Characterization and Evaluation of Mobile CrowdSensing Performance and Energy Indicators

Riccardo Pinciroli
Politecnico di Milano
Via Ponzio 34/5, 20133 Milano, Italy.
riccardo.pinciroli@polimi.it

Salvatore Distefano
Università di Messina
Contrada di Dio, S. Agata, 98166 Messina, Italy.
sdistefano@unime.it
Kazan Federal University
35 Kremlyovskaya street, 420008 Kazan, Russia.
sdistefano@kpfu.ru

ABSTRACT

Mobile Crowdsensing (MCS) is a contribution-based paradigm involving mobiles in pervasive application deployment and operation, pushed by the evergrowing and widespread dissemination of personal devices. Nevertheless, MCS is still lacking of some key features to become a disruptive paradigm. Among others, control on performance and reliability, mainly due to the contribution churning. For mitigating the impact of churning, several policies such as redundancy, over-provisioning and checkpointing can be adopted but, to properly design and evaluate such policies, specific techniques and tools are required.

This paper attempts to fill this gap by proposing a new technique for the evaluation of relevant performance and energy figures of merit for MCS systems. It allows to get insights on them from three different perspectives: end users, contributors and service providers. Based on queuing networks (QN), the proposed technique relaxes the assumptions of existing solutions allowing a stochastic characterization of underlying phenomena through general, non exponential distributions. To cope with the contribution churning it extends the QN semantics of a service station with variable number of servers, implementing proper mechanisms to manage the memory issues thus arising in the underlying process. This way, a preliminary validation of the proposed QN model against an analytic one and an in depth investigation also considering checkpointing have been performed through a case study.

Keywords

Mobile Crowdsensing; queuing networks; G/G/x; performance; energy consumption; checkpointing.

1. INTRODUCTION

Internet of Things (IoT) is gaining more and more importance thanks to recent advancements in mobile, pervasive sensing and communication technologies. It aims at connecting several physical objects into a unique and larger space, the *cyberspace* [21, 14], opening up new application scenarios that may potentially involve billions devices and users, a big challenge to properly address. A promising attempt in this direction is Mobile CrowdSensing (MCS), which proposes to deploy and run IoT applications on mobiles by actively involving their owners in a volunteer-/crowd-based fashion [13]. This way, MCS allows to reach a large number of users, which may also act as contributors sharing their devices to mainly provide sensing facilities.

Several success stories confirm the effectiveness of the crowdsourcing approach in IoT contexts [7, 10, 23], thus attracting interests from both academic and business communities that are investing resources and efforts to implement middleware mechanisms [25, 29] and to investigate on a commercial exploitation for MCS. This imposes to revise the MCS paradigm, extending its scope to business contexts where service level agreements on functional and non functional properties have to be enforced to meet quality of service (QoS) requirements. Therefore, proper mechanisms and tools for supporting the design and the operation of MCS services are required. In particular, modeling and evaluation techniques dealing with MCS contribution dynamics issues such as churning (i.e., the random and unpredictable join and leave process characterizing contributors) must be specified.

These arrival and service processes, mainly identifying the time behavior of an MCS system, have to be properly characterized indeed, recurring, when possible, to stochastic models able to adequately represent their fluctuations. This way, the resulting model is often not memoryless and the corresponding stochastic process is non-Markovian, implying to take into account related memory issues in its analysis. This is particularly true in MCS systems where the service requests are usually split in simple tasks then assigned to contributor nodes, whose processing could be interrupted due to churning, and thus rescheduled. Sometimes, the results obtained by the partial processing of an interrupted task are not wasted, e.g., for purely sensing activities, or also in the case the MCS services implement checkpointing policies to limit the impact of rescheduling on QoS and performance. To take into account this aspect in modeling, i.e., to properly represent restart from the conditions before the interruptions or from checkpoints, specific memory preservation mechanisms are required.

State space based solutions have been mainly proposed in literature to address these issues [9, 8, 17, 16, 22]. Nevertheless, the main drawback of the state space models is the well-known *"state space explosion"* when dealing with large-complex problems, limiting their applicability, especially in

This work is based on earlier work from the proceedings of the workshop InfQ2016 - New Frontiers in Quantitative Methods in Informatics, held in Taormina, Italy, October 2016.

MCS-IoT contexts where the number of requests and contributors could easily reach thousands or even millions. Furthermore, none of them except [9] takes into account checkpointing policies and related issues, allowing anyway to represent contribution dynamics and churning, but not dealing with the memory problem.

A solution to this problem has been proposed in [24], where we provided a technique based on QNs for dealing with all such issues altogether. In particular we introduced a new service station policy copying with variable number of servers explicitly adopted in the MCS system model to represent the contribution dynamics. Memory policies have been also implemented in order to allow the representation of checkpoint-restart processing, also allowing to consider three different perspectives, i.e., contributors, service provider and end-users, in the evaluation through specific performance and energy metrics.

This paper extends our previous work [9, 24] by improving the overall technique and the model they proposed. New features, such as the request decomposition into tasks, have been considered, revising the original model by introducing new elements (i.e., fork-join). Furthermore, the evaluation part has been significantly extended by new measurements and insigths, reporting on the validation experiments against a stochastic model (a continuous time Markov chain, CTMC) and another QN model developed and evaluated by the JMT tool [3] in the exponential case.

Details are showed in the remainder of this paper, organized as follows: Section 2 provides an overview of MCS, explaining the problem at hand. A QN-based modeling technique for MCS systems is proposed in Section 3, then adopted in Section 4 for the evaluation of an example also exploited to validate the model in the exponential case. Related work is overviewed in Section 5, while final discussions, remarks and cues for future work close this paper in Section 6.

2. A CLOSE-UP VIEW OF MCS

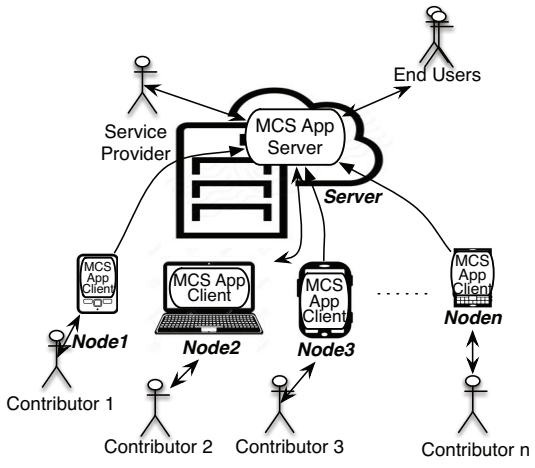

Figure 1: MCS application scenario.

MCS [13] is a computing paradigm where contributors share their mobile devices (e.g., smartphones, tablets, PDAs, laptops, etc.) with specific applications in order to gather and aggregate their data for further processing. From a high level perspective, the architecture of an MCS application mainly implements a client-server model as shown in Figure 1. The MCS application client deployed on each involved node (usually after it has been explicitly downloaded and installed by the contributor on his/her device) produces, (pre-)processes and sends data to the server that collects, aggregates, further processes (if required) and stores such data, providing information and results to the end-users. More specifically, three different stakeholders can be thus identified:

- *contributors*, i.e., the owners of the devices shared to build up the MCS sensing infrastructure;
- the *service provider*, which manages the whole system and application by gathering, aggregating and processing data and results from contributing nodes;
- *end-users*, which use the MCS application to access the services and the information it provides.

