Data Collection and Upload under Dynamicity 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, Irvine, California, USA

Abstract

The Internet of Things has enabled new services to communities in many domains, e.g. smart healthcare, environmental awareness, and public safety. These services require timely and accurate event delivery, but such in-situ deployments are often limited by the coverage of sensing/communication infrastructures. In this paper we develop effective, scalable, and realistic data collection and upload solutions using mobile data collectors in community IoT systems. Specifically, we address the optimized upload planning problem, i.e. determine the optimal schedule for communication to enable timely data delivery under dynamicity in network connectivity, data characteristics/heterogeneity, and mobility. We develop a two-phase approach and associated policies, where an initial upload plan is generated offline with prior knowledge of networks and data, and a subsequent runtime adaptation alters the plan under multiple dynamics. To validate our approach, we designed and built SCALECycle, our mobile data collection platform, and deployed it in real communities in Rockville, MD and Irvine, CA. Measurements from these testbeds are used to drive extensive simulations. Experimental results indicate that compared with opportunistic operation, our two-phase approach using a judicious combination of policies can result in 30–60% improvement in overall data utility, 30% reduction in collection delays, along with greater resilience to dynamicity and improved scalability.

Keywords: community, Internet of Things, data collection, upload planning, mobility

1. Motivation and Approach

With the emerging popularity of smart personal devices and the Internet of Things (IoT), designing platforms and applications to enable pervasive computing in smart and connected communities has become a hot topic. The scale and form of IoT deployments exhibit significant diversity – use cases range from personal sensing in smart homes, to creating environmental awareness in smart instrumented communities and cities [1–3]. Recent events such as the NIST/US IGNITE Global City Teams Challenge have 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 example of a smart community effort is the Safe Community Awareness and Alerting Network (SCALE) project [4, 5], which aims at creating an IoT assisted community awareness system to improve the safety and health of residents. To build a "smart" community with enhanced levels of convenience and safety for individuals, we aim to exploit the heterogeneity of collected data and leverage any and all available sensing modalities for increased sensing coverage in communities. Increasing sensing coverage implies potentially large upgrades to the infrastructure – a large number of sensing devices and improved communication coverage are often needed to fulfill fine grained data collection requirements.

Enabling full-fledged deployment of IoT devices and supporting continuous operation of community scale IoT applications is difficult. Inherently, sensing and communication infrastructures are intermittent and provide varying levels of coverage. First, IoT systems depend heavily on network infrastructures, such as the Internet and the community Wi-Fi systems. Public network infrastructures are not uniformly and continuously available/accessible. Furthermore, communication in networks that support large community deployments may be congested or damaged in large disruptive events such as disasters – it is often in such situations when dynamic information from impacted regions and communities becomes very useful [6, 7]. Second, providing complete coverage by blanketing the entire region with sensors [8, 9], networks [10], and power supply can be costly, difficult, and even infeasible in certain regions and terrains. Note also that some types of sensed information may only be useful in rare events (e.g. chemical concentration during fires) – custom deployments for such rare events are not cost-effective. Therefore, intuitive and flexible approaches for instrumentation based on need are essential for enhancing the resilience and dependability of current large-scale IoT systems.

One promising solution is to deploy mobile agents/entities for data collection as augmentation to in-situ deployments to provide more complete, timely, and accurate coverage of events and phenomena (e.g. disaster damage). This is feasible due to the growth in the number of personal mobile devices with sufficient computational power, including laptops, smartphones, tablets, customized computing platforms, etc. A large number of participatory systems [11–13], such as BikeNet [14], ZebraNet [15, 16], and CarTel [17], have been designed to work with these mobile devices in specialized application domains. We argue that effectively leveraging such mobile agents in smart communities [18] and planning their behavior with adequate prior knowledge can provide expanded coverage and reduced dependency on public infrastructures, and enable cost effectiveness in the collection of community related information. Alongside data collection, it is important that data get delivered to the backend (e.g. cloud services) in time to enable online analysis and ensure quick response.

A motivating scenario: The following application scenario is based on our experience working with first response organizations on emergency response techniques [19, 20] and similar related projects [21–23]. Consider a community fire event. Enabling improved awareness for first responders and fire response agencies requires situational information regarding the fire and its impact on the community in a timely fashion to guide the residents and nearby crowds to safety efficiently. This includes information on toxic and explosive gases, air contaminants, ambient temperatures, pictures and video clips of damages gathered from sensors mounted on heterogeneous in-situ and/or mobile devices.

Challenges arise in timely data collection due to varying data size; furthermore, the sensed data exhibit diverse timing requirements and importance levels that cannot be determined accurately a priori. For example, information on presence of explosive gas near the fire scene is more useful than simply obtaining pictures of damage taken near it. Assume that first responders and community volunteers that move around in the community, whom we refer to as mobile data collectors (MDCs), carry heterogeneous devices with capability to collect data on their own or obtain data from deployed in-situ nodes. Each MDC is provided with a trajectory that guides its travel. Data need to be collected at specific landmark points and uploaded to the backend using any available network in the community. The intermittent coverage of networks, unpredictability of data characteristics/needs, and dynamicity in the surrounding environment add to the challenges in determining how MDCs should operate and when and where to upload data.

As illustrated in the above scenario, achieving timely and reliable upload of time critical information is challenging and essential. This paper focuses on techniques to ensure such timely and reliable data upload. The most straightforward and naïve approach is to upload all data at a final collection point in the path (depot) or the first opportunity (completely opportunistically) and has been adopted in many existing works [14, 17]. However, such an approach fails to deal with the heterogeneous nature of networks, data, and applications, and fails to handle environmental dynamics effectively, resulting in inaccurate, incomplete or delayed information transfer. At the same time, mobility inherently causes the devices to be exposed to rapidly changing surroundings; further more, mobility patterns and trajectories of human data collectors can deviate from plans. These facts add to the heterogeneity and dynamicity of the system. Current efforts to design mobile data collection in large-scale events (e.g. earthquakes) or in communities with poor network coverage [19, 20, 24–30] focus on path planning and assume that the mobile data collectors (MDCs) have consistent connectivity to the backend server – this is unlikely to be true in real-world settings, especially 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.

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 network contexts, and how to best leverage planning in realistic settings where dynamicity exists in network capacity (i.e. bandwidth), data characteristics (e.g. size, importance, timing), and movements. We present a two-phase approach to manage data upload for mobile data collectors to enhance the effectiveness of data collection in terms of timeliness, efficiency, and resilience. The key idea is to address the dynamically changing networks, data, and movements by exploiting prior knowledge of (a) community networks (e.g. location and quality of Wi-Fi access points or other upload opportunities), (b) the IoT deployments (e.g. where the sensors nodes are), and (c) the heterogeneous nature of IoT applications and associated data characteristics (e.g. volume and modality of data generated during a certain period). Prior knowledge could be learned from daily patterns of mobility and operation, and dynamics could be observed and resolved in real-time. Note also, that much community related IoT traffic (e.g. air quality data) is inherently delay-tolerant to some level. Leveraging any available information and context while being able to adapt to dynamics and uncertainties is the main focus of our proposed upload planning solutions.

1.1. Problem Description

As illustrated in Figure 1, in our community data collection and upload scenario, the MDC travels through the community following a given path, where there are several places to collect data, and several “upload opportunities” that it can use to deliver the collected data to the backend. As we discussed previously, based on our need in the fire scenario, different types of data we would like to collect may have diverse degrees of importance and delay sensitivity. Upload opportunities enable Internet connectivity using multiple methods. These diverse approaches may lead to different costs in time, money, and energy. The

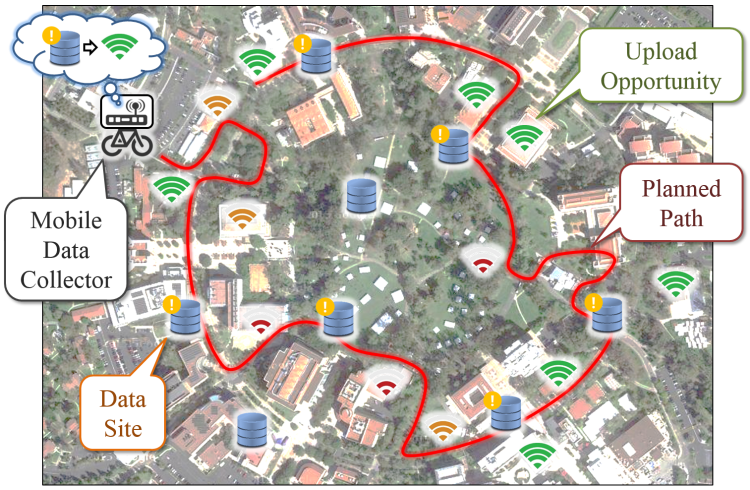


Figure 1: An upload planning problem, where data sites and upload opportunities lie on a planned path, and the MDC needs to decide where to deliver the data.

community/city setting is complex, hence causes dynamicity in network connectivity and MDC mobility.

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 the state of the system is available, albeit a little stale. Knowledge about opportunities can be estimated with system specifications, Internet service providers (ISPs), device reports through limited communication channels, or via crowdsourcing. The mobility pattern of the MDC on a given path (i.e. the time it spends moving on each segment of the path), can be estimated with the knowledge about the MDC and the terrain, or past data from previous runs. Given such knowledge, we aim to plan the data 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). The objective of planning is to determine the optimal schedule for data upload to maximize the overall data utility in terms of timeliness. While our work is motivated in a smart community scenario, 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 emergencies to take pictures or videos (data collection) at targeted locations, where public wireless networks may serve as "upload opportunities". Our scheme aims to assign several opportunities to each volunteer for data upload.

1.2. 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. At the same time, due to the limited accuracy of predictions/estimations on future values of parameters (e.g. upstream data rate, data characteristics, moving time), there does not seem to be a straightforward way to determine whether a plan is optimal at any specific time. To handle the dynamic nature of the underlying networks, the application contexts, and the environment, 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 and estimations of parameters. In the second **dynamic adaptation** phase, the static plan is adjusted by the MDC at runtime. In this phase, the MDC executes the dynamic adaptation schemes to adapt to the varying connectivity to the backend, determining how to best adapt the static plan based on data upload deadlines vs. the current timeline, and expectations on network availability and data characteristics vs. actual observations.

Key contributions of this paper include: (a) Formalization of upload planning (Section 2) as a constrained optimization problem that is NP-hard. A specific contribution here is that we have considered multiples types of runtime dynamics and addressed them in the formulation. (b) Design of a two-phase solution and associated algorithms (Section 3). The solution combines a static planning stage that leverages known upload needs and opportunities and a dynamic adaptation phase that reconfigures the upload plan based on conditions at the upload site. A noteworthy contribution here is our Balanced Delay-Opportunity-Priority algorithm for static planning, with a Lyapunov-based control theoretic upload adaptation at runtime. The Lyapunov-based adaptation policy is general enough to handle common sources of dynamicity. (c) Development of a prototype mobile sensing platform (Section 4) and validation/measurements in real community testbeds. (d) Extensive performance evaluation of proposed techniques via simulations driven by real-world traces from deployments (Section 5) to demonstrate the effectiveness and investigate the stability and scalability of the combined solution in large community settings.

