CrowdSenSim 2.0: a Stateful Simulation Platform for Mobile Crowdsensing in Smart Cities

Federico Montori
federico.montori2@unibo.it
Department of Computer Science and Engineering, University of Bologna
Bologna, Italy

Emanuele Cortesi
emanuele.cortesi2@studio.unibo.it
Department of Computer Science and Engineering, University of Bologna
Bologna, Italy

Luca Bedogni
luca.bedogni4@unibo.it
Department of Computer Science and Engineering, University of Bologna
Bologna, Italy

Andrea Capponi
andrea.capponi@uni.lu
FSTC-CSC, University of Luxembourg
Esch-sur-Alzette, Luxembourg

Claudio Fiandrino
claudio.fiandrino@imdea.org
IMDEA Networks Institute
Madrid, Spain

Luciano Bononi
luciano.bononi@unibo.it
Department of Computer Science and Engineering, University of Bologna
Bologna, Italy

ABSTRACT

Mobile crowdsensing (MCS) has become a popular paradigm for data collection in urban environments. In MCS systems, a crowd supplies sensing information for monitoring phenomena through mobile devices. Typically, a large number of participants is required to make a sensing campaign successful. For such a reason, it is often not practical for researchers to build and deploy large testbeds to assess the performance of frameworks and algorithms for data collection, user recruitment, and evaluating the quality of information. Simulations offer a valid alternative. In this paper, we present CrowdSenSim 2.0, a significant extension of the popular CrowdSenSim simulation platform. CrowdSenSim 2.0 features a stateful approach to support algorithms where the chronological order of events matters, extensions of the architectural modules, including an additional system to model urban environments, code refactoring, and parallel execution of algorithms. All these improvements boost the performances of the simulator and make the runtime execution and memory utilization significantly lower, also enabling the support for larger simulation scenarios. We demonstrate retro-compatibility with the older platform and evaluate as a case study a stateful data collection algorithm.

CCS CONCEPTS

• **Human-centered computing → Mobile computing**; • **Software and its engineering → Simulator / interpreter**; *Publish-subscribe / event-based architectures*; • **General and reference →** *Validation*; • **Networks →** *Network simulations*.

KEYWORDS

mobile crowdsensing, simulation, modeling, distributed algorithms

ACM Reference Format:
Federico Montori, Emanuele Cortesi, Luca Bedogni, Andrea Capponi, Claudio Fiandrino, and Luciano Bononi. 2019. CrowdSenSim 2.0: a Stateful Simulation Platform for Mobile Crowdsensing in Smart Cities. In *22nd Int'l ACM Conf. on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM'19), Nov. 25–29, 2019, Miami Beach, FL, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3345768.3355929

1 INTRODUCTION

Mobile crowdsensing (MCS) gained exponential interest in the last years and has become one of the most promising paradigms for data collection in urban environments within the scope of smart cities [3]. MCS systems gather data from sensors typically embedded in citizens' mobile devices, such as smartphones, tablets, and wearables. The number of worldwide smartphones sales is still increasing according to Gartner statistics, reaching 1.55 billion units in 2018 [7]. The crowd analytics market is projected to reach USD 1 142.5 million by 2021, raising from USD 385.1 million of 2016 at a compound annual growth rate of 24.3% [11].

The success of a MCS campaign typically relies on large participation of users [14]. Unfortunately, often it is not feasible to develop testbeds and platforms that involve a multitude of citizens[1]. On the one hand, the cost of recruitment scales with the number of users involved and the amount of data collected. On the other hand, the required time for setting up a large-scale sensing campaign is prohibitively long. To this end, simulators offer a valid alternative to assess the performance of MCS systems in city-wide scenarios with large user participation in a reasonable time. Specifically, simulators are well-suited to assess and compare the performance of specific aspects of MCS systems (e.g., the decision process to sense and report data).

In this work, we present CrowdSenSim 2.0, a stateful simulation platform for developing MCS systems in urban environments for Smart Cities applications. The simulator has been developed in order to be general-purpose, i.e., to cover many use cases and architectures as well as implementing several MCS aspects such as energy, coverage, realistic user mobility, real urban environments, communication infrastructures, recruitment, and data collection

[1]In the rest of the paper, we will use the terms users, citizens, and participants interchangeably

algorithms. CrowdSenSim 2.0 is based on the architecture of CrowdSenSim [5]. From the original version, we kept its core architecture and re-implemented almost integrally the simulator engine, besides several other improvements. As a matter of fact, the legacy version of CrowdSenSim can simulate with a high level of detail MCS systems in urban scenarios and assess the energy consumption of mobile devices. However, it lacks adaptation to several MCS applications that require features such as statefulness and flexible event triggering. Indeed, as it will be explained in § 3, the original CrowdSenSim featured only stateless use cases and was oriented to model network and energy consumption characteristics rather than algorithmic ones.

With respect to the original CrowdSenSim [5], we make the following contributions:

(1) We significantly improve the original platform by implementing a set of crucial features tailored to embrace a larger class of MCS algorithms and frameworks. In a nutshell, CrowdSenSim 2.0 supports stateful simulations (i.e., where the simulation events are chronologically dependent and the algorithm operation relies on such dependence), MGRS spatial encoding, a flexible time interval for event generation, and the integration of a new algorithm to determine user trajectories.
(2) We optimize the computational performance by means of a full code refactoring and the introduction of algorithm-level parallelism, which enables researchers to run several MCS algorithms simultaneously or several runs of the same algorithm at the same time.

Furthermore, besides the above contributions directly inherent to the simulation platform, the paper makes these additional contributions:

- We validate the benefits CrowdSenSim 2.0 brings in terms of runtime execution and memory utilization.
- We present as use case an analysis of a stateful distributed data collection algorithm implemented in CrowdSenSim 2.0.

