An Algorithm for Sink Positioning in Bus-assisted Smart City Sensing

Pedro Cruz[a], Rodrigo S. Couto[b], Luís Henrique M. K. Costa[a]

[a] *Universidade Federal do Rio de Janeiro; GTA/PEE/COPPE*
[b] *Universidade do Estado do Rio de Janeiro; PEL/DETEL/FEN*

Abstract

Smart Cities use data obtained from different sensors to offer better services. Such data is usually sent to sinks, using the paradigm of Internet of Things. However, covering the whole city area with static sensors might be expensive. This paper addresses the utilization of the public transportation system as a mobility platform for sensor nodes and bus stops acting as sinks. This platform can improve the coverage of a sensor network and take advantage of opportunistic communication. Since this platform can be used for delay-constrained applications, we propose an algorithm to choose sensor sinks in the bus stops of a smart city that minimizes the maximum delay when delivering messages. We compare our solution with an optimal formulation and use real data, obtained in buses from Rio de Janeiro, as an input of our algorithm. Our experiments show that we can use approximately 16% of bus stops as sinks, having less than 10% of increase in the network maximum delay and no significant loss in the spatial coverage[1].

Keywords: Internet of Things, Wireless Sensor Networks, Smart Cities, Resource Positioning.

Email addresses: cruz@gta.ufrj.br (Pedro Cruz), rodrigo.couto@uerj.br (Rodrigo S. Couto), luish@gta.ufrj.br (Luís Henrique M. K. Costa)

[1]The final publication is available at Elsevier via https://doi.org/10.1016/j.future.2017.09.018

1. Introduction

Projections show that human population should reach 8.5 billion people by 2030, mainly in urban areas[2], posing many challenges to the cities and directly affecting the well-being of citizens. Smart Cities are an umbrella of different technologies to answer to those challenges. A recurrent strategy is to gather information about the city and intelligently use it to improve the services offered to the citizens, or create new services [1]. There are possible applications in weather, surveillance, pollution monitoring, and others [2]. Naturally, gathering data about a whole city is a huge challenge, and an infrastructure based on the Internet of Things (IoT) [3] is an important tool. The IoT paradigm is often denoted as the enhancement of daily-life objects with sensing, actuation, and communication capabilities [4]. In many scenarios, wireless sensor networks (WSN) are part of the IoT infrastructure [5]. WSNs are usually composed of many sensing nodes that sense data about the environment and send this data to one or more sinks. The sink, then, sends data to a so-called task manager node, responsible for storing, processing and serving data to the users [6].

Including mobile nodes in the wireless sensing network is a way to enlarge the sensed area [7]. In the context of a city, it is an alternative to having a (potentially expensive) fixed sensor network covering the whole city. In that scenario, mobile sensors opportunistically deliver data to gateways or to sinks [8]. In this paper, we depart from the observation that different metropolises count on bus lines on their public transportation system. We therefore build upon the idea of having buses as mobile sensing nodes. In the considered application scenario, buses are elements of the IoT infrastructure. This setup enables applications in the smart city dimensions of smart mobility, smart environment, and smart living [9]. Then, the problem of where and when buses will deliver data to the Internet arises. We assume that there will be sinks located at bus stops to receive the data and deliver it to the place in the cloud where the data will be concentrated, and decisions made based on the information gathered, such as taking actions over the city infrastructure. Another advantage of this sensing bus network is that bus mobility is predictable; buses are already part of the public infrastructure, meaning that the mobile nodes come for free, and the additional cost of having a bus carrying a sensor is small.

[2] http://www.un.org/en/development/desa/news/population/2015-report.html.

One of the issues with the opportunistic communication performed by sensor nodes embedded in buses is the delay to deliver data. Sensors have to wait until the bus is close to a sink in order to deliver the gathered data. On the other hand, the type of application defines the freshness required for the information to be useful [10]. Therefore, in this paper we analyze the communication latency experienced in a citywide sensing bus network. We consider a mobile WSN where sensor nodes are on board buses and sinks are located on a subset of the bus stops. Thus, we model the delay when the number of sinks is constrained, as an Integer Linear Programming (ILP) problem, and also propose an approximate polynomial algorithm to locate sinks at some of the bus stops. The polynomial algorithm is executed using as input real data of all the buses and bus stops of Rio de Janeiro. Our results show that installing sinks in only 16% of bus stops can increase network maximum delay in less than 10% with no loss to the area covered.

The paper is organized as follows. Section 2 presents related work and positions our proposal within the literature of the field. Section 3 models the application scenario of our sensing bus network. In Section 4, we model the problem of positioning sinks in the network as an ILP problem. In Section 5 we propose an approximate algorithm to select bus stops to install sinks, minimizing the network maximum delay and compare its performance with the optimal solution. In Section 6 we run the algorithm for a real dataset. Section 7 concludes the work and points out future directions for our work.

2. Related Work

There is a rich literature on sink positioning for static WSNs, with focus on different quality of service metrics. Wong *et al.* propose an algorithm to decide a location for the sinks on a WSN that achieves the lowest latency possible [11]. The work shows that an optimal solution is not always feasible because of its time complexity and proposes an approximation algorithm, capable of obtaining a satisfactory solution in a feasible time. Since the time to store and forward data is usually higher than data propagation time over links, Wong *et al.* uses number of hops as a metric to minimize. Our work differs from [11] since we consider dynamic WSNs, where buses provide mobility to nodes, reducing the costs of deployment, but keeping the covered area.

Gathering data for smart cities with urban vehicles is also a studied topic. Opensense project uses embedded sensors in public transportation vehicles to

monitor pollution in urban areas [12]. This project employs a GSM network to deliver gathered data and does not employ opportunistic communication. Other similar project is Mosaic [13], that measures pollution through embedded sensors in public transportation vehicles. Data is delivered to access points that work as sinks, using WiFi, GPRS or Bluetooth. In Opensense and Mosaic, the focus is data processing in order to achieve accuracy in the presence of mobility. Ours work focuses on the sink infrastructure, by choosing sink locations that minimizes data delay.

The BusNet [14] project uses public transport buses to obtain data about pollution and road condition. The work does not take into account the delay in data delivery. Therefore, data gathered is limited to applications that are tolerant to indefinite delays. In contrast, our work focuses on delay-constrained sensing applications.

Umer *et al.* [15] proposes a routing protocol that employs clustering to select sinks on a static WSN deployed in a hospital. This network is delay sensitive and thus critical information employs dedicated paths. The proposed protocol saves energy and improves flexibility, fault-tolerance, and reliability in the considered network. Using Hilbert curves, Ghafoor *et al.* [16] define trajectories for mobile sinks on static WSNs. Their work increases the order of the Hilbert curve in areas where node density is higher. These strategies were designed to operate on networks with static sensors and their use with mobile sensor nodes, such as considered in our scenario, is not completely investigated.

Dias *et al.* [17] analyze the throughput of an opportunistic network composed of public buses and show that this network is capable of transmitting enough data for many applications, including sensing. Our work considers a similar network, proposing an algorithm to allocate its sinks and analyzing its behavior to delimit the applications that can use buses as a mobility platform.

