MobSink: a Visual Mobile Wireless Sensor Networks Positioning Simulator

João Paulo Just Peixoto
Federal Institute of Bahia
Vereador Romeu Agrário Martins St., Tento
Valença, BA 45400-000
joao.just@ifba.edu.br

Daniel G. Costa
State University of Feira de Santana
Transnordestina Av., Novo Horizonte
Feira de Santana, BA 44036-900
danielgcosta@uefs.br

ABSTRACT

The planning and deployment of a WSN (Wireless Sensor Network) in a Smart City can be a very challenging work. People involved in such implementation must be aware of how sensors and sinks may behave in the target environment. To aid the development of new WSNs, simulation tools are often used to predict how nodes will interact before spending time and money in a real deployment. Also, simulator can help students better understand WSNs. In this paper, we present MobSink, a simulator for WSNs with multiple mobile sinks. MobSink also allows to perform simulations in a Smart City scenario, with streets and movements constraints. It also shows how to configure MobSink for a generic scenario and describes how it works internally.

KEYWORDS

simulation, WSN, sinks mobility, smart city

1 INTRODUCTION

A Wireless Sensor Network (WSN) is an ad-hoc network composed of several sensor nodes that gather environment data as temperature, humidity, luminosity and also multimedia data as audio and video. The sensed data is then transmitted in a hop-by-hop manner to centralized nodes called *sinks*, which can be connected to the Internet or to a local server to deliver all the collected data to an application [9].

The main purpose of WSNs is to monitor environments, applying the data obtained from the network in specific applications [1]. Sensor nodes can also be deployed in a Smart City to collect many useful information as pollution status, urban noise, local temperature, traffic density, etc. There are several possibilities of applications of WSNs in Smart Cities, like smart surveillance, smart transportation, smart services, among others [3]. In such scenarios, sinks can be installed in vehicles to run across the roads, collecting data from sensor nodes as they pass by them [6].

In an ordinary WSN, the sensor nodes have some hardware constraints: restricted energy supply, reduced processing power, low memory, low communication range, etc. Most of the constraints of a sensor node are related to the low power available (most sensor nodes run with non-rechargeable batteries [2]).

In: XVII Workshop de Ferramentas e Aplicações (WFA 2018), Salvador, Brasil. Anais do XXIV Simpósio Brasileiro de Sistemas Multimídia e Web: Workshops e Pôsteres. Porto Alegre: Sociedade Brasileira de Computação, 2018.

ISBN 978-85-7669-435-9.

In order to create a sensing environment, several sensor nodes must be deployed into the area of interest, which can be time consuming and costly if the network is not well designed [2]. When trying to predict the performance of a WSN before its deployment, simulators can be used to support better designing of the network [4, 7, 8]. Also, they can help students to learn more about WSN configurations without the need to buy and deploy real hardware.

In this paper we describe the development of the new tool MobSink, a software specially designed to simulate WSNs in a Smart City environment. The main purpose of MobSink is to test energy consumption and data delivery in a WSN with mobile sinks in a Smart City. A traffic model is used to calculate the fastest route of a mobile sink in such a way it can avoid traffic jams and reach its destination as fast as possible. The use of this type of simulator can aid the designing and deployment of WSNs in a Smart City scenario, as well to help students to better understand how a WSN works with multiple mobile sinks. MobSink is focused in multiple sinks positioning and do not consider routing algorithms, transport protocols, etc.

MobSink is licensed under GNU GPLv3 (GNU General Public License version 3) and its source code is available on GitHub. This makes possible to students to learn how to code tools for WSN simulation and also contribute with this project.

The remainder of this paper is organized as follows. Section 2 presents the energy model and positioning scheme used in MobSink, while section 3 explains how these models were implemented; section 4 describes how to simulate a Smart City scenario and 5 concludes this paper and presents future works.