An initial version of this work, albeit in a more static setting, is available at [31].

2. Upload Planning for Mobile Data Collection

In this section, we will discuss our assumptions to simplify the problem and define the terms and notations we used in the formulation. With the assumptions and notations, we formulate the upload planning problem as an optimization problem and prove it is NP-hard.

2.1. Assumptions

Exploiting the prior knowledge of upload planning requires comprehensive analysis that considers multiple factors. We make the following assumptions about our uploading planning problem to simplify the formalism.

We assume that: (a) We plan for **only one mobile agent** at any time. If there is more than one mobile agent, they work independently in isolation. (b) Tasks are short, so that power consumption and storage constraints are neglected. (c) A path planning phase exists a priori that generates a path for the mobile agent based on desired optimization needs. **This path is not changed** in upload planning. (d) Data are organized in **indivisible data chunks**. (e) Due to the limited range and obvious delay in connection setups, the mobile agent will have to **stop and stay** before the connection and data transmission take place. (f) During **static planning**, **we know necessary parameters** of data chunks, upload opportunities, and movements, which we refer to as their expectations or estimates.

2.2. 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 the mobile agent does not deviate from the path as we assumed, 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. Note that the MDC may stop for communication at several points. When it is moving, its speed at any location x is $v(x), \forall x > 0, x \neq x(\boldsymbol{a}), \forall \boldsymbol{a}, x \neq x(\boldsymbol{u}), \forall \boldsymbol{u}$. We use the "speed vs. location" representation $v(x)$ here to indicate that the path and terrain are the dominant factors to limit the speed. In the static planning, $v(x)$ is estimated as $v_e(x)$.

A **data site** is a place where 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 that share some characteristics and have to be kept/transferred together. For 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 $p \in (0, 1]$. The expectations of s and p are respectively s_e and p_e. Let $\{\boldsymbol{a}\}$ denote the set of all data chunks and N be the total number of data chunks, i.e. $N = |\{\boldsymbol{a}\}|$. We also use $x(\boldsymbol{a}_i)$ to refer to the x (i.e. location) of $\boldsymbol{a}_i$, and the similar notations for other attributes (size, deadline and priority). A data chunk is considered delivered when it is uploaded. The *actual* delivery time of $\boldsymbol{a}_i$ in a given upload plan $P = (\lambda, l)$ (defined later) is written as $t(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l)$. The *estimated* delivery time used in the static planning is denoted as $t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)$.

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 (upstream data rate). Let $\{\boldsymbol{u}\}$ denote the set of all upload opportunities and M be the total number of upload opportunities, i.e. $M = |\{\boldsymbol{u}\}|$. 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 a very important piece of prior knowledge in the static planning. The data sites, upload opportunities, and speed function, that is, $\{\boldsymbol{a}\}$, $\{\boldsymbol{u}\}$, and v, together form the input of the upload planning problem.

An **upload plan** describes the upload activities of an MDC, which is ultimately the solution to a specific upload planning problem. 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 in which the chunks at the same opportunity are uploaded. Therefore, the global plan λ is a mapping over data chunks $\{\boldsymbol{a}\}$ to the set of upload opportunities, $\{\boldsymbol{u}\}$. We represent a global plan 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 $\mathbf{\Lambda}$, where $\mathbf{\Lambda}_{i,j} = 1$ iff $\lambda(\boldsymbol{a}_i) = \boldsymbol{u}_j$. The third representation is an N-dimensional plan vector $\mathbf{j} = [j_1, j_2, \cdots, j_N]^T$, 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, note that when uploading time is much shorter than moving time, l becomes less influential, so we can decouple it from λ. Therefore, in static planning, l is assumed to be 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 shows 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 $x(\boldsymbol{a}_i) \leqslant x(\boldsymbol{u}_j)$. A valid global plan λ, with its matrix representation being $\mathbf{\Lambda}$, may have $\lambda(\boldsymbol{a}_i) = \boldsymbol{u}_j$, i.e. $\mathbf{\Lambda}_{i,j} = 1$, only if $\mathbf{C}_{i,j} = 1$.

2.3. Utility of Data Chunk Delivery

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. $\Delta(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l) = t(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l) - d(\boldsymbol{a}_i)$. The utility function, f, should be chosen based on application-specific requirements. One important characteristic of the function 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 early does not matter much as long as the chunk arrives prior to the deadline. In this way, $f(\Delta) = 1, \forall\Delta \leqslant 0$. In the planning phase, the utility can be estimated by $f(\Delta_e)$, based on the estimated delivery time, i.e. $\Delta_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l) = t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l) - d(\boldsymbol{a}_i)$.

We always hope to deliver all data chunks in a timely manner – when this is not possible, we tend to find a plan that maximizes the overall utility of them.

The **weighted overall utility** (WOU) $U = U(\{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l)$ 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.

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

The WOU is used as a major performance indicator to compare various plans. A plan that ends up with a higher WOU is considered to be a better plan. Our objective is to find an upload plan for a MDC so as to maximize this WOU. However, when we make a plan before the MDC departs, due to the possible dynamics in the task which should not have occurred yet in this phase, the actual delivery time is unknown, so is the actual delay Δ. Therefore, in this phase, we have to use the **estimated weighted overall utility** (EWOU) $U_e = U_e(\{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)$ in planning heuristics as the objective function to maximize, which is

$$U_e = \sum_{i=1}^{N} p_e(\boldsymbol{a}_i) \cdot f\big(\Delta_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)\big) \Bigg/ \sum_{i=1}^{N} p_e(\boldsymbol{a}_i) \quad (2)$$

Unlike Δ, which is obtained directly at runtime, $\Delta_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l) = t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l) - d(\boldsymbol{a}_i)$ has to be estimated by the planner. We provide detailed steps in Section 2.6 to show how the planner makes this estimation.

2.4. Problem Formulation—the Upload Planning Problem

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

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

$$\max_{\lambda} \; \sum_{i=1}^{N} p(\boldsymbol{a}_i) \cdot f\big(\Delta(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l)\big) \Big/ \sum_{i=1}^{N} p(\boldsymbol{a}_i)$$
$$\text{s.t. } \boldsymbol{\Lambda}_{i,j} \leqslant \mathbf{C}_{i,j}, \forall i = 1, \ldots, N, \forall j = 1, \ldots, M \qquad (3)$$

The upload planning problem above is proven to be NP-hard, by showing the 0-1 knapsack problem can be reduced to an upload planning problem.

2.5. Proof of Computational Complexity

We prove the constrained optimization formulation of the upload planning problem, shown in Equation (3), is NP-hard, even with all information made available, complexity significantly reduced, and no dynamicity introduced as is assumed in the static planning phase. Such computational complexity is proven by showing that the 0-1 knapsack problem, which is known to be NP-complete, can be reduced to an (actually simple case of) upload planning problem, where (a) There is only one upload opportunity that represents the knapsack; (b) There are multiple data chunks representing items, such that the size and priority of data chunks are the weight and value of items; (c) An on-time upload indicates an item being placed in the knapsack.

Consider the following 0-1 knapsack problem: Given a set of N items, numbered $1, ..., N$, each with a value $v_i > 0$ and a weight $w_i > 0$, along with a maximum weight capacity W,

$$\begin{aligned} &\max \ \sum_{i=1}^{N} v_i x_i \\ &\text{s.t.} \ \sum_{i=1}^{N} w_i x_i \leqslant W, \text{and } x_i \in \{0,1\} \end{aligned} \tag{4}$$

According to our definition of utility, $f(\Delta) \in [0,1]$, and f should be monotonically decreasing over $\Delta \in \mathbf{R}$. Therefore, according to the monotone convergence theorem,

$$\lim_{x \to +\infty} f(\Delta) = 0$$
$$\lim_{x \to -\infty} f(\Delta) = 1$$

Let

$$\begin{aligned} \Delta_+ &= \min\{\Delta \geqslant 0 | f(\Delta) \leqslant \min\{v_i\}/\max\{v_i\}\} \\ \Delta_- &= \max\{\Delta \leqslant 0 | 1 - f(\Delta) \leqslant \min\{v_i\}/\max\{v_i\}\} \\ \Delta_0 &= \max\{\Delta_+, -\Delta_-, 1\} \end{aligned}$$

In this way, we can reduce the original 0-1 knapsack problem as in Equation (4) to an upload planning problem with $M = 1$ (only one) upload opportunity and N data chunks, along with randomly chosen positive constant moving speed $v(x) = V, \forall x > 0, x \neq x_1 > 0$ and data collecting rate R_a.

Let

$$\begin{aligned} &T_c = 0 \\ &\boldsymbol{u}_1 = (x_1, r_1), \text{ where } x_1 > 0, \text{ and } r_1 = 1/2\Delta_0 \\ &x(\boldsymbol{a}_i) = 0 \\ &s(\boldsymbol{a}_i) = w_i/\min\{w\}, \forall i = 1, ..., N \\ &p(\boldsymbol{a}_i) = v_i/\max\{v\}, \forall i = 1, ..., N \end{aligned}$$

If we set all $d(\boldsymbol{a}_i), \forall i = 1, ..., N$ to

$$d(\boldsymbol{a}_i) = d_0 = \sum_{k=1}^{N} \frac{w_k/R_a}{\min\{w\}} + \frac{x_1}{V} + \frac{W/r_1}{\min\{w\}} + \Delta_0$$

then only a subset of all data chunks with their total size being no greater than $W/\min\{w\}$ can be uploaded before their common deadline. According to our previous settings, any data chunk that is scheduled to be overdue cannot contribute a utility more than even the least important data chunk that is scheduled on-time. In this way, choosing from items to put into the knapsack is equivalent to choosing from the data chunks to upload at $\boldsymbol{u}_1$, i.e. $x_i = \boldsymbol{\Lambda}_{i,1}$.

Hence, the 0-1 knapsack problem in Equation (4) can be reduced to an upload planning problem formulated as in Equation (3). This means our simplified formulation of the upload planning problem is at least as hard as the 0-1 knapsack problem, which is known to be NP-complete. In other words, the upload planning problem is at least as hard as the hardest problems in NP. Therefore, the upload planning problem is NP-hard.

2.6. Estimation of Overall Utility

As we mentioned earlier in Section 2.3, the planner needs to make an estimation of the overall utility, i.e. the EWOU U_e, when making a plan for the MDC before it departs. Therefore, we provide detailed steps below to show how $t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)$ can be calculated by static planners with expectations/estimation on several data/opportunity parameters, so that EWOU U_e can be computed based on t_e.