3. Network and Delay Model

This paper considers a sensing bus network: a mobile wireless sensor network where sensing nodes are embedded into buses and sink nodes are installed on bus stops. In the studied scenario, illustrated in Fig. 1, the buses carry nodes capable of sensing environmental data. This data is stored in the buses and transmitted to a sink node through a wireless interface. The sink nodes, located at bus stops, receive data and send them to the

Task Manager node through the Internet. The sink nodes might be capable of executing some pre-processing on data, like filtering, aggregating, or taking local decisions, on a paradigm called fog computing [18].

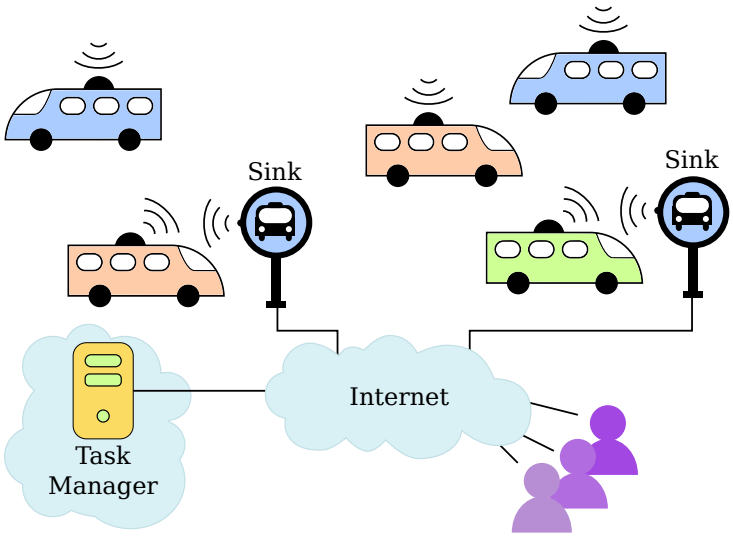


Figure 1: Overview of the bus-based information gathering and distribution system.

The sensing node architecture is illustrated in Fig. 2 and was initially presented in [19]. The sensing node follows the Smart Object requirements presented in [20]: low energy consumption, small physical size and low cost. In this architecture, the controller manages all the other modules. Its memory, energy and communication tasks can be managed by an operating system, such as TinyOS [21]. The sensor bank gathers raw data, which is read by the controller. After reading the raw data, the controller reads the time and position from the Global Positioning System (GPS) receiver module and appends this information to the sensed raw data, building a tuple. The tuple is stored into the memory unit until the wireless interface is able to connect to a sink. When the wireless interface connects to a sink, the wireless interface communicates with the controller and the controller unloads the memory unit contents to the wireless interface. The wireless interface then sends the data to a sink, ending the sensing cycle. The sink delivers the data to the Task Manager node, and the data is finally available to the users. In terms of power supply, the sensing node can drain power from the bus, use solar power or heat harvesting systems, or other energy resources [22]. This architecture is compatible with the one presented by Akyildiz in [6]. Nevertheless, it is

worth noting that the mobilizer role of the architecture in [6] is performed by the bus carrying the node.

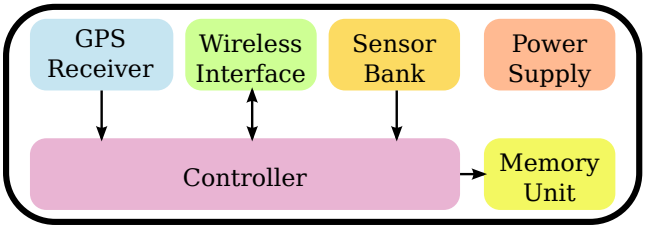


Figure 2: The node-level architecture of the proposed sensing bus system.

In our scenario, the sensing node and the sink are one hop apart. Additionally, we assume that a single encounter is enough to deliver all data gathered between the present encounter and the immediately previous encounter. This assumption is better analyzed in Section 3.3. Regarding the wireless technology, we can employ IEEE 802.11p, IEEE 802.11n, LoRa or other wireless technologies. Each technology has different range and data rates, producing different communication coverage and throughput.

When data is gathered, the sensing node waits until a connection with a sink is established. Next, the sensing node sends the data to the sink. Finally, the sink sends this data to the Task Manager node through the Internet. The delay from the data being gathered until it is available to the users is the time the sensing node waits for a connection with a sink plus the time the sink takes to send it to the Task Manager. The maximum time that the sensing nodes waits for a connection is the time a bus takes to travel from a bus stop vicinity to the next bus stop vicinity, in a case where both bus stops have sinks installed. Typically, the time for a bus to travel from one bus stop vicinity to the vicinity of another bus stop is of some minutes magnitude, and the time to send data between nodes, using wireless interfaces, is of some milliseconds. In this sense, we expect that the time to send the data is negligible when compared to the time waiting a connection. Therefore, this paper models the delay as the time between contacts, neglecting the time to send the data from the sensing node until the Task Manager node.

3.1. Delays on a constrained number of sinks

This work assumes that, given a certain budget, the number of chosen sinks is smaller than the number of bus stops. Therefore, the problem is to decide which bus stops should work as sinks. This section models the

envisioned network and the delay added to data sensed by this network. Table 1 summarizes the notations used in this article.

Let $b \in \mathcal{B}$ be a bus equipped with a sensing node, $s \in \mathcal{S}$ a bus stop that is a sink candidate, S_b an ordered list in which every element $S_b(i)$ is the i-th ($1 \leq i \leq m$) stop s that b makes contact with, and $T_b(i)$ the instant when the bus b makes contact with the i-th stop in S_b. It is possible to see the sequence S_b as the path of b through the stops with which b makes contact. When a bus $b \in \mathcal{B}$ gathers data throughout its path in S_b, such data gets delayed when the bus does not contact some sink. Data gathered right after b loses contact with $S_b(1)$ is delayed by $T_b(2) - T_b(1)$. Every other piece of data gathered by b on the way between $S_b(1)$ and $S_b(2)$ has smaller delay, until b makes contact with $S_b(2)$. The same can be applied to any pair $(S_b(i), S_b(i+1))$ that happens in the path S_b. For the sake of simplicity, the expression $(T_b(i+1) - T_b(i))$ is defined as the delay between $S_b(i)$ and $S_b(i+1)$.