2 MOBSINK INTERNALS

The MobSink tool works by simulating sensor nodes that generate data and that forward the buffered data to the next hop, which is another sensor node, until the data reaches the nearest sink. Actually, there are different possible behaviors for the sensor nodes, but the MobSink tool was designed to incorporate the relevance-based scheme proposed in [5]. In that proposal, each sensor node has a *RL* (Relevance Level) that varies from 0 (no relevance) to 15 (highest relevance). Generally speaking, the relevance level of each sensor node can be defined in different ways, but the values of *RL* will usually be a reflection of the applications monitoring requirements (e.g. a temperature sensor near a nuclear reactor may have higher relevance than a sensor node monitoring the ambient temperature in the same WSN). It is reasonable to expect that the amount of data generated by a sensor node depends on its *RL* (*RL* = 0 means no data is generated by the sensor and it works only as a relay node).

In wireless sensor networks, data transmission and relaying tasks are expected to consume most energies of the nodes. Therefore, the position of the sinks influence the path data packets must follow to reach their destinations, also impacting in the amount of consumed energy. This particularity of WSNs was specially considered when designing the MobSink tool.

The following subsections describe how MobSink computes the energy consumed by each sensor node and how this tool moves the sink nodes inside the modelled WSN.

2.1 Energy consumption

The MobSink tool operates generating data packets at source nodes and transmitting them to the nearest sink, which is performed in the scope of a "simulation". At each iteration of a simulation, every sensor node generates PDUs (Packet Data Units) according to its RL (sensor nodes with $RL = 0$ generate no data). Each generated packet is then inserted into a buffer for further transmission. If the buffer is full, the exceeding packets are discarded and this information is accounted for a final report. After inserting the generated PDUs into the local buffer, the sensor node transmits to the next hop, which receives the packets and stores them into its buffer (again, if the receiver node's buffer is full, the packets are dropped).

This whole process is performed once at each iteration for every sensor node in the WSN simulation. Data transmission and reception procedures result in energy consumption in each node that participated in the considered communication. Currently, MobSink uses the power scheme described in [5]: each PDU is 127 bytes sized, the power used in transmission and reception of packets is 60 mW and 4 μs are needed to transmit each bit of data (computed according to the maximum transmission rate, which is based on the IEEE 802.15.4 standard). Also, each sensor node is supplied by a 3 V battery with 1000 mAh capacity and has a 128 KB buffer. Although the current version of MobSink does not allow for configurable power source and buffer, future releases will feature a configuration file.

When a packet leaves the source node, it is relayed by some intermediate nodes until it reaches the sink. The energy consumed in each intermediate node is computed to account the total energy spent in the simulation. Let $D_{tx}(i)$ be the size of the packet transmitted from node i in bits, $P_{tx}(i)$ the power needed by node i to transmit a bit and $t(i)$ the time needed by node i to transmit a bit, $E(i)$ is then the total power needed by node i to transmit a packet, as shown in Equation 1.

$$E_{tx}(i) = D_{tx}(i) \cdot P_{tx}(i) \cdot t(i) \tag{1}$$

In the same way, MobSink computes the energy needed to receive a packet in each sensor node. The equation is almost the same, but MobSink considers the size of the received packet in node i as $D_{rx}(i)$ and the reception power as $P_{rx}(i)$ in Equation 2.

$$E_{rx}(i) = D_{rx}(i) \cdot P_{rx}(i) \cdot t(i) \tag{2}$$

In a sequence $S = (s_0, s_1, ..., s_n)$ of sensor nodes, the total energy spent to send a packet from its source to the sink is obtained by summing the energy spent by each node to transmit the packet to the next hop and the energy spent by each node to receive the packet from its origin, as described in Equation 3.

$$\sum_{i=0}^{n-1} E_{tx}(i) + E_{rx}(i+1) \tag{3}$$

2.2 Sinks positioning

MobSink provides three main positioning schemes for sinks: static, fixed movement and relevance-based.

In the static scheme, the sinks are positioned in a grid centralized manner and they keep their initial positions during the whole simulation time. No movements are performed in this scheme. Distant sensor nodes must relay its data through intermediate nodes to reach the sink. Although currently MobSink does not allow for a custom fixed position, this is a feature to come in a future release.

