the edges creation, the old edges that represent the next RSU of RSU-2 (RSU-2, RSU-3) and previous RSU of RSU-5 (RSU-5, RSU-4) have their weight decremented. Now, from the route registered a bus coming from RSU-2, has two possible next RSUs, as well as RSU-5 has two possible previous RSUs. In addition, one can infer by the weight of the outgoing edges of RSU-2 that the most recent route avoids RSU-3 and RSU-4. This is indicated by the highest weight of the edge connecting RSU-2 to RSU-5. Then, the route registered at the Central is illustrated in Figure 5(b).

Consider another bus leaving S1 to S2. After messages exchange, the Central updates the weights of the RSUs. The weight changes can be seen in Figure 5(c). Note that the weights of RSU-3 and RSU-4 are almost half the initial weight because buses have stopped passing by.

When the bus approaches RSU-5, the weight update process is triggered again. Afterwards, the Central verifies again that the previous RSU as informed by the bus is RSU-2, instead of RSU-4. As a consequence, the next RSU is again RSU-5 instead of RSU-3. Then, the weights of the edges maintaining such information are incremented, becoming equal to the initial and maximum value allowed. On the other hand, the weights of the other edges representing the next RSU of RSU-2 and the previous RSU of RSU-5 are decreased. Upon reaching zero, these edges are removed from the bus line route representation in the Central, as observed in Figure 5(d). The removal of the edges confirms the route change. This can be seen by checking Figure 5(d) that despite RSU-3 and RSU-4 still exist in the route representation, they are not reachable by following any edges on the route beginning at RSU-1. We can note that because the maximum and initial weight of the edges is equal to two in the example, two buses are needed to confirm the route change. Nevertheless, as mentioned earlier, one can change this parameter to determine how responsive the algorithm is to route changes.

With the confirmation of the route, no other change happens on the edges between RSUs in the route. Yet, with buses continuing to follow the route, RSU-3 and RSU-4 are removed from the bus route and the maximum weight allowed to the RSUs decreases. The new route is depicted in Figure 5(e).

Since the main purpose of WiBus is to provide estimates of buses arrival times at bus stops, it is worth mentioning that during a route change process, when many routes are possible, the estimates are calculated using the old route until the new route is confirmed. Therefore, even if a bus driver follows the wrong route, estimates will be calculated correctly. On the other hand, if a route change is really in progress, the estimates will only take the new route into account after it has been confirmed. Hence, one must consider the tradeoff between

avoiding errors on arrival time estimates either because of premature conclusions (weight values too low) or excessive delay for bus route adaptation (weight values too high) when choosing the edges maximum and initial weight values.

Algorithm 2 details the combination of the route update and maintenance algorithms implemented at the Central.

Algorithm 2: Bus line route update and maintenance algorithms.

Input: message m received from bus b of bus line l.
Output: updated bus line route

```
if first message or previous message received then
    search l described in m;
    if found l then
        obtain RSU_current from m and search it in l;
        if found RSU_current then
            Update edges to previous RSUs of RSU_current;
            Update weight of RSUs in l;
            Update edges to next RSUs of RSU_previous;
            Exclude RSUs no longer in l;
        else
            Create RSU_current from m and insert it in l;
            RSUs weight in l = weight_maximum;
            Create edges for RSU_previous and RSU_current;
        end
    else
        Create RSU_current from m;
        Create l from m with RSU_current;
    end
    if b bus line changed compared with the message before m then
        Add 1 bus to l and subtract 1 from previous bus line;
    end
    if b seen by the first time then
        Add 1 to number of buses in bus line l;
    end
end
```

V. Graphical Interface

Users make requests to WiBus using a graphical user interface. The first interface created was a web page, `BuZoom Web`. Nevertheless, assuming that most users would like to access the system via a customized interface for smartphones, we have also implemented the `BuZoom` application, shown in Figure 6. This interface is available for devices with Android 2.2 operating system or above. The tools used in the `BuZoom` development are the Android SDK and JDK. A QR code recognition is also implemented.

The use of QR code gives an alternative for users to obtain information about a given bus stop. Using a QR code, a user can read the code available at the bus stop to identify his/her location and to obtain buses arrival times. This is useful, in case users do not know exactly where they are. Besides the utilization of the QR code, the user can also use the `BuZoom` interface using a typical application menu. In this case, the user must choose the bus stop from a dropdown list and inform the bus line of interest through another list. The first alternative, using QR code, is more suitable for mobile users who are already at the bus stop, whereas the second alternative can be preferable for users who are not yet at the bus stop. To send the request to the Central and to receive information back, the