It is possible to define D_b as the sequence of delays for a bus b as follows:

$$D_b = \{T_b(2) - T_b(1), T_b(3) - T_b(2), ..., T_b(m) - T_b(m-1)\}. \quad (1)$$

Note that, in Equation 1, the element $D_b(i)$ is the time a bus takes to go from the stop $S_b(i)$ to the stop $S_b(i+1)$. Given that a given bus b contacts m_b bus stops, $S_b(m_b)$ is the last element in the bus path, an thus element $D_b(m_b)$ cannot be defined. Thus, we define the maximum delay of network, D_{max}, as the highest delay of every delay sequence from the buses:

$$D_{max} = \max_{b \in \mathcal{B}} \left(\max_{i \in \{2,...,(m_b-1)\}} (D_b(i)) \right). \quad (2)$$

3.2. Sensor node memory requirements

The presented model assumes that data is gathered by a bus b after the contact with a sink on its path $S_b(i)$ and is completely delivered on the contact with the next sink, $S_b(i+1)$. Throughout the travel time, represented by $D_b(i)$, data is incrementally stored in the memory module. Assuming that the memory module is empty when the bus leaves $S_b(i)$, the total data stored in the memory module when the bus arrives in $S_b(i+1)$ is:

$$M_g = gen_{rate} \times D_b(i), \quad (3)$$

where gen_{rate} is the data generation rate, i.e., the amount of data generated by the sensor bank and GPS module by unit of time. The memory module

Table 1: Notations.

Notation	Description	Type
$\mathcal{B}$	Buses moving around the city	Set
$\mathcal{S}$	Bus Stops that are sink candidates	Set
$\mathcal{S}_b^p$	Sink candidates that make contact with bus b before some other candidate	Set
$\mathcal{S}_b^a$	Sink candidates that make contact with bus b after some other candidate	Set
$\mathcal{S}_{bs}^p$	Sink candidates that make contact with bus b before candidate s	Set
$\mathcal{S}_{bs}^a$	Sink candidates that make contact with bus b after candidate s	Set
$\mathcal{I}$	Bus stops that are the starting or final stop of some bus	Set
S_b	Sequence of bus stops that make contact with bus b, ordered by contact time	Parameter
$S_b(i)$	Function that returns the i-th element of the sequence S_b	Parameter
$T_b(i)$	Time when bus b makes contact to the i-th element of the sequence S_b	Parameter
D_b	Sequence of time intervals between contacts of bus b and the bus stops in S_b	Parameter
$D_b(i)$	The time between the contact of b with $S_b(i)$ and the contact with $S_b(i+1)$	Parameter
M_g	Data gathered between the contact of a bus with two consecutive bus stops	Parameter
M_t	Data transferred by a bus on a single contact with a bus stops	Parameter
tx_{rate}	The transmission rate between a bus and a bus stop	Parameter
c_{time}	The contact time between a bus and a bus stop	Parameter
d_{bpq}	Total time, for a bus b, to make contact with bus stop p and then with bus stop q	Parameter
N_{budget}	The amount of sinks allowed by the budget	Parameter
D_{max}	Maximum possible delay between any two sinks	Variable
x_s	Boolean that indicates whether s is chosen as a sink	Variable
y_{bsq}	Boolean that indicates, for a bus b, whether q is the next sink when departing from s	Variable

must be capable of storing the maximum amount of data expected for it. Since we assume a constant gen_{rate}, the maximum amount of data is achieved when $D_b(i)$ is maximum. In Section 3.1, the maximum $D_b(i)$ was defined as D_{max}. In Section 6.1 we show that, for a network using the buses of the city of Rio de Janeiro, it is reasonable to use the value of 2 h for D_{max}. Additionally, using the results from Sinaeepourfard *et al.* [23], we estimate that the average amount of data generated by typical smart city sensors in 2 h is 717 B. Therefore, the amount of data to be stored in the memory module is 717 B for each sensor in the sensor bank.

3.3. The assumption of full unload on a single contact

In this paper, we assume that sensing nodes are able to completely deliver the data gathered between stops $S_b(i)$ and $S_b(i+1)$ when a contact with the sink in $S_b(i+1)$ occurs. We refer to this as the assumption of full unload on a single contact. Given that M_t is the amount of data transferred from the bus to the sink at the bus stop $S_b(i+1)$, our assumption holds if:

$$M_g = M_t. \tag{4}$$

Inside the sensing node, when the wireless interface makes contact to a sink, data is transferred first from the memory unit to the controller, than from the controller to the wireless interface and, finally, from the wireless interface to the sink. Each one of these transmissions might have different transmission rates, and the minimum of these rates limits the transmission as a whole. Therefore, we define tx_{rate} as the minimum of these transmission rates. The transmission between the sink and the sensor node is only possible while there is contact between them. The time of contact, denoted by c_{time} is the time in which the bus is connected to a sink. The time c_{time} is a function of the bus speed, route and the transmission range between a sink and a sensing node. In order for the assumption of full unload on a single contact to be valid, the tx_{rate} must be big enough to unload, during c_{time}, all the data generated by the sensor bank during $D_b(i)$. This holds true if the inequality on Inequation 5 is satisfied, with M_g as defined in Section 3.2:

$$tx_{rate} \times c_{time} \geq M_g. \tag{5}$$

To estimate the tx_{rate} and c_{time} of a worst case scenario, we used measurements and estimations from different works, respecting the worst case

scenarios when applicable. The work of Borges *et al.* [24] measured the contact time between a bus and a bus stop, using IEEE 802.11. Borges concluded that, on average, a bus makes contact with a bus stop for 65 s if it stops at it. Additionally, it makes contact for 32 s if the bus does not stop. The work of Rubinstein *et al.* [25] measured vehicle to vehicle communication using IEEE 802.11, achieving average throughputs of 1.81 MB/s throughout the interval of contact, using a relative speed of 120 km/h. Defining c_{time} as 32 s and tx_{rate} as 1.81 MB/s, we can use Inequation 5 to estimate that the maximum amount of gathered data is 57.92 MB.

Using the results from Section 3.2, we can conclude that a bus that makes contact with a sink for 32 s, every 2 h and achieve and average throughput of 1.81 MB/s could carry more than 80,000 sensors in its sensor bank and still unload all the contents in the memory module. Therefore, it is reasonable to assume that the sensing node can completely deliver gathered data on a single contact.

3.4. Candidate sink removal

In the studied problem, every bus stop $s \in \mathcal{S}$ is initially a sink candidate. We define the bus stop s as removed if s is not chosen as a sink. When a bus stop s is removed, the nodes cannot deliver data to it and must deliver the data to the next sink on their path.

The removal of s has effects on the sequence D_b, illustrated in Figure 3. If a bus b has s as the element $S_b(i)$ and s is removed, b must deliver to $S_b(i+1)$ all the data that otherwise would be delivered to s, in $S_b(i)$. This means that, when $S_b(i)$ is removed, $D_b(i-1)$ becomes the old $D_b(i-1)$ plus $D_b(i)$. Additionally, the old $S_b(i)$ is removed from the sequence S_b, the old $D_b(i)$ is removed from the sequence D_b, and every element after $S_b(i)$ and $D_b(i)$ is shifted one position behind.