In the fixed movement scheme, the sinks move on straight lines which can be a horizontal or a vertical path through the WSN. The sink moves at a constant speed and just like the static scheme, its initial position is defined by the simulator.

At last, in the relevance-based scheme, MobSink uses the approach defined in [6]. In such way, the sinks positions are defined by the relevance levels of sensor nodes. The goal of this approach is to make sinks closer to more relevant nodes, potentially improving the global performance of the network, since sinks will be closer to the source nodes that transmit more data (higher values of *RL*. If there are more than one sink in the WSN, this algorithm creates clusters of sensors according to their relevances. The total *RL* of the clusters, defined by the sum of the sensors relevances in a cluster, are balanced in a way that each cluster total *RL* do not differ too much from the others.

When using the relevance-based scheme, a Smart City scenario can be easily simulated. The user can model streets of a city and the sinks will move through these streets to reach its final positions. Every time the topology changes (e.g.: a sensor node *RL* changes or runs out of energy), MobSink recalculates new positions for the sinks and make them move. The sinks paths always obey roads restrictions, if defined by the user.

3 SOFTWARE ARCHITECTURE

The MobSink was designed to model the operation of any WSN that has one or more static or moving sinks, which complies with a lot of different real sensor networks. For that, the operation details of wireless sensor networks were carefully considered.

This section explains how a WSN is simulated and how sinks positions are calculated in a Smart City scenario. Subsection 3.1 presents the mechanism used to calculate energy consumption and data traffic in a simulation, while subsection 3.2 presents the developed mechanisms to model a city and its associated traffic behavior.

3.1 WSN simulation

To simulate a WSN, some classes were implemented to represent the actors in a WSN. Figure 1 presents a class diagram showing the relations of nodes and sinks in MobSink. When the user inserts any new sensor node in the WSN, an object of "Node" class is instantiated. After specifying how many sinks will be used in the simulation, MobSink instantiates a "Sink" object for each sink in simulation and a "Cluster" object for each sink. Each "Cluster" object has a

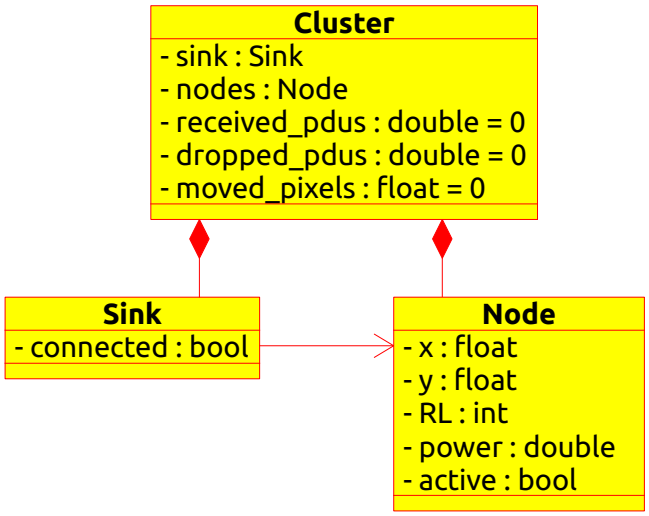

Figure 1: Class diagram for MobSink.

sink associated to it, besides several nodes. The nodes associations are made depending on the chosen positioning algorithm.

When simulation starts, MobSink positions every sink according to the employed positioning algorithm, associating each node to its nearest sink. This is done by adding a reference of the node object to the correspondent Cluster object. After this, MobSink has some clusters (one for each sink), containing its nodes. The sensor nodes can only communicate with other nodes in the same cluster, including the sink.

MobSink simulates "one second" intervals. At each iteration, it checks every sensor node and sends it a message to "work" (generate data and update its status according to user's programmed schedule). The sensor node will generate packets according to its relevance level and sinks may be repositioned if there was any topology change in the network (e.g: a sensor toggles active/inactive or changes its relevance level). At last, active sensors with buffered data transmit to the next hop. At the end of the iteration, one second has passed in the simulated scenario and every active sensor has generated and/or relayed data. Figure 2 presents the flowchart of MobSink simulation.