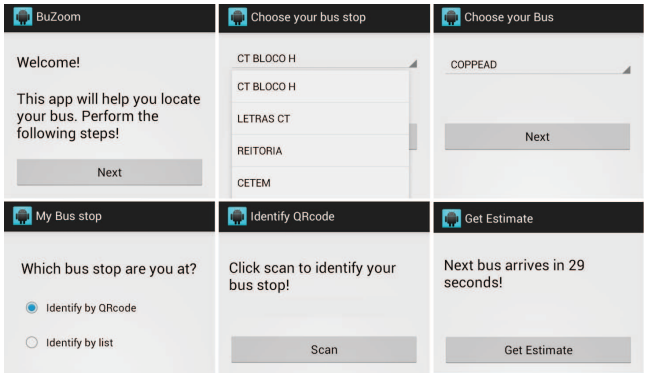


Figure 6. Screenshots of `BuZoom` for Android.

user needs an Internet connection. Then, the response from the Central is displayed on the device screen.

VI. Experimental Setup and Evaluation

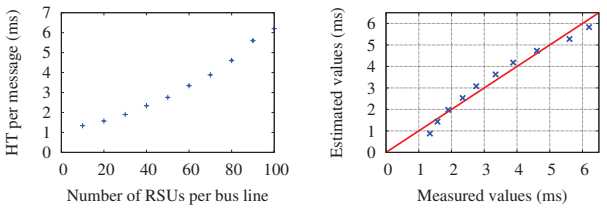
WiBus is evaluated using a prototype assembled in our lab. We have emulated the elements of the architecture using PCs and Wi-Fi routers. The Central entity is a PC configured with Debian operating system, 8 GB of RAM, and Intel Core i7 860 processor. The RSUs and the buses are represented by D-Link DIR-320 Wi-Fi routers. We connect the Central to the RSUs via an Ethernet switch. The D-Link DIR-320 is equipped with a 240 MHz ARM processor and 32 MB of RAM. It also has a USB port, which is used to increase the storage capacity, originally limited to 4 MB flash. The operating system used in routers is OpenWRT Backfire 10.03.1, a Linux distribution for embedded devices.

In our experiments, the initial weight of edges is set to twice the number of buses in the bus line (Section IV-C). The first experiment evaluates the capacity of WiBus, i.e., measures the maximum number of buses the system is able to track, and also tests its scalability with respect to the number of buses and number of RSUs per bus line. The second experiment, which is performed with real data of a university scenario, aims at evaluating WiBus estimation errors.

A. WiBus Service Capacity

The capacity test emulates the conditions of the city of Rio de Janeiro, which currently has around 8,000 buses [9]. We assume, for evaluation purposes, that these buses are evenly distributed over 800 fictitious bus lines. In our test, we evaluate the performance of a single Central upon message reception from a bus. We have written a script to emulate the buses connecting to RSUs and sending messages to the Central. As the number of bus stops can be different for each bus line, we have also varied the number of RSUs from 10 to 100. After the creation of the routes of the fictitious bus lines, one of all the registered buses runs through its bus line 10 times, sending messages to the Central whenever it changes of RSU. The Central treats these messages and measures the time spent during the process, the message Handling Time (HT). The same procedure is repeated three times for each different

number of RSUs in a bus line. The HT per message results are shown in Figure 7(a) with 95% confidence intervals.

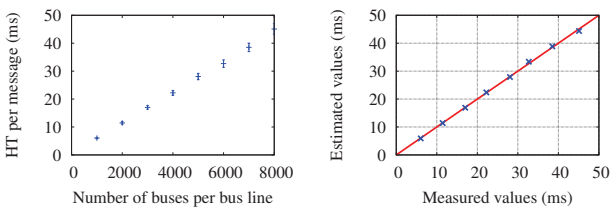


(a) Handling time (HT) per message. (b) Linear regression residual plot.

Figure 7. Rio de Janeiro emulated scenario.

From Figure 7(a), we can observe that WiBus would be able to monitor and estimate the entire city of Rio de Janeiro with just one server similar to the one used in the experiment. This inference is made by observing that the HT of a single message is less than 6.5 ms and assuming that buses take, in average, more than a minute to reach the next RSU on its route. With this assumption, the system has at least 60 seconds to handle 8,000 messages. Nevertheless, dividing the 60 seconds by the worst case 6.5 ms, we obtain a rate of over 9,000 messages per minute. This amount is higher than the number of buses, meaning that the Central entity is able to meet these demands.

To assess WiBus scalability with respect to the number of RSUs per bus line, we modeled the HT per message obtained in the Rio de Janeiro scenario through a linear regression. To evaluate the accuracy of this model we use a residual plot along with the R^2 metric [10]. In the residual plot, measured and estimated values are plotted and a unit slope is also shown indicating a perfect model. The linear regression model, which achieves a R^2 value of 0.969, together with the residual plot shown in Figure 7(b) are very close to the reference slope, suggesting that the experimental HT per message fits well a linear function. Hence, we can conclude that WiBus scales with respect to the number of RSUs per bus line.