The estimated delivery time $t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)$ is the sum of three major components: the total moving time $t_m(\boldsymbol{a}_i, v_e)$, the total data collection time $t_a(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \lambda)$, and the total uploading time $t_u(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, \lambda, l)$, referring to the time spent on the corresponding activities before $\boldsymbol{a}_i$ is uploaded at $\lambda(\boldsymbol{a}_i)$.

The speed of MDC $v(x)$ has no definition at any data site or any upload opportunity. However, for its estimation $v_e(x)$, if we assign a finite positive value for it at those sites, then the moving time can be computed with

$$t_m(\boldsymbol{a}_i, v_e) = \int_0^{x(\boldsymbol{a}_i)} \frac{dx}{v_e(x)}$$

This integral, however, might be too heavy for planners. An easier way to estimate the moving time is to estimate the average moving speed from the beginning of path to location x as $\bar{v}_e|_0^x$, so the estimated moving time

$$t_m(\boldsymbol{a}_i, v_e) = \frac{x(\boldsymbol{a}_i)}{\bar{v}_e|_0^{x(\boldsymbol{a}_i)}} \tag{5}$$

The total data collection time is the time spent on all data sites that are located before $\lambda(\boldsymbol{a}_i)$. Since $\mathbf{C}_{i,j} = 1$ iff $x(\boldsymbol{a}_i) \leqslant x(\boldsymbol{u}_j)$, we have

$$t_a(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \lambda) = \sum_{k=1}^{N} \mathbf{C}_{k,\lambda_i} \cdot \left(T_c + \frac{s_e(\boldsymbol{a}_k)}{R_a} \right) \tag{6}$$

where T_c is an estimation of the time to establish connection at an upload opportunity. Therefore, for a specific $\boldsymbol{a}_i$, it takes $O(N)$ time to compute $t_a(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \lambda)$.

The total uploading time $t_u(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, \lambda, l)$ spent before $\boldsymbol{a}_i$ is uploaded can be further divided into three parts: the uploading time spent before $\lambda(\boldsymbol{a}_i)$, denoted as $t_{u1}(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, \lambda)$, the uploading time spent at $\lambda(\boldsymbol{a}_i)$ to upload other data chunks before $\boldsymbol{a}_i$ according to l, denoted as $t_{u2}(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, \lambda, l)$, and the time to upload $\boldsymbol{a}_i$ itself, denoted as $t_{u3}(\boldsymbol{a}_i, \{\boldsymbol{u}\}, \lambda)$.

Since $\{\boldsymbol{u}_j\}$ is given in increasing order of $x(\boldsymbol{u}_j)$, given an M-by-M up triangular binary matrix $\mathbf{B}$, where $\mathbf{B}_{i,j} = 1$ iff $i < j$, the uploading time spent before $\lambda(\boldsymbol{a}_i)$ can be written as

$$t_{u1}(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, \lambda) = \sum_{h=1}^{M} \mathbf{B}_{h,\lambda_i} \cdot \left(T_c \cdot \bigvee_{k=1}^{N} \mathbf{\Lambda}_{k,h} + \sum_{k=1}^{N} \mathbf{\Lambda}_{k,h} \cdot \frac{s_e(\boldsymbol{a}_k)}{r_e(\boldsymbol{u}_h)} \right) \tag{7}$$

whose computation takes $O(MN)$ time for a specific $\boldsymbol{a}_i$.

The uploading time spent at $\lambda(\boldsymbol{a}_i)$ to upload other data chunks before $\boldsymbol{a}_i$ is

$$t_{u2}(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, \lambda, l) = T_c + \sum_{k=1}^{N} \mathbf{\Lambda}_{k,\lambda_i} \cdot l(\boldsymbol{a}_k, \boldsymbol{a}_i, \lambda) \cdot \frac{s_e(\boldsymbol{a}_k)}{r_e(\lambda(\boldsymbol{a}_i))} \tag{8}$$

whose computation takes $O(N)$ time.

The time to upload $\boldsymbol{a}_i$ itself is

$$t_{u3}(\boldsymbol{a}_i, \{\boldsymbol{u}\}, \lambda) = \frac{s_e(\boldsymbol{a}_i)}{r_e(\lambda(\boldsymbol{a}_i))} \tag{9}$$

Then the estimated delivery time $t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)$ can be acquired from Equations (5-9) for any $\boldsymbol{a}_i$ in $O(MN)$ time, using

$$\begin{aligned} t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l) = & \; t_m(\boldsymbol{a}_i, v_e) + t_a(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \lambda) \\ & + t_{u1}(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, \lambda) + t_{u2}(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, \lambda, l) \\ & + t_{u3}(\boldsymbol{a}_i, \{\boldsymbol{u}\}, \lambda) \end{aligned} \tag{10}$$

Finally, we are able to calculate the EWOU $U_e(\{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)$ out of the estimated delivery time $t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l)$ of all data chunks using Equation (2). Therefore, the overall time complexity to calculated the EWOU of a given static plan λ is $O(MN^2)$.

3. Algorithms/Policies for Upload Planning

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 described in Section 1.2.

3.1. Static Planning Algorithms

Static planning is the first phase of the proposed approach, which is executed offline on the backend server, before the departure of the MDC. Therefore, in this phase, it is acceptable and expected to optimize upload plans using comprehensive and computational intensive algorithms.

In this paper, we propose two computational techniques for constructing the upload plan. One is based on the genetic algorithm (GA), and another uses an iterative greedy heuristic, Balanced Deadline-Opportunity-Priority (BDOP).

3.1.1. Genetic Algorithm

Due to the NP-hardness of our upload planning problem, we use heuristics to help with the optimization. Based on the definitions and problem formulation, we derived a simple implementation of a genetic algorithm (GA). The intuition here is, in a given problem setting, for each individual data chunk, some upload opportunities are generally "better" (though this is possibly affected by the schedule of other data chunks) than other opportunities, in terms of data rates and distance from the data site where the data chunk is collected. If we compare data chunks to loci (locations of genes) and different choices of opportunities to genes/alleles, then “better” opportunities are alleles who contribute more to the overall fitness. In this way, the upload planning problem is comparable to the evolution of a population, where each individual is a feasible solution to the optimization problem.

Based on such intuition, our genetic algorithm (GA) is derived as follows: Plan vector $\mathbf{j}$ is a *chromosome*, where j_i (the upload opportunity chosen for data chunk i) is a gene. An individual has only one chromosome. The EWOU $U_e(\{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)$ is used as the *fitness*. The initial population contains 100 individuals. Since it is easy to obtain a “plan vector” $\mathbf{j}$ that corresponds to the naïve approach – simply let $j_i = \min\{j | x(\boldsymbol{u}_j) \geqslant x(\boldsymbol{a}_i)\}$, we add such a “naïve plan vector” to the initial population to guarantee the bottom line of GA based optimization. In the reproduction phase, 5% elite individuals are kept as is, and 80% of children are generated from a random binary array based crossover with constraint checks.

According to Section 2.3, the time it takes to calculate the EWOU of a given plan λ is $O(MN^2)$. Therefore, the fitness calculation (bottleneck step) of each generation takes $O(MN^2)$ multiplied by the population size to complete. Note that it generally takes tens of generations of “evolution” before GA finds a maxima, so in practice, execution time of GA is usually longer.

3.1.2. Balanced Deadline-Opportunity-Priority

In this section we propose the **Balanced Deadline-Opportunity-Priority** (BDOP) algorithm. BDOP chooses data chunks in an earliest deadline first (EDF) based 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*. BDOP is designed with the following concerns: (a) Data chunks with earlier deadlines should be uploaded at earlier opportunities, in case they expire in the future. (b) Upload opportunities with faster upstream bandwidth should be preferred, in order to save time for other data chunks. (c)

In limited time/resource, important data chunks should be given higher priority in planning compared to less important data chunks, in order to improve the overall utility of collected data.

This algorithm is sketched in Algorithm 1. BDOP starts with an empty plan $\mathbf{j}$, and all data chunks are put in set W. In every big loop (Ln 4–24), 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. In fact, 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 be sacrificed for $\boldsymbol{a}_i$ (Ln 11–22). At each iteration, a sacrificed chunk is picked in a lowest-priority-first manner, with the size being the tiebreaker (Ln 12). The algorithm then stays in the inner loop (Ln 7–23) 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 16). Then it either delays $\boldsymbol{a}_i$ and reverts sacrificed chunks (Ln 18), or plans $\boldsymbol{a}_i$ for in-time delivery and puts sacrificed chunks back into W (Ln 20). To simplify the computation of the utility gain brought by $\boldsymbol{a}_i$, we use $p(\boldsymbol{a}_i)$ as its estimation. Note that $p(\boldsymbol{a}_i)$ is the maximum possible utility gain.

In BDOP, $\boldsymbol{a}_k$ 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 $\boldsymbol{a}_k$. Note that sacrifice is monotonically made in an increasing order of importance levels, so $\boldsymbol{a}_k$ will not be sacrificed for $\boldsymbol{a}_i$ for more than once. Thus, the total number of sacrifices is no greater than $O(N^2)$. In each loop the time spent on calculating $f(\Delta_e)$ (the bottleneck step) is $O(MN)$. Therefore, the algorithm terminates in $O(MN^3)$ time.

3.2. Dynamic Adaptation Policies

Dynamic adaptation is the second phase in our two-phase approach and 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, more information becomes available in addition to the prior knowledge: (a) The MDC has a static plan λ. (b) When the MDC arrives at an opportunity $\boldsymbol{u}_j$, it has information regarding the *actual* characteristic of data that will be uploaded. (c) It is able to estimate the bandwidth r as it uploads. This phase is intended to keep the benefits of the static plan (e.g. the global view of the task in optimization) while adapting to possible dynamics in networks, data, and movement along the task execution.

In this part we design three adaptation policies, namely (a) strict to the plan, (b) strict to the plan's timeline, and (c) an adaptively balanced policy based on Lyapunov control.

3.2.1. Strict Static Plan

Strict static plan represents a purely planned approach. It does not adapt to runtime dynamics, but makes full use of the static plan. 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 for any reason, these data chunks will not be uploaded in later opportunities, but carried back by the MDC physically.

3.2.2. Strict Timeline

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

$$t_e(\boldsymbol{u}_j, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda) = \max\{t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l) | \lambda(\boldsymbol{a}_i) = \boldsymbol{u}_j\} \tag{11}$$

In the actual implementation, $t_e(\boldsymbol{u}_j)$ for each opportunity can be calculated by the static planner in the planning phase, so it would not become an extra overhead for the MDC.

If the MDC has finished all data chunks planned at the current opportunity but there is still time left, it tries to find some high-priority data chunks (better still within their deadlines) to upload with the objective of achieving higher utility.