These roles are not mutually exclusive and, for example, contributors may also act as end-users and vice-versa.

In the MCS application processing, the main activities are decomposed into simpler *tasks* to be performed by the nodes independently. Each client may elaborate zero, one or many tasks, whose results are collected and recomposed by the application server. To prevent and minimize churning issues due to random and unpredictable joins and leaves of contributors, an active contributing node may periodically send partially processed data (i.e., checkpoints) to restart future elaborations on other available nodes in the case of its departure. Indeed, the application client can sometimes predict the leave of a contributor (e.g., in the case of low battery).

The MCS paradigm has been already successfully adopted in several applications such as OpenStreetmap [15], Waze [28], Nericell [23], as an effective solution for problems related to mobility [30, 10, 7], traffic monitoring [28, 23], public safety [2], Smart Cities [4] environment and pollution monitoring [30], emergency and crowd management [20], to mention but a few. Nevertheless, most of the potential of MCS is still untapped due to the limits of the contribution-based approach, often unreliable and unpredictable and therefore reflecting as high uncertainty in service fruition.

Thus, it is of strategic importance to focus on the overall MCS system rather than on a specific application, since the MCS application performance is strongly and mainly affected by the MCS system performance, reliability and energy related metrics, which are governed by the contribution dynamics. The MCS application client and server processing can be just considered as a constant bias compared to the latter.

Specifically, several metrics of interest can be identified from the different perspectives above identified, as reported in Table 1. A contributor is mainly interested in knowing the impact of contribution on his/her device or node, in terms of computing resource utilization (CPU, memory, storage) and battery charge. Instead, service provider is mainly interested in managing the resources, i.e., maximizing their utilization and reducing power consumption, while reducing the time for processing a task and increasing the system throughput. This also affects the response time for processing a request, which is the main metric of interest for the end-users.

PERSPECTIVE METRIC	*Contributor*	*Service Provider*	*End User*
Resource Utilization	Node	Server	-
Energy Consumption	Battery	Server	-
Response Time	-	Task	Service
Throughput	-	Task	

Table 1: MCS system metrics of interests.

A model or a modeling technique can provide a valid support to the design, deployment and assessment of the MCS applications. However, several aspects must be taken into account to properly represent an MCS system. As said above, the most challenging one is the fluctuation of the number of contributors, which strongly impacts on the MCS underlying infrastructure reflecting on the non functional properties of the applications. Thus, an MCS system model has to primarily take into account the contribution dynamics in the stochastic characterization of both arrival (join) and departure (leave) times of contributors, by using specific probability distributions. A stochastic characterization is anyway required for the whole system, including the end-user requests arrival process and service time. In general these are not Markovian, since their stochastic behavior is not memoryless. Furthermore, the model has also to deal with the complexity of an MCS system, for example, taking into account queueing effects as well as checkpointing policies.

3. MODELING MCS

Starting from the description of Section 2, it is evident the need of a modeling technique that allows to evaluate the metrics of interest identified in Table 1, while dealing with churning issues. This strongly suggests to address the problem in a hierarchical, layered way, separating contribution aspects and concerns from provisioning and fruition ones. To this purpose, based on the scenario of Figure 1, a two-layer modeling technique is proposed splitting the contribution and the processing subsystems. In the contribution subsystem, the contribution dynamics due to node fluctuations joining and leaving the MCS system is mainly represented. The processing subsystem is instead focused on the MCS application requests and tasks processing and therefore on the service dynamics. However the two subsystems are not independent since contributors mainly serve application requests and tasks. Indeed, when a contributor joins the contribution network, a server in the processing subsystem is turned on and can serve incoming requests. The server is turned off when the contributor associated with that server leaves the contribution network. It is important to remark that a connection between a server and a contributor lasts until the latter leaves the system. Then, the server in the processing network may be switched on by another contributor. In other words, at most one active contributor at a time can be associated with any idle server.

Another aspect to take into account in the MCS modeling is the complexity. In an MCS system the number of users and contributors is commonly in the order of thousands, which excludes state space-based models. Due to its well known capability in dealing with complexity we choose the QN formalism.

3.1 The contribution QN

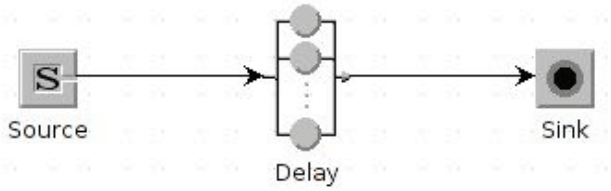

Figure 2: The contribution queuing network submodel.

The QN submodel representing the MCS contribution dynamics is shown in Figure 2. Contributors can join and leave the MCS system randomly and the time they spend in the system is the contribution time. Therefore, they are stochastically represented by general distributions. Thus, a job in the `Delay` station (i.e., a $G/G/\infty$ queue) represents a contributor node that is sharing its resources. The arrival rate to `Delay` and the time a job spends in it may be characterized by any general distribution. They represent the arrivals of contributors in the system and their contribution times (i.e., the time they are available to serve the end-users requests), respectively. As discussed above, the `Delay` station of this network is connected to the `Service Center` one in the processing QN. The number of servers in the processing submodel is equal to the number of jobs in the `Delay` station of the contribution QN.

3.2 The processing QN

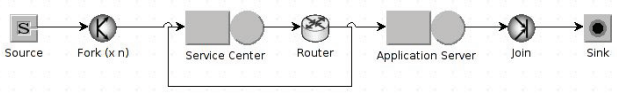

Figure 3: The processing queuing network submodel.

The processing submodel represents the end-user requests incoming to the MCS system, which have to be processed by the contributor nodes. Figure 3 shows the proposed QN submodel that is just a part of the model, the overall QN model is composed of the two submodels of Figures 2 and 3, strictly dependent. The task requests arrival process is characterized by a general distribution and does not depend on the contribution QN. It is modeled by the `Source` station. As well, the `Application Server` station does not depend on the contribution QN and its service time is generally distributed, thus resulting in a $G/G/1$ queue. This station models the commit of a request after it has been completely processed. As discussed above, an end-user request is decomposed into n tasks for processing, represented by the `Fork-Join` elements in the QN, thus splitting an end-user request before being served and recomposing it afterwards in the `Application Server`.

The processing in the `Service Center` station depends on the contribution QN. To represent this dependence, we refer to the `Service Center` as a $G/G/x$ queue with generally distributed arrival and service times and variable number of

servers $x \in [0, \infty)$ specifically representing the contribution churning dynamics. The number of servers x is associated with the number of contributors that are in the contribution system QN of Figure 2. To represent fluctuations on the number of servers for the `Service Center` station, each contributor arrival/departure turns on/off a server.

A task request is served by only one server at a time but, since it could be rescheduled due to churning, it may be served by different servers till it is completely processed. Indeed, a request exits from the `Service Center` after completion or if the i-th server processing the task is switched off, meaning that the associated contributor left the system. In the former case, the task processing results are sent to the `Application Server` by the `Router` to be aggregated with the other $n-1$ tasks of the same end-user request, and after the synchronization (through the `Join` station) further processed before leaving the system. In the latter case, the task request is rescheduled, thus sent back by the `Router` to the `Service Center`, waiting to be processed by an idle server.

