Upload Planning for Mobile Data Collection in Smart Community Internet-of-Things Deployments

Qiuxi Zhu, Md Yusuf Sarwar Uddin, Zhijing Qin, Nalini Venkatasubramanian
Department of Computer Science, University of California, Irvine
qiuxiz@ics.uci.edu, yusuf.sarwar@uci.edu, zhijing.qin@gmail.com, nalini@ics.uci.edu

***Abstract*—In this paper, we develop effective solutions for enabling mobile sensing/data collection in community IoT deployments where sensing/communication coverage is intermittent and varying. Specifically, we address the optimized upload planning problem, i.e. determine optimal schedules for upload of gathered information to enable timely data collection in dynamic settings. We develop a two-phase approach and associated policies, where an initial upload plan is generated with prior knowledge of upload opportunities and data needs, and a subsequent runtime adaptation phase alters the plan based on dynamic network and data conditions. To validate our approach, we designed and built SCALECycle, a prototype mobile data collection platform and deployed it in real world community settings; measurements from testbeds in Rockville, MD and Irvine, CA are used to drive extensive simulations. Experimental results indicate that a judicious combination of policies in the two phases of upload planning (a balanced delay-opportunity-priority method with Lyapunov-inspired upload adaptation) can result in a 30-60% improvement in overall utility of collected data compared with opportunistic operation along with 30% reduction in collection delays /overheads.**

I. Motivation and Approach

With the emerging popularity of smart personal devices and the Internet of Things (IoT), designing platforms and applications to enable smart and connected communities has become a hot topic. The scale and form of such IoT deployments are quite diverse and range from personal devices and homes, to smart communities and cities [1-3]. Recent events such as the NIST/US IGNITE Global City Teams Challenge has brought together teams from governments, universities, and industry to demonstrate how lives of citizens can be improved by providing them with effective approaches to monitor and control their surroundings. One such example is the Safe Community Awareness and Alerting Network (SCALE) project [4], which aims at to create an IoT assisted community awareness system to improve the safety and health of residents.

Despite several advances, the full-fledged deployment and continuous operation of community scale IoT applications are still limited due to intermittent and varying coverage of sensing/communication infrastructures. First, IoT systems depend heavily on network infrastructures. The public network infrastructures are not uniformly and continuously available or accessible. Furthermore, they may be congested or damaged in large disruptive events such as disasters – it is often in such situations when dynamic information from impacted regions can be very useful. Second, providing complete coverage by blanketing the entire region with sensors is costly, infeasible, or difficult in certain regions and terrains. Additionally, some forms of sensed information may only be useful in rare events (e.g. chemical release during fires) – custom deployments for such rare events are not cost-effective. Intuitive and flexible approaches for instrumentation based on need are essential to enhancing the resilience and dependable operation of current large-scale IoT systems.

One promising solution is to deploy mobile agents/entities to augment in-situ deployments to provide more complete, timely, and accurate coverage of events/phenomena (e.g. disaster damage). We argue that effectively leveraging the growing mobile computing/communication capability in communities [5] can provide expanded coverage, reduce dependency on public infrastructures, and cost-effectively gather community related information. Mobile platforms that have been augmented with sensors and multiple networks [6-8], i.e. participatory systems such as BikeNet [9], ZebraNet [10,11], and CarTel [12] have been designed for specialized application domains.

An interesting question in this context is to ensure timely upload of gathered information to sites, e.g. cloud platforms, where deeper analysis can be conducted. In most existing platforms, uploading of data to the backend is mostly done in a purely opportunistic (upload at depot or first opportunity) way. However, simplistic approaches, e.g. purely opportunistic, may fail to deal with the heterogeneous nature of networks, data, and applications, and fail to handle environmental dynamics effectively, resulting in inaccurate, incomplete or delayed transfer of information. Efforts to design mobile data collection in large-scale events (e.g. earthquakes) or in communities with poor network coverage [16-21] focus on path planning and assume that the mobile data collectors (MDCs) have consistent connection setup to the backend server- this is unlikely to be true in disasters. Consistent and complete communication in all spaces is hard to create due to physical and policy limitations; even in highly developed communities, public networks are not likely to be uniformly good everywhere.

In this paper, we focus on how to optimally plan the upload of real-time information gathered by mobile data collectors that operate in conditions of intermittently available and dynamically changing network contexts. We present a two-phase approach to enhance the effectiveness (in terms of reliability, timeliness, efficiency) of upload. The key idea is to exploit (a) knowledge of the IoT deployments, e.g. where the sensors nodes are, location of Wi-Fi access points and upload opportunities; (b) knowledge of the heterogeneous nature of IoT applications and associated data characteristics – volume and modality of data generated during a certain period etc.

978-1-5090-0898-8/16/$31.00 ©2016 IEEE

Such information can be gathered/learned from daily patterns of mobility and use. Note also, that much community related IoT traffic (e.g. air quality data) is inherently delay-tolerant. Leveraging this available information and context is indeed the intuition behind our proposed upload planning solution.

A. Problem Description

As illustrated in Figure 1, in the upload planning problem, the mobile data collector (MDC) is given a path that travels through the community (or city), where there are several places to visit to fetch data, and several "upload opportunities" that can be used to deliver those data to the backend. The data we would like to collect can have different importance levels and deadlines. Upload opportunities enable Internet connectivity in any possible way, but may lead to different costs in time, money, and energy. Knowledge about data (e.g. size, urgency) can be acquired from system specifications or application requirements. Deployed sensor nodes may also selectively send metadata to the backend based on bandwidth or cost limits. In general, we assume that knowledge about current data and state is available, albeit a little stale. Knowledge about opportunities can be acquired either from system specifications and Internet service providers (ISPs), or via crowdsourcing. Given such knowledge, we want to *plan* the data collection and uploading behavior of the MDC, to ensure timely delivery of critical data.

We formalize upload planning as a constrained optimization problem (shown to be NP-hard). This optimization problem can be applied to many mobile data collection and mobile sensing scenarios. For example, volunteers (mobile data collectors) can be dispatched in emergency scenarios to take pictures or videos at targeted locations (the data collection tasks). Public wireless networks may serve as "upload opportunities". Our scheme would aim to assign several opportunities for each volunteer for data upload.

B. The Two-Phase Proactive Approach

Developing an optimal and comprehensive plan requires thorough knowledge about the state of the entire system, which is computationally complex and difficult to achieve in real-time. To handle the dynamic nature of the underlying network and application context, we propose a two-phase approach. In the first phase, referred to as **static planning**, a comprehensive plan is computed before the departure of an MDC at the server, based on collected information about the deployment and infrastructures. In the second **dynamic adaptation** phase, the static plan is adjusted by the MDC at runtime to adapt to changing network and data dynamics. At runtime, connectivity to the backend can vary. Schemes to handle this are required Dynamic adaptation algorithms execute on the MDC to determine how best to adapt the static plan based on current data upload deadlines and network availabilities.