3.2.3. Adaptation Using Lyapunov Control Strategy

We devise another dynamic adaptation technique using Lyapunov control [32]. Lyapunov control 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 constraint that all "queues" remain bounded. These queues are modeled as system states, and their growth refers to the system's tendency towards 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 the Lyapunov 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 λ constructed a priori. This value is updated at every upload opportunity when the MDC arrives there. 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

$$L\big(\phi(t)\big) = \frac{1}{2}Q^2(t) + \frac{1}{2}\big(T(t) - \beta\big)^2 \tag{12}$$

Then the Lyapunov drift is

$$\Delta(t) = \mathbf{E}\big[L\big(\phi(t+1)\big) - L\big(\phi(t)\big)\big|\phi(t)\big] \tag{13}$$

Let $R(t)$ define some reward function that the system tries to maximize. Then, the strategy to adopt according to the Lyapunov theory [32] is to take actions at time slot t that minimizes

$$\Delta(t) - V \cdot R(t) \tag{14}$$

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

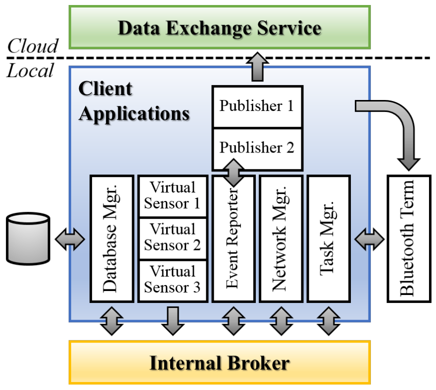
$$\begin{aligned} X(t) &= \sum_{\boldsymbol{a}} \sigma(\boldsymbol{a}) \cdot s(\boldsymbol{a}) \\ R(t) &= \sum_{\boldsymbol{a}} \sigma(\boldsymbol{a}) \cdot p(\boldsymbol{a}) \cdot f\big(T(t) + s(\boldsymbol{a})/r(t) - d(\boldsymbol{a})\big) \end{aligned}$$

Hence minimizing Equation (14) is equivalent to maximizing

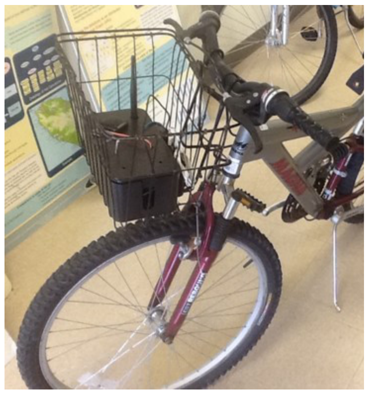
$$\sum_{\boldsymbol{a}} \Bigg(\sigma(\boldsymbol{a}) \cdot s(\boldsymbol{a}) \cdot \Big(Q(t) - \big(T(t) - \beta\big)/r(t) \Big) + V \cdot p(\boldsymbol{a}) \cdot f\Big(T(t) + s(\boldsymbol{a})/r(t) - d(\boldsymbol{a})\Big) \Bigg) \tag{15}$$

which reaches its maxima when we let $\sigma(\boldsymbol{a})$ be 1 for each $\boldsymbol{a}$ that makes the summed expression in Equation (15) 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 results in the maximum positive value for the summed expression in Equation (15) and loops until no such chunk is found. This takes $O(1)$ for each data chunk since all the parameters should be known to the MDC when this adaptation occurs, and takes at most $O(N)$ at each upload opportunity, for all data chunks collected but yet to be uploaded. Since $Q(t)$, $T(t) - \beta$, and $p(a) \cdot f$ all have different scales and units, we need to normalize them with some coefficients. 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.



(a) Architecture of SCALECycle client



(b) Prototype system

Figure 2: The SCALECycle platform.

4. 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 at various locations in Montgomery County, MD, including Victory Court Senior Apartments and Thingstitute, a re-purposed courthouse for community-centric innovation programs. To validate the problem motivations and assumptions in real smart community/city settings and collect data to drive our experiments, we developed SCALECycle, a mobility enabled IoT system and its 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 [33], which justified the need for the proposed problem and validated several assumptions we made.

4.1. The SCALECycle Mobile Sensing Platform

SCALECycle leverages SCALE [4], a previous developed community IoT platform. It provides us with a real-world implementation of smart community IoT systems, and augments the in-situ sensing capabilities in community deployments with mobility. A SCALECycle node (MDC) is a sensor box with a Raspberry Pi Model B as the basic computing device. It supports multiple types of sensors (see below) and networks interfaces. Compared to the SCALE box, it is enhanced with mobility support (GPS, battery, local storage, etc.)

In our SCALECycle prototype, the mobile node has an air quality sensor (TGS 2600), a Wi-Fi adapter, and a Bluetooth adapter. The air quality sensor collects air pollution data. The Wi-Fi adapter is used as both the network interface and a sensor that reports Wi-Fi coverage and quality data. The user can interact with the node using an Android phone running a Bluetooth terminal to send commands and acquire statistics and debug information. A Bluetooth GPS module is paired with the Raspberry Pi to provide location information, so the client can associate sensed events with geotags. The mobile node is powered by a 19,000 mAh USB battery, which is sufficient for 10 hours' operation. The device is mounted on a bicycle as is shown in Figure 2b – that is where the name SCALECycle comes from.

The SCALECycle client running on the Pi provides a local pub/sub broker that supports different types of applications. The software architecture of SCALECycle is shown in Figure 2a. Client applications include virtual sensors that generate sensed data and events, event sinks that consume these events and publish them to the backend with various protocols, a network manager that monitors the availability of multiple networks, an event reporter that dispatches events to different events sinks, and several other applications that implement auxiliary functions (e.g. a lightweight location manager that records GPS virtual sensor readings and tags other events with GPS coordinates). This flexible framework allows developers to build virtual sensors and event sinks for their own scenarios, or even deploy their own auxiliary applications to implement more complicated functionality. Applications exchange data and events via the

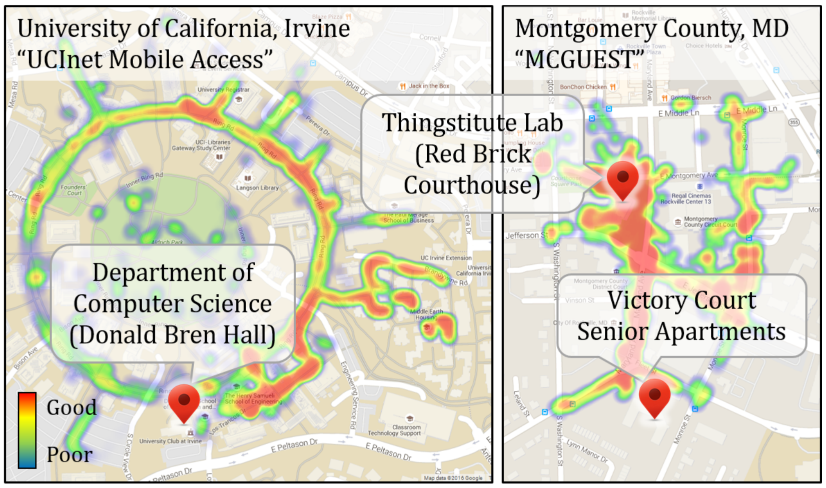
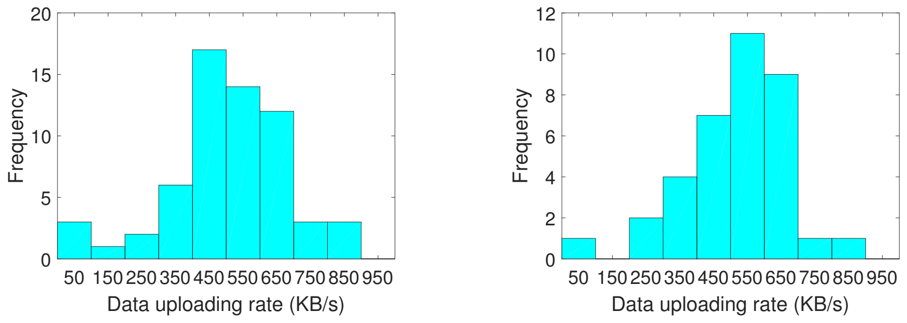

University of California, Irvine
"UCInet Mobile Access"
Department of Computer Science (Donald Bren Hall)
Good
Poor
Montgomery County, MD
"MCGUEST"
Thingstitute Lab (Red Brick Courthouse)
Victory Court Senior Apartments

Figure 3: Wi-Fi heatmaps created with SCALECycle for UCI campus and Montgomery County testbeds.

internal broker. Most of the time, the client publishes data to the cloud data exchange service using the MQTT [34] protocol. When the device is offline, the event reporter redirects the data to the database manager, which keeps them in a local database for delivery at a latter time when the device is reconnected. Cloud applications subscribe to the data exchange service, and use the data in further analysis.

4.2. Initial Measurements on Real Testbeds

The initial measurements are carried out in the two real-world testbeds mentioned earlier. 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 business park as in Figure 3. Unfortunately, this network blocks the MQTT protocol for security concerns. For validation, we used two free public Wi-Fi APs we found close to our deployments, plus two SCALE Wi-Fi APs we placed in the Victory Court Senior Apartment and the Red Brick Court House for extended coverage. We tested the upload through-



(a) Montgomery County testbed

(b) UCI campus testbed

Figure 4: Initial measurements for Wi-Fi upload rates.

put to the MQTT broker at these APs. Figure 4a is a histogram for 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 second testbed, the UCI campus 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 Figure 3 shows the coverage of the campus network on the ring road around the park in the center of the campus (known as 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 Figure 4b. The mean is 513 KB/s, and the standard deviation is 160 KB/s.

During the tests, detailed JSON messages containing original data are generated by our three virtual sensors at a rate of approximately 1–5 KB/s – it varies since the Wi-Fi coverage virtual sensor reports more data when it detects more access points. Similarly, an in-situ sensing node should generate several megabytes of data in a few hours. Traffic conditions and slopes may lead to an up-to-50% change in movement time compared with the expectation computed from the average speed. These measurements were used to drive the simulation studies below.

5. Performance Evaluation and Results

Due to the limited scale of real-world deployments, the relatively long time to run real experiments, and the lack of diversity in real settings, we derive synthetic configurations for simulated experiments to demonstrate the effectiveness of our proposed approach. Simulation parameters have been designed/tuned to track the real-world observations and measurements to best reflect the real-world scenarios and environment.

5.1. Experimental Environments and Simulation Setup

Our simulations are mostly done in the QualNet simulator [35]. Data exchange is implemented in a customized application-layer traffic generation protocol in the user library. All wireless communication goes through 802.11a/g. 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 of DDR2 SDRAM. The OS is Ubuntu 14.04 LTS 64-bit. The simulator is a QualNet EDU 7.3 compiled from the source with our customized 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. However, comparing results on small test sets from QualNet simulations and from MATLAB simulations, we observed significant correlation – the results were mostly similar in values and consistent in trends. Therefore, we believe the MATLAB simulations provide reasonable estimations on actual performance in medium and large settings.