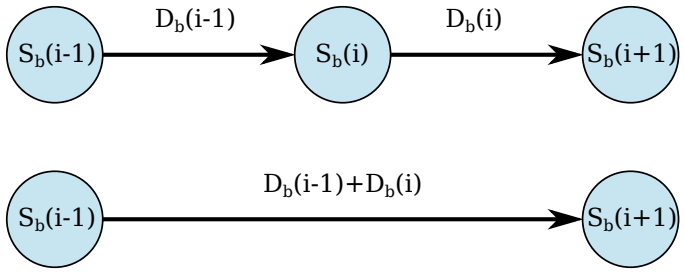


Figure 3: Effects in delay caused by the removal of a sink candidate.

Different buses are affected differently by a sink candidate removal, since buses can have different paths. Figure 4 illustrates the removal of the candidate $q \in \mathcal{S}$ for two different buses, $b, c \in \mathcal{B}$. The buses b and c are moving along the bus stops $p, q, r, s, t \in \mathcal{S}$ along different paths. The sequence of bus stops for b is $S_b = (p, q, r)$ and its sequence of delays is $D_b = (1, 2)$. The sequence of bus stops for c is $S_c = (s, q, t)$ and $D_c = (3, 1)$ is its sequence of delays. When q is removed, S_b becomes (p, r), D_b becomes (3), S_c becomes (s, t), and D_c becomes (4).

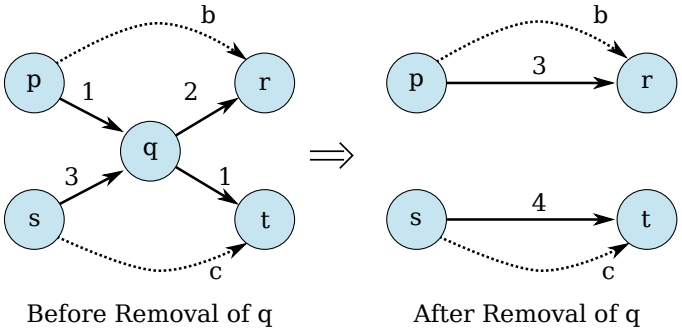


Figure 4: Effects of sink removal for different buses.

We then define the operation of sink candidate removal $s \in \mathcal{S}$ as:

1. Remove s from the set $\mathcal{S}$
2. For every $b \in \mathcal{B}$:
   (a) If there is some $S_b(i) = p$, for every i:
      i. Remove of $S_b(i)$ from S_b;
      ii. Assign $D_b(i-1) := D_b(i-1) + D_b(i)$;
      iii. Remove $D_b(i)$ from D_b.

The removal of the sink candidate s changes one or more delays in D_b, if and only if s is part of S_b. We define the *removal delay* of s ($D_{rem}(s)$) as the maximum delay produced by the removal of s:

$$D_{rem}(s) = \max_{b \in \mathcal{B}} \left(\max_{i \in \{2,\ldots,(m_b-1)\}} (\{D_b(i-1) + D_b(i) | S_b(i) = s\}) \right). \quad (6)$$

A last consideration about the model is the fact that a sensing node might make contact with the same sink more than once, because the bus can drive nearby the same bus stop during different stages of its path. Additionally, a contact between a bus b and a sink on the bus stop $S_b(i)$ does not imply that the bus serves passengers on $S_b(i)$.

4. Formulation of the Optimal Solution

The problem formulated in this work chooses N_{budget} sinks, given all $\mathcal{S}$ bus stops. This choice minimizes the maximum delay between two sinks in the network, which is defined by Equation 2. The optimal solution for this problem is obtained through an ILP problem, modeled as follows:

$$\text{minimize } D_{max} \tag{7}$$

$$\text{subject to} \quad \sum_{s \in \mathcal{S}} x_s = N_{budget}; \tag{8}$$

$$\sum_{q \in \mathcal{S}^a_{bs}} y_{bsq} = x_s \quad \forall b \in B, \quad \forall s \in \mathcal{S}^p_b; \tag{9}$$

$$\sum_{q \in \mathcal{S}^p_{bs}} y_{bqs} = x_s \quad \forall b \in B, \quad \forall s \in \mathcal{S}^a_b; \tag{10}$$

$$D_{max} - \sum_{q \in \mathcal{S}^a_{bs}} d_{bsq} y_{bsq} \geq 0 \quad \forall b \in B, \quad \forall s \in \mathcal{S}^p_b; \tag{11}$$

$$x_s = 1 \quad \forall s \in \mathcal{I}; \tag{12}$$

$$D_{max} \in \mathbb{Z}; \quad y_{bij} \in \{0,1\} \ \forall \ b \in \mathcal{B}, \ \forall \ i,j \in \mathcal{S}; \quad x_s \in \{0,1\} \ \forall \ s \in \mathcal{S}. \tag{13}$$

The objective of this ILP, given by Equation 7, is to minimize D_{max}. Equation 8 specifies that the number of sinks must be equal to N_{budget}. Equation 9 indicates that if bus stop s is chosen as a sink and bus b makes contact to another stop after s, then s must be a predecessor of some bus stop q. Similarly, Equation 10 indicates that if the bus stop s is chosen and if the bus b makes contact to any other stop before s, then s must be a successor of some bus stop q. Hence, Equations 9 and 10 define together the variables y_{bsq}, which are used to evaluated the delay as shown in Figure 3. This evaluation is performed by Equation 11, stating that if in a given path of bus b the stops s and q are chosen and all the stops between s and q are not used, the delay between s and q is accounted in the evaluation of D_{max}. Equation 12 specifies that, if a stop s is the first, or the last one, that any of the buses $b \in \mathcal{B}$ makes contact with (i.e., $s \in \mathcal{I}$)[3], then the choice of s as sink is mandatory, as seen in Section 3.4. Finally, Equation 13 defines the domain of each variable.

[3]Formally, $\mathcal{I} = \left(\bigcup_{b \in B} \mathcal{S}^p_b \setminus \mathcal{S}^a_b\right) \bigcup \left(\bigcup_{b \in B} \mathcal{S}^a_b \setminus \mathcal{S}^p_b\right)$.

5. A Fast Algorithm for Sink Selection

The optimal problem is an ILP, therefore, NP-Hard. Hence, the solution of this problem in real life scenarios might not be feasible. Therefore, this work proposes a greedy solution, specified in Algorithm 1, that is initialized as if every sink candidate was an actual sink. At every iteration, the algorithm removes the sink candidate with the minimum removal delay, defined in Equation 6. The algorithm terminates when the set of sink candidates has N_{budget} elements or when every bus route has only two sink candidates: the start and final stops. If every bus route has only two candidates, it is not possible to remove any more sink candidates. In both cases, the algorithm returns the current set of sink candidates.

The algorithm parameters are: the set $\mathcal{B}$; the set $\mathcal{S}$; the vector $\boldsymbol{paths}$, that stores a list $\boldsymbol{S_b}$ for every element $b \in \mathcal{B}$; the vector $\boldsymbol{d_{next}}$, that stores, in every index i the delay sequence $\boldsymbol{D_b}$ for b; and N_{budget}, the amount of desired sinks.

5.1. Complexity analysis

Given the parameters of Algorithm 1, we define M as the size of the largest list in $\boldsymbol{paths}$. Consequently, the largest list in $\boldsymbol{d_{next}}$ has size $M - 1$.