Key contributions of this paper include: (a) Formalization of upload planning (Section II) as a constrained optimization problem that is NP-hard. (b) Design of a two-phase solution and associated algorithms (Section III) that combines a static planning stage given known upload needs and opportunities and a dynamic adaptation phase that reconfigures the upload parameters based on conditions at upload site. A key contribution here is a balanced delay-opportunity-priority algorithm for *static planning* with a Lyapunov based control theoretic upload *adaptation* at runtime. (c) Development of a prototype mobile sensing platform (Section IV) and validation in real community testbeds, (d) Extensive performance evaluation of proposed techniques via simulations driven by real-world traces from deployments (Section V) to demonstrate the effectiveness and study the scalability of the combined solution in community settings.

II. Upload Planning for Mobile Data Collection

In this section, we will discuss our assumptions and define the frequently used terms and notations. With the assumptions and notations, we formulate the upload planning problem as an NP-hard optimization problem.

A. Assumptions

Exploiting prior knowledge for upload planning requires comprehensive analysis that considers multiple factors. In this paper, we make the following assumptions to get a simple formalism, which is still computationally complex.

We assume: (a) We plan for **only one mobile agent** at any time. If there are more than one mobile agents, they work independently. (b) Tasks are short so that power consumption and storage are not constraints. (c) An earlier planning phase has generated a path for the mobile agent, and **the path is not changed** by upload planning. (d) Data are organized in **indivisible chunks**. (e) The mobile agent will **stop and stay** to wait for connection and data transmission. (f) During **static planning**, we assume that **we know all necessary parameters** about data chunks, upload points, and movements, which we refer to as their expectations.

B. Terms and Notations

A **mobile data collector (MDC)** is a mobile agent that is able to collect data and upload them to the backend server. It follows a **path**, which is an ordered sequence of geographical sites to visit. Since we are not deviating from the path as we assumed above, each point on the path can be mapped into a **location** x, which is the distance from the path origin to that point along the path. As is assumed, the MDC moves at a constant speed V.

A **data site** is a place we are interested in sending the MDC for data collection. The data collection rates at all data sites are the same R_a. A **data chunk** consists of indivisible data. For

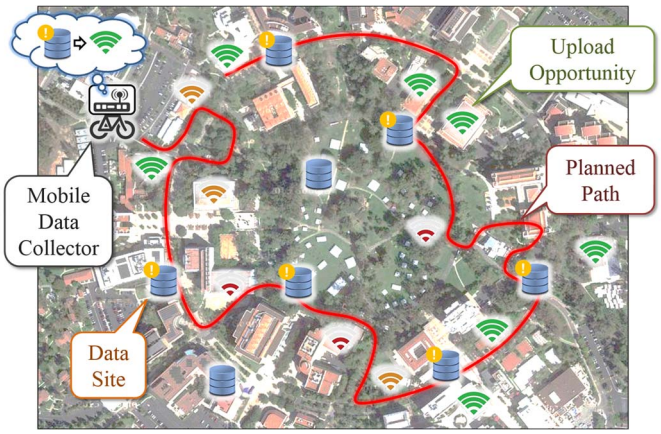


Fig. 1. An upload planning problem, where data sites and upload opportunities lie on a planned path, and the MDC needs to decide where to deliver the data.

simplicity, we assume each data chunk is held by a data site. A data chunk $\boldsymbol{a}$ can be described by quadruple $\boldsymbol{a} = (x, s, d, p)$, where x is the location of the data site holding $\boldsymbol{a}$, and s, d, and p are respectively the size, deadline, and importance level (priority) of $\boldsymbol{a}$. We assume $p \in (0,1]$. N is the number of data chunks. We also use $x(\boldsymbol{a}_i)$ to refer to the x of $\boldsymbol{a}_i$, and the same to all other characteristics. The actual delivery time of $\boldsymbol{a}_i$ in a given upload plan λ (defined later) is written as $t(\boldsymbol{a}_i, \lambda, l)$. It is estimated as $t_e(\boldsymbol{a}_i, \lambda, l)$ in planning phase.

An **upload opportunity** is a place where the MDC can upload data to the backend. An upload opportunity $\boldsymbol{u}$ is represented by $\boldsymbol{u} = (x, r)$, where x is its location, and r is its bandwidth (uploading rate). M is the number of opportunities. We also use $x(\boldsymbol{u}_j)$ to denote the x of $\boldsymbol{u}_j$, and the same to r. The estimated uploading rate r_e is known as part of the prior knowledge, but r is unknown until the MDC arrives at $\boldsymbol{u}_j$.

An **upload plan** describes the upload activities of an MDC. A plan P is a tuple $P = (\lambda, l)$ where λ is the **global plan** that defines which data chunk goes to which opportunity, and l is the **local ordering** function that determines the order of chunks at the same opportunity. It is the solution to a specific upload planning problem. Hence, the global plan λ is a mapping over data chunks $\{\boldsymbol{a}\}$ to the set of upload opportunities, $\{\boldsymbol{u}\}$. In this paper, a global plan is represented in three different ways for convenience in different contexts. The first one is the mapping representation, where $\lambda(\boldsymbol{a}_i) = \boldsymbol{u}_j$ if data chunk $\boldsymbol{a}_i$ is planned at opportunity $\boldsymbol{u}_j$. The second one is an N-by-M binary value plan matrix $\boldsymbol{\Lambda}$, $\boldsymbol{\Lambda}_{i,j} = 1$ iff $\lambda(\boldsymbol{a}_i) = \boldsymbol{u}_j$. The third representation is an N-dimensional plan vector $\mathbf{j} = [j_1, j_2, \dots, j_N]^T$, s.t. $\boldsymbol{u}_{j_i} = \lambda(\boldsymbol{a}_i)$. The local ordering function $l(\boldsymbol{a}_i, \boldsymbol{a}_k, \lambda) = 1$ iff $\lambda(\boldsymbol{a}_i) = \lambda(\boldsymbol{a}_k)$, and $\boldsymbol{a}_i$ goes before $\boldsymbol{a}_k$ in the local order. To reduce the complexity of solutions, when uploading time is much shorter than moving time, l becomes less influential so we decouple it from λ. Therefore, in static planning, l is assumed a simple highest-priority comparator using deadline as a tiebreaker.

The constraint we have in the upload planning problem is the **cause-and-effect constraint**, which means a data chunk cannot be uploaded before it is collected. The constraint can be represented as a binary matrix $\mathbf{C}$, $\mathbf{C}_{i,j} = 1$ iff $x(\boldsymbol{a}_i) \le x(\boldsymbol{u}_j)$.