Using data collected on community testbeds, we created test cases at multiple scales as is shown in Table 1. 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, the different 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 (0–2.23 Mbps for all cases). To test the effect of data heterogeneity and data dynamics, the approaches are compared using different average size of data chunks (1–10 MB for all cases) and different level of dynamics in data size. To test the effect of mobility dynamics, we experiment with different number of upload opportunities (4–40 for medium cases) and different level of dynamics in movement. To test the scalability of the approaches, we use 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 100–2,000 for large cases).

5.2. Performance Evaluation Metrics

We use three metrics to compare the performance of studied approaches: the weighted overall utility (WOU), the fraction of important data chunks uploaded in time, and the total time spent to complete all data collection and upload.

5.2.1. Weighted Overall Utility (WOU)

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

$$f = \begin{cases} 1 & , \Delta \leqslant 0 \\ \exp(-\Delta/T_f) & , \Delta > 0 \end{cases} \tag{16}$$

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.

The weighted overall utility (WOU) is a comprehensive performance index we defined in Equation 1 to measure the overall performance of a plan. In dynamic simulations, WOU can be calculated from statistics. In static simulations, we calculate EWOU $U_e(\{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)$ from Equation (2) for comparison.

5.2.2. 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 chunks first to maximize the overall utility. This fraction can be calculated by comparing $t(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l)$ with $d(\boldsymbol{a}_i)$. In static tests, $t_e(\boldsymbol{a}_i, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda, l)$ is used in place of t.

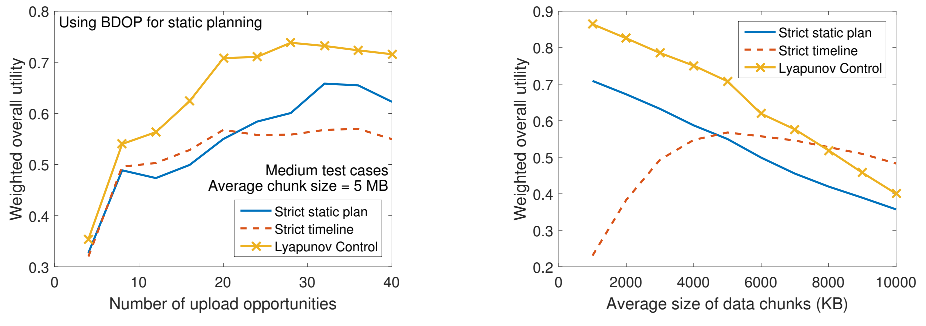
5.2.3. Time Consumption to Complete All Data Collection

The total time required to complete all data collection can also be used as a benchmark to evaluate the planning algorithms, as we expect the MDC to complete the assigned task in as short time as possible. The total time is the completion time at the last opportunity, $t(\boldsymbol{u}_M, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v, \lambda)$. In MATLAB static simulations, its estimation $t_e(\boldsymbol{u}_M, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda)$ is used.

5.3. Simulation Results

With the simulation results we have, we are able to compare static planning algorithms presented in Section 3.1 and dynamic adaptation policies discussed in Section 3.2 respectively to validate and demonstrate the applicability and effectiveness of our proposed algorithms in both phases. We also compare our two-phase approach using different algorithm combinations, with corresponding static-only (completely planned) approaches and the naïve (completely opportunistic) approach to show the effectiveness and competitive performance of the two-phase optimization.

As we proceeded with our experiments, algorithms or algorithm combinations that often appeared to be defective (e.g. worse than the naïve approach) or obviously inefficient were removed from future comparisons, which allowed us to focus on techniques that worked well.



(a) (b)

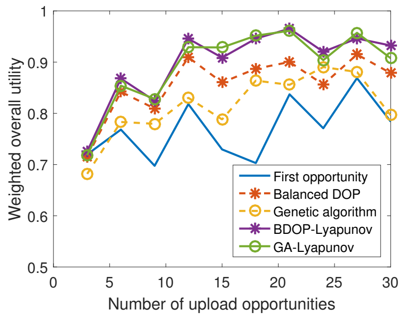
Figure 5: Simulation results for effect of network availability and capacity, using BDOP for static planning, comparing WOU of dynamic adaptation policies.

5.3.1. Basic Comparisons and Effect of Network Availability

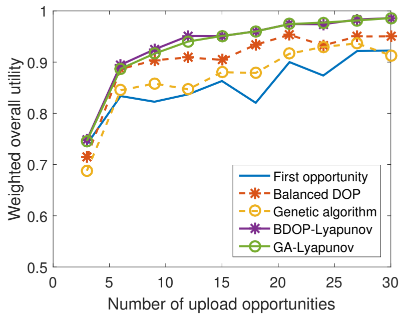
In these basic comparisons, we first compared algorithms for the two phases separately, and then compared our two-phase approaches with static-only approaches and the naïve approach (a.k.a. "first opportunity"). These results are displayed in Figure 5 and 6.

Figure 5 shows the results on medium sets that compares the three dynamic policies. In all cases, the static plans were generated by the Balanced Delay-Opportunity-Priority (BDOP) algorithm. Results show that the Lyapunov optimization performed better than the "strict static plan", and that the "strict timeline" policy exhibited the worst performance, especially for larger number of upload opportunities as shown in Figure 5a and for smaller data chunks as shown in Figure 5b. Since the "strict static plan" represents the completely planned approach and requires least computation, we concluded that "strict timeline" was not effective, and would be omitted from future experimentation. Comparisons of static algorithms on small sets show that GA and BDOP performed similarly well; hence other combinations with both these static planning algorithms were compared with the naïve approach.

In Figure 6, we compare the performance of multiple combinations of static planning algorithms and dynamic adaptation policies, as the number of upload



(a)



(b)

Figure 6: Simulation results for effect of network availability and capacity, comparing WOU of multiple approaches.

opportunities increases. Figure 6a shows the QualNet simulation 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%. Also note that both of the two-phase approaches performed better than corresponding static-only approaches, which shows the effect of the dynamic adaptation phase in our two-phase approach. Figure 6b shows the MATLAB simulation results on the same test set. Compared to the QualNet experiments, MATLAB simulations modeled fewer details of the physical networks and protocol stack implementation. This led to less fluctuation and relatively better results. We also note that the relative positions and trends of the lines were similar.

Figure 7 shows the effect of network heterogeneity and variance in network availability. Two static algorithms were compared with the naïve approach in a medium test set. We can see both BDOP and GA performed much better than the naïve approach, especially when the variance among upload opportunities increased. The reason might be that planning with a global view of all data chunks and upload opportunities (by the static planner) enables the MDC to leverage the faster opportunities and avoid the slower ones, while there was no way for the opportunistic naïve approach to do the same. This comparison

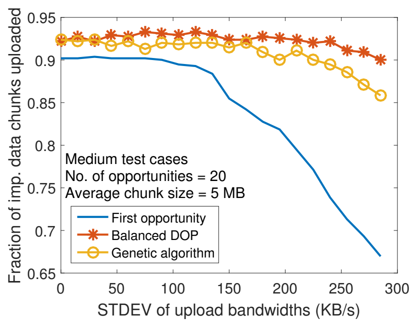


Figure 7: Simulation results for effect of variance in network availability.

shows the significance of a global plan, especially in communities and cities where connectivity is not uniformly good.

5.3.2. *Effect of Data Heterogeneity*

Results in Figures 8a and 8b show the QualNet simulation results for our two-phase approach were more stable as the size of chunks increased, compared with the static-only approaches and the naïve approach. The improvement in WOU can be as big as 36–60% for larger data chunks, and the reduction in total time was up to 30%, compared with the naïve approach. Interestingly, though the BDOP based approaches (static and two-phase) performed similarly well with GA based approaches in terms of WOU as is shown in Figure 8a, BDOP seemed to save a little more time than GA in Figure 8b. Figures 8c and 8d are MATLAB simulation results on the same test sets. Again, we can see the relative positions and trends of the lines were mostly the same. Together with results in Figure 6b, the use of MATLAB simulations on larger-scale experiments (e.g. in Section 5.3.5) is justified.

Figure 9 compares the running time of static planning algorithms, where GA ran much slower than BDOP. Results also show that, as the complexity of problem increased (e.g. data chunks became larger), the running time of GA increased like linearly. With larger test sets, we found it took more than 20 minutes to generate a plan for one scenario with 1,000 data chunks using GA, but the performance outcome was not better than BDOP (actually slightly

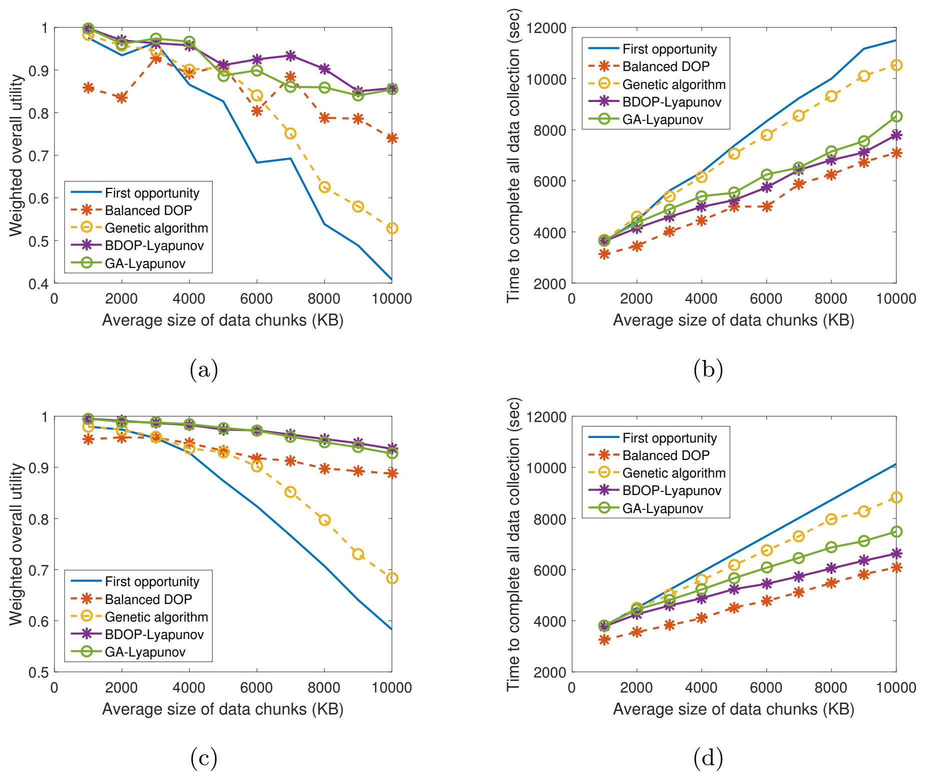


Figure 8: Simulation results for effect of data heterogeneity.

worse, perhaps due to the increased size of the solution space). Therefore, we will not evaluate GA in scalability studies.