In conclusion, the paper has the following structure: Section 2 outlines the main research efforts in the area, Section 3 describes the simulator with particular focus on the improvements, Section 4 validates its computational performance in comparison to CrowdSenSim, Section 5 describes the distributed data collection algorithm that we integrated, implementation details of such algorithm, and its results. Finally, Section 6 concludes the work.

2 RELATED WORKS

After having scanned the state-of-the-art thoroughly, it has become evident that it does not exist a simulation tool that covers all the components of MCS. This is because MCS is a general paradigm which groups highly heterogeneous components. For example, data collection can be opportunistic or participatory according to the degree of user involvement. Users might contribute freely to a sensing campaign or can be recruited through specific policies. The objective of MCS research spans over a number of areas, including quality and coverage of information over an area of interest, user recruitment, and incentive mechanisms [3, 17]. Thus, such research areas typically propose optimization frameworks or algorithms that are evaluated standalone and often leave apart important components that impact on the correct modeling. These components include realistic user mobility [16] and modeling of urban environments [1] as well as modeling of the network that transfers sensing readings from end-users to the cloud where it is typically processed.

For the above reasons, a number of proposed simulation platforms are not suitable to properly evaluate MCS systems because they typically focus on one component at a time [9]. For example, in [19] the authors propose to leverage the capabilities of Network Simulator 3 (NS-3) to simulate ad-hoc scenarios for reporting incidents. NS-3 is a highly detailed simulation tool for networking purposes and models network protocols down to the granularity of the single packet across all the layers of the network stack. This strongly limits scalability, as the level of detail in such simulations is too high and modeling typical MCS sensing campaigns with thousands of users overall contributing during hours/days timescale becomes prohibitive. The same applies to similar simulation tools such as OMNeT++, used in [18]. CupCarbon, proposed in [12], is a WSN-based simulator in which the researcher can individually deploy both sensors and base stations on realistic urban environments obtained from OpenStreetMap (OSM)[2]. Sensors can be mobile and can have dedicated paths along the roads, which makes it suitable for MCS scenarios. However, CupCarbon limits the size of the scenario, which precludes scalability to thousands of nodes. The most notable effort in the last years is given by CrowdSenSim [5], a simulator for MCS scenario capable of supporting a high number of users (order of hundreds of thousands) and their motion along the roads of cities imported by OSM without modeling in full the network stack, yet providing a sufficient level of detail on battery consumption statistics and number of tasks executed. The focus of the simulator is heavily energy-driven, implementing a number of algorithms, both in participatory and opportunistic scenarios, aiming to reduce the energy consumption per device. Being primarily implemented for energy consumption oriented scenarios, CrowdSenSim lacks adaptability to many MCS use cases as it does not support a number of features, such as statefulness, that are required by the majority of MCS systems.

3 THE CROWDSENSIM 2.0 ARCHITECTURE

This section presents the architecture of CrowdSenSim 2.0 by highlighting its novelties over the original CrowdSenSim. In particular, we detail the architecture of the simulator outlining the role of each module and we expose the new features and improvements.

3.1 General Architecture

Figure 1 shows the architecture of CrowdSenSim 2.0, which includes major modifications and added features to the previous version of CrowdSenSim. Main novel contributions include a stateful approach that is fundamental for specific classes of MCS applications, support for a more flexible generation of events in terms of temporal granularity and other configurable parameters, MGRS spatial encoding, and generation of highly-precise user trajectories. These features will be explained in detail throughout the section.

The simulator generates a set of participants moving within a street network, contributing data through the sensors of their

[2]https://www.openstreetmap.org/

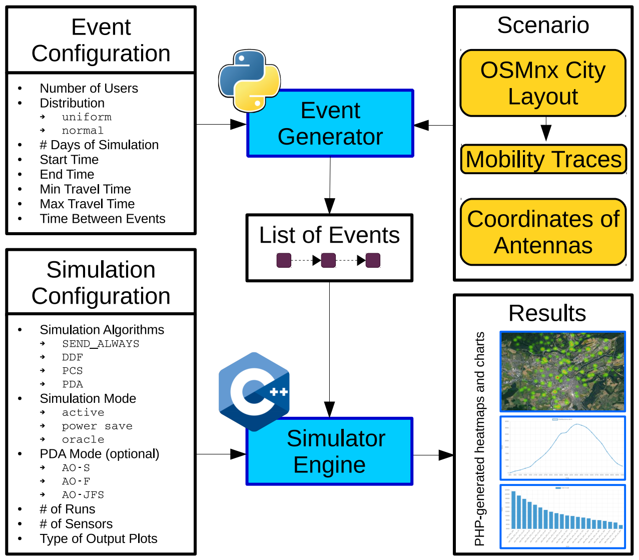


Figure 1: Simulator modular architecture, including original features of CrowdSenSim and novelties of CrowdSenSim 2.0.

mobile devices, and reporting it through the closest cellular base station or WiFi access point, according to the design of the MCS campaign. The Event Generator module consists of creating events, defined as "the arrival of a participant in a given location at a defined time". To this end, it takes in input the City Layout, the User Mobility, the Coordinates of the Antennas, and a set of parameters from the Event Configuration file. After such a macro step has been performed, the list of events is passed to the Simulator Engine, which defines the behavior of each participant upon each event. Both the Simulator Engine and the Event Generator have been integrally rewritten to make possible the integration of a set of necessary functionalities, which are explained thereafter. Additionally, significantly more code cleanliness was enforced.

3.2 City Layout

The City Layout module allows researchers to define the urban street network of a city-wide scenario over which the participants move. The street network is defined as a set of coordinates where pedestrians can be located, including *latitude*, *longitude*, and *altitude*.