C. Definition of Utility

We argue that the utility of a data chunk depends on whether it is uploaded in a timely fashion or not. Therefore, we define a utility function, f, as a function of Δ, which is the difference between the delivery time and the deadline, i.e. $\Delta(\boldsymbol{a}_i, \lambda, l) = t(\boldsymbol{a}_i, \lambda, l) - d(\boldsymbol{a}_i)$. f should be chosen based on application-specific requirements. One important characteristic of utility is that it should be monotonically decreasing over $\Delta \in \mathbf{R}$, which indicates that utility declines as delivery gets late. For many applications, arriving way early does not matter much as long as the chunk arrives prior to the deadline. In this way, $f(\Delta) = 1, \forall \Delta \le 0$. In the planning phase, the utility can be estimated by $f(\Delta_e)$, based on the estimated delivery time, i.e. $\Delta_e(\boldsymbol{a}_i, \lambda, l) = t_e(\boldsymbol{a}_i, \lambda, l) - d(\boldsymbol{a}_i)$.

The **weighted overall utility** (WOU) $U(\lambda, l)$ of a plan λ is the weighted average utility of all data chunks, with weights being their importance levels, assuming local ordering l. i.e.

$$U(\lambda, l) = \sum_{i=1}^{N} p(\boldsymbol{a}_i) \cdot f\big(\Delta(\boldsymbol{a}_i, \lambda, l)\big) \Big/ \sum_{i=1}^{N} p(\boldsymbol{a}_i), \qquad (1)$$

A plan that ends up with a higher WOU is considered to be a better plan. As a comprehensive performance index, it is also the objective function to maximize in the formulated problem.

In static planning, the **estimated weighted overall utility** (EWOU) $U_e(\lambda, l)$ is used in static planning heuristics as the objective function to maximize, which is

$$U_e(\lambda, l) = \sum_{i=1}^{N} p(\boldsymbol{a}_i) \cdot f\big(\Delta_e(\boldsymbol{a}_i, \lambda, l)\big) \Big/ \sum_{i=1}^{N} p(\boldsymbol{a}_i), \qquad (2)$$

where $\Delta_e(\boldsymbol{a}_i, \lambda, l) = t_e(\boldsymbol{a}_i, \lambda, l) - d(\boldsymbol{a}_i)$ is to be estimated by the planner. The estimated delivery time $t_e(\boldsymbol{a}_i, \lambda, l)$ is the sum of the total moving time $t_m(\boldsymbol{a}_i)$, the total collecting time $t_a(\boldsymbol{a}_i, \lambda)$, and the total uploading time $t_u(\boldsymbol{a}_i, \lambda, l)$. Since we have assumed a constant moving speed V of the MDC, the moving time is $t_m(\boldsymbol{a}_i) = x(\boldsymbol{a}_i)/V$. The total data collecting time $t_a(\boldsymbol{a}_i, \lambda)$ and the total uploading time $t_u(\boldsymbol{a}_i, \lambda, l)$ can be obtained by iterating through $\{\boldsymbol{a}\}$ and $\{\boldsymbol{u}\}$ respectively. Then, $U_e(\lambda, l)$ can be calculated using (2) in $O(MN^2)$ time. See the technical report [27] for details.

D. Problem Formulation – the Upload Planning Problem

With the assumptions and terms above, the upload planning problem is formulated as the following constrainted optimization problem:

Given an ordered list of data chunks $\{\boldsymbol{a}_i\}, i = 1, \dots, N$, with increasing $x(\boldsymbol{a}_i)$, and an ordered list of opportunities $\{\boldsymbol{u}_j\}, j = 1, \dots, M$, with increasing $x(\boldsymbol{u}_j)$, we want to find plan λ and its corresponding plan matrix $\boldsymbol{\Lambda}$, to maximize the WOU subject to the cause-and-effect constraint, i.e.

$$\text{maximize } U(\lambda, l) = \sum_{i=1}^{N} p(\boldsymbol{a}_i) \cdot f\big(\Delta(\boldsymbol{a}_i, \lambda, l)\big) \Big/ \sum_{i=1}^{N} p(\boldsymbol{a}_i),$$
$$\text{s.t. } \boldsymbol{\Lambda}_{i,j} \le \mathbf{C}_{i,j}, \forall i = 1, \dots, N, \forall j = 1, \dots, M.$$

The upload planning problem above is proven NP-hard, by showing the 0-1 knapsack problem can be reduced to an upload planning problem. See the technical report [27] for the proof.

III. Algorithms /Policies for Upload Planning

The naïve purely opportunistic approach is to upload all data as long as there is an opportunity available. In this section, we will propose a family of feasible techniques, including two static planning algorithms and three dynamic adaptation policies, to address the problem using our two-phase approach (I.B).

A. Static Planning Algorithms

Static planning is the first phase of the proposed approach. We propose two computation techniques for constructing the upload plan. One is based on Genetic algorithm (GA), and another uses an iterative greedy heuristic, Balanced Deadline-Opportunity-Priority (BDOP).

1) Genetic Algorithm

Due to the NP-hardness of our problem, we use heuristics to help with the optimization. Based on the definitions and

TABLE I PROPOSED ALGORITHM FOR STATIC PLANNING (BALANCED DOP)

Balanced DOP for Static Planning

Algorithm: planBalancedDOP($\{\boldsymbol{a}\}, \{\boldsymbol{u}\}$)
Objective: Find a plan λ to maximize $U_e(\lambda)$
Inputs: $\{\boldsymbol{a}\}$, $\{\boldsymbol{u}\}$, **Output**: Plan vector **j**

1 Initialize $N \leftarrow \{\boldsymbol{a}\}$.**size**, $M \leftarrow \{\boldsymbol{u}\}$.**size**, $\mathbf{j} \leftarrow [-1, \dots, -1]_N^T$
2 Put all chunks in $\{\boldsymbol{a}\}$ into W
3 **While** W **is not Empty**:
4 $\quad \boldsymbol{a}_i \leftarrow$ Take the chunk with the earliest deadline in W
5 $\quad$ Initialize an empty set of data chunks S and let totalSac $\leftarrow 0$
6 $\quad \mathbf{b} \leftarrow \mathbf{j}$ // Keep the temporary plan vector before planning $\boldsymbol{a}_i$
7 $\quad$ **While** 1: // If $\boldsymbol{a}_i$ is not successfully planned
8 $\quad\quad \boldsymbol{u}_j \leftarrow$ Fastest $\boldsymbol{u}$ to upload $\boldsymbol{a}_i$ in time, after other planned chunks
9 $\quad\quad$ **If** $\boldsymbol{u}_j$ **is not Null**: // Successful
10 $\quad\quad\quad$ Plan $\boldsymbol{a}_i$ at $\boldsymbol{u}_j$ in **b**, and put chunks in S back into W
11 $\quad\quad\quad$ Overwrite $\mathbf{j} \leftarrow \mathbf{b}$; **Break**
12 $\quad\quad$ **Else**:
13 $\quad\quad\quad \boldsymbol{a}_k \leftarrow$ A planned chunk with least importance level to sacrifice
14 $\quad\quad\quad$ **If** $\boldsymbol{a}_k$ **is not Null**: totalSac += $f(\Delta_e(\boldsymbol{a}_k), \lambda, l)$ //Calculate totalSac
15 $\quad\quad\quad$ **If** totalSac $> p(\boldsymbol{a}_i)$: // Sacrifice more than estimated utility gain
16 $\quad\quad\quad\quad$ Revert $\mathbf{j} \leftarrow \mathbf{b}$
17 $\quad\quad\quad\quad$ Plan $\boldsymbol{a}_i$ at the fastest $\boldsymbol{u}$ after all other chunks; **Break**
18 $\quad\quad\quad$ **Else**: Put $\boldsymbol{a}_k$ into S, wipe it from **b**, and put $\boldsymbol{a}_i$ back into W;
19 **Return j**