After simulation finishes, the cluster object has accounted the total of received and dropped packets, distance traveled by its sink and power consumption of all sensor nodes.

3.2 Smart City modeling

In oder to model a Smart City scenario, MobSink simulates roads and their traffic behavior. Before simulation begins, MobSink checks every road added by the user and finds all intersections between them. Each road is modeled as an edge in a directed weighted graph. Roads' source and destination points are modeled as vertices and so are their intersections.

To set a weight in each edge, MobSink uses the total time to traverse the road, calculated using the road's length and average speed as parameters. Our approach to simulate traffic jams was to use the roads legal speed limit defined as $V(r)$ as a maximum

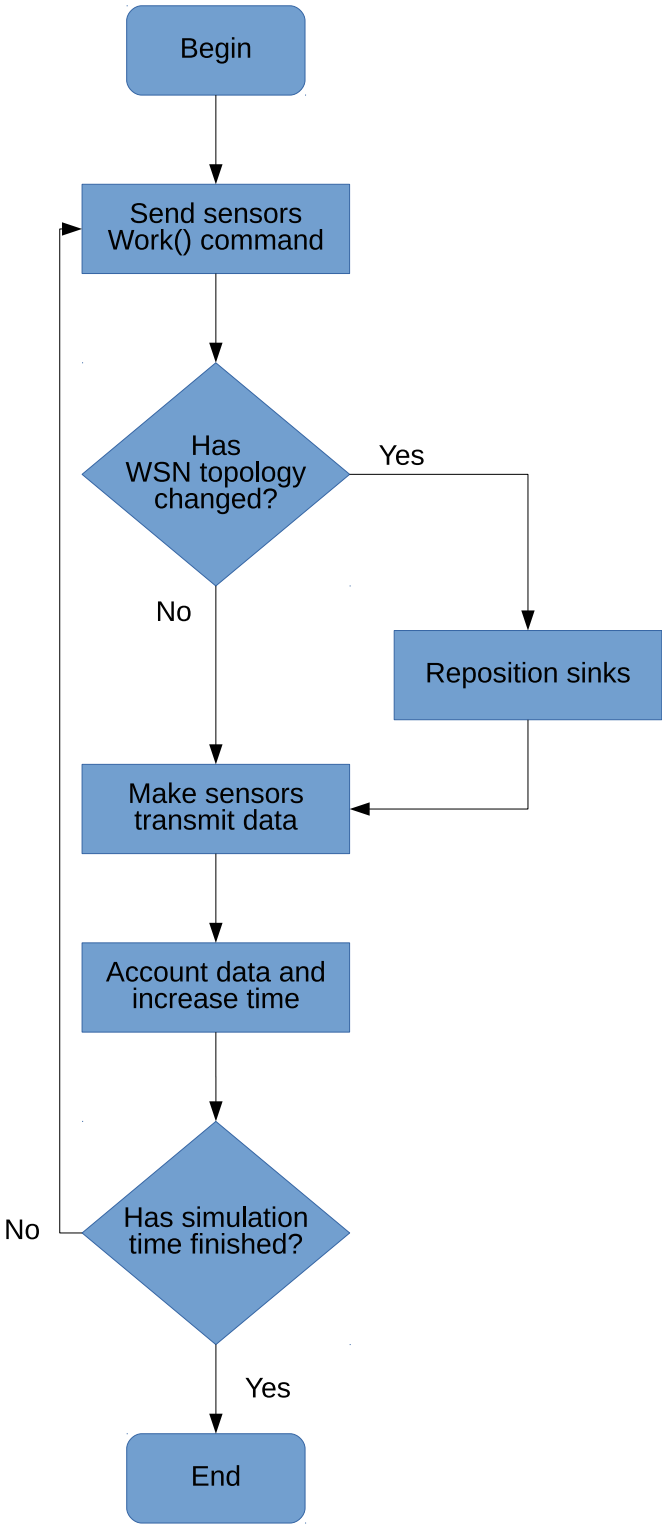

Figure 2: Flowchart of MobSink simulation.