The time complexity of the algorithm is the complexity of the initial test, on line 1, plus the complexity of the removal delay list initialization, on line 2, plus the complexity of sink candidates removal, on line 6.

The initial test on line 1 is a cardinality comparison of the sets $\mathcal{S}$ and $\mathcal{I}$ with N_{budget}. Using O notation for worst case scenario, the initial test complexity is $O(1)$.

The initialization of the list of removal delays, on line 2, is an iteration over every M elements of $\boldsymbol{P_b}$, nested in an iteration over every K elements of $\mathcal{B}$. Hence, the complexity of this part is $O(KM)$.

The sink candidates removal has an external loop, starting on line 6, in which $|\mathcal{R}|$ grows one unit per iteration. This is limited by the value of N_{budget}. Since N_{budget} is limited by N, the complexity of the sink candidates removal is the complexity of its inner loops multiplied by N. Nested to the loop on line 6, there are two other loops, in sequence. On line 7, a loop iterates over all N elements in $\mathcal{S}$, checking if $s \in \mathcal{I}$ and if $s \in \mathcal{R}$. Since it is possible to use data structures where the complexity of such checks is $O(1)$, the time complexity of loop in line 7 is $O(N)$.

Algorithm 1 Algorithm for Sink Selection

Require: $\mathcal{B} = \{b_1, \ldots, b_K\}, \mathcal{S} = \{s_1, \ldots, s_N\}, \mathcal{I} = \{s_{sy}, \ldots, s_{sz}\}, \boldsymbol{paths} = (\boldsymbol{S_{b1}}, \ldots, \boldsymbol{S_{bK}}), \boldsymbol{d_{next}} = (\boldsymbol{D_{b1}}, \ldots, \boldsymbol{D_{bK}}), N_{budget}$

1: **if** $|\mathcal{S}| - |\mathcal{I}| \leq N_{budget}$ **then return** $\mathcal{I}$ ▷ It is not possible to obtain N_{budget} sinks
2: **for** $b \in \mathcal{B}$ **do** ▷ Initialize removal delay list
3: **for** $i \leftarrow 1, |\boldsymbol{S_b}|$ **do**
4: $\boldsymbol{max\_ts}[\boldsymbol{S_b}[i]] \leftarrow \max(\boldsymbol{D_b}[i-1] + \boldsymbol{D_b}[i], \boldsymbol{max\_ts}[\boldsymbol{S_b}[i]])$
5: $\mathcal{R} \leftarrow \emptyset$
6: **while** $|\mathcal{R}| + |\mathcal{I}| + N_{budget} < |\mathcal{S}|$ **do** ▷ Remove sink candidates until budget is achieved
7: **for** $p \in \mathcal{S}$ **do** ▷ Select a candidate with minimum removal delay
8: **if** $(d_{min} == NULL \vee d_{min} < \boldsymbol{max\_ts}[s]) \wedge s \notin I \wedge s \notin R$ **then**
9: $d_{min} \leftarrow \boldsymbol{max\_ts}[s]$
10: $s\_to\_remove \leftarrow s$
11: **for** $\boldsymbol{S_b} \in \boldsymbol{paths}$ **do**
12: **for** $i \leftarrow 1, |\boldsymbol{S_b}|$ **do**
13: **if** $\boldsymbol{S_b}[i] = s\_to\_remove$ **then**
14: $\boldsymbol{D_b}[i-1] \leftarrow \boldsymbol{D_b}[i-1] + \boldsymbol{D_b}[i]$ ▷ Refresh the delay between the previous and the next candidates
15: **if** $\boldsymbol{max\_ts}[\boldsymbol{S_b}[i+1]] < \boldsymbol{D_b}[i-1] + \boldsymbol{D_b}[i])$ **then**
16: $\boldsymbol{max\_ts}[\boldsymbol{S_b}[i+1]] \leftarrow \boldsymbol{D_b}[i-1] + \boldsymbol{D_b}[i])$ ▷ Refresh removal delays
17: **if** $i > 1$ **then**
18: **if** $(\boldsymbol{max\_ts}[\boldsymbol{S_b}[i-1]] < \boldsymbol{D_b}[i-2] + \boldsymbol{D_b}[i-1])$ **then**
19: $\boldsymbol{max\_ts}[\boldsymbol{S_b}[i-1] \leftarrow \boldsymbol{D_b}[i-2] + \boldsymbol{D_b}[i-1]$
20: Remove($\boldsymbol{S_b}[i]$) ▷ Remove i from $\boldsymbol{S_b}[\boldsymbol{i}]$
21: $\mathcal{R} \leftarrow \mathcal{R} \cup \{s\_to\_remove\}$ ▷ Insert the removed candidate into the set of removed candidates
22: **return** $\{\mathcal{S} \setminus \mathcal{R}\}$ ▷ Return the set of non-removed candidates

The loop in line 20 iterates over all K elements S_b of vector $\boldsymbol{paths}$. Nested to the loop on line 20, another loop iterates over all the elements of every S_b, with a maximum of M. On every iteration, the element D_b of vector $\boldsymbol{max\_ts}$ is accessed and the elements $D_b(i-2)$, $D_b(i-1)$, and $D_b(i)$ are also accessed. To avoid iterating over the list D_b to find these elements,

a pointer to these elements can be coupled to $S_b(i)$. This way, the access to $D_b(i-2)$, $D_b(i-1)$, and $D_b(i)$ in this loop has worst case complexity $O(1)$. Therefore, the loop in line 20 has complexity $O(KM)$.

The insertion of s into the set $\mathcal{R}$ has complexity $O(1)$. Given the complexities of the nested loops, the complexity of the loop that starts on line 6 is $O(N^2 + KMN + N)$. As a consequence, the time complexity of the Algorithm 1 is given by $O(1 + KM + N^2 + KMN + N)$. By the notation O, the time complexity is the biggest between $O(N^2)$ and $O(KMN)$.

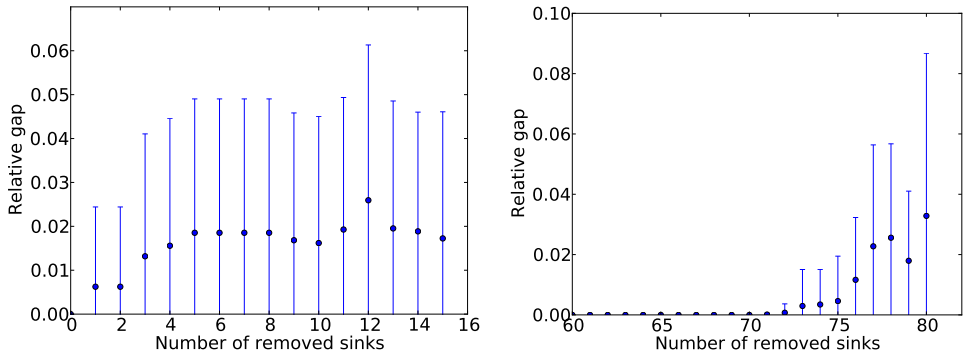
5.2. Comparison with the optimal solution

To analyze the approximation offered by our greedy approach, we compare the solutions found by our algorithm with the ones obtained by the optimal problem solution. To this end, we generate artificial data that simulates a bus mobility scenario, composed of sink candidates. The sink candidates are organized in a grid and bus mobility is simulated. We perform the evaluation for five buses moving around a grid with 25 bus stops as well as a scenario with 10 buses moving around a grid with 100 bus stops. For every scenario, we generate 30 datasets.