Priority policies may be applied in the `Service Center` to prioritize dropped requests. Since the number of available servers may be lower than the number of requests that need to be served, only x requests can be served at the same time by the `Service Center`. This way, a task request that has been interrupted can be rescheduled for processing with a higher priority. The rescheduling has to also take into account possible memory policies and strategies adopted to reduce churning, such as the checkpointing one, thus implementing proper memory mechanisms to restart from checkpoints.

3.3 Measurements

Once the model has been specified, the performance and energy consumption metrics of interest must be identified on it in terms of QN measurements . Referring to Table 1, an end-user is interested in the average *Push time*, i.e., the time needed by the system to process an end-user request, that is equivalent to n task requests. Therefore, this is the time for processing the n task requests and their further elaboration by the application server after aggregation and, considering a generic end-user request i, it can be expressed as:

$$PushTime_i = max_{(j=1,..,n)}(R_i^j) \tag{1}$$

where n is the number of task requests that the system collects before returning some results to the end-user and R_i^j is the response time of the j-th task of the i-th end-user request, obtained by the processing system QN analysis. The average *Push time* is:

$$PushTime = \frac{\sum_{i=1}^{I_{max}} PushTime_i}{I_{max}} \tag{2}$$

where I_{max} is the number of end-user requests processed by the system.

The service provider may be interested in the average system response time R and throughput X. The former is obtained observing each end-user request processing, evaluating how long it spends in the system. The latter as $X = C/T$ where T is the observation time and C the number of completed end-users requests.

Finally, a contributor is interested in knowing the impact of contribution on his/her devices, quantified by the utilization U_C and the battery energy consumed by the MCS application during contribution. To this purpose, we evaluate U_C of each node as:

$$U_C = \frac{\sum_{i=0}^{x_{max}} ActivityTime_i}{\sum_{i=0}^{x_{max}} OnTime_i} \tag{3}$$

where $ActivityTime_i$ is the time spent by the server i in the `Service Center` serving a request, $OnTime_i$ is the total time that server i is available for the MCS application, either busy to serve a task request or idle, waiting for a task request to serve, and x_{max} is the maximum number of server in the `Service Center`. Assuming that the device is mainly used for computations, we evaluate the power consumption P_C of each node as shown in [12]:

$$P_C(U_C) = P_C^{idle} + (P_C^{max} - P_C^{idle})U_C \tag{4}$$

where P_C^{idle} is the power consumption when the device is idle, P_C^{max} is the power consumption of the device when it is fully utilized and U_C is the utilization of the device as obtained by eq. (3). Once the power consumption has been defined, the average energy consumption E_C can be specified as:

$$E_C = P_C \cdot S_C \tag{5}$$

where S_C is the average contributor service time, i.e., the overall time the contributor spends in the system, obtained by the contribution QN of Figure 2. We estimate the percentage of battery depleted by the contributor for sharing his/her resources with the MCS system as:

$$Battery\ consumption = \frac{E_C}{W_C} \tag{6}$$

where W_C is the battery capacity of the node.

4. VALIDATION AND EVALUATION

To explain in details the proposed technique, in this Section we analyze a case study on an MCS system. Therefore, we first discuss the parameters and configuration settings used for the analysis based on the values taken from existing literature when available. This way, we evaluate the model to demonstrate its validity through comparison against other models (analytic and QN), restricting the scope to the exponential case. The exponential assumption is then relaxed to perform further investigations on the MCS system performance and energy aspects from the different NCS stakeholder perspectives.

4.1 Parameters and Configurations

As above discussed in Section 3, each request submitted by the end-user is split into n simpler tasks then assigned and processed by contributor nodes. Once the n tasks have been processed, the provider application server aggregates the results and sends the obtained outcomes to the end-users. To take into account checkpointing-restart policies, we investigate MCS systems with and without memory strategies. In the former case, the checkpoint strategy lets the system keep memory of the task requests processing at restarting, while in the latter the processing restarts from scratch.

As stated above, the parameters used in the experiments have been taken from literature when possible. Specifically, the contributor and request arrivals are exponentially distributed, as well as the service time of the `Delay` and the `Application Server` stations. The service time of each

server in the `Service Center` station is characterized by a 3-stage Erlang, similarly to [9]. The average service time of the `Application Server` S_{AS} and the `Delay` S_C is the same in all experiments, i.e., 10 seconds and 30 minutes, respectively. The inter-arrival time of the two networks (i.e., A_C and A_R for contribution and processing QN, respectively) and the average service time of a server in the `Service Center` S_{SC} have been changed in the experiments: $A_C = \{1, 10, 20, 30\}$ minutes; $A_R = \{100, 200, 300\}$ minutes; $S_{SC} = \{5, 10, 15, 20, 25, 30, 35, 40\}$ minutes.

The strategy used in the `Service Center` to select the next request to be processed is a FIFO with priority. The priority of a request increases every time the request is sent from the server back to the queue. The job with the largest priority is the first to be served. If two or more jobs have identical priority, FIFO strategy is adopted. The number of task requests that must be committed before the system returns some results to end-user is $n = 10$ (i.e., the number of tasks generated by each end-user request).

4.2 Model Validation

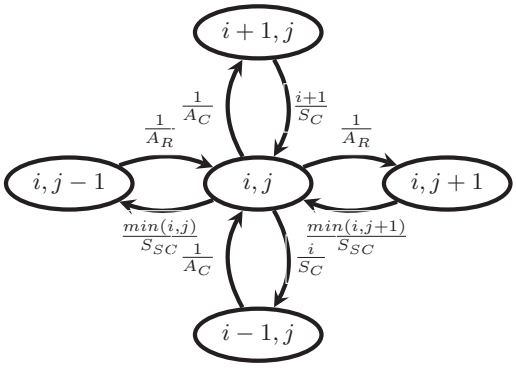

Figure 4: Part of the CTMC of the MCS system used for validation.

To validate the proposed model, we compare it against the CTMC of the MCS system shown in Figure 4, thus restricting the validation scope to the exponential case. Each state of this CTMC is characterized by a pair (i, j) where i and j are the number of contributors and requests in the MCS system, respectively. When the number of contributors changes, we can have a transition to state $(i-1, j)$ at rate i/S_C if a contributor leaves the system or to state $(i+1, j)$ at rate $1/A_C$ if a new one joins the MCS system. If a request is completed the state can move from (i, j) to $(i, j-1)$ with rate $min(i, j)/S_{SC}$ or to $(i, j+1)$ with rate $1/A_R$ if a request enters the `Service Center`.

To validate our QN model against the CTMC, all the QN time parameters must be approximated by exponential distributions. This way, the service time of each server in the `Service Center` is exponentially distributed with mean time equal to S_{SC} and the `Fork/Join` stations of the processing QN are neglected, thus the new A_R is given by the original one divided by the number of task requests n. Furthermore, for validation purposes, we evaluated only one case, characterized by $A_R = 20$ minutes and $A_C = 10$ minutes (i.e., 3 contributors into the system on average). We also truncated the state space, assuming that both the number of contributors and the number of requests cannot be larger than 20. This does not affect the results since the system we are considering for validation rarely has a number of contributors and requests larger than 20.

We consider two different discrete event simulation tools for the validation phase. OMNeT++ [27], an extensible, modular, component-based C++ simulation library and framework used for network simulations, and JMT [3], a comprehensive framework for performance evaluation of QN models. All the performance indexes have been estimated with 99% confidence intervals.