vehicle speed, applying then a traffic coefficient $m(r)$ (as shown in Equation 4) to set the approximated time to traverse each road. By using these parameters with the road's length, defined as $S(r)$ in that Equation, it is possible to calculate the time spent to traverse a road r. The user can then program a traffic schedule and simulate traffic jams along the simulation time.

$$t(r) = \lim_{x \to m(r)} \frac{S(r)}{V(r) \cdot x} \tag{4}$$

Every time a sink has to be positioned in the map, Dijkstra's algorithm is run, using the specified weight ($t(r)$) to find the fastest route, avoiding traffic jams. The simulator also offers an option to use the shortest routes, ignoring traffic information. A control schedule is implemented in a WSN XML file, setting traffic data for each road.

3.3 WSN XML programming

Sensors behavior and traffic can be dynamically changed during the simulation. The way MobSink allows it to be done is by specifying the changes and the times they occur in a well-formatted WSN XML file. Each sensor node is defined by a *sensor* tag in the XML file, containing its coordinates in the map and the initial relevance level. Inside the *sensor* tag, a *control* tag is used to define a change in the sensor behavior. The *control* tag has three attributes: *time*, to specify at which time of simulation the change will be applied; *rl*, to specify the new relevance level of the sensor; and *enabled*, to enable or disable the sensor node at that time of simulation.

The roads, if any, are defined by a *path* tag, containing its two vertices coordinates and flow (it may be an one-way road). Like the sensor nodes, the roads may have *control* tags with three attributes: *time*, like sensor nodes, to specify the time of change; *speedlimit*, to specify the new speed limit from that time on; and *traffic*, varying from 0 (stopped traffic) to 1 (free road), to specify the traffic behavior at that moment. The *traffic* attribute is used as $m(r)$ in Equation 4. Listing 1 shows an example of a WSN XML file for MobSink.

Listing 1: MobSink XML sample

```
<?xml version="1.0" encoding="UTF-8"?>
<network width="200" height="200">
 <path xa="0" ya="50" xb="100" yb="50">
  <traffic time="1" speedlimit="60"
  traffic="1"/>
  <traffic time="3600" speedlimit="60"
  traffic="0.3"/>
 </path>
 <sensor x="10" y="40" rl="1">
  <control time="0" rl="0" enabled="false"/>
  <control time="60" rl="5" enabled="true"/>
 </sensor>
 <sensor x="70" y="70" rl="1">
  <control time="0" rl="0" enabled="false"/>
  <control time="120" rl="7" enabled="true"/>
  <control time="180" rl="0" enabled="false"/>
 </sensor>
</network>
```

4 SIMULATING A WSN WITH MOBSINK

A simulation of a generic WSN deployed on a city will be presented, demonstrating the operation of the MobSink tool. In this section, a map of the city of Feira de Santana (Brazil) is considered, assuming that sensors will be deployed in this city. The MobSink can be used as shown in Figure 3 (main screen of the tool) to model every street of the city or the user can convert maps exported from OpenStreetMaps directly to the MobSink format, which is a feature also provided by the MobSink tool.

Once the user has created a map for simulation or imported a map from OpenStreetMaps, he/she can insert sensor nodes to create a WSN. There are three ways to insert sensor nodes:

- Manually: the user can click the sensor tool, type a relevance level in "RL" text box and click the position on the map where he/she wants to insert the sensor.
- Grid: the user can type the amount of sensors to be inserted into the "Sensors" text box and click the "Insert sensors in a grid manner" button. The sensors will be inserted onto the map forming a regular grid.
- Randomly: the user can type the amount of sensors to be inserted into the "Sensors" text box and click the "Insert sensors randomly" button. The sensors will be inserted onto the map in random positions.

In any of these options, there is also a possibility to randomly select the relevance level of the new sensor nodes by clicking the "Insert sensors with random RLs" button. In this case, every time the user inserts a new node, its relevance level will be randomly chosen.

After inserting sensors nodes, the sinks have to be inserted. The current version of MobSink does not allow to insert sinks in user-defined positions. Instead, the user has to type how many sinks he/she wants to insert into the "Sinks" text box, choose a sink positioning algorithm from the combo box labeled "Sink positioning" and click the "car" button. MobSink will place the sinks according to the chosen sink positioning method.