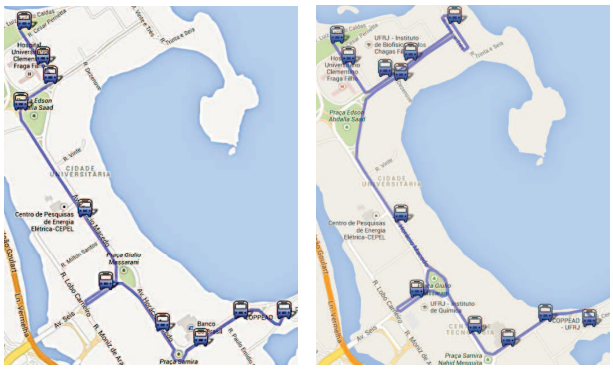
(a) Handling time (HT) per message. (b) Linear regression residual plot.

Figure 8. Number of buses per bus line scalability test.

To show that WiBus scales with respect to the number of buses per bus line, we create a new scenario where we varied the number of buses in the system from 1000 to 8000. All the buses were registered in a single bus line with a route with 5 RSUs. Then, one of the buses follows the route 10 times and, as a consequence, the HT per message is calculated. The whole process is repeated three times for each different number of buses in a bus line. The HT per message results are shown in Figure 8(a). Note that the HT values obtained in this test are greater than the obtained in the Rio de Janeiro scenario because in this scenario all the buses were registered in the same bus line, which does not happen in the real world. Again, we model the HT per message, resulting in a linear regression. This shows that WiBus scales in respect to the number of buses per bus line. The linear regression model achieves R^2 value of 0.999, together with the residual plot shown in Figure 8(b) almost overlapping the reference slope suggest that the experimental HT per message is well fitted with a linear function. Thus, we can infer that WiBus scales with respect to the number of buses per bus line, even with unrealistic numbers of buses in the same bus line. The next experiment evaluates the quality of the arrival time estimates.

B. Experiments in a University Scenario

For this test, we measured nine times the time spent on all segments of the UFRJ's internal bus lines "COPPEAD" and "Estação UFRJ", as in Figure 9. These data are used to evaluate the quality of WiBus estimates, which calculate arrival times using moving averages of size K. In our tests, the parameter K is varied from 1 to 8. Figure 10 plots the average absolute error for each segment concerning buses arrival time estimates for both lines. We show results for K equal to 1, 4, 5 and 8, for the sake of clarity.

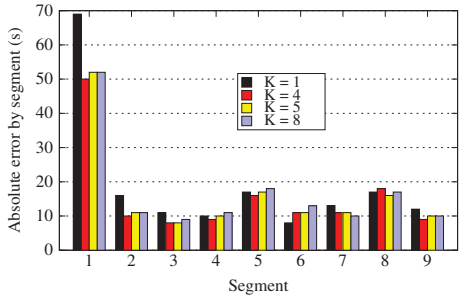


(a) "COPPEAD" Bus Line. (b) "Estação UFRJ" Bus Line.

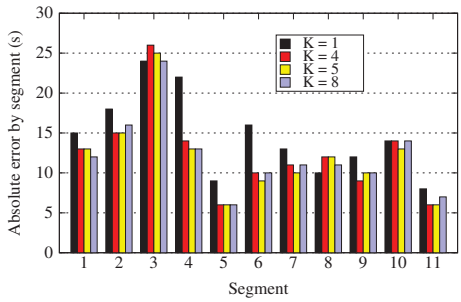
Figure 9. University bus line routes.

The value of K is chosen to minimize the average error of the obtained estimates from the first until the last bus stop of the evaluated lines. The values of K are those that produce the lowest absolute error according to Figure 11, which are $K = 4$ for "COPPEAD" and $K = 3$ for "Estação UFRJ".

Picking different values of K for each bus line is justified by the fact that different lines have different behaviors. The characteristics per segment of the two bus lines are summarized in Tables I and II. We can observe that some segments have a high coefficient of variation (CV), defined as the standard deviation (σ) divided by the mean. The high coefficient of variation of these segments is a consequence of traffic lights or pedestrian crossings, which can produce time variation. Also, from the Tables I and II, one could calculate the average coefficient of variation for the two lines. The line "COPPEAD"

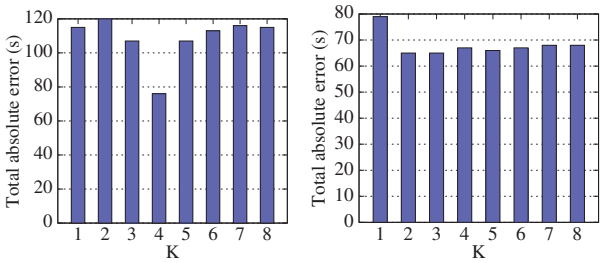


(a) "COPPEAD" Bus Line.