The results obtained by analyzing these models, mainly referred to the utilization, the system response time and the average number of requests in the `Service Center`, are shown in Figure 5. From Figures 5(a), 5(b) and 5(c) we can definitely argue that the performance of the MCS system obtained by the CTMC and the QN models are almost identical. Indeed, Figure 5(d) shows the relative error

$$\frac{|\Theta_{CTMC} - \Theta_{OMNeT++}|}{\Theta_{CTMC}} \qquad \Theta = \{U, R, N_{SC}\},$$

of the QN OMNeT++ model against the CTMC. The maximum relative error for all the considered indexes is always lower than 4%. In particular, the average relative errors for the utilization and the system response time are lower than 1%, while it is lower than 1.26% for the number of requests in the `Service Center`. Similar results can be obtained by the QN model evaluated through JMT.

4.3 Evaluation

After validation, the QN model has been used to perform parametric experiments by varying one between A_C and A_R, while keeping the other constant, and relaxing the exponential assumption. The MCS system has been analyzed varying A_C (or A_R) and setting $A_R = 200$ minutes (or $A_C = 20$ minutes). In the experiments also S_{SC} has been varied, assuming that the tasks allocated to the server may have different complexity and service demands, thus performing tests where 2 out of 3 parameters are variable.

With regard to the energy measurements, in the experiments we assume $P_C^{idle} = 0.268W$, $P_C^{max} = 1W$ as the idle and the maximum power consumption values [5] and $W_C = 7.77$ Watthour as the battery capacity for all the devices. The MCS system has been observed for $T = 7$ days.

In order to evaluate the QN model described in Section 3, considering the non-exponential parameters and the performance indexes there specified, we extended OMNeT++ implementing the dependency between the contribution and the processing submodels, and modifying the `Job` implementation to consider different memory strategies as discussed in Section 2. The results thus obtained are discussed in the following Sections, taking into account the different MCS stakeholders perspectives.

4.3.1 Contributor

Contributors are mainly interested in quantifying the impact of the contribution on their resources. It can be evaluated by the utilization and the battery consumption of their nodes. We analyze these quantities with eq. (3) and (6). Results are shown in Figure 6, where the average utilization per contributor as a function of the average service time of `Service Center` is depicted. In Figures 6(a) and 6(b) different inter-arrival time of the contributors are considered with requests arrival time of 200 minutes and $n = 10$,

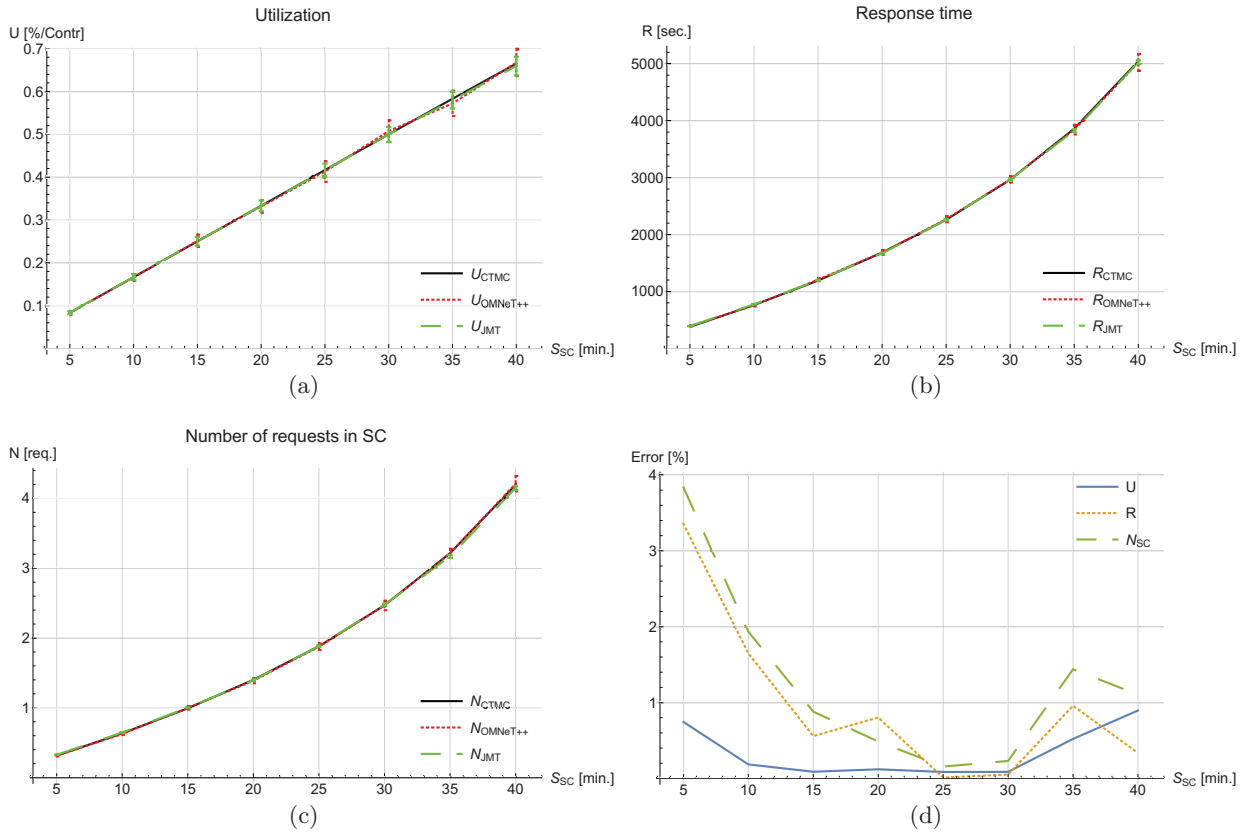

Figure 5: Performance indexes of the MCS system obtained by the CTMC and two QN simulations (OMNeT++ and JMT). Utilization (a), system response time (b), number of requests in `Service Center` (c) and relative errors of OMNeT++ QN results against CTMC ones (d).

comparing two strategies (i.e., checkpoint and no memory, respectively). As expected, the nodes utilization decreases with the contribution inter-arrival time since the total number of contributors (i.e., $N_C = S_C/A_C$) increases when the inter-arrival time A_C decreases, being the average contribution time constant, i.e., $S_C = 30$ minutes. Furthermore, if a checkpoint memory strategy is adopted, the utilization of each device is lower than the service without memory, since contributors have to work more in the latter case. In Figures 6(c) and 6(d) the inter-arrival time of contributors is set to 20 minutes, whereas the request arrival time varies. Also in this case the checkpoint contributor utilization is lower than the no memory one. A higher utilization is also observed when the request inter-arrival time is small since a larger number of requests is in the system.

The utilization is then used to investigate the power absorption through eq. (4), by which the energy consumption is derived. For this reason, energy consumption curves have the same trend of utilization ones and have not been plotted for the sake of space. However, considering the parameters given in Section 4, the battery depletion for a 30 minutes contribution is never greater than 7%, when $U = 100\%$.

4.3.2 *Service provider*

Several interesting metrics for the MCS service provider could be obtained using the QN model. One of them is the number of contributors in the system, since the number of end-users requests that can be served depends on it. Figure 7 shows the contribution dynamic for $A_C = 1$ minute and $S_C = 30$ minutes. The evolution of the contribution QN is represented only for a day in stationary conditions. In this case, the number of contributors is in the range between 14 and 47, and its average is 30.