problem formulation, we derived the following implementation of genetic algorithm (GA). Plan vector **j** is a chromosome, where a plan j_i for one data chunk is a gene. An individual has only one chromosome. EWOU $U_e(\lambda, l)$ is used as the fitness.

2) Balanced Deadline-Opportunity-Priority

The **Balanced Deadline-Opportunity-Priority (BDOP)** algorithm chooses chunks using an earliest deadline first (EDF) greedy manner, but it changes the plan when a previously planned low priority data chunk can be removed to fit in a high priority chunk, which we refer to as a "sacrifice". The algorithm is sketched in Table I.

BDOP starts with an empty plan **j**, and all data chunks are put in set W. In every big loop (Ln 3-18), it takes out one $\boldsymbol{a}_i$ from W. First, the algorithm tries to find if $\boldsymbol{a}_i$ can be planned in time and after all other previously planned chunks, so these chunks are not affected by $\boldsymbol{a}_i$ (Ln 8). If it succeeds, then $\boldsymbol{a}_i$ is planned. Otherwise, it tries to find a set of previously planned chunks to sacrifice for $\boldsymbol{a}_i$ (Ln 12-18). Each time, a sacrificed chunk is picked in a lowest-priority-first manner, with the size being the tiebreaker (Ln 13). It then stays in the inner loop (Ln 7-18) until $\boldsymbol{a}_i$ can be planned in time (Ln 9), or the sacrifice is too big compared with the utility gain $\boldsymbol{a}_i$ brings (Ln 15). Then it either plans $\boldsymbol{a}_i$ late and reverts sacrificed chunks (Ln 16,17), or plans $\boldsymbol{a}_i$ in time and puts sacrificed chunks back into W (Ln 18). The utility gain $\boldsymbol{a}_i$ brings is hard to calculate accurately, so we are using $p(\boldsymbol{a}_i)$ as the estimation because it is the cap of the possible gain. In this algorithm, $\boldsymbol{a}_k$ can be sacrificed for $\boldsymbol{a}_i$ only if it is less important than $\boldsymbol{a}_i$. Therefore, if $\boldsymbol{a}_k$ has been sacrificed for $\boldsymbol{a}_i$, $\boldsymbol{a}_i$ will never be sacrificed for $\boldsymbol{a}_k$. Since sacrifice is made in an increasing order of importance levels, $\boldsymbol{a}_k$ will not sacrifice for $\boldsymbol{a}_i$ for more than once. Therefore, the total number of sacrifices is limited to $O(N^2)$. In each loop the time spent on $f(\Delta_e)$ (bottleneck step) is $O(MN)$. Therefore, the algorithm terminates in $O(MN^3)$ time.

B. Dynamic Adaptation Policies

Dynamic adaptation is the second phase in our two-phase approach, which is executed on the MDC during task runtime. Hence, the policies for this phase should be simple enough to run on mobile devices. Since the MDC cannot change the path or the points to visit, dynamic adaptation only happens at upload opportunities. In this phase, in addition to the prior knowledge we have, the MDC also has a static plan λ, and when it arrives at an opportunity $\boldsymbol{u}_j$, it estimates the bandwidth r as it uploads. This phase is intended to keep the benefits of the static plan while adapting to possible dynamics along task execution. We employ three policies, which are respectively strict to the plan, strict to the plan's timeline, and an adaptively balanced policy based on Lyapunov control.

1) Strict Static Plan

Strict static plan represents a purely planned approach. In this policy, when the MDC arrives at $\boldsymbol{u}_j$, it tries to upload all $\boldsymbol{a}_i$ s.t. $\lambda(\boldsymbol{a}_i) = \boldsymbol{u}_j$ as specified by plan λ. If $\boldsymbol{u}_j$ is not available, the data chunks will be carried back by the MDC physically.

2) Strict Timeline

The strict timeline policy is another attempt to stick to the original plan. Instead of keeping the plan unchanged, we try to keep the plan's timeline unchanged. In other words, we try to complete at each opportunity $\boldsymbol{u}$ at the expected time given by plan λ. To implement this, we need to compute the expected completion time, $t_e(\boldsymbol{u}_j, \lambda)$ as follows.

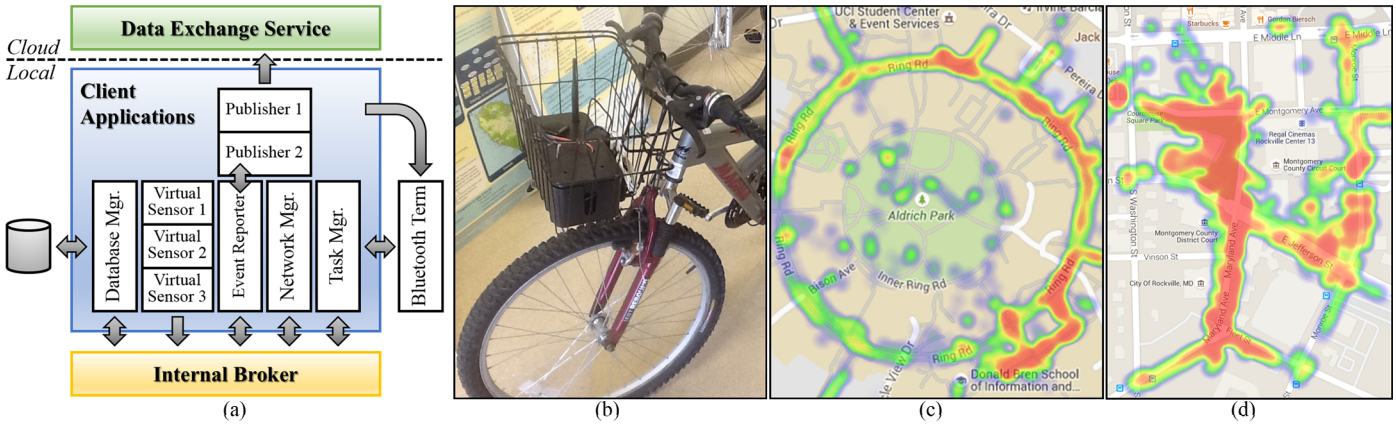
$$t_e(\boldsymbol{u}_j, \lambda) = \max\{t_e(\boldsymbol{a}_i, \lambda, l) \big| \lambda(\boldsymbol{a}_i) = \boldsymbol{u}_j\}, \tag{3}$$

If the MDC has finished all data chunks planned at the current opportunity but there is still time left, it tries to find some important data chunks (better still within their deadlines) to upload in seek of a higher U.