(b) "Estação UFRJ" Bus Line.

Figure 10. Average absolute errors for different values of K.



(a) "COPPEAD" Bus Line. (b) "Estação UFRJ" Bus Line.

Figure 11. Average estimate error from the first bus stop to the last one for different values of K.

has an average coefficient of 0.21, whereas the line "Estação UFRJ" has 0.18. These values help understanding the plots in Figure 11. Bus lines with larger variations, such as "COPPEAD", tend to show a greater tradeoff between recent and long-term information. In opposition, for bus lines with minor variations, like "Estação UFRJ", this behavior is less clear.

Finally, from Figure 11, we verify that even for the entire bus route, the average absolute error of the estimates remains below 80s.

VII. Conclusions and Future Work

This paper proposed a system to estimate the time of arrival of buses at bus stops. WiBus uses information obtained from an IEEE 802.11 network with devices installed inside buses and on infrastructure along their routes. The idea is to track buses positions in order to estimate their arrival times. To avoid system failures in case of unforeseen route changes,

#	Mean (s)	σ (s)	CV
1	116.44	52.04	0.45
2	42.67	12.43	0.29
3	44.33	9.90	0.22
4	107.56	11.75	0.11
5	79.67	17.26	0.22
6	140	14.73	0.11
7	85.33	10.83	0.13
8	63.67	16.59	0.26
9	61.22	9.00	0.15

Table I
"COPPEAD" bus line.

#	Mean (s)	σ (s)	CV
1	58.56	13.62	0.23
2	42.11	15.31	0.36
3	85.44	24.69	0.29
4	122.11	15.14	0.12
5	173.89	5.92	0.03
6	128	9.75	0.08
7	127.56	10.88	0.09
8	80.44	11.64	0.14
9	27.33	8.94	0.33
10	97.11	13.56	0.14
11	61.56	7.63	0.12

Table II
"Estação UFRJ" bus line .

an algorithm for dynamic creation and maintenance of digital routes was proposed. The arrival time estimate is based on a simple moving average of size K, where examples of K were calculated with real data for two university bus lines. Results have shown that the estimates made by WiBus present errors on the order of a few minutes, for the entire route followed by the bus lines. Also, we have shown that WiBus is scalable and is able to meet the demands of a big city like Rio de Janeiro, being able to handle more than 9,000 messages per minute, in the worst case. WiBus implementation provides two user interfaces, a web page and an Android app called `BuZoom`.

As future work, we plan to extend our measurements with more real data regarding the time spent to traverse route segments. With more information, it would be possible to evaluate more precisely the choice of the parameter K, and possibly allowing the creation of a dynamic algorithm for K adjustment as a function of the estimate error.

Acknowledgement

This work was funded by CNPq, CAPES, and FAPERJ.

References

[1] DENATRAN, www.denatran.gov.br/frota.htm (in Portuguese), 2014.
[2] R. F. Casey, L. N. Labell, S. P. Prensky, and C. L. Schweiger, "Advanced public transportation systems: the state of the art," U.S. DOT, Tech. Rep., 1991.
[3] J. Ribeiro Junior, M. Mitre Campista, and L. Costa, "COTraMS: A collaborative and opportunistic traffic monitoring system," *IEEE Transactions on ITS*, vol. 15, no. 3, pp. 949–958, June 2014.
[4] M. Caceres, F. Sottile, and M. Spirito, "WLAN-based real time vehicle locating system," in *IEEE VTC*, Apr. 2009, pp. 1–5.
[5] K. Manolis and D. Kwstis, "Intelligent transportation systems - travelers' information systems the case of a medium size city," in *IEEE ICM*, Jun. 2004, pp. 200–204.
[6] W.-H. Lin and J. Zeng, "Experimental study of real-time bus arrival time prediction with GPS data," *Transportation Research Record*, vol. 1666, no. 1, pp. 101–109, 1999.
[7] S. Chien, Y. Ding, and C. Wei, "Dynamic bus arrival time prediction with artificial neural networks," *J. Transp. Eng.*, vol. 128, no. 5, pp. 429–438, 2002.
[8] A. Karbassi and M. Barth, "Vehicle route prediction and time of arrival estimation techniques for improved transportation system management," in *IEEE Intelligent Vehicles Symposium*, Jun. 2003, pp. 511–516.
[9] Instituto Municipal de Urbanismo Pereira Passos, http://www.armazemdedados.rio.rj.gov.br/ (in Portuguese), 2011.
[10] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.