3.2.1 High-precision street network design. While CrowdSenSim received as input a *.txt* file with a list of all coordinates to generate the city layout, CrowdSenSim 2.0 automatically gets the coordinates by exploiting OSMnx, an open-source Python package to download and simplify street networks from OSM [2]. Furthermore, CrowdSenSim 2.0 implements the AOP algorithm [21] to augment the precision of the graph describing the street network, with a granularity chosen according to the needs and the objectives of the MCS campaign under study. Figure 2(a) shows the map of the city layout and the street network where users can walk. Pedestrian movement is generated over the points, which correspond to the set of downloaded coordinates.

3.2.2 MGRS Support. CrowdSenSim 2.0, in addition, supports Military Grid Reference System (MGRS) spatial encoding [8], which allows the developer to design data collection algorithms on top of such hierarchical spatial encoding. As a matter of fact, the usage of MGRS is crucial in data collection algorithms for MCS and several applications are built on top of it [6, 10, 15]. In particular, each event is generated along with its MGRS coordinates with the finest possible granularity and such data is then passed to the Simulator Engine module for processing.

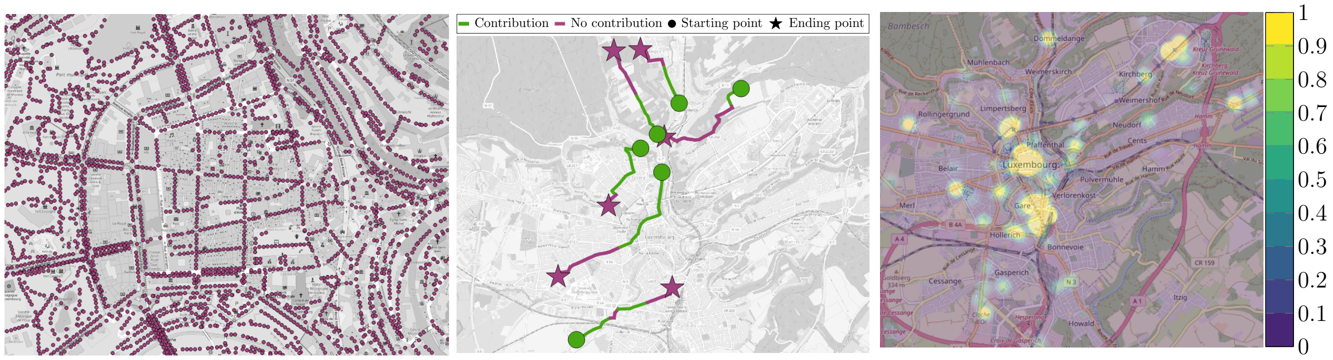
3.3 User Mobility and Event Generation

The User Mobility module defines the initial user placement, at what time they "spawn" in the urban environment and how they move. Users are generated using a spatial distribution function and move according to different possible models. For instance, mobility can be uniformly or randomly distributed, based on real traces, or built upon different weights according to the point of interests and time of the day (e.g., following the distribution of Google Popular Times[3]). Each participant has a certain travel time (e.g., 20 min walk) and its trajectory is generated consequentially as a sequence of events, defined in § 3.1, equally spaced in time. After an event is generated, the participant jumps to a location over the urban network topology reachable in a certain time given its walking speed, which is generated uniformly within the interval 1 - 1.5 m/s.

3.3.1 User Trajectories. User mobility is generated as pedestrian trajectories with a random start and end point according to the walking period of each citizen over the street network of the chosen city. This feature allows highlighting the periods of active contribution of users along their paths according to the data collection framework (DCF) under analysis. For instance, Figure 2(b) illustrates the trajectories of 5 participants walking in Luxembourg City and contributing with a Deterministic Distributed Framework (DDF) [4] in which users stop the sensing process after a certain amount of collected data. The circle represents the starting walking point and the star the ending point. The green path indicates when users contribute data, the purple one when they do not sense data. Each user sends data to the closest cellular base station (BS) or WiFi access point (AP) according to the chosen communication technology. For instance, Figure 2(c) shows the concentration of users connected to each BS in Luxembourg City at a given instance of time of the simulation runtime.

3.3.2 Event Configuration. In CrowdSenSim 2.0 many options in the generation of events have been made configurable (see the block "Event Configuration" in Figure 1). The distribution function for generating users can be selected when configuring the simulation, e.g., a normal or uniform distribution, whereas in the old version users were generated only uniformly. By selecting among various generation functions, it is now possible to simulate different density of the users throughout the simulation runtime. The amount of time each user moves in the urban environment can be configured like in the original CrowdSenSim. The time interval Δt between two events has been made configurable as well, while in CrowdSenSim was fixed to 60s. This enables to simulate possibly more complex scenarios, in which the time between updates is not decided at

[3]https://support.google.com/business/answer/2721884



(a) City Layout

(b) User trajectories

(c) Heatmap

Figure 2: City layout, user trajectories and distribution of users connected to BSs in Luxembourg City

design time, but can change through configuration. This makes the number of supported applications significantly higher.

3.4 Simulator Engine

The simulator engine is written in C++ and, as shown in Figure 1, it takes in input a list of events, corresponding to the time in which users perform an action. In turn, each event triggers the sampling of each sensor as well as the communication module of each device because the Simulator Engine implements an apposite callback function. In practice, the Simulator Engine defines the behavior of each participant by implementing the action performed upon each event.

3.4.1 Global Statefulness. CrowdSenSim 2.0 executes the events in absolute chronological order. To make the present CrowdSenSim 2.0 simulator more oriented to the algorithmic side rather than energy consumption of MCS systems, we implemented the Simulator Engine in a way in which events are ordered chronologically, whereas, in the previous version, events were executed per-participant – i.e., all the events of the first participant were executed before all the events of the second, and so on – making the implementations of certain algorithms unfeasible. In fact, the original CrowdSenSim could implement any stateless algorithms as well as any algorithm requiring only local statefulness, i.e., a state maintained only internally by each participant and does not interact with other participants in any case. With the novel version of the simulator presented in this paper, CrowdSenSim 2.0, we can implement any algorithm with global statefulness, i.e., the state is maintained by each participant as well as a central entity, and each event can be influenced by the past ones. Note that this increases the expressiveness of the simulator without preventing data collection algorithms that were implementable in the past version to be also implemented in CrowdSenSim 2.0.