5.3.3. Effect of Data Dynamics

Figure 10 displays two sets of simulation results on a medium test set, showing the effect of dynamics in data chunks (i.e uncertainties in chunk size and data priority). Different markers represent different levels of dynamics. The dynamics are implemented by the half proportional range of the uniformly distributed noise added to the size of each data chunk, this also represents the probability that the priority of each data chunk is elevated by one level. Different line styles represent different approaches: solid lines for **two-phase approaches**, dashed lines for **static-only approaches**, and dotted lines for **the naïve approach**.

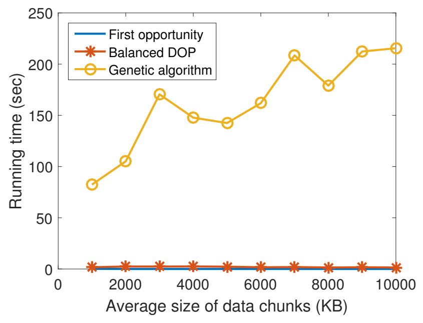


Figure 9: Comparison of running time of static planning algorithms.

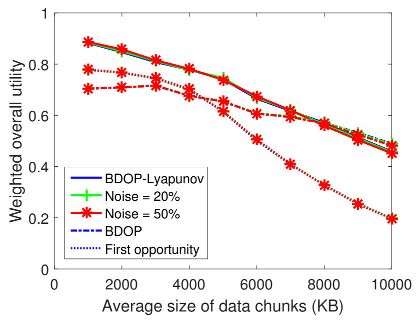
In the first set of experiments (Figures 10a and 10b), the expectation of chunk size remained unchanged. In the second set of experiments (Figures 10c and 10d), the expectation increased by the half proportional range. For example, if half range $l/2 = 20\%$ chunk size, a data chunk that was expected to be 5 MB may end up being $[5 \times (1 \pm 20\%)]$ MB in the first experiment, or $[5 \times (1 + 20\% \pm 20\%)]$ MB in the second experiment, and a data chunk of medium importance may turn out to be of high importance with a chance of 20%.

In Figures 10a and 10b, the three groups of tightly bundled curves in both figures suggested the dynamics in chunk size and data priority had limited effect on the overall utility of collected data, as long as the noise added to chunk size had a mean value of 0.

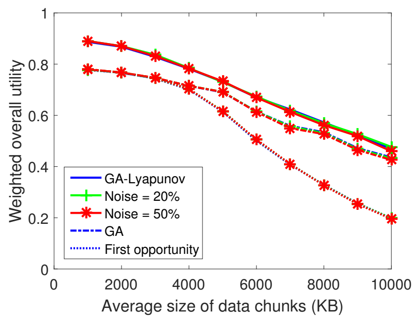
Figures 10c and 10d show the two-phase approaches using Lyapunov for dynamic adaptation always result in higher WOU, compared to corresponding static-only approaches, when chunks are bigger than expected in average. The decrease in WOU caused by data dynamics is also smaller in tests using two-phase approaches, which means having Lyapunov control based dynamic adaptation for the second phase improves the resilience of planned operation over data dynamics.

5.3.4. Effect of Mobility Dynamics

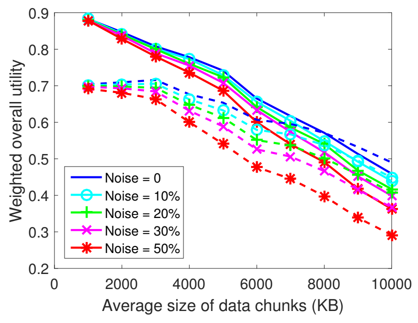
Figure 11 displays two sets of simulation results on a medium test set, showing the effect of mobility dynamics (i.e uncertainties in moving time). We il-



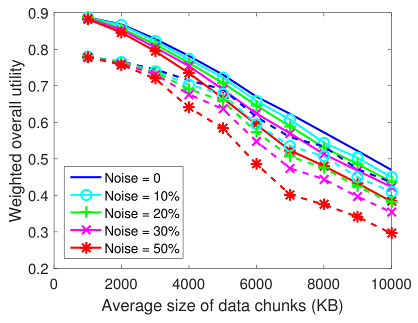
(a) BDOP and BDOP-Lyapunov



(b) GA and GA-Lyapunov



(c) BDOP and BDOP-Lyapunov

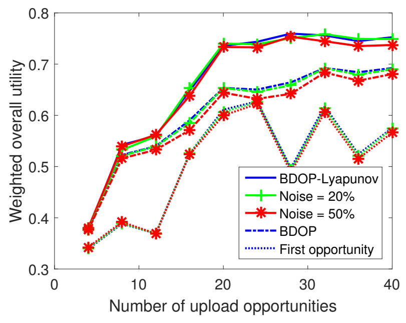


(d) GA and GA-Lyapunov

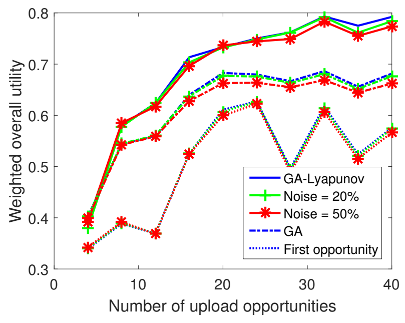
Figure 10: Simulation results for effect of different levels of data dynamics. CS stands for "chunk size".

lustrate results under different levels of dynamics (i.e. noise). In the graphs, different line styles represent different approaches: solid lines for **two-phase approaches**, dashed lines for **static-only approaches**, and dotted lines for **the naïve approach**.

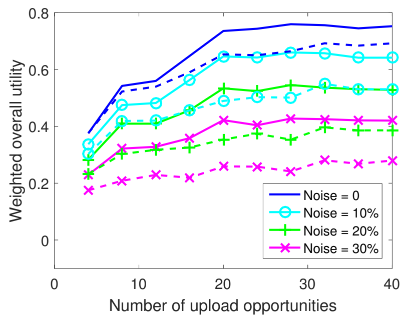
In the first set of experiments (Figures 11a and 11b), the expectation of moving time was unchanged. In comparison, in the second set of experiments (Figures 11c and 11d), the expectation of moving time increased. For example, under standard deviation $\mu = 20\%$ moving time, a movement that was expected to take 100 sec may end up taking $[100 \times (1-20\%)+y]$ sec in the first experiment, or $(100+y)$ sec in the second experiment, where y is an exponentially distributed random variable with $\lambda = 1/\mu = 1/20$.



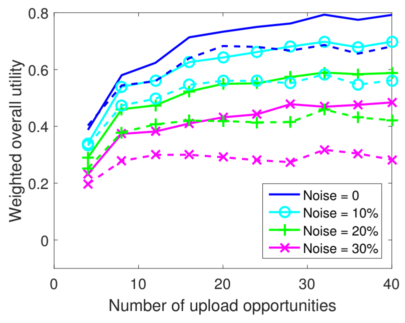
(a) BDOP and BDOP-Lyapunov



(b) GA and GA-Lyapunov



(c) BDOP and BDOP-Lyapunov



(d) GA and GA-Lyapunov

Figure 11: Simulation results for effect of different levels of dynamics in moving time.

In Figures 11a and 11b, the three groups of tightly bundled lines in both figures suggested the dynamics in moving time had limited effect on the overall utility of collected data, as long as the noise had a mean value of 0.

Figures 11c and 11d (when moving time is longer than expected in average) show that the two-phase approaches which use the Lyapunov control for dynamic adaptation always resulted in higher WOU, compared to the corresponding static-only approaches. Similar to the results for data dynamics, the decrease in WOU caused by mobility dynamics is also smaller with the two-phase approaches, which means using Lyapunov control based dynamic adaptation for the second phase also improved the resilience of planned operation under mobility dynamics.

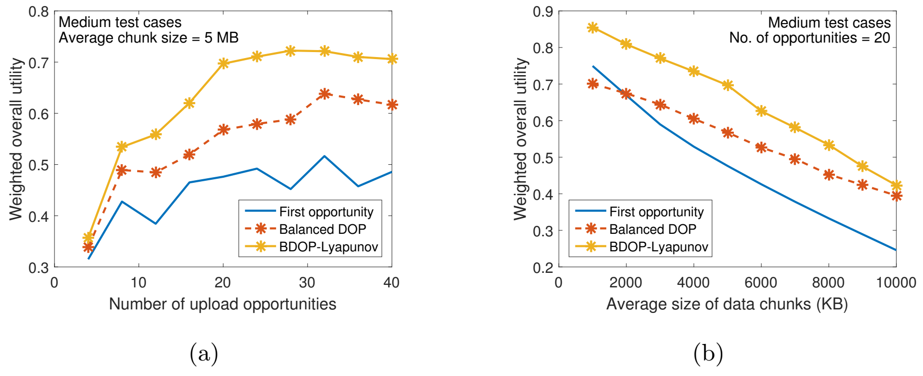


Figure 12: Simulation results for demonstration of scalability.

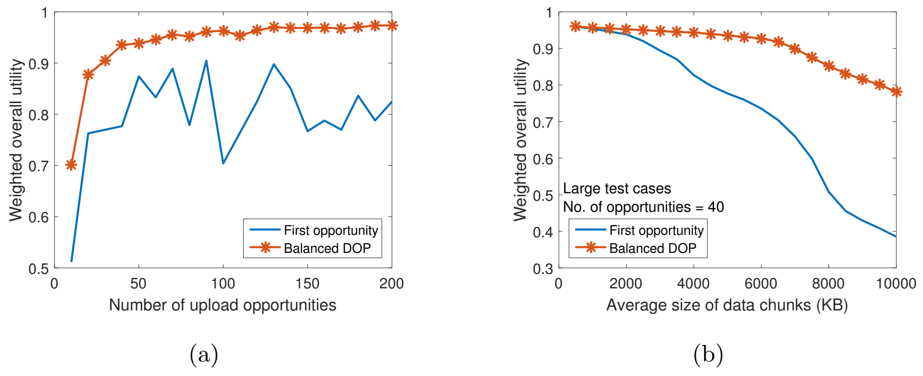


Figure 13: Simulation results for demonstration of scalability on large test sets.

5.3.5. *Scalability*

Results in Figures 12 and 13 show that our approach worked well as the scale of deployment increased. Figure 12 shows the results for medium test sets, and Figure 13 for large test sets. In Figure 12a, 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. In comparing Figure 12 with Figure 6 allows us to conclude that the advantage of our two-phase approach was more obvious when the deployment scales up.

Figure 13 compares the performance of BDOP with "first opportunity" on large test sets, and shows that BDOP performs well and is more stable as the system scales up. In summary, our two-phase approach using BDOP-Lyapunov exhibits superior performance as compared to other strategies.