The parameters used on each dataset are: the size of the path of each bus, the bus stops on each path, and the delays between stops along a bus path. To generate the datasets, we sample a real dataset, detailed on Section 6. First, we sample real bus path sizes and normalize the path sizes to the size of the scenario. After that, we sample, for each bus, a sequence of bus stops from the grid to be the bus path S_b. The number of stops in the sequence is the path size defined before. Finally, for every pair $(S_b(i),\ S_b(i+1))$, we sample a delay from the real dataset and build $D_b(i)$. Every sampling is executed with uniform distribution over the population.

To solve the optimal problem, we use IBM ILOG CPLEX 12.5.1. Figures 5(a) and 5(b) show the results for 25 and 100 buses, respectively, by plotting the relative gap in D_{max} achieved by the proposed algorithm and the optimal solution, for every number of removed candidates. We evaluate the relative gap as the difference between the two results divided by the optimal result. The error bars represent a 95% confidence interval.

The results show that the proposed algorithm finds solutions close to the optimal. In the worst case of the studied scenarios, the algorithm is less than 10% distant from the optimal solution.



(a) Grid with 25 sink candidates and 5 buses.

(b) Grid with 100 sink candidates and 10 buses.

Figure 5: Relative gap in network maximum delay in function of removed sink candidates on a 5x5 and 10x10 sink candidates grid.

6. Real Scenario Case Study

Our case study consists in running Algorithm 1 using a dataset containing the positions of buses and bus stops in the city of Rio de Janeiro. The instant GPS position of buses[4] and the position of bus stops[5] are published in the website of the Rio de Janeiro's Federation of Passenger Transportation Companies (FETRANSPOR in the Brazilian acronym). Bus positions are collected and stored in one file with all instant positions per minute. Since the bus stops are stationary, their positions are presented as a single file. From these datasets, we generate the data needed to run the algorithm proposed in this article.

Instant positions of buses are collected for a 24h interval, from 0:00h of November, 30^{th}, 2016 to 0:00h of December, 1^{st}, 2016. This data is inserted into a database, together with the positions of bus stops. We define that a bus and a bus stop make contact if the distance between them is less than or equal to 300 m. This range is chosen because it is a typical value for V2I (Vehicle to Infrastructure) communication [26], even though the range of communication does not affect the algorithm, just the generated network. A tuple (instant, bus, bus stop) is selected for every instant when a bus makes contact to a bus stop and a new dataset is built with all the tuples.

[4]http://data.rio/dataset/gps-de-onibus.

[5]http://data.rio/dataset/pontos-de-parada-de-onibus.

This dataset contains data from 6,272 bus stops and 6,683 buses. From this dataset, it is possible to obtain the sets $\mathcal{B}$, $\mathcal{S}$, S_b, and D_b.

The set $\mathcal{B}$ is obtained as the union of all buses in all tuples (instant, bus, bus stop). Tuples are then separated by bus and ordered by instant. We state that tuple (instant, bus, bus stop) belongs to bus b if bus in the tuple equals to b. The list S_b is obtained by the sequence of bus stops in the tuples of b, when ordered by instant. Finally, the list D_b is built using the difference of instant in consecutive tuples of the bus b.

6.1. Dataset analysis

The analysis of the dataset shows the need of some filtering, especially when it comes to large intervals between contacts. Figure 6 shows the cumulative probability function of the maximum delay for each bus. It shows that approximately 20% of buses have maximum delay greater than 7,200 s. This means that 20% of the buses travel more than two hours without stopping at a bus stop to serve passengers. While it is extremely rare for urban buses to remain two hours without stopping at a bus stop, it is expected that a sampling rate of 1 position/minute leaves out many contacts that last less than two minutes. Moreover, it is possible that some buses are out of service, but with their GPS equipment turned on. Our solution to this problem is to filter out buses that have large intervals between contacts. If the number of reached stops is still similar after filtering, this means that the applied filter does not affect the spatial coverage.

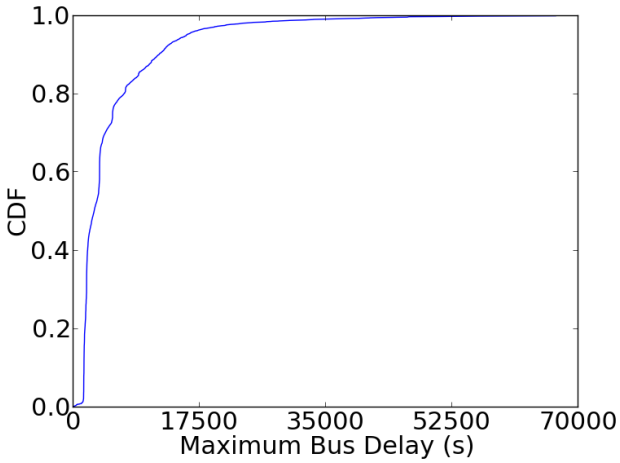


Figure 6: Cumulative distribution of maximum delay of each bus.

Table 2: Dataset parameters.