3.4.2 Algorithm-level Data Collection Parallelism. Comparing the performance of multiple algorithms (e.g., for data collection) at a time is often a desirable feature in simulation platforms. CrowdSenSim 2.0 makes it possible to run different algorithms within the same run, simply by defining more than one callback function relative to each event. More in detail, any time the Simulator Engine is triggered upon the occurrence of an event, it is possible to specify more than one function to be called separately. In this way, it is possible to define a number of algorithms to run at the same time, that will output their results separately just as if they were separate runs. Obviously, one can also run the same algorithm several times in parallel with a different random seed. Such advance boosts the performance of the simulator significantly, as in a single run several algorithms can be tested at the same time. We show performance evaluation of this aspect in § 4. As shown in Figure 1, parameters such as the algorithms used as well as the number of runs can be specified in the Simulation Configuration file. The simulator implements DDF, PCS, and PDA algorithms as in [20], in particular, PDA has been re-designed in its global stateful version [13] (§ 5).

3.4.3 Participant Awareness. To enable applications and algorithms that require privacy or fine-grained energy savings mechanisms, CrowdSenSim 2.0 allows the simulated users to be aware of certain information detained by the central entity organizing the sensing campaign. For example, instructions about the amount of yet to be delivered information in a certain area. Therefore, users can be in:

- **Power-save mode**, thus eligible to receive such information when it is piggybacked on another communication (e.g., when they are pushing data).
- **Active mode**, thus eligible to receive such information upon each of their events occur.
- **Oracle mode**, thus aware of such information at any time.

When looking at such division from a privacy perspective, users in power-save mode are those with limited access to global information because they are not trustworthy, whereas users in oracle mode have higher privileges. Clearly, the additional state may be implemented in case the simulated scenario requires it.

4 PERFORMANCE EVALUATION

The newly developed CrowdSenSim 2.0 has undergone a complete code refactoring procedure as detailed in Section 3. Such operation has been extremely delicate as we needed to assure that every feature of the original CrowdSenSim was left intact. In other words, the novel simulator should be retro compatible with the previous implementation, to achieve results reproducibility regardless of the simulator version used. In this section, we show that both CrowdSenSim and CrowdSenSim 2.0 exhibit the same behavior on common use cases and we highlight the benefits in terms of RAM utilization that the new version brings.

4.1 Validation of CrowdSenSim 2.0 over CrowdSenSim

Both the instances of CrowdSenSim and CrowdSenSim 2.0 that we used throughout the paper have been launched on a virtual machine using 1 core of the host machine with 4 GB of dedicated RAM and running Ubuntu 16.04.6 LTS. The host machine is an AMD Ryzen 5 1600 at 3.2 GHz (6 core, 12 thread) with 16 GB RAM and running Windows 10 Pro 1809.

In order to efficiently validate CrowdSenSim 2.0, we referred to an energy consumption analysis of the DDF data collection algorithm originally proposed in [4] that was implemented and practically evaluated in [20]. DDF is a locally stateful data collection algorithm in which participants keep on generating data up to a certain threshold of energy consumption depending on their battery capacity. For the sake of energy-related analysis, we left the energy calculation of the original CrowdSenSim untouched. In detail, we equipped each participant with a mobile device carrying an accelerometer, a pressure sensor, and a temperature sensor. As in [5], the sensors generate readings with the same sampling frequency. For all the simulations, we resort to 10 000 participants in the center of Luxembourg City, which covers an area of 51.73 km^2 with a perimeter of 52.5 km and a population of 119 214 inhabitants as of the end of 2018. The simulation has a duration of 12-hours (starting at 12:00 PM and ending at 11:59 PM) and paths are generated with a duration uniformly distributed between 20min and 40min.

We fed both CrowdSenSim and CrowdSenSim 2.0 with the same set of events, which are sampled per-participant with a frequency of 60s. We ran extensive simulations and measured the current drain of the device of each participant in relation to the sensing and reporting activity, and we plot the results in Figure 3. Devices use the WiFi technology for communication as in [5]: a number of WiFi hotspots are deployed in the area of interest, and every time a participant needs to transmit it sends data through the closest AP in the map. We observed that both CrowdSenSim and CrowdSenSim 2.0 generate the exact same values, which proves their equivalence in terms of results output.

In order to further strengthen our claim, we also compared the transmission consumption between the two simulators. As we did for the sensors, we did not modify the way in which participants transmit data. Again the values produced by the two simulators match perfectly, validating their equivalence.

4.2 Performance Analysis of Runtime Execution and Memory Utilization

The code refactoring and the parallel processing feature brought a significant boost in the simulation performance in terms of the time of execution and memory consumption.

Figure 4 shows the simulator runtime in seconds on top of the number of algorithms run. Even in one single run, CrowdSenSim 2.0 achieves a lower run time than CrowdSenSim (7s in average against 8s). Furthermore, as CrowdSenSim 2.0 allows for multiple algorithms to run in parallel – or multiple runs of the same algorithm, – the time execution remains almost constant whereas in the original CrowdSenSim it scales linearly. Indeed, to perform multiple runs, the original CrowdSenSim needs to be launched several times sequentially.