3) Adaptation Using Lyapunov Control Strategy

We devise a dynamic adaptation technique using Lyapunov control [23], which is usually applied to systems that evolve over time with active "queues". The overall goal of the framework is to maximize the time-averaged reward under the constraints that all "queues" remain bounded. These queues are modeled as system states, and their growth refers to system's tendency to instability. The framework defines a Lyapunov function over the system states (i.e., the current queue sizes) and tries to keep the Lyapunov drift, the expected difference between function values at two successive steps, as small as possible, which ultimately ensures the system reaching its goal over time.

In our upload plan, we define the state of our system, $\phi(t)$, at time slot t by a vector of two queues: $\phi(t) = [Q(t), T(t)]$. A time slot t actually corresponds to an upload opportunity $\boldsymbol{u}$ encountered by our MDC. Queue $Q(t)$ refers to queue backlog at the MDC, i.e. the total size of items yet to be uploaded, and $T(t)$ measures the amount of time elapsed since the MDC started its operation. Let β be the time that is supposed to have elapsed according to the static plan constructed apriori, and is updated at every opportunity. Ideally, we want the MDC to take actions to keep $Q(t)$ low and $T(t)$ close to β. Hence, a quadratic Lyapunov function is defined as



(a) (b) (c) (d)

Fig. 2 The SCALECycle platform architecture (a), its prototype (b), and the Wi-Fi heatmaps collected with it on UCI campus (c) and in Montgomery County (d).

$$L(\phi(t)) = \frac{1}{2}Q^2(t) + \frac{1}{2}(T(t) - \beta)^2. \quad (4)$$

Then the Lyapunov drift is

$$\Delta(t) = \mathbf{E}\left[L(\phi(t+1)) - L(\phi(t))\middle|\phi(t)\right]. \quad (5)$$

Let $R(t)$ define some reward function that the system tries to maximize. Then, according to the Lyapunov theory [23], the strategy suggests taking actions at time slot t that minimizes

$$\Delta(t) - V \cdot R(t), \quad (6)$$

with control parameter, V. At each upload opportunity, our MDC chooses items to upload that reduces $Q(t)$ but increases $T(t)$. Let $X(t)$ be the total size of the items selected to upload. $Q(t+1) = Q(t) - X(t)$, $T(t+1) = T(t) + X(t)/r(t)$, where $r(t) = r(\boldsymbol{u})$ is the data uploading rate. It is shown that $\Delta(t) \sim B + \mathbf{E}[-Q(t) \cdot X(t) + (T(t) - \beta) \cdot X(t)/r(t)]$, where B is a constant that approximates $(1 + 1/r^2(t)) \cdot \mathbf{E}[X^2(t)]$. In our case, $R(t)$ is the total utility of selected chunks. Let $\sigma(\boldsymbol{a})$ be 1 if chunk $\boldsymbol{a}$ is selected (and 0, otherwise), then

$$X(t) = \sum_{\boldsymbol{a}} \sigma(\boldsymbol{a}) \cdot s(\boldsymbol{a}),$$

$$R(t) = \sum_{\boldsymbol{a}} \sigma(\boldsymbol{a}) \cdot p(\boldsymbol{a}) \cdot f\big(T(t) + s(\boldsymbol{a})/r(t) - d(\boldsymbol{a})\big).$$

Hence minimizing (6) is equivalent to maximizing

$$\sum_{\boldsymbol{a}} \sigma(\boldsymbol{a}) \cdot s(\boldsymbol{a})(Q(t) - (T(t) - \beta)/r(t)) + V \cdot p(\boldsymbol{a}) \cdot f\big(T(t) + s(\boldsymbol{a})/r(t) - d(\boldsymbol{a})\big), \quad (7)$$

which reaches its maxima when we let $\sigma(\boldsymbol{a})$ be 1 for each $\boldsymbol{a}$ that makes the summed expression in (7) positive.

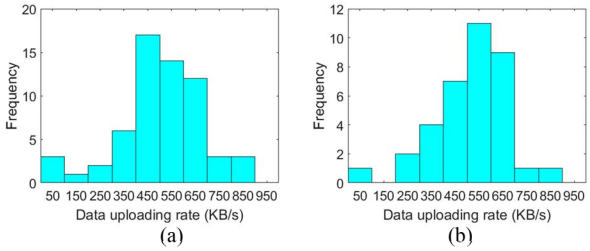
In actual implementation, each time the MDC arrives at an opportunity or finishes uploading a data chunk, it checks its buffer and picks up the data chunk that gives the maximum positive value for the summed expression in (7) and loop until no such chunk is found. Since $Q(t)$, $T(t) - \beta$, and $p(\boldsymbol{a}) \cdot f$ all have different scales and units, we need to add coefficients so these three items are at similar scales. The tuning results are, $V = 1$, the coefficient of $Q(t)$ be set to the magnitude of 10^{-6}, and that of $T(t)$ be set to the magnitude of 10^{-4}. The performance is not very sensitive to coefficient changes as long as they are at those magnitudes.

IV. Prototype Platform and Testbed

The upload planning solutions above were tested in the context of the Safe Community Awareness and Alerting Network (SCALE) project, an IoT system implemented by our team [4] to extend smart communities with commodity sensor devices. We have already deployed SCALE on the campus of University of California, Irvine, and around the Thingstitute lab and Victory Court Senior Apartments in Montgomery County, Maryland. To validate the problem motivations and assumptions in real smart community/city settings and collect data to drive our experiments, we extended SCALE into SCALECycle, a mobile sensing system, and implemented a prototype. We conducted initial measurements in our two SCALE deployments, which are real-world smart community IoT systems. The data we collected in those real settings have revealed the heterogeneity in community IoT deployments and network infrastructures [24], which justified the need for the proposed problem and validated several assumptions we made.

A. The SCALECycle Mobile Sensing Platform

SCALECycle is implemented as the mobility enhancement to SCALE. It provides us with a real-world implementation of mobility-enabled smart community IoT systems, and also helps to augment the resilience of SCALE. A SCALECycle MDC is a multi-sensor box, which has a Raspberry Pi Model B as the computing device, and supports multiple types of sensors (see below) and networks. The SCALE client running on the Pi provides a local broker that supports different applications. The architecture is shown in Fig. 2 (a). Applications include virtual sensors, event sinks, network manager, event reporter, and several others that implement auxiliary functions. Most of the time, the client publishes data to the backend using MQTT [25] protocol. In our prototype of SCALECycle, the mobile node has an air quality sensor (TGS 2600), a Wi-Fi adapter, and a



(a) (b)

Fig. 3 Results for Wi-Fi upload rate tests on Montgomery County testbed (a) and on UCI campus (b).