Similarly, the system utilization is important to identify saturation conditions (i.e., when the utilization of each server is 100%) where the performance degrades and the requested QoS cannot be satisfied, driving to SLA violations. The system response time is plotted in Figure 8 on the service time, also considering different memory strategies. In Figures 8(a) and 8(b) the arrival time of end-user requests is set to 200 minutes and the contributor one varies. In the former, the checkpoint memory strategy is used, in the latter no memory strategies are adopted. It can be observed that the system response time is inversely proportional to the number of contributors in the system. Indeed, if enough contributors are into the system, the task requests are served without interruption and they are completed soon. Instead, when the number of the contributors is too low, each task request may spend several time waiting to be served. This behaviour worsens without checkpointing, since a task request must be processed from scratch if rescheduled.

Figures 8(c) and 8(d) show the system response time for the requests when the contributor arrival time is 20 minutes and the request inter-arrival time may vary. Similar considerations to the previous case can be drawn. Indeed, the system response time is also in inverse proportion to the

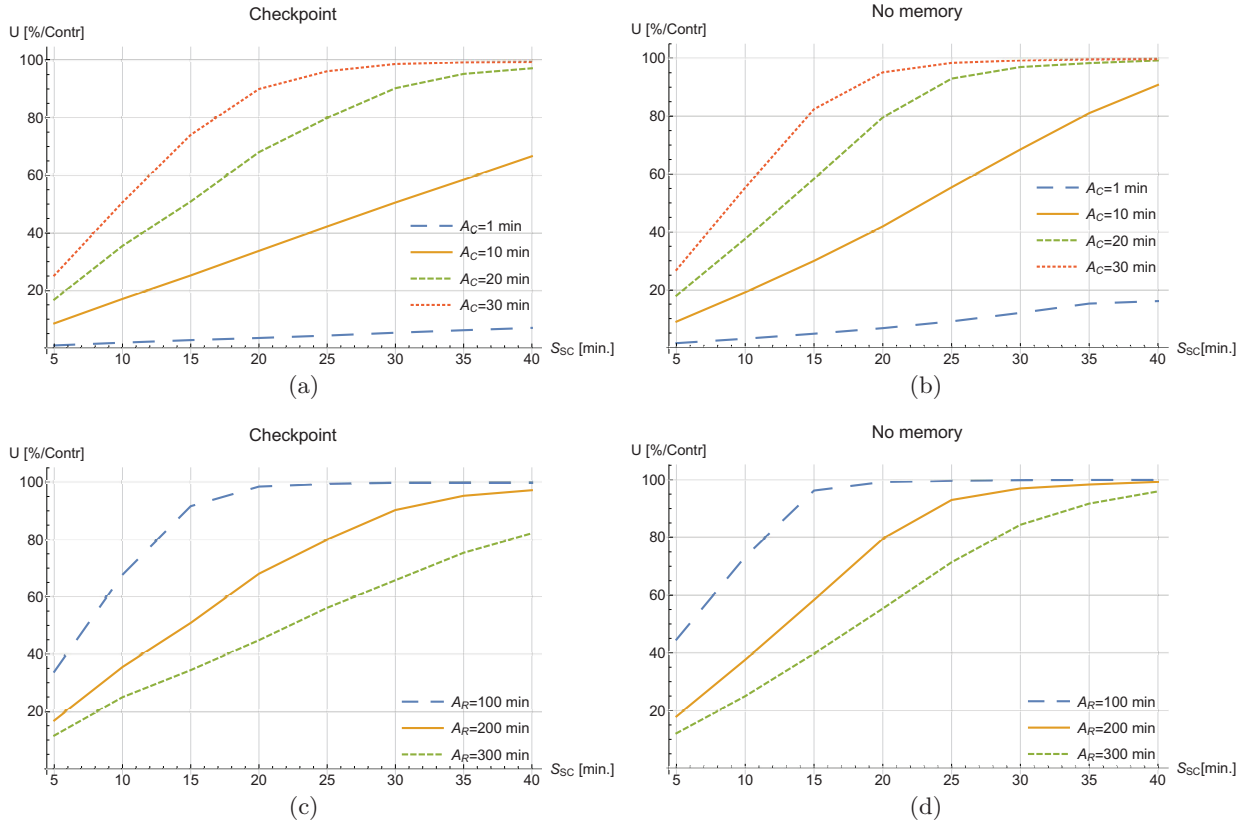

Figure 6: Average utilization per contributor, as a function of service time with fixed request inter-arrival time and either checkpoints (a) or without memory (b), and with fixed contributor inter-arrival time and either checkpoints (c) or without memory (d).

request arrival time.

To be sure to not incur in penalties, the provider is also usually interested in the system throughput, since the provisioning is often constrained by SLAs on this performance metric. The system throughput is shown in Figure 9 as a function of the service time, where the checkpoint memory strategy is compared to the no memory one also considering different inter-arrival time for contributors and requests. A larger number of contributors (i.e., smaller value of contributor arrival time) lets the system provide the largest throughput also when complex requests are in the system. Being an open QN model, if the system is not saturating, the average requests throughput is equal to the request arrival rate. The throughput has its maximum value when the tasks to be completed are simple, whereas it decreases when the requests are complex. Also in this case the checkpoint memory strategy positively impacts on the MCS system, ensuring larger throughput than without memory.

4.3.3 End-user

End-users are mainly interested in the push time, i.e., the time elapsed since the submission of the end-user request to the result feedback returned by the MCS system. In our example, the system has to process $n = 10$ task requests before returning the results to the user. The push time thus evaluated by the QN model, as specified by eq. (1) and (2), is shown in Figure 10 on the service time, comparing checkpoint and no memory strategies, considering different values of either contributors or requests inter-arrival time. In Figures 10(a) and 10(b) the contributors arrival time may vary. Considering requests inter-arrival time of 200 minutes, the push time behaviour is similar to the response time one, even if its absolute values are larger. The push time increases when the requests are complex (i.e., for larger service time values) or when the number of available contributors is low (i.e., for larger A_C values). It is worth noticing that the larger the number of requests in the system, the faster the push time increases. Indeed, requests with the same complexity can easily affect the system performance when several other requests are already in the system. In both the considered cases, checkpoint strategy provides the best results.