At last, the user can type the simulation time in seconds into the "Time" text box and start a simulation. To do so, the user has to click the "Start simulation" button. MobSink will simulate the network for the time the user has chosen and display its results in the text box at the right corner of the main window.

After a simulation had run, it is possible to save all the simulation data into a CSV file by clicking the "Save report" button. This file can be opened in most spreadsheet applications and can be parsed to generate graphics of energy consumption in the WSN. Along with energy and transmission data, it contains every movement the sinks may have done during the simulation. Figure 4 contains an example of a chart created with GNU Plot just processing MobSink provided data.

Also, it is possible to run an animated simulation if the user clicks the "Enable sinks animation" button. Sometimes, this is worth to watch the sinks moving during the simulation (the paths chosen by the sinks are highlighted every time they move). Actually, this is a helpful resource provided by our tool that can support different teaching methods in networks and WSN classes.

Finally, if the user needs to run several simulations in a batch manner, MobSink offers a command line interface.

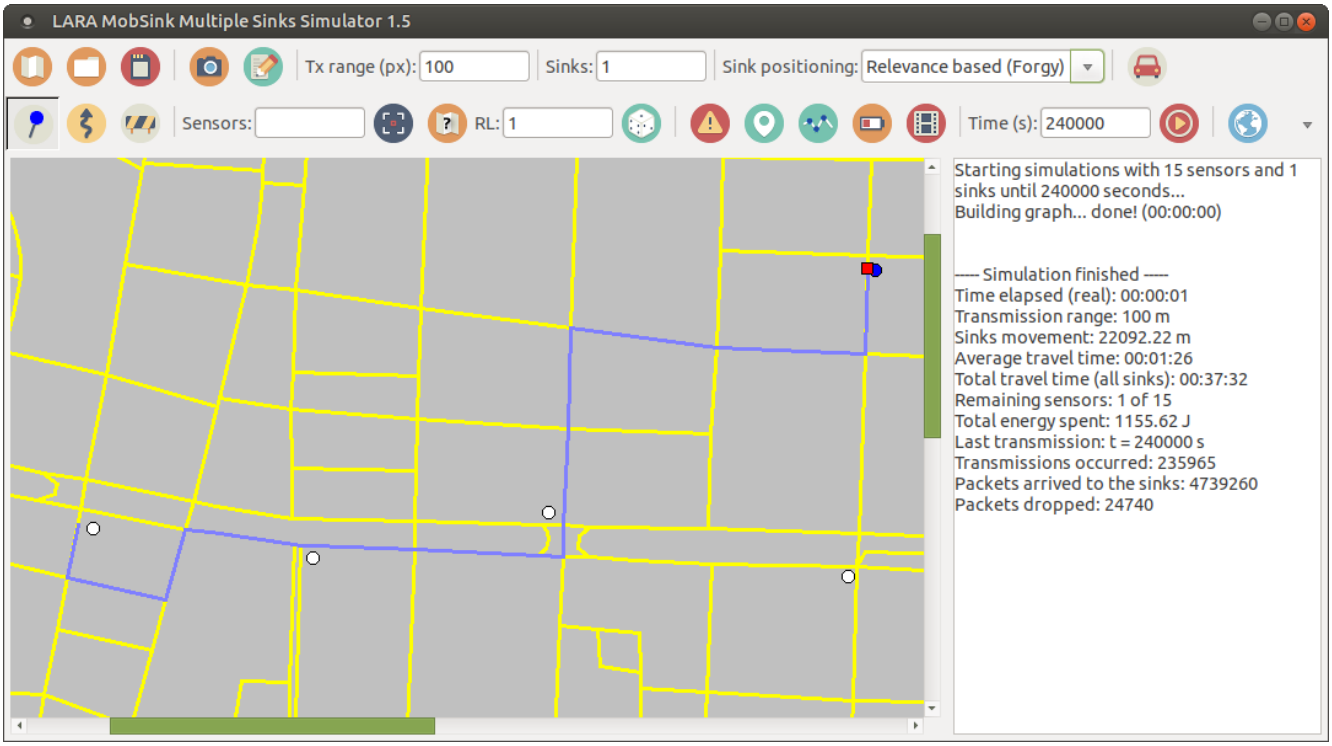

Figure 3: MobSink main screen.