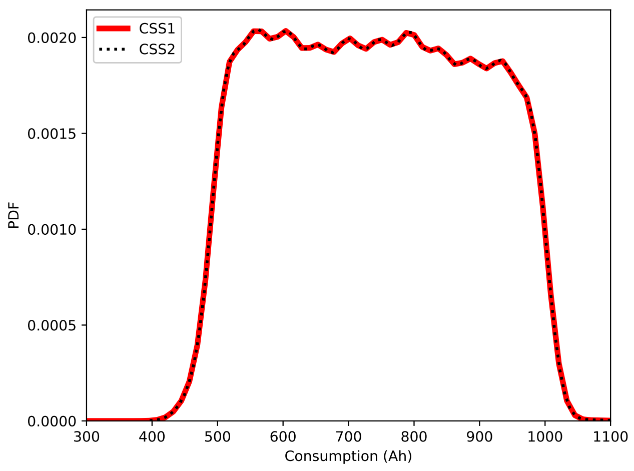


Figure 3: Density of the energy consumed by participants with CrowdSenSim and CrowdSenSim 2.0.

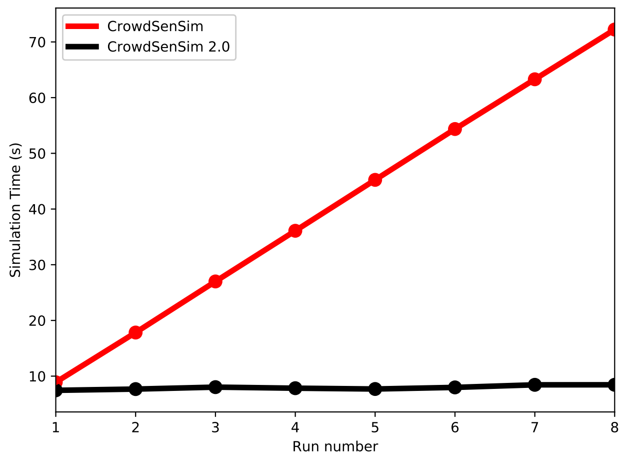


Figure 4: Runtime execution of DDF with multiple runs.

Figure 5 shows in boxplot form the performance of CrowdSenSim and CrowdSenSim 2.0 in terms of RAM consumption. For a fair comparison, we used only one algorithm at a time. CrowdSenSim 2.0 outperforms CrowdSenSim significantly. This is due to several code optimizations, such as the use of integers as identifiers instead of strings.

5 CASE STUDY: A STATEFUL DISTRIBUTED OPPORTUNISTIC ALGORITHM

In this section, we outline our Asymptotic Opportunistic algorithm for Joint Fairness and Satisfaction index (AO-JFS) that was initially

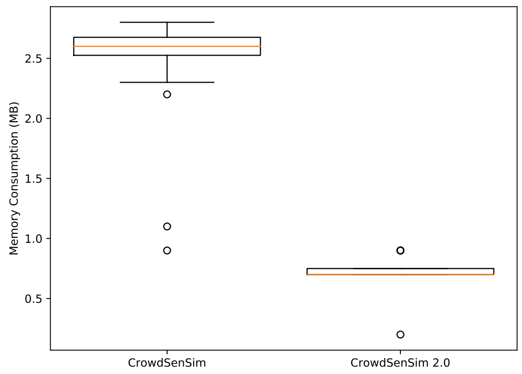


Figure 5: Performance of CrowdSenSim and CrowdSenSim 2.0 in terms of RAM consumption.

proposed in [13] along with its simplified versions AO-F and AO-S, both implemented in CrowdsSenSim 2.0 as members of a family of algorithms called Probabilistic Distributed Algorithms (PDA). Originally, performance evaluation was conducted using an ad-hoc simulator. The algorithm has been designed for data collection control, that is, preventing the whole scenario from generating too much or excessively less data. Too less data would result in a poor mapping of a phenomenon on an urban (or rural) environment, whereas too much data may result in too much noise to get rid of as well as an unbearable amount of users to reward for data that is much more than required and, consequently, an unnecessary energy consumption. We model such scenario in a push-based and totally anonymous distributed algorithm in which the central entity has no direct control over the single users – i.e., it cannot specifically query users about certain resources – neither it has knowledge of where the users are and how many of them are contributing. Participants can contribute pushing their data at a defined frequency to the central entity, which will only reply with a Satisfaction Index (*SI*), a number that resembles how much the server is "satisfied" about the number of observations received in a defined time window about a resource. Consequentially, participants decide with a probabilistic distribution whether to contribute or not at the following time slot on top of the received *SI*: if the satisfaction is low, they are pushed to contribute more, if it is high, their contribution is discouraged.

A similar version of such algorithm was implemented in the original CrowdSenSim [20] and called PDA, although CrowdSenSim 2.0 for its stateful approach would have been required to assess AO-JFS properly. Indeed, AO-JFS requires the central entity to be aware of all data delivered by the participants at each instant of time in order to correctly calculate the *SI*. With the original CrowdSenSim, each participant executes all its events completely before the events of another participant can take place. Hence, the chronological order of the events is violated. Therefore, CrowdSenSim 2.0 with its stateful approach is necessary for any MCS algorithm that heavily depends on the chronological sequence of the events. In the rest of the section, we will outline in detail the behavior of AO-JFS.

5.1 AO-JFS Core Idea

We can model the problem as N different stations (we use the terms stations, users, and participants interchangeably) that adhere to the MCS campaign and perform observations against a given phenomenon. Such number N can vary over time due to mobility, in particular, participants may leave the interested area, whereas new ones may join it. We assume to split our timeline in time slices Δt_i, that represent the atomic units during which a station cannot transmit more than once due to internal clocks. We also assume that the stations will send observations relative to v certain resources $\Psi_0, \ldots, \Psi_v$ periodically. The central entity's goal is to obtain exactly M_j observations about Ψ_j for $j \in [0, v]$ within every time window T_i, the length of which is given by $|T| = w$. We follow the approach of the sliding window, thus $T_i = \{\Delta t_{i-w}, \ldots, \Delta t_i\}$, this means that T_i and T_{i-1} are overlapping by $w-1$ time slots. The central entity displays the performances of the data collection process through the above-cited *SI*. Such value is calculated upon each time window T_i and it is defined as $SI_{i,j} = \frac{m_{i,j}}{M_j}$, where $m_{i,j}$ is the actual number of observations received by the central entity within T_i for the resource Ψ_j. The aim of the central entity is to obtain a *SI* equal to 1 for each resource.