6. Related Work and Conclusion

This paper addresses issues at the intersection between IoT systems and mobile computing when deployed at scale in real world community settings. Today, we observe increasing public interest in using IoT technologies in applications at personal and societal levels ranging from personalized healthcare to clean environments. Recent research in the design and deployment of large scale IoT architectures in community and city-scale deployments have brought new challenges to the forefront [1–5]. For example, the Community Seismic Network (CSN) [6, 7] is a participatory IoT system to help with early alerting of earthquakes in Southern California using cheap accelerometers attached to residents' personal computers and devices – resilient operation when there is large scale damage to infrastructures is a key challenge. The primary focus of IoT systems has been the enabling of in-situ sensing; our goal is to leverage the capabilities of mobile computing to further enhance and augment such deployments.

Mobile sensing has been a research topic for some years now has applications in several domains – significant advantages include extended coverage and reduced dependency on infrastructures. It has typically been used for data collection in large areas that cannot be blanketed uniformly with sensing devices. CarTel [17] is a delay-tolerant mobile computing system deployed in cars; it supports on-board mobile sensing and data uploading through the public network infrastructure and is used for gathering air quality and wireless network coverage information in cities. ZebraNet [15, 16] is an energy efficient mobile sensing network designed for long-term tracking and observation of wild animals. Due to the sparse distribution and low-energy design of nodes, dedicated data collectors wander around to collect data from the mobile sensors periodically. BikeNet [14] is a sensing platform for mapping cyclists experiences and incorporates both planned operation (tasking) and opportunistic operation. Recent work on mobile participatory sensing and crowdsourcing has enabled the creation of high-resolution sensing maps for communities and cities using large groups of cheap commodity sensors [6–8, 36] [11–13]. Often, in such settings, the coverage and deployment of sensing capabilities is tied to the availability of continuous network connectivity [8–10].

Using mobile devices for ferrying messages and data has been studied in the context of delay tolerant networks (DTN)[24–28, 37]; and data mules [29, 30] ; these approaches often assume absence or very limited access to communication infrastructure. Applications are assumed to be delay tolerant, i.e. meeting timing deadlines is not a critical goal and system design typically involves the use of multi-hop networks [29, 30]. Techniques for proactive and adaptive data transmission or uploading in this setting have been designed [38–48]. Delay tolerance of applications has also been exploited in the context of mobile crowdsourcing. O2SM (Offline Online Social Media) [38, 39] is a delay tolerant application framework that pre-fetches online social media content when connectivity is available so as to enable efficient offline access to social media streams based on its likelihood of being viewed. Another example is Piggyback Crowdsensing (PCS) [40, 41], a framework designed to reduce the energy overhead of smartphone based crowdsensing. Using prior knowledge about user habits and CPU/network profiles of applications, PCS determines when to trigger on-board sensors to minimize energy consumption. In these cases too, when and how to upload the collected information is not a significant concern.

The above efforts have inspired our approach to enhance IoT deployments with mobility – where new technologies, solutions and deployment settings have brought about a new set of dynamicity challenges. Prior work on IoT system architectures have considered using mobile nodes (mostly smartphones) to extend the coverage of sensing capability [2, 7] of in-situ nodes. Other efforts [49, 50] have also attempted to leverage node mobility to improve the overall performance and resilience of deployed IoT systems. Our focus is on application scenarios where communication infrastructure exists, but maybe non-uniform; based on our experience, network availability and connectivity are seldom uniform over space and time, even in highly developed communities and cities. Connectivity conditions can also be affected by external factors, e.g. disasters, large public events. We also place emphasis on the timely delivery of data and events since the IoT deployments may be used in time-sensitive applications, e.g. involving public safety. Overall, this places more stringent requirements on timely delivery of IoT data using the available communication infrastructure. Upload planning is a critical problem in this setting – this paper focuses on addressing this issue. In community deployment settings, one can exploit planned mobility (e.g using regularity of transit vehicles and community volunteers) for data gathering and upload planning; in such settings, there is a priori knowledge of communication network availability and mobility paths/-trajectories. Our findings through measurement studies also provide practical guidance on how this mobility should be tailored. We stress on the realistic "stop–operate–go" mobility pattern to ensure complete data collection. The two-phase upload planning approach and associated scheduling algorithms and adaptation policies are based on factors derived from real world measurement studies. In our work, we also highlight the need for timely data delivery under non-uniform network settings and show how planned mobility can be utilized to realize the dynamic communication needs. In this paper, we hone in on a specific problem – the upload planning problem in smart community IoT deployments and propose a two-phase approach to solve the optimization in community/city settings. We design SCALECycle, a prototype system, to conduct measurements in our real community testbeds. Experiments using data collected with SCALECycle, show that our two-phase approach using BDOP-Lyapunov outperforms both the naïve approach and the planned approach, in complex community settings of different scales with multiple different types of dynamics.

Our future work will leverage the use of multiple MDCs and multiple access networks in a cooperative manner. Currently, we assume that the path planning phase is done separately a priori, potentially by other planners; hence, we decouple the two aspects of path and upload planning and focus on details and realistic issues associated with upload scheduling. If multiple MDCs are all given different paths and work independently with the fixed paths, the proposed techniques and individual scheduling approaches will work. An interesting direction of future work is the collaboration of multiple MDCs that enables joint data collection and upload. Multiple MDCs become handy when the trajectories can be manipulated or MDCs could be arranged to deviate from their original plans to capture dynamic events – this brings about multiple questions. Which MDCs should have their path/plan modified? Should this be done for each MDC one at a time or should we design a joint planning mechanism? Should path planning and upload scheduling for each MDC be conducted in separate phases or jointly? How can MDCs be made to deviate from their planned and expected paths as new data collection events arise? In fact, we have addressed similar path planning and detour planning problems in previous efforts [51] without considering data uploading. In future work, we plan to study the interaction between MDCs (e.g. their collaboration and competition) and its impact on upload planning and path planning. Integrating a growing number of mobile devices into a changing and heterogeneous IoT ecosystem requires innovative networking approaches as well. New network architectures and protocols to handle wireless access networks into IoT settings are being designed and developed [52–54]. Our recent work on multi-network IoT deployments [55] illustrates the role of upcoming technologies such as SDN in managing the heterogeneous nature of IoT systems. Our future directions also include incorporating a multi-network upload setting to improve the overall efficiency of data collection in large-scale community deployments, which could be the key to driving future smart communities worldwide.

Acknowledgement

We thank the ISG and DSM teams for discussion. This project is funded under National Science Foundation Award No. CNS 1450768.

References

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

[2] R. Jalali, K. El-Khatib, C. McGregor, Smart city architecture for community level services through the internet of things, in: 2015 18th International Conference on Intelligence in Next Generation Networks (ICIN), 2015, pp. 108–113.

[3] C. Perera, A. Zaslavsky, P. Christen, D. Georgakopoulos, Sensing as a service model for smart cities supported by Internet of Things, Transactions on Emerging Telecommunications Technologies 25 (1) (2014) 81–93.

[4] K. Benson, C. Fracchia, G. Wang, Q. Zhu, S. Almomen, J. Cohn, L. D'arcy, D. Hoffman, M. Makai, J. Stamatakis, N. Venkatasubramanian, SCALE: Safe community awareness and alerting leveraging the internet of things, IEEE Communications Magazine 53 (12) (2015) 27–34.

[5] M. Y. S. Uddin, A. Nelson, K. Benson, G. Wang, Q. Zhu, Q. Han, N. Alhassoun, P. Chakravarthi, J. Stamatakis, D. Hoffman, L. Darcy, N. Venkatasubramanian, The Scale2 Multi-Network Architecture for IoT-Based Resilient Communities, in: 2016 IEEE International Conference on Smart Computing (SMARTCOMP), 2016, pp. 1–8.

[6] M. D. Kohler, T. H. Heaton, M.-H. Cheng, The Community Seismic Network and Quake-Catcher Network: Enabling structural health monitoring through instrumentation by community participants, in: SPIE Smart Structures and Materials+ Nondestructive Evaluation and Health Monitoring, International Society for Optics and Photonics, 2013, pp. 86923X–86923X.

[7] R. W. Clayton, T. Heaton, M. Chandy, A. Krause, M. Kohler, J. Bunn, R. Guy, M. Olson, M. Faulkner, M. Cheng, others, Community seismic network, Annals of Geophysics 54 (6).

[8] A. Boubrima, W. Bechkit, H. Rivano, Optimal deployment of dense wsn for error bounded air pollution mapping, in: International Conference on Distributed Computing in Sensor Systems (DCOSS) 2016, 2016.

[9] S. Toumpis, L. Tassiulas, Optimal deployment of large wireless sensor networks, IEEE Transactions on Information Theory 52 (7) (2006) 2935–2953.

[10] C. Zhu, C. Zheng, L. Shu, G. Han, A survey on coverage and connectivity issues in wireless sensor networks, Journal of Network and Computer Applications 35 (2) (2012) 619–632.

[11] S. Hachem, A. Pathak, V. Issarny, Probabilistic registration for large-scale mobile participatory sensing, in: 2013 IEEE International Conference on Pervasive Computing and Communications (PerCom), 2013, pp. 132–140.

[12] D. Christin, C. Rosskopf, M. Hollick, L. Martucci, S. Kanhere, IncogniSense: An anonymity-preserving reputation framework for participatory sensing applications, in: 2012 IEEE International Conference on Pervasive Computing and Communications (PerCom), 2012, pp. 135–143.

[13] W. Sun, Q. Li, C.-K. Tham, Wireless deployed and participatory sensing system for environmental monitoring, in: 2014 Eleventh Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), 2014, pp. 158–160.

[14] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, A. T. Campbell, The BikeNet Mobile Sensing System for Cyclist Experience Mapping, in: Proceedings of the 5th International Conference on Embedded Networked Sensor Systems, SenSys '07, ACM, New York, NY, USA, 2007, pp. 87–101.

[15] P. Zhang, C. M. Sadler, S. A. Lyon, M. Martonosi, Hardware Design Experiences in ZebraNet, in: Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems, SenSys '04, ACM, New York, NY, USA, 2004, pp. 227–238.

[16] T. Liu, C. M. Sadler, P. Zhang, M. Martonosi, Implementing Software on Resource-constrained Mobile Sensors: Experiences with Impala and ZebraNet, in: Proceedings of the 2Nd International Conference on Mobile Systems, Applications, and Services, MobiSys '04, ACM, New York, NY, USA, 2004, pp. 256–269.

[17] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, S. Madden, CarTel: A Distributed Mobile Sensor Computing System, in: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, SenSys '06, ACM, New York, NY, USA, 2006, pp. 125–138.

[18] L. Guo, Q. Han, Reliable data collection from mobile users with high data rates in wireless sensor networks, in: World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a, 2012, pp. 1–6.