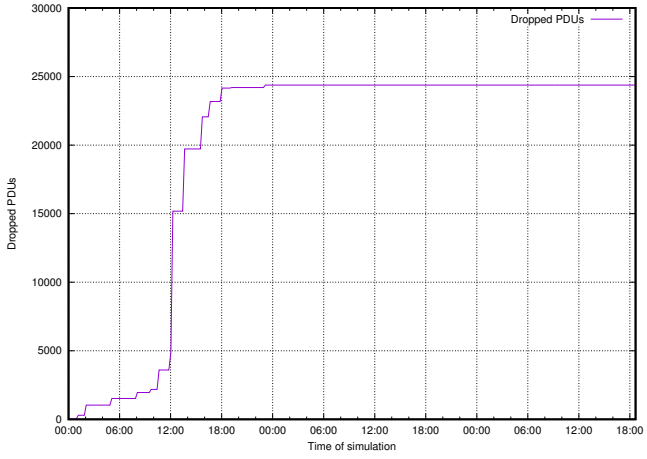

Figure 4: Number of dropped PDUs.

5 CONCLUSIONS

This paper presented MobSink, a novel tool to simulate multiple sinks positioning in WSNs and associated energy consumption. MobSink can be used to simulate dynamic sensors behavior and traffic jams, allowing the user to assess energy consumption and packet losses. The energy consumption accounting and sinks positioning algorithms were described in this paper, as well a flowchart explaining how the tool works when simulating a WSN. Also, the mechanism for sensors and traffic configurations were shown.

We believe MobSink can be very useful to students who want to better understand the basics of WSNs, sinks positioning and Smart Cities. And this will be even more evident in future versions of the tool, which will include additional features in this sense.

6 DOWNLOAD

The MobSink software can be downloaded from http://just.pro.br/blog/mobsink

REFERENCES

[1] Y. Charfi, N. Wakamiya, and M. Murata. 2009. Challenging Issues in Visual Sensor Networks. *Wireless Commun.* 16, 2 (April 2009), 44–49.
[2] E Egea-Lopez, J Vales-Alonso, AS Martinez-Sala, P Pavon-Marino, and J García-Haro. 2005. Simulation tools for wireless sensor networks. In *Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*. 24.
[3] G. P. Hancke and G P. Silva, B.and Hancke Jr. 2012. The Role of Advanced Sensing in Smart Cities. *Sensors* 13, 1 (2012), 393.
[4] M. Jevtic, N. Zogovic, and G. Dimic. 2009. Evaluation of Wireless Sensor Network Simulators. 17th Telecommunications Forum.
[5] J. P. J. Peixoto and D. G. Costa. 2015. QoE-aware multiple sinks mobility in wireless sensor networks. In *Conference on New Technologies, Mobility and Security (NTMS)*. 1–4.
[6] J. P. J. Peixoto and D. G. Costa. 2017. Wireless visual sensor networks for smart city applications:A relevance-based approach for multiple sinks mobility. *Future Generation Computer Systems* 76 (2017), 51 – 62.
[7] D. Rosario, Z. Zhao, C. Silva, E. Cerqueira, and T. Braun. 2013. An OMNeT++ Framework to Evaluate Video Transmission in Mobile Wireless Multimedia Sensor Networks. In *Conference on Simulation Tools and Techniques (ICST)*.
[8] H. Sundani, H. Li, V. Devabhaktuni, M. Alam, and P. Bhattacharya. 2008. Wireless Sensor Network Simulators: A Survey and Comparisons. 2 (April 2008). International Journal Of Computer Networks.
[9] J. Yick, B. Mukherjee, and D. Ghosal. 2008. Wireless sensor network survey. *Computer Networks* 52, 12 (aug 2008), 2292–2330.