4.3.4 Further results

In this section we want to investigate on the likelihood of exponential and non exponential models by comparing the CTMC validation results against those obtained by a non-Markovian QN model characterizing by an Erlang distribution the `Service Center` service time and also including the `Fork/Join` stations to represent the end-user request decomposition. To this purpose, we compare the system response times of the two models with $A_C = 10$ minutes and $A_R^{ex} = 200$ minutes, where end-user requests are decomposed into $n = 10$ tasks (thus $A_R^{ex} = A_R^{ex}/n = 20$ minutes for the exponential model). For non-exponential model we considered both the case with checkpoint memory strategy

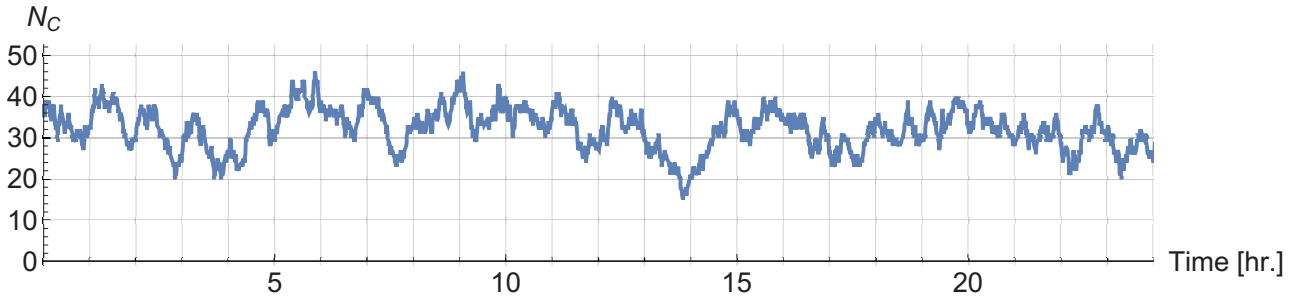

Figure 7: Number of contributors in the MCS system in the case of $A_C = 1$ minute and $S_C = 30$ minutes, for 24 hours of observation.

and the one without memory, to be compared with the exponential model that, being memoryless, cannot represent any memory policy. The results thus obtained, shown in Figure 11, clearly highlight that exponential models cannot be used as a valid approximation of non-exponential ones. The exponential values are closer to the ones obtained by the non exponential model when considering the checkpointing memory policy. This allows us to argue that dealing with non exponential distributions and related memory policies is mandatory to properly represent MCS systems.

5. RELATED WORK

Several works in literature deals with the MCS challenges above identified. With specific regard to performance and energy modeling and evaluation, it is important to investigate on the MCS application behaviours under different scenarios, dealing with complexity and churning issues. In this context, simulation is proposed for the evaluation of MCS system performance in [8], while Karaliopoulos et al. [18] used stochastic model to face the problem of user mobility, selecting users for crowdsensing campaign through an optimization problem. Stochastic models are also used to investigate how to maximize utility [17] and profit [16] in MCS applications. In these works, Lyapunov optimization is used to find the best solution for the considered problem. Liu et al. [19] adopted a CTMC to study the data collected by sensing vehicles, focusing on their spatial distribution. In [9, 22] MCS applications are modeled to analyze their QoS. In particular, the authors adopted an extended version of Stochastic Petri net formalism taking into account contributors and workload fluctuations in MCS system performance and reliability evaluation. The main problem of all such techniques lies on largeness and complexity, in terms of numbers of user requests and contributors. State space based models have intrinsic limitations in dealing with such an issue due to the well know state space explosion problem. Also simulation time could be significantly affected by these parameters, even if in a less critical way than the state space models.

In this paper we recurred to queueing theory to model MCS user and contributors arrival processes, since it allows to represent complex systems with both open and closed workloads. The main issue to address is on the contribution modeling, which is related to the queuing station server. In MCS this parameter is variable in time, due to churning issue, and in the QN domain it means that a technique able to deal with variable number of servers is required. An interesting solution could be provided by QN with breakdowns and repairs [26, 6, 11, 1], considering the departure and the arrival of a contributor as a breakdown and a repair event for a QN station server, respectively. In [26] only closed QN are considered, whereas [6, 11, 1] analyze the performability of an open QN. All these techniques adopts Markov processes to model the failure state of the systems, thus restricting the scope to exponentially distributed sojourn times for MCS contributors, that are considered as faulty/reparable servers. In our work, we proposed to deal with the contributor/server fluctuations representing them as job arrivals in the contribution QN (i.e., contributors) and as servers in the processing network. In this way, we relax the memoryless-exponential limitation on contributors/servers allowing to representation of any kind of distribution in the QN model.

Furthermore, to provide quantitative results from different perspectives (contributors, service provider and end-users), we also took into account energy related metrics on the nodes and servers. Conti et al. [8] identified this aspect as one of the key challenges to motivate participants in contributing. Indeed, since the battery capacity of these devices is often very limited, MCS applications should have a low impact on the battery depletion, letting users be almost unaware of their contributions to MCS. The proposed technique also allows to derive energy indexes from performance metrics, thus providing a multiple-view and a comprehensive analysis of an MCS system.

6. CONCLUSIONS AND FUTURE WORK

In this paper, a technique for evaluating MCS system performance and energy consumption properties is proposed. Through queuing network models, this technique allows to assess corresponding indicators from different perspectives including end-users, contributors and providers. In this way, the contribution dynamics issues due to the random and unexpected leave of contributors, i.e., churning, can be taken into account, as well as related policies for mitigating their effects on the MCS application performance such as the checkpoint memory strategy. To deal with churning dynamics the QN model has been extended to allow variable number of servers in the contribution service station, thus properly representing contributor join and leave fluctuations. The proposed technique therefore overcomes the limits of existing solutions, coping with non exponential distributions stochastically characterizing the contribution, the end-user request and the service time dynamics. To this purpose, memory mechanisms have been implemented providing a powerful mechanisms for the evaluation of advanced policies such as the checkpointing one, allowing to