Bluetooth adapter. The user can interact with the node using an Android phone running a Bluetooth terminal. A Bluetooth GPS module is used to collect location information and associates the events with geotags. The mobile node is powered by a USB battery and is mounted on a bicycle as is shown in Fig. 2 (b).

B. Initial Measurements on Real Testbeds

The initial measurements are carried out in two real-world testbeds. Measurements include the RSSI and link quality of Wi-Fi access points, the upload bandwidth at several spots with relatively good Wi-Fi coverage, and the total amount of sensor data that is generated in a certain amount of time. In the Montgomery County, MD testbed, the county's guest network MCGUEST covers areas around the office buildings as in Fig. 2 (d). Unfortunately, this network blocks the MQTT protocol. Instead, we used the two SCALE Wi-Fi networks we created in the Victory Court Senior Apartment and the Red Brick Court House, plus two free public Wi-Fi APs we found close to our deployments. We tested the upload throughput to the MQTT broker at these APs. Fig. 3 (a) is the speed test results at one of the four APs. The mean rate is 500 KB/s, and the standard deviation is 212 KB/s. The UCI campus testbed is a good example for a Wi-Fi covered community. The Wi-Fi network UCInet Mobile Access covers most areas on campus. The Wi-Fi heatmap in Fig. 2 (c) is the coverage of the campus network along the path around the central Aldrich Park. We can see even the campus network(intended for ubiquitous Internet access) does not provide uniformly good coverage. On the heatmap, we picked seven points with relatively good link quality and tested the upload throughput to the MQTT broker. A histogram of the speed test results is shown in Fig. 3 (b). The mean is 513 KB/s, and the standard deviation is 160 KB/s.

During the tests, the detailed JSON messages generated by our three virtual sensors are coming at approximately 1-5 KB/s. Therefore, a similar in-situ sensing node may generate several megabytes of data in a few hours. These measurements were used to drive the simulation studies below.

TABLE II. SPECIFICATIONS OF SMALL, MEDIUM, AND LARGE TEST SETS

Scenario Characteristics		Small	Medium	Large
MDC	Speed V/ms^{-1}	5		
	Est. Conn. Time T_c/s	12		
Path	Length Exp. $\mathbf{E}[L]/m$	12,000	24,000	200,000
Data Sites and Chunks	Bandwidth $R_a/Mbps$	12		
	Total Number N	120	120	1000
	Distance $\Delta x(\boldsymbol{a})/m$	$N(90,20^2)$	$N(180,60^2)$	
	Size s/MB	$U(2,8)$ [a]		
	Importance Level p	$\{0.3,0.6,1.0\}$ [b]		
	Deadline Inc. $\Delta d/s$	$U(-40,200)$		
Upload Opportunities	Total Number M	3~30 [a]	4~40 [a]	10~200 [a]
	Distance $\Delta x(\boldsymbol{u})/m$	$U(0, 2\mathbf{E}[L]/M)$ [a]		
	Bandwidth $r_e/Mbps$	$N(4,2^2)$ [c]		

[a] Subject to change if this parameter is chosen as an independent variable.
[b] Importance level is chosen from this set with probability 0.6, 0.3, 0.1, respectively.
[c] The minimum valid bandwidth is 200 Kbps, and the same to other test cases.

V. PERFORMANCE EVALUATION AND RESULTS

A. Experimental Environments and Simulation Setup

Our simulations are mostly done in the QualNet simulator [26]. Bandwidth and transmission powers are tuned to reflect the data uploading rate and RSSI we measured in real settings. The QualNet simulations were done on a Dell OptiPlex 755 PC, which has an Intel Core 2 Duo E6550 dual core 2.67 GHz CPU, and 4.00 GB DDR2 SDRAM. The OS is Ubuntu 14.04 LTS 64-bit. The simulator is a QualNet EDU 7.3 compiled with our user application. Since QualNet EDU license allows only up to 50 nodes per scenario, and the simulation takes long time, we could not scale up the experiments in QualNet. Thus, medium and large cases were simulated in MATLAB R2015B.

Using data collected on community testbeds, we created test cases in different scales as is shown in Table II. We compared our two-phase approach with the naïve and purely opportunistic approach (referred to as "first opportunity"). For our two-phase approach, since we have two algorithms for static planning and three policies for dynamic adaptation, there will be six combinations to compare: GA only, BDOP only, GA-ST, BDOP-ST, GA-Lyapunov, and BDOP-Lyapunov.

To test the **effect of network availability and capacity**, approaches are compared in experiments with different number of upload opportunities (3-30 for small cases, 4-40 for medium cases, and 10-200 for large cases) and different standard deviation of uploading bandwidth (0-285 KBps for all cases). To test the **effect of data characteristics**, the approaches are compared in simulations using different average size of data chunks (1-10 MB for all cases). To test the **scalability** of the approaches, they are compared in simulations using different number of upload opportunities (the same settings for medium and large cases) and different number of data chunks (10-200 for medium cases and 100-2000 for large cases).

B. Performace Evaluation Metrics

1) Weighted Overall Utility (WOU)

In our experiments, we used the utility function f defined below as a piecewise function, which is flat for $\Delta \leq 0$ and decreases exponentially for $\Delta > 0$.

$$f = \begin{cases} 1 & , \Delta \leq 0 \\ \exp(-\Delta/T_f), & \Delta > 0 \end{cases}, \tag{8}$$

where $1/T_f > 0$ is the attenuation coefficient. In our tests, we used $T_f = 30/\ln 2$, so the utility of a single data chunk halves every 30 seconds after its deadline.In static simulations, we are using EWOU $U_e(\lambda, l)$ calculated from (2).

2) Fraction of Important Data Chunks Uploaded In Time

The fraction of important data chunks that are uploaded in time is a straightforward metric. Intuitively, an intelligent plan should accommodate important data first to maximize the overall utility. This fraction can be calculated by comparing $t(\boldsymbol{a}_i, \lambda, l)$ with $d(\boldsymbol{a}_i)$. In static tests, t_e is used in place of t.