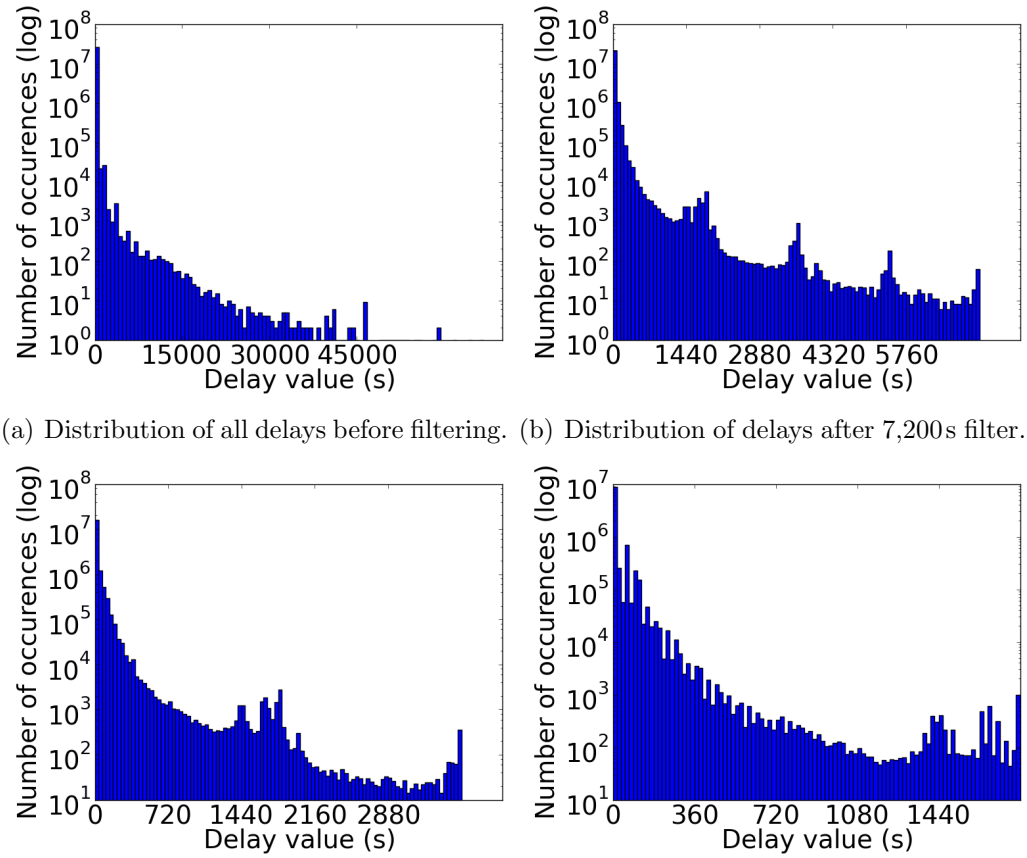
Network maximum delay (s)	**Number of Buses**	**Stops visited**	**Candidates in $\mathcal{I}$**
No filter	6,683	6,272	1,266
7,200	5,429	6,272	1,085
3,600	4,104	6,266	933
1,800	4,104	6,218	641

Three different filters are applied to the dataset, eliminating every bus b that, for some i, has $D_b(i) > 7{,}200$ s, $D_b(i) > 3{,}600$ s, or $D_b(i) > 1{,}800$ s. Figure 7(a) shows the distribution of all delays in the network before the filter. Figure 7(b) shows the distribution of the delays in the network after the filter of 7,200 s, Figure 7(c) shows the distribution of the delays in the network after the filter of 3,600 s, and Figure 7(d) shows the distribution of the delays in the network after a 1,800 s filter.

Figure 7 shows that the filters do not significantly change the delay distribution of the network, except for setting a threshold for the maximum delay. Table 2 shows the parameters of the original dataset and the filtered ones. These results show that, after all the filters, the remaining buses still make contact to approximately 99% of all the 6,272 bus stops somewhere in their paths, meaning that there was no significant loss in the spatial coverage. On the other hand, limiting the number of buses that are used for sensing means that the area covered is less visited by sensing buses, making it less likely for a sensor to detect some event of interest [7].

6.2. Algorithm results

Algorithm 1 is employed to remove sink candidates according to the maximum budget to each one of the datasets. Figure 8(a) shows the network delay D_{max} when executing the algorithm using the dataset obtained after the 7,200 s filter, for different numbers of removed candidates. Similarly, Figure 8(b) and Figure 8(c) show, respectively, the same results for the filter of 3,600 s and 1,800 s. The values for fewer removals were omitted because there was no modification on them. The dots in these figures are inflexion points, meaning that the delay is the same for a given range of removed candidates. These inflexion points happen because the algorithm removes successive sink candidates without penalties to the network maximum delay until a certain threshold, limited by some bottleneck sink. When the bottleneck sink can-



(a) Distribution of all delays before filtering. (b) Distribution of delays after 7,200 s filter.

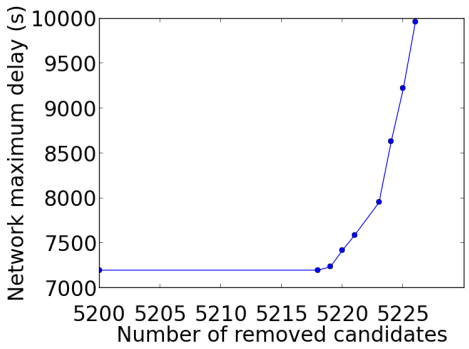
(c) Distribution of delays after 3,600 s filter. (d) Distribution of delays after 1,800 s filter.

Figure 7: Distribution of delays before filtering and after 1,800 s, 3,600 s, and 7,200 s filters.

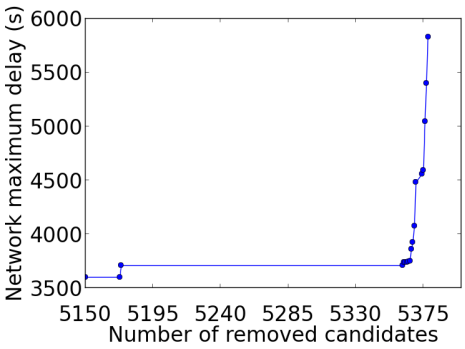
didate is removed, the network maximum delay finally increases. Then the cycle repeats for a new threshold.

All results show that our solution is capable of eliminating more than 84% of a total 6,272 bus stops while having less than 10% increase in the delay. In the case of the filter of 1,800 s, it is possible to use 20% of the bus stops as sinks and still have a network maximum delay of 30 minutes.

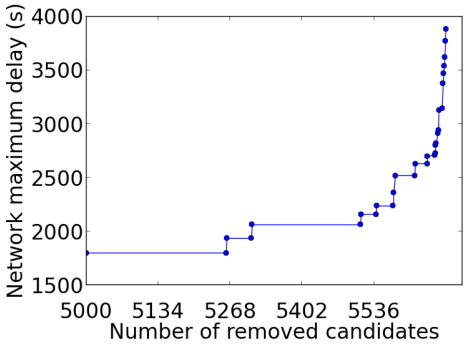
The results also show that the network maximum delay starts increasing faster when more sink candidates are removed. Furthermore, the analysis shows that different filters can provide different delay levels. A possible scenario is that buses are divided into different delay levels, balancing the trade-off between delay constraints and amount of data available.



(a) Network maximum delay (D_{max}) filtered for 7,200 s, as a function of the number of removed candidates.



(b) Network maximum delay (D_{max}) filtered for 3,600 s, as a function of the number of removed candidates.



(c) Network maximum delay (D_{max}) filtered for 1,800 s, as a function of the number of removed candidates.

Figure 8: Network maximum delay (D_{max}) for different filters, as a function of the number of removed candidates.

7. Conclusions and Future Work

In this work, we have considered a bus sensing network where buses play the role of mobile sensors, improving the coverage of a citywide wireless sensor network. While on the one hand the opportunistic communication provided by buses reduce costs, on the other hand it may impact the communication latency, since the sensor node has to wait for the next contact opportunity with a sink node located at a bus stop.

Therefore, we assumed a network where mobile nodes traverse the city at regular travels and communicate with a limited number of sink nodes.