We assume that each participant knows the *SI* values at all times (i.e., the central entity broadcasts the *SI* constantly) and, for each atomic time slot, performs a decision of whether to send or not the local measurement of the sensor (for each resource). In detail, at the time Δt_{i+1}, each participant calculates a probability of sending the measurement relative to the sensor Ψ_j that is basically the inverse of the received $SI_{i,j}$, therefore $P_{i,j} = 1 - CSI_{i-1,j}$ with CSI_i, j being the Constrained Satisfaction Index:

$$CSI_{i,j} = \begin{cases} \epsilon & \text{if } SI_{i,j} < \epsilon, \\ 1-\epsilon & \text{if } SI_{i,j} > 1-\epsilon, \\ SI_{i,j} & \text{otherwise,} \end{cases} \quad (1)$$

with ϵ being a very small number (in our case 0.001). This forces the *SI* to range from a very small number close to 0 to a number close to 1 for the purpose of probability calculation.

5.2 Boosting Runtime Execution

To prevent contributions to stabilize at a too low or too high *SI*, we introduced boosters for the probability calculation. We define a booster as an exponent E to which we elevate the *CSI* in the probability calculation: $P_{i,j} = 1 - CSI^{E}_{i-1,j}$. Figure 6 shows the probability curve for different values of E (the central straight line

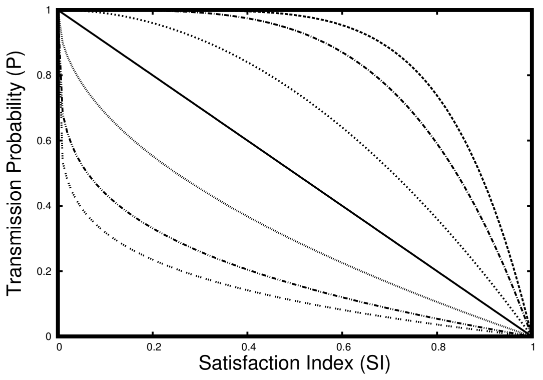


Figure 6: Base probability plus different booster values

is obviously for $E = 1$). Specifically, we introduce an overall booster value b_j and an individual value k_j per sensor.

Suppose the aim is to maintain SI in the range $SI \in [\sqcap_{SI}; \sqcup_{SI}]$, where the bounds are, for example, set as 0.95 and 1.15. Then $\forall i, j$, if $SI_{i,j} > \sqcap_{SI}$ then $b_j = dec(b_j)$, whereas if $SI_{i,j} < \sqcup_{SI}$ then $b_j = inc(b_j)$ with:

$$inc(b) = \begin{cases} \frac{1}{(1/b)-1} & \text{if } b < 1, \\ b + 1 & \text{otherwise.} \end{cases} \tag{2}$$

$$dec(b) = \begin{cases} b - 1 & \text{if } b > 1, \\ \frac{1}{(1/b)+1} & \text{otherwise.} \end{cases} \tag{3}$$

k is the attempting factor, calculated individually by each station on top of the received b as

$$k_j = \begin{cases} \lfloor \log_2(\eta) \rfloor & \text{if } b_j < 1, \\ \eta \cdot b_j & \text{if } b_j \geq 1, \end{cases} \tag{4}$$

where $\eta > 0$ is the number of Δt_i slots elapsed since the last transmission. In the end, the probability is calculated as

$$P_{i,j} = 1 - CSI_{i-1,j}^{inc^k(b)}. \tag{5}$$

Note that the term $f^n(x)$ indicates the iterative composition as $f^n(x) = f \circ f^{n-1}(x)$.

5.3 Implementation and Testing of AO-JFS

We outline how to integrate the AO-JFS algorithm (§ 5) with CrowdSenSim 2.0. AO-JFS is a distributed algorithm where the active part is given by the participants and the central entity is only "reactive". In more details, each event generated by the event generator on the map triggers an action by a participant. Since events are processed in chronological order, each of them depends on the sequence of events previously occurred. Thus CrowdSenSim 2.0 can be employed for its analysis. We implemented AO-JFS only in Active Mode, which means that each participant receives information about the status of the SI once every time slot Δt. Upon the occurrence of an event, each participant at time slot t_i (for each sensor j, with $j \in [0, v]$):

(1) Retrieves the known value of the $SI_{i,j}$ and transform it to a $CSI_{i,j}$ for the purpose of the probability calculation using Equation 1.
(2) Retrieves the known value of the global booster b.
(3) Sets the local attempting factor k using Equation 4.
(4) Calculates the actual probability $P_{i,j}$ to send the observation for the resource Ψ_j using Equation 5.

Therefore, events for an AO-JFS simulation are merely the instants in which a participant decides whether to send observations or not. When all the events for t_i occurred, then the values of the SI are updated before taking into account the next time slot.