3) Time Consumption to Complete All Data Collection

The total time to complete all data collection can also be used as a benchmark to evaluate the planning algorithms, as we would like the MDC to complete the assigned task. The total

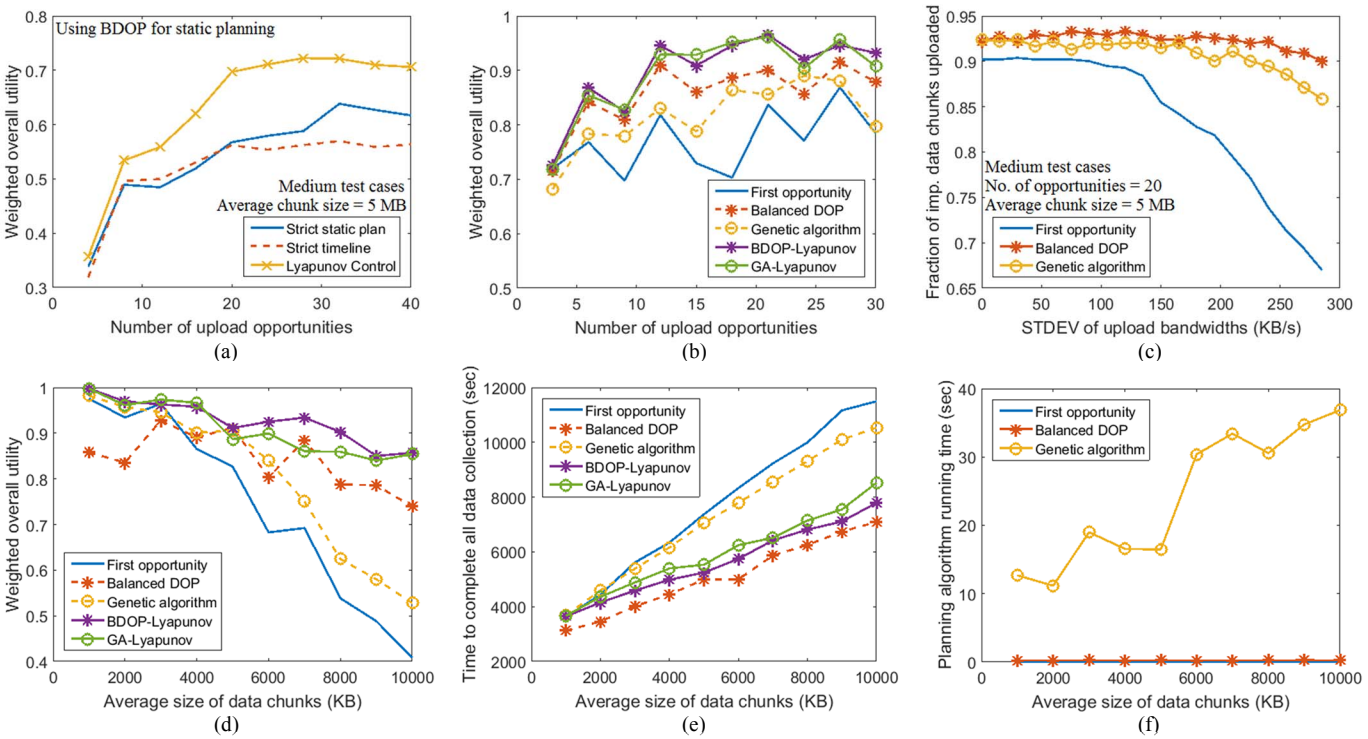


Fig. 4 Simulation results for effect of network availability and capacity (a,b,c), and effect of data heterogeneity (d,e,f).

time is the completion time at the last opportunity, $t(\boldsymbol{u}_M, \lambda)$. In MATLAB static simulations, its estimation $t_e(\boldsymbol{u}_M, \lambda)$ is used.

C. Experimental Results

1) Basic Comparisons and Effect of Network Availability:

In these tests, we first compared the algorithms for the two phases separately, and then compared our two-phase approach (different algorithm combinations) with the naïve opportunistic approach ("**first opportunity**"). The results are shown in Fig. 4 (a,b,c). (a) shows the results on a medium set that compares the three dynamic policies, with static plans generated by Balanced DOP. It shows the Lyapunov optimization performed better than "strict static plan", and "strict timeline" was the worst. Since "strict static plan" represents the purely planned approach, that requires least computation, we concluded that "strict timeline" was not useful, so it would not appear in further experiments. Comparisons of static algorithms on small sets show that GA and BDOP performed similarly well, so the other combinations were compared with the naïve approach.

Fig. 4 (b) shows results on a small test set. GA-Lyapunov and BDOP-Lyapunov performed equally well. Compared with the naïve approach, the improvement in WOU was about 14~24%. Fig. 4 (c) shows the effect of network heterogeneity. Two static algorithms were compared with the naïve approach in a medium test set. We can see BDOP and GA made it possible for the MDC to utilize the faster ones and avoid slower ones, while the opportunistic approach failed to do so.

2) Effect of Data Heterogeneity:

Results in Fig. 4 (d,e) show the performance of our two-phase approach was stable as the size of chunks increased. The improvement in WOU was 36-60% for larger data chunks, and the deduction in total time was up to 30%, compared with the naïve approach. Fig. 4 (f) compares the running time of static planning algorithms, where GA ran much slower than BDOP. With larger test sets, we found it took more than 20 minutes to generate a plan for one scenario with 1000 data chunks using GA, but the performance outcome was not better than BDOP (actually slightly worse, due to the size of solution space to search). Therefore, we will not use GA in scalability study.

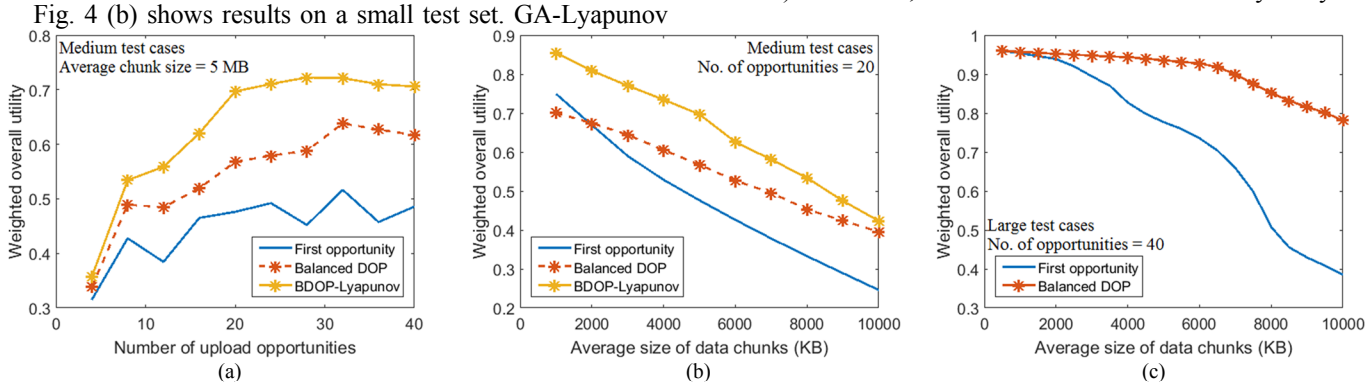


Fig. 5 Simulation results on medium (a,b) and large (c) test sets, for demonstration of scalability.