We have then investigated the relationship between the number of chosen sinks and the delay experienced in the network. This delay was defined as the maximum time that a bus takes to travel between two consecutive sinks in the IoT infrastructure. Hence, we have formulated an integer linear programming (ILP) problem to choose a limited amount of bus stops as sinks, trying to minimize the network delay. Given that ILP is NP-Hard, we have proposed a greedy algorithm to solve this problem and shown that this solution finds results close to the optimal ILP solution. Finally, we have provided a case study using the algorithm to choose sink locations in a network of real buses, based on the GPS data provided by the public transport system in Rio de Janeiro. The results have shown that, for the city of Rio de Janeiro, it is possible to obtain a network using approximately 16% of the bus stops as sinks and have a maximum delay of 32 minutes in data delivery with no significant losses on the spatial coverage. It is worth noting that the maximum delay is 30 minutes when this network has all its bus stops acting as sinks.

As future work, we plan to combine the average delay with the used maximum delay metric. The basic idea will be to try to reduce the average delay of message delivery, but preserving the maximum network maximum delay as small as possible. Moreover, we aim to model the time of bus contact with sinks, to better define which wireless technologies are more suitable for such a network. Another possible approach is to employ post-optimization techniques to improve another metrics, such as the average delay. For example, we can employ clustering solutions after running our algorithm.

Acknowledgments

The authors would like to thank CAPES, CNPq and FAPERJ for their financial support to this work.

References

[1] H. Chourabi, T. Nam, S. Walker, J. R. Gil-Garcia, S. Mellouli, K. Nahon, T. A. Pardo, H. J. Scholl, Understanding smart cities: An integrative framework, in: Hawaii International Conference on System Science (HICSS), IEEE, 2012, pp. 2289–2297.

[2] B. Rashid, M. H. Rehmani, Applications of wireless sensor networks for urban areas: A survey, Journal of Network and Computer Applications 60 (2016) 192–219.

[3] L. Atzori, A. Iera, G. Morabito, The internet of things: A survey, Computer Networks 54 (15) (2010) 2787–2805.

[4] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (IoT): A vision, architectural elements, and future directions, Future Generation Computer Systems 29 (7) (2013) 1645–1660.

[5] A. Zanella, N. Bui, A. Castellani, L. Vangelista, M. Zorzi, Internet of things for smart cities, IEEE Internet of Things Journal 1 (1) (2014) 22–32.

[6] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, A survey on sensor networks, IEEE Communications magazine 40 (8) (2002) 102–114.

[7] B. Liu, P. Brass, O. Dousse, P. Nain, D. Towsley, Mobility improves coverage of sensor networks, in: ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), ACM, 2005.

[8] E. Ekici, Y. Gu, D. Bozdag, Mobility-based communication in wireless sensor networks, IEEE Communications Magazine 44 (7) (2006) 56 – 62.

[9] V. Albino, U. Berardi, R. M. Dangelico, Smart cities: Definitions, dimensions, performance, and initiatives, Journal of Urban Technology 22 (1) (2015) 3–21.

[10] J. Merino, I. Caballero, B. Rivas, M. Serrano, M. Piattini, A data quality in use model for big data, Future Generation Computer Systems 63 (2016) 123–130.

[11] J. L. Wong, R. Jafari, M. Potkonjak, Gateway placement for latency and energy efficient data aggregation, in: IEEE International Conference on Local Computer Networks, IEEE, 2004, pp. 490–497.

[12] A. Marjovi, A. Arfire, A. Martinoli, High resolution air pollution maps in urban environments using mobile sensor networks, in: International

Conference on Distributed Computing in Sensor Systems (DCOSS), IEEE, 2015, pp. 11 – 20.

[13] W. Dong, G. Guan, Y. Chen, K. Guo, Y. Gao, Mosaic: Towards city scale sensing with mobile sensor networks, in: IEEE International Conference on Parallel and Distributed Systems (ICPADS), IEEE, 2015, pp. 29–36.

[14] K. D. Zoysa, C. Keppitiyagama, G. P. Seneviratne, W. W. A. T. Shihan, A public transport system based sensor network for road surface condition monitoring, in: ACM SIGCOMM Workshop on Networked Systems for Developing Regions (NSDR), ACM, 2007.

[15] T. Umer, M. Amjad, M. K. Afzal, M. Aslam, Hybrid rapid response routing approach for delay-sensitive data in hospital body area sensor network, in: Proceedings of the 7th International Conference on Computing Communication and Networking Technologies, ACM, 2016, p. 3.

[16] S. Ghafoor, M. H. Rehmani, S. Cho, S.-H. Park, An efficient trajectory design for mobile sink in a wireless sensor network, Computers & Electrical Engineering 40 (7) (2014) 2089–2100.

[17] D. S. Dias, L. H. M. K. Costa, M. D. de Amorim, Capacity analysis of a city-wide V2V network, in: International Conference on the Network of the Future (NoF), IFIP/IEEE, 2016, pp. 1–3.

[18] F. Bonomi, R. Milito, P. Natarajan, J. Zhu, Fog computing: A platform for internet of things and analytics, in: Big Data and Internet of Things: A Roadmap for Smart Environments, Springer, 2014, pp. 169–186.

[19] P. Cruz, J. B. P. Neto, M. E. M. Campista, L. H. M. K. Costa, On the accuracy of data sensing in the presence of mobility, in: International Conference on the Network of the Future (NoF), IFIP/IEEE, 2016.

[20] J. Vasseur, A. Dunkels, Interconnecting Smart Objects with IP: The Next Internet, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010.

[21] M. Amjad, M. Sharif, M. K. Afzal, S. W. Kim, Tinyos-new trends, comparative views, and supported sensing applications: A review, IEEE Sensors Journal 16 (9) (2016) 2865–2889.

[22] F. Akhtar, M. H. Rehmani, Energy replenishment using renewable and traditional energy resources for sustainable wireless sensor networks: A review, Renewable and Sustainable Energy Reviews 45 (2015) 769–784.

[23] A. Sinaeepourfard, J. Garcia, X. Masip-Bruin, E. Marín-Tordera, J. Cirera, G. Grau, F. Casaus, Estimating smart city sensors data generation, in: Ad Hoc Networking Workshop (Med-Hoc-Net), 2016 Mediterranean, IEEE, 2016, pp. 1–8.

[24] V. B. Da Silva, F. O. Da Silva, M. E. M. Campista, L. H. M. Costa, A trajectory-based approach to improve delivery in drive-thru internet scenarios, in: Communications Workshops (ICC), 2013 IEEE International Conference on, IEEE, 2013, pp. 489–494.

[25] M. G. Rubinstein, F. B. Abdesslem, M. D. De Amorim, S. R. Cavalcanti, R. D. S. Alves, L. H. M. K. Costa, O. C. M. B. Duarte, M. E. M. Campista, Measuring the capacity of in-car to in-car vehicular networks, IEEE Communications Magazine 47 (11).

[26] J. Gozálvez, M. Sepulcre, R. Bauza, IEEE 802.11p vehicle to infrastructure communications in urban environments, IEEE Communications Magazine 50 (5).