Our implementation of AO-JFS is evaluated for 2 hours long simulations in the center of Luxembourg City. Parallelism is used to test 50 runs of AO-JFS at the same time, each of them using a different random seed. Δt is set to 10s and $w = 30$, therefore the time window T is 5min. We set $v = 3$, in particular, we used the three sensors mentioned in § 4 as our resources, and fixed the desired amount of observations as $M_1 = 7500, M_2 = 5000, M_3 = 2500$. We generated users using a uniform distribution and set the total number G as 2500, 5000 and 10000. Figure 7 shows the number of

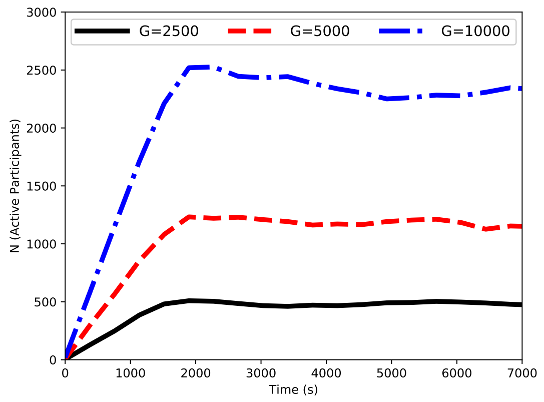


Figure 7: Number of active users N over time for 3 different values of generated users G (G=2500, G=5000, G=10000).

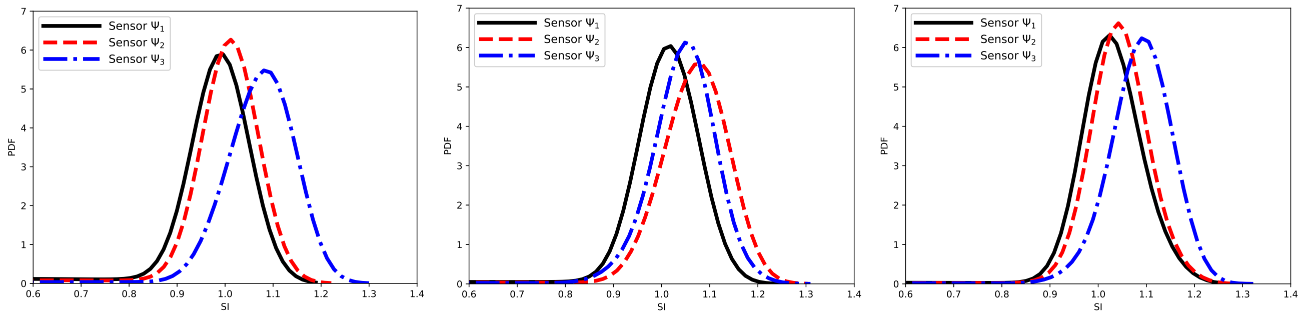
active users N over time. As the distribution is uniform, N tends to reach a steady state after an initial transient.

Figure 8 shows the results of the simulations. In particular, the values of the SI for the three sensors Ψ_1, Ψ_2, and Ψ_3 are shown for each configuration in the form of a Probability Distribution Function (PDF) in which the data points are the value of the SI sampled each Δt. For each sensor, the SI value stabilizes around 1, which is the goal of AO-JFS. In more details, Figure 8(b) shows the behavior of the SI at $G = 5000$, with the number of active users over time N floating around 1200. This number is in a good balance with the number of required observations, in fact the SI value for all the sensors tends to cluster almost regularly around 1. Figure 8(c) shows the behavior of the SI with $G = 10000$, with N settling around 2300. This number is very high in comparison to the number of required measurements. Therefore the effort of AO-JFS is in limiting the number of contributions by the participants. This is even more evident for Ψ_3, the resource for which the fewer contributions are needed, as the respective SI values tend to cluster at a slightly higher value (i.e., around 1.1). On the other hand, Figure 8(a) shows the SI for $G = 2500$ and, consequently, N floating around 500, which is a low number in comparison to the contribution required. Although the behavior of the SI is quite similar to the other plots, we can appreciate a slight difference for Ψ_3, as its values are more spread. This is due to b being quite high: as the participants are pushed to contribute more, the probability curve gets very high for most of the CSI values in input and causes more likely peaks and troughs in the contribution over time.

6 CONCLUSIONS

This paper presents CrowdSenSim 2.0[4], which extends with major advances the existing CrowdSenSim platform for simulations of MCS activities in realistic urban environments. CrowdSenSim 2.0 exhibits two main novel aspects. First, the simulator features a stateful approach that enforces all events to be executed in chronological order with a higher fine-grained temporal resolution. Second, it features two models to generate the city layouts over realistic street networks where users move, based on the popular OSM and the

[4]We make publicly available all the source files and scripts of CrowdSenSim 2.0 under https://crowdsensim.gforge.uni.lu/download.html.



(a) G = 2500 (b) G = 5000 (c) G = 10000

Figure 8: PDF of the SI for the three sensors for different values of G.

Military Grid Reference System. In a nutshell, other advances include extensions of the architectural modules, code refactoring, and algorithms' parallel execution that boost performance by making significantly lower the runtime execution and memory utilization. This clearly enables the simulation of larger scale scenarios, which is of paramount importance for research in MCS. We demonstrate that when feeding CrowdSenSim 2.0 and the original CrowdSenSim with the same list of events, they perform identically in terms of mobile device energy consumed for sensing and reporting, thus making CrowdSenSim 2.0 compatible with previous studies. As an example of use case, we showcase the performance evaluation of a distributed opportunistic algorithm for data collection that shows how the stateful approach is fundamental for specific applications.

ACKNOWLEDGMENT

Dr. Fiandrino's work is supported by the Juan de la Cierva grant from the Spanish Ministry of Economy and Competitiveness (FJCI-2017-32309).