3) Scalability: Results in Fig. 5 show that our approach worked well when the scale of deployment increased. Fig. 5 (a,b) are results for medium test sets, and (c) for a large test set. In (a), the WOU was improved by 38-46% by our two-phase approach when there were more than eight opportunities, in comparison to the naïve opportunistic operation. Comparing (a,b) with Fig. 4, we can conclude that the advantage of our two-phase approach was more obvious when the deployment scales up. Fig. 5 (c) compares the performance of BDOP with "first opportunity" on a large test set, and shows BDOP was well performing and stable when the system scaled up. In summary, our two-phase approach performed the best using BDOP-Lyapunov, and the performance was significantly better than the naïve approach.

VI. Related Works and Conclusion

Works about IoT-based smart city and smart community systems [1-3] revealed the growing popularity of large-scale IoT deployments in cities. Research works [6-8] demonstrated the effectiveness of combining participatory mobile sensing with crowdsourcing. Taking such techniques into IoT systems should bring great benefits. CarTel [12] was a delay-tolerant mobile computing system for cars. ZebraNet [10,11] was an energy efficient mobile sensing network design for long-term tracking and observation of wild animals. BikeNet [9] was a sensing platform on bikes. There were both planned operation (tasking) and opportunistic operation in BikeNet, but they were not integrated to improve data uploading. Such works exploited the applications and demonstrated the advantages of mobility. However, they mostly focused on a single application, instead of building a complete IoT ecosystem. T. Black, et al. [15] proposed to use mobile agents in sensor networks for efficient data collection. It inspired our idea to enhance IoT systems with mobility, and provided with a good example of solving the IoT challenge of dynamics. M. Zhao, et al. [16] proposed an architecture for WSNs of three layers, and realized load blancing with it. A trajectory-planning layer was used for planned data collection with time constraints. Several works [17-19] have proposed the method of using mobile nodes as message ferries to facilitate data exchange, and the path planning algorithms for message ferries. There are also works using MDCs for data collection in shantytowns, and studied environmental uncertainties in detail [20,21]. However, they focused mostly on path planning, instead of network dynamics.

In this paper, we motivated and formulated the upload planning problem in smart community IoT settings and proposed a two-phase approach to solve the optimization in community settings. We designed SCALECycle, a prototype system, to help conduct measurements in our real community testbeds. Experiments using data collected with SCALECycle, showed that our two-phase approach using BDOP-Lyapunov worked significantly better and performed more stable than both the naïve approach and the planned approach, in complex community settings of different scales. Future research aims at incorporating a multi-network upload setting and integrating upload behavior planning with path planning and tasking to improve the overall efficiency of data collection in large-scale community deployments. Such techniques will be key to driving future smart communities worldwide.

Acknowledgement: We thank the ISG and DSM teams for discussions. This project is funded under NSF awards CNS-1450768.

References

[1] A. Zanella, et al., "Internet of Things for Smart Cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, Feb. 2014.

[2] R. Jalali, et al., "Smart city architecture for community level services through the internet of things," in *ICIN* 2015, pp. 108–113.

[3] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Sensing as a service model for smart cities supported by Internet of Things," *Trans. Emerging Tel. Tech.*, vol. 25, no. 1, pp. 81–93, Jan. 2014.

[4] K. Benson, et al., "SCALE: Safe community awareness and alerting leveraging the internet of things," *IEEE Communications Magazine*, vol. 53, no. 12, pp. 27–34, Dec. 2015.

[5] L. Guo and Q. Han, "Reliable data collection from mobile users with high data rates in wireless sensor networks," in *WoWMoM* 2012, pp. 1–6.

[6] S. Hachem, A. Pathak, and V. Issarny, "Probabilistic registration for large-scale mobile participatory sensing," in *PerCom* 2013, pp. 132–140.

[7] D. Christin, C. Rosskopf, M. Hollick, L. A. Martucci, and S. S. Kanhere, "IncogniSense: An anonymity-preserving reputation framework for participatory sensing applications," in *PerCom* 2012, pp. 135–143.

[8] W. Sun, et al., "Wireless deployed and participatory sensing system for environmental monitoring," in *SECON* 2014, pp. 158–160.

[9] S. B. Eisenman, et al., "The BikeNet Mobile Sensing System for Cyclist Experience Mapping," in *SenSys* 2007, pp. 87–101.

[10] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi, "Hardware Design Experiences in ZebraNet," in *SenSys* 2004, pp. 227–238.

[11] T. Liu, C. M. Sadler, P. Zhang, and M. Martonosi, "Implementing Software on Resource-constrained Mobile Sensors: Experiences with Impala and ZebraNet," in *MobiSys* 2004, pp. 256–269.

[12] B. Hull, et al., "CarTel: A Distributed Mobile Sensor Computing System," in *SenSys* 2006, pp. 125–138.

[13] N. D. Lane, et al., "Piggyback CrowdSensing (PCS): Energy Efficient Crowdsourcing of Mobile Sensor Data by Exploiting Smartphone App Opportunities," in *SenSys* 2013, pp. 7:1–7:14.

[14] H. Xiong, et al., "CrowdTasker: Maximizing coverage quality in Piggyback Crowdsensing under budget constraint," in *PerCom* 2015.

[15] T. Black, et al., "Using Autonomous Mobile Agents for Efficient Data Collection in Sensor Networks," in *WAC* 2006, pp. 1–6.

[16] M. Zhao, Y. Yang, and C. Wang, "Mobile Data Gathering with Load Balanced Clustering and Dual Data Uploading in Wireless Sensor Networks," *IEEE Transactions on Mobile Computing*, Apr. 2015.

[17] W. Zhao, et al., "A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks," in *MobiHoc* 2004, pp. 187–198.

[18] A. Monfared, et al., "Computational ferrying: Challenges in deploying a Mobile High Performance Computer," in *WoWMoM* 2015, pp. 1–6.

[19] M. M. Bin Tariq, et al., "Message Ferry Route Design for Sparse Ad Hoc Networks with Mobile Nodes," in *MobiHoc* 2006, pp. 37–48.

[20] G. Jain, et al., "On disaster information gathering in a complex shanty town terrain," in *GHTC-SAS* 2014, pp. 147–153.

[21] R. Raj, et al., "Efficient Path Rescheduling of Heterogeneous MDCs for Dynamic Events in ShantyTown Emergency Response", in *GLOBECOM* 2015.

[22] A. Petz, J. Enderle, and C. Julien, "A Framework for Evaluating DTN Mobility Models," in *ICST* 2009, pp. 94:1–94:8.

[23] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[24] Z. Qin, et al., "MINA: A reflective middleware for managing dynamic multinetwork environments," in *NOMS* 2014, pp. 1–4.

[25] MQTT. *Online*. http://mqtt.org/.

[26] QualNet. *Online*. http://qualnet.com/

[27] Q. Zhu, et al., "A technical report on the upload planning problem," *Online*. https://sites.google.com/site/zhuqiuxiuci/misc/upload-planning.