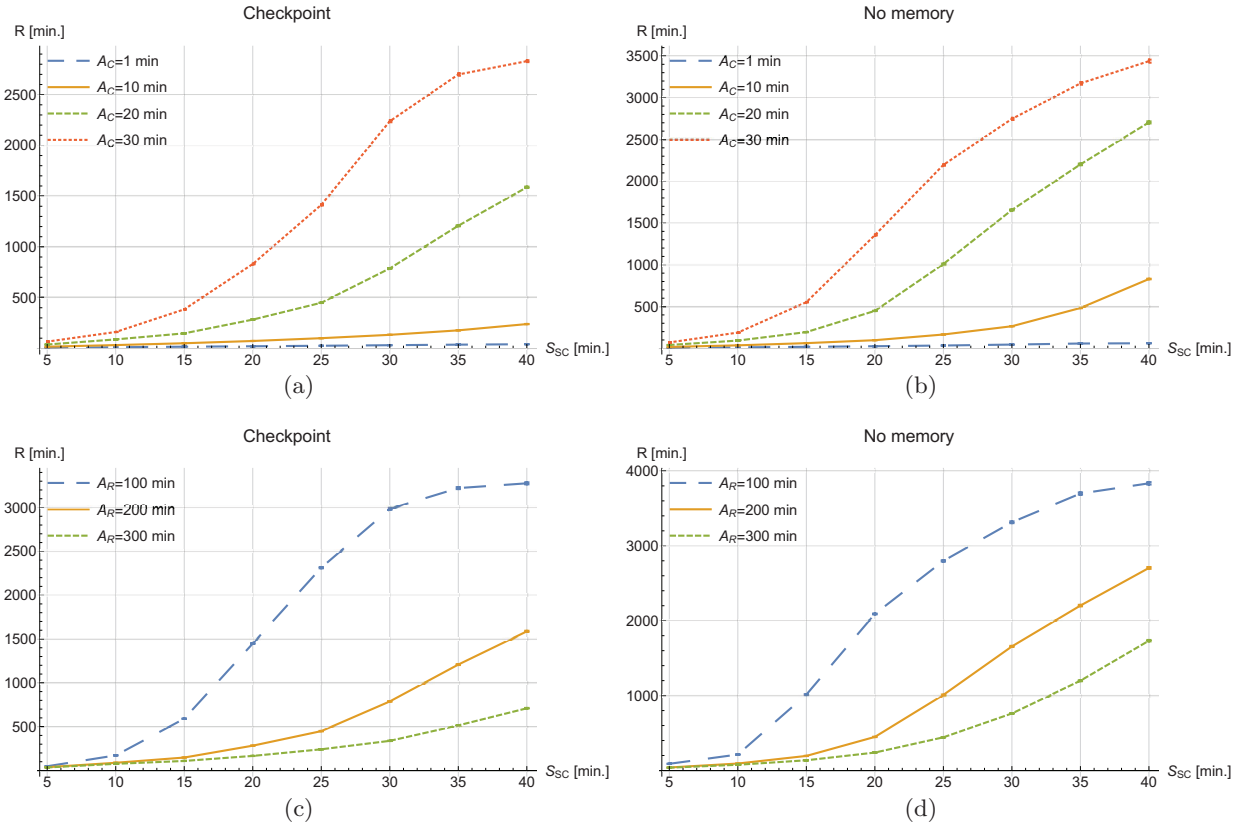

Figure 8: Average system response time for task requests, as a function of service time with fixed request inter-arrival time and either checkpoints (a) or without memory (b), and with fixed contributor inter-arrival time and either checkpoints (c) or without memory (d)

properly represent restarts from checkpoints in stateful, non-memoryless processes.

A case study has been discussed to explain by example how to use the proposed technique. In the experiments, our simulation-based implementation, extending OMNeT++, has been validated through a continuous time Markov chain, also comparing the results thus obtained against the ones computed by using the JMT tool to solve our QN model. Then, further evaluation allowed us to investigate the effect of checkpointing strategies on the performance and energy indicators of the MCS system taken into account from the contributor, end-user and provider perspectives. Eventually, the validity, the suitability and the effectiveness of the proposed technique have been demonstrated through a case study.

7. REFERENCES

[1] S. Andradóttir, H. Ayhan, and D. G. Down. Compensating for failures with flexible servers. *Operations Research*, 55(4):753–768, 2007.

[2] E. Aubry, T. Silverston, A. Lahmadi, and O. Festor. Crowdout: A mobile crowdsourcing service for road safety in digital cities. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, pages 86–91, March 2014.

[3] M. Bertoli, G. Casale, and G. Serazzi. Jmt: performance engineering tools for system modeling. *SIGMETRICS Perform. Eval. Rev.*, 36(4):10–15, 2009.

[4] G. Cardone, L. Foschini, P. Bellavista, A. Corradi, C. Borcea, M. Talasila, and R. Curtmola. Fostering participaction in smart cities: a geo-social crowdsensing platform. *Communications Magazine, IEEE*, 51(6):112–119, June 2013.

[5] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *USENIX annual technical conference*, volume 14. Boston, MA, 2010.

[6] R. Chakka, E. Ever, and O. Gemikonakli. Joint-state modeling for open queuing networks with breakdowns, repairs and finite buffers. In *2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 260–266. IEEE, 2007.

[7] Y. Chon, N. D. Lane, F. Li, H. Cha, and F. Zhao. Automatically characterizing places with opportunistic crowdsensing using smartphones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 481–490. ACM, 2012.

[8] M. Conti, C. Boldrini, S. S. Kanhere, E. Mingozzi, E. Pagani, P. M. Ruiz, and M. Younis. From manet to people-centric networking: milestones and open research challenges. *Computer Communications*, 71:1–21, 2015.

[9] S. Distefano, F. Longo, and M. Scarpa. Qos assessment of mobile crowdsensing services. *Journal of*

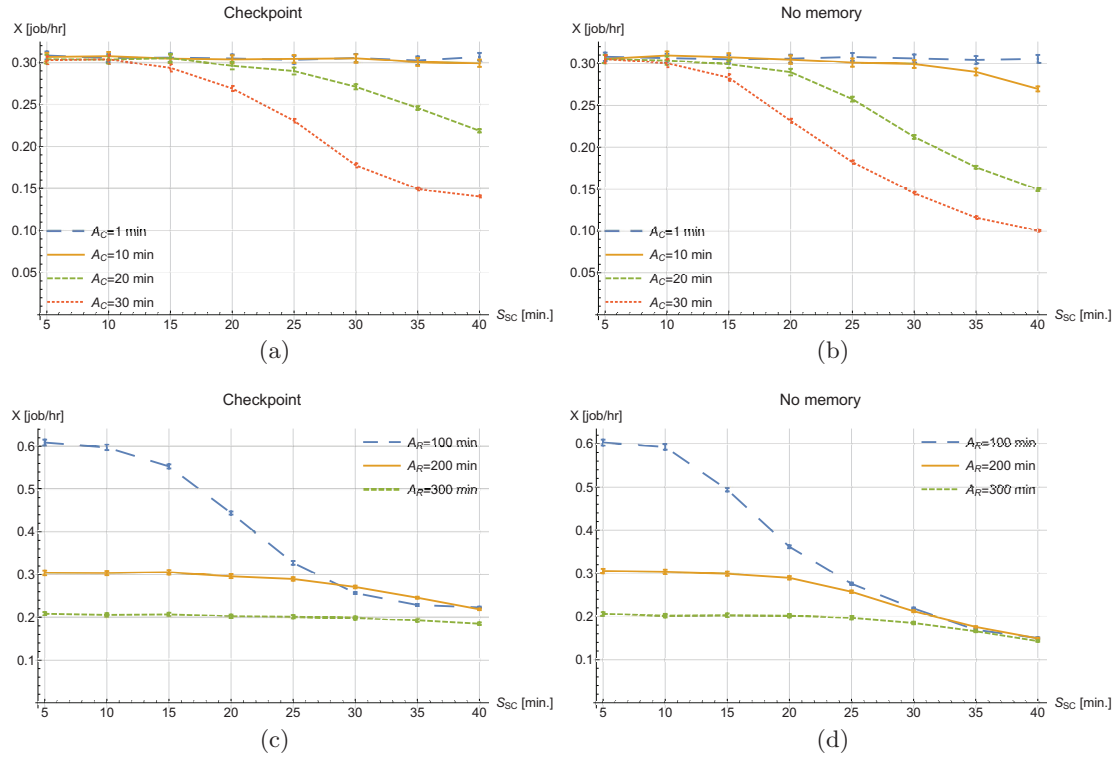

Figure 9: Average system throughput, as a function of service time with fixed request inter-arrival time and either checkpoints (a) or without memory (b), and with fixed contributor inter-arrival time and either checkpoints (c) or without memory (d).

Grid computing, 14, 2016.

[10] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell. Bikenet: A mobile sensing system for cyclist experience mapping. *ACM Transactions on Sensor Networks (TOSN)*, 6(1):6, 2009.

[11] E. Ever. Fault-tolerant two-stage open queuing systems with server failures at both stages. *IEEE Communications Letters*, 18(9):1523–1526, 2014.

[12] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 13–23. ACM, 2007.

[13] R. K. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: current state and future challenges. *Communications Magazine, IEEE*, 49(11):32–39, 2011.

[14] B. Guo, D. Zhang, Z. Wang, Z. Yu, and X. Zhou. Opportunistic iot: Exploring the harmonious interaction between human and the internet of things. *Journal of Network and Computer Applications*, 36(6):1531–1539, 2013.

[15] M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE*, 7(4):12–18, Oct 2008.