REFERENCES

[1] Luca Bedogni, Marco Fiore, and Christian Glacet. 2018. Temporal Reachability in Vehicular Networks. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*. 81–89. https://doi.org/10.1109/INFOCOM.2018.8486393

[2] Geoff Boeing. 2017. OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems* 65 (2017), 126–139. https://doi.org/10.1016/j.compenvurbsys.2017.05.004

[3] Andrea Capponi, Claudio Fiandrino, Burak Kantarci, Luca Foschini, Dzmitry Kliazovich, and Pascal Bouvry. 2019. A Survey on Mobile Crowdsensing Systems: Challenges, Solutions and Opportunities. *IEEE Communications Surveys Tutorials* (May 2019), 1–49. https://doi.org/10.1109/COMST.2019.2914030

[4] Andrea Capponi, Claudio Fiandrino, Dzmitry Kliazovich, Pascal Bouvry, and Stefano Giordano. 2017. A Cost-Effective Distributed Framework for Data Collection in Cloud-Based Mobile Crowd Sensing Architectures. *IEEE Transactions on Sustainable Computing* 2, 1 (Jan 2017), 3–16. https://doi.org/10.1109/TSUSC.2017.2666043

[5] Claudio Fiandrino, Andrea Capponi, Giuseppe Cacciatore, Dzmitry Kliazovich, Ulrich Sorger, Pascal Bouvry, Burak Kantarci, Fabrizio Granelli, and Stefano Giordano. 2017. Crowdsensim: a simulation platform for mobile crowdsensing in realistic urban environments. *IEEE Access* 5 (Feb 2017), 3490–3503. https://doi.org/10.1109/ACCESS.2017.2671678

[6] Yali Gao, Xiaoyong Li, Jirui Li, and Yunquan Gao. 2017. A dynamic-trust-based recruitment framework for mobile crowd sensing. In *Proc. of IEEE International Conference on Communications (ICC)*. 1–6. https://doi.org/10.1109/ICC.2017.7997420

[7] Gartner. 2019. Gartner Says Global Smartphone Sales Stalled in the Fourth Quarter of 2018. https://www.gartner.com/en/newsroom/press-releases/2019-02-21-gartner-says-global-smartphone-sales-stalled-in-the-fourth-quart

[8] Raino Lampinen. 2001. Universal transverse mercator (UTM) and military grid reference system (MGRS).

[9] Imre Lendák and Károly Farkas. 2015. Evaluation of simulation engines for crowdsensing activities. (2015).

[10] Martina Marjanović, Lea Skorin-Kapov, Krešimir Pripužić, Aleksandar Antonić, and Ivana Podnar Žarko. 2016. Energy-aware and quality-driven sensor management for green mobile crowd sensing. *Journal of Network and Computer Applications* 59 (Jan 2016), 95–108. https://doi.org/10.1016/j.jnca.2015.06.023

[11] MarketsandMarkets. 2018. Crowd analytics market worth $1,142.5 million by 2021. https://www.marketsandmarkets.com/PressReleases/crowd-analytics.asp

[12] Kamal Mehdi, Massinissa Lounis, Ahcène Bounceur, and Tahar Kechadi. 2014. Cupcarbon: A multi-agent and discrete event wireless sensor network design and simulation tool. In *Proc. of 7th International ICST Conference on Simulation Tools and Techniques*. 126–131. https://doi.org/10.4108/icst.simutools.2014.254811

[13] Federico Montori, Luca Bedogni, and Luciano Bononi. 2017. Distributed data collection control in opportunistic mobile crowdsensing. In *Proc. of ACM SMARTOBJECTS*. 19–24. https://doi.org/10.1145/3127502.3127509

[14] Federico Montori, Luca Bedogni, and Luciano Bononi. 2018. A Collaborative Internet of Things Architecture for Smart Cities and Environmental Monitoring. *IEEE Internet of Things Journal* 5, 2 (Apr 2018), 592–605. https://doi.org/10.1109/JIOT.2017.2720855

[15] Federico Montori, Luca Bedogni, Alain Di Chiappari, and Luciano Bononi. 2016. SenSquare: A mobile crowdsensing architecture for smart cities. In *Proc. of IEEE 3rd World Forum on Internet of Things (WF-IoT)*. 536–541. https://doi.org/10.1109/WF-IoT.2016.7845471

[16] Federico Montori, Marco Gramaglia, Luca Bedogni, Marco Fiore, Farid Sheikh, Luciano Bononi, and Andrea Vesco. 2017. Automotive Communications in LTE: A Simulation-Based Performance Study. In *Proc. of IEEE 86th Vehicular Technology Conference (VTC-Fall)*. 1–6. https://doi.org/10.1109/VTCFall.2017.8288297

[17] Federico Montori, Prem Prakash Jayaraman, Ali Yavari, Alireza Hassani, and Dimitrios Georgakopoulos. 2018. The Curse of Sensing: Survey of techniques and challenges to cope with sparse and dense data in mobile crowd sensing for Internet of Things. *Pervasive and Mobile Computing* 49 (Jul 2018), 111–125. https://doi.org/10.1016/j.pmcj.2018.06.009

[18] Riccardo Pinciroli and Salvatore Distefano. 2017. Characterization and evaluation of mobile crowdsensing performance and energy indicators. *ACM SIGMETRICS Performance Evaluation Review* 44, 4 (2017), 80–90. https://doi.org/10.1145/3092819.3092829

[19] Cristian Tanas and Jordi Herrera-Joancomartí. 2013. Crowdsensing simulation using ns-3. In *International Workshop on Citizen in Sensor Networks*. Springer, 47–58. https://doi.org/10.1007/978-3-319-04178-0_5

[20] Mattia Tomasoni, Andrea Capponi, Claudio Fiandrino, Dzmitry Kliazovich, Fabrizio Granelli, and Pascal Bouvry. 2018. Why energy matters? Profiling energy consumption of mobile crowdsensing data collection frameworks. *Pervasive and Mobile Computing* 51 (2018), 193–208. https://doi.org/10.1016/j.pmcj.2018.10.002

[21] Piergiorgio Vitello, Andrea Capponi, Claudio Fiandrino, Paolo Giaccone, Dzmitry Kliazovich, and Pascal Bouvry. 2018. High-precision design of pedestrian mobility for smart city simulators. In *Proc. of IEEE International Conference on Communications (ICC)*. 1–6. https://doi.org/10.1109/ICC.2018.8422599