[16] Y. Han and Y. Zhu. Profit-maximizing stochastic control for mobile crowd sensing platforms. In *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*, pages 145–153. IEEE, 2014.

[17] Y. Han, Y. Zhu, and J. Yu. Utility-maximizing data collection in crowd sensing: An optimal scheduling approach. In *Sensing, Communication, and Networking (SECON), 2015 12th Annual IEEE International Conference on*, pages 345–353. IEEE, 2015.

[18] M. Karaliopoulos, O. Telelis, and I. Koutsopoulos. User recruitment for mobile crowdsensing over opportunistic networks. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 2254–2262. IEEE, 2015.

[19] Y. Liu, J. Niu, and X. Liu. Comprehensive tempo-spatial data collection in crowd sensing using a heterogeneous sensing vehicle selection method. *Personal and Ubiquitous Computing*, 20(3):397–411, 2016.

[20] T. Ludwig, C. Reuter, T. Siebigteroth, and V. Pipek. Crowdmonitor: Mobile crowd sensing for assessing physical and digital activities of citizens during emergencies. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 4083–4092, New York, NY, USA, 2015. ACM.

[21] H.-D. Ma. Internet of things: Objectives and scientific challenges. *Journal of Computer science and Technology*, 26(6):919–924, 2011.

[22] G. Merlino, S. Arkoulis, S. Distefano, C. Papagianni, A. Puliafito, and S. Papavassiliou. Mobile crowdsensing as a service: a platform for applications on top of sensing clouds. *Future Generation Computer Systems*, 56:623–639, 2016.

[23] P. Mohan, V. N. Padmanabhan, and R. Ramjee.

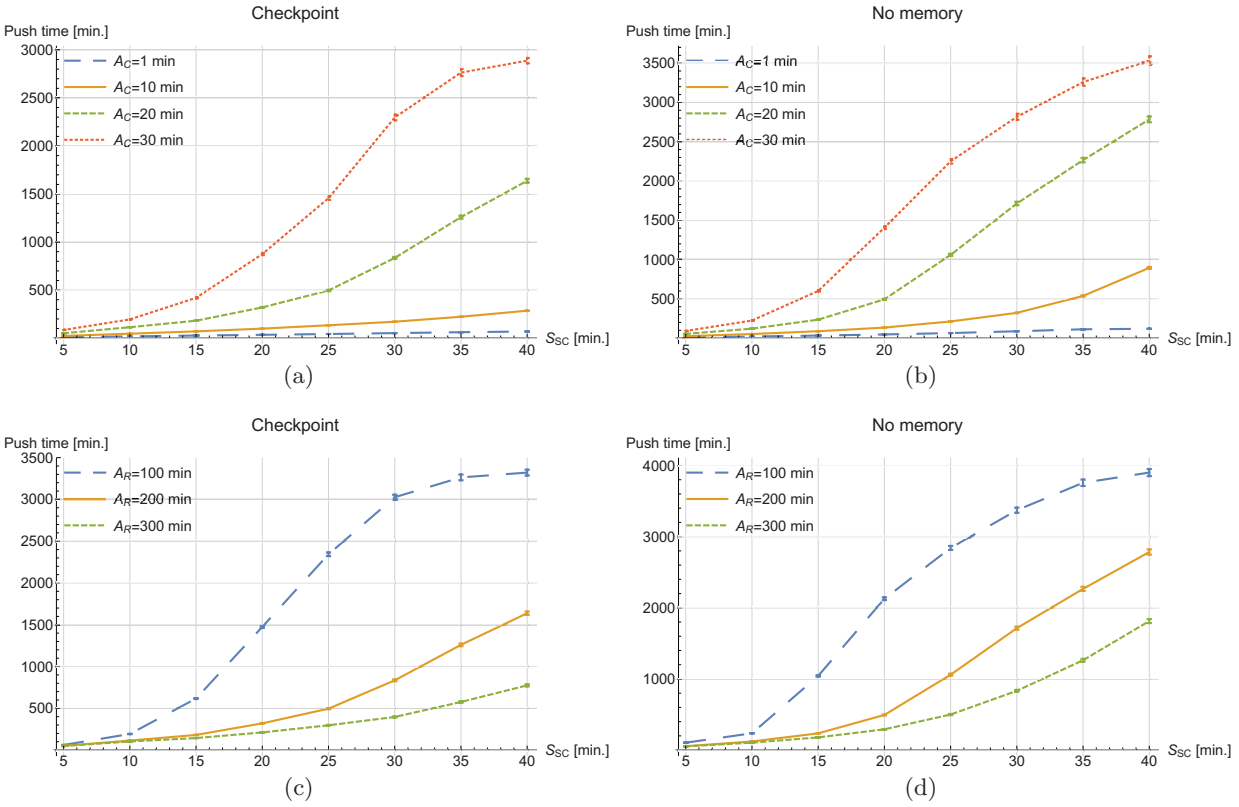

Figure 10: Average push time, as a function of service time with fixed request inter-arrival time and either checkpoints (a) or without memory (b), and with fixed contributor inter-arrival time and either checkpoints (c) or without memory (d).

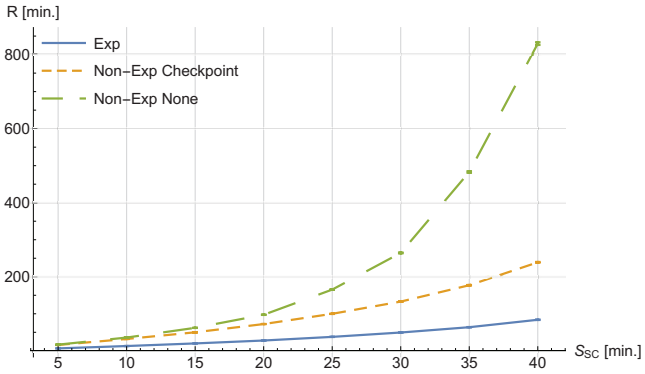

Figure 11: Comparison between the exponential model and the non-exponential one for system response time.

Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 323–336. ACM, 2008.

[24] R. Pinciroli and S. Distefano. Extending Queuing Networks to Assess Mobile CrowdSensing Application Performance. In *The sixth Workshop of the Italian group on Quantitative Methods in Informatics (InfQ2016) - Valuetools 2016*. ACM.

[25] M.-R. Ra, B. Liu, T. F. La Porta, and R. Govindan. Medusa: A programming framework for crowd-sensing applications. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 337–350. ACM, 2012.

[26] C. Ramanjaneyulu and V. Sarma. Modeling server-unreliability in closed queuing-networks. *IEEE transactions on reliability*, 38(1):90–95, 1989.

[27] A. Varga and R. Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[28] Waze Mobile. Waze website, 2006.

[29] Y. Xiao, P. Simoens, P. Pillai, K. Ha, and M. Satyanarayanan. Lowering the barriers to large-scale mobile crowdsensing. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, page 9. ACM, 2013.

[30] X. Yu, W. Zhang, L. Zhang, V. O. Li, J. Yuan, and I. You. Understanding urban dynamics based on pervasive sensing: An experimental study on traffic density and air pollution. *Mathematical and Computer Modelling*, 58(56):1328 – 1339, 2013. The Measurement of Undesirable Outputs: Models Development and Empirical Analyses and Advances in mobile, ubiquitous and cognitive computing.