[19] G. Jain, S. Babu, R. Raj, K. Benson, B. Manoj, N. Venkatasubramanian, On disaster information gathering in a complex shanty town terrain, in: 2014 IEEE Global Humanitarian Technology Conference - South Asia Satellite (GHTC-SAS), 2014, pp. 147–153.

[20] R. Raj, S. Babu, K. Benson, G. Jain, B. S. Manoj, N. Venkatasubramanian, Efficient Path Rescheduling of Heterogeneous Mobile Data Collectors for Dynamic Events in Shanty Town Emergency Response, in: 2015 IEEE Global Communications Conference (GLOBECOM), IEEE, 2015, pp. 1–7.

[21] B. Xing, S. Mehrotra, N. Venkatasubramanian, Radcast: Enabling reliability guarantees for content dissemination in ad hoc networks, in: INFOCOM 2009, IEEE, IEEE, 2009, pp. 1998–2006.

[22] G. Denker, N. Dutt, S. Mehrotra, M.-O. Stehr, C. Talcott, N. Venkatasubramanian, Resilient dependable cyber-physical systems: a middleware perspective, Journal of Internet Services and Applications 3 (1) (2012) 41–49.

[23] S. Sarma, N. Venkatasubramanian, N. Dutt, Sense-making from distributed and mobile sensing data: A middleware perspective, in: Proceedings of the 51st Annual Design Automation Conference, ACM, 2014, pp. 1–6.

[24] M. Zhao, Y. Yang, C. Wang, Mobile Data Gathering with Load Balanced Clustering and Dual Data Uploading in Wireless Sensor Networks, IEEE Transactions on Mobile Computing 14 (4) (2015) 770–785.

[25] W. Zhao, M. Ammar, E. Zegura, A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks, in: Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc '04, ACM, New York, NY, USA, 2004, pp. 187–198.

[26] A. Monfared, M. Ammar, E. Zegura, D. Doria, D. Bruno, Computational ferrying: Challenges in deploying a Mobile High Performance Computer, in: World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2015 IEEE 16th International Symposium on a, 2015, pp. 1–6.

[27] M. M. Bin Tariq, M. Ammar, E. Zegura, Message Ferry Route Design for Sparse Ad Hoc Networks with Mobile Nodes, in: Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc '06, ACM, New York, NY, USA, 2006, pp. 37–48.

[28] T. Wang, C. P. Low, The general message ferry route (MFR) problem and the An-Improved-Route (AIR) scheme, Computer Networks 56 (4) (2012) 1442–1457.

[29] D. Kim, R. N. Uma, B. H. Abay, W. Wu, W. Wang, A. O. Tokuta, Minimum latency multiple data mule trajectory planning in wireless sensor networks, Mobile Computing, IEEE Transactions on 13 (4) (2014) 838–851.

[30] S. Y. Wu, J. S. Liu, Evolutionary path planning of a data mule in wireless sensor network by using shortcuts, in: 2014 IEEE Congress on Evolutionary Computation (CEC), 2014, pp. 2708–2715.

[31] Q. Zhu, M. Y. S. Uddin, Z. Qin, N. Venkatasubramanian, Upload planning for mobile data collection in smart community internet-of-things deployments, in: Smart Computing (SMARTCOMP), 2016 IEEE International Conference on, IEEE, 2016, pp. 1–8.

[32] M. J. Neely, Stochastic network optimization with application to communication and queueing systems, Synthesis Lectures on Communication Networks 3 (1) (2010) 1–211.

[33] Z. Qin, L. Iannario, C. Giannelli, P. Bellavista, G. Denker, N. Venkatasubramanian, MINA: A reflective middleware for managing dynamic multinetwork environments, 2014, pp. 1–4.

[34] MQTT, `http://mqtt.org/`.

[35] QualNet, `http://qualnet.com/`.

[36] W. Sun, Q. Li, C.-K. Tham, Wireless deployed and participatory sensing system for environmental monitoring, in: Sensing, Communication, and Networking (SECON), 2014 Eleventh Annual IEEE International Conference on, IEEE, 2014, pp. 158–160.

[37] T. Black, V. Mak, P. Pathirana, S. Nahavandi, Using Autonomous Mobile Agents for Efficient Data Collection in Sensor Networks, in: Automation Congress, 2006. WAC '06. World, 2006, pp. 1–6.

[38] Y. Zhao, N. Do, S.-T. Wang, C.-H. Hsu, N. Venkatasubramanian, O2SM: Enabling efficient offline access to online social media and social networks, in: ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, Springer, 2013, pp. 445–465.

[39] N. Do, Y. Zhao, C.-H. Hsu, N. Venkatasubramanian, Crowdsourced mobile data transfer with delay bound, ACM Transactions on Internet Technology (TOIT) 16 (4) (2016) 28.

[40] N. D. Lane, Y. Chon, L. Zhou, Y. Zhang, F. Li, D. Kim, G. Ding, F. Zhao, H. Cha, Piggyback CrowdSensing (PCS): Energy Efficient Crowdsourcing of Mobile Sensor Data by Exploiting Smartphone App Opportunities, in: Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems, SenSys '13, ACM, New York, NY, USA, 2013, pp. 7:1–7:14.

[41] H. Xiong, D. Zhang, G. Chen, L. Wang, V. Gauthier, CrowdTasker: Maximizing coverage quality in Piggyback Crowdsensing under budget constraint, in: 2015 IEEE International Conference on Pervasive Computing and Communications (PerCom), 2015, pp. 55–62.

[42] J. Ma, N. Lu, H. Zhang, Pso-based proactive routing in delay tolerant network, in: Cyberspace Technology (CCT 2014), International Conference on, IET, 2014, pp. 1–4.

[43] C. Raffelsberger, H. Hellwagner, A hybrid manet-dtn routing scheme for emergency response scenarios, in: Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on, IEEE, 2013, pp. 505–510.

[44] R. Wohlers, N. Trigoni, R. Zhang, S. Ellwood, Twinroute: Energy-efficient data collection in fixed sensor networks with mobile sinks, in: Mobile Data Management: Systems, Services and Middleware, 2009. MDM'09. Tenth International Conference on, IEEE, 2009, pp. 192–201.

[45] A. Petz, J. Enderle, C. Julien, A Framework for Evaluating DTN Mobility Models, in: Proceedings of the 2Nd International Conference on Simulation Tools and Techniques, Simutools '09, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 2009, pp. 94:1–94:8.

[46] Z. Lu, X. Sun, T. La Porta, Cooperative data offloading in opportunistic mobile networks, in: Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on, IEEE, 2016, pp. 1–9.

[47] M. Y. S. Uddin, H. Wang, F. Saremi, G.-J. Qi, T. Abdelzaher, T. Huang, Photonet: a similarity-aware picture delivery service for situation awareness, in: Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd, IEEE, 2011, pp. 317–326.

[48] W. Gao, G. Cao, A. Iyengar, M. Srivatsa, Supporting cooperative caching in disruption tolerant networks, in: Distributed Computing Systems (ICDCS), 2011 31st International Conference on, IEEE, 2011, pp. 151–161.

[49] D. P. Abreu, K. Velasquez, M. Curado, E. Monteiro, A resilient internet of things architecture for smart cities, Annals of Telecommunications (2016) 1–12.

[50] A. Capponi, C. Fiandrino, C. Franck, U. Sorger, D. Kliazovich, P. Bouvry, Assessing performance of internet of things-based mobile crowdsensing systems for sensing as a service applications in smart cities, in: 8th IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 2016.

[51] Y. Zhao, C.-C. Liao, T.-Y. Lin, J. Yin, N. Do, C.-H. Hsu, N. Venkatasubramanian, Smartsource: A mobile q&a middleware powered by crowdsourcing, in: Mobile Data Management (MDM), 2015 16th IEEE International Conference on, Vol. 1, IEEE, 2015, pp. 145–156.

[52] A. Venkataramani, J. F. Kurose, D. Raychaudhuri, K. Nagaraja, M. Mao, S. Banerjee, Mobilityfirst: a mobility-centric and trustworthy internet architecture, ACM SIGCOMM Computer Communication Review 44 (3) (2014) 74–80.

[53] D. Raychaudhuri, K. Nagaraja, A. Venkataramani, Mobilityfirst: a robust and trustworthy mobility-centric architecture for the future internet, ACM SIGMOBILE Mobile Computing and Communications Review 16 (3) (2012) 2–13.

[54] S. Li, J. Chen, H. Yu, Y. Zhang, D. Raychaudhuri, R. Ravindran, H. Gao, L. Dong, G. Wang, H. Liu, MF-IoT: A MobilityFirst-Based Internet of Things Architecture with Global Reach-Ability and Communication Diversity, in: 2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI), 2016, pp. 129–140.

[55] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, N. Venkatasubramanian, A software defined networking architecture for the internet-of-things, in: Network Operations and Management Symposium (NOMS), 2014 IEEE, IEEE, 2014, pp. 1–9.

[56] R. Zivan, H. Yedidsion, S. Okamoto, R. Glinton, K. Sycara, Distributed constraint optimization for teams of mobile sensing agents, Autonomous Agents and Multi-Agent Systems 29 (3) (2015) 495–536.

[57] H. Huang, A. V. Savkin, Path planning algorithms for a mobile robot collecting data in a wireless sensor network deployed in a region with obstacles, in: 2016 35th Chinese Control Conference (CCC), 2016, pp. 8464–8467.

[58] D. Gavalas, I. E. Venetis, G. Pantziou, C. Konstantopoulos, An iterated local search approach for multiple itinerary planning in mobile agent-based sensor fusion, in: 2015 11th International Conference on Mobile Ad-hoc and Sensor Networks (MSN), IEEE, 2015, pp. 1–7.

[59] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (iot): A vision, architectural elements, and future directions, Future generation computer systems 29 (7) (2013) 1645–1660.

Algorithm 1: Balanced Deadline-Opportunity-Priority static planning algorithm for finding a plan λ to maximize $U_e(\{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)$

```
function planBalancedDOP ($\{\boldsymbol{a}\}$,$\{\boldsymbol{u}\}$);
Input  : Data chunks $\{\boldsymbol{a}\}$ and upload opportunities $\{\boldsymbol{u}\}$
Output: Plan vector $\mathbf{j}$ corresponding to plan $\lambda$
Initialize $N \leftarrow \{\boldsymbol{a}\}$.size, $M \leftarrow \{\boldsymbol{u}\}$.size, and $\mathbf{j} \leftarrow [-1, \ldots, -1]_N^T$;
W $\leftarrow \{\boldsymbol{a}\}$; // Put all chunks in $\{\boldsymbol{a}\}$ into W
while $W$ is not empty do
    $\boldsymbol{a}_i \leftarrow$ Take the chunk with the earliest deadline in W;
    Initialize S $\leftarrow \{\}$, totalSac $\leftarrow 0$, and $\mathbf{b} \leftarrow \mathbf{j}$;
    loop
        $\boldsymbol{u}_j \leftarrow$ Fastest $\boldsymbol{u}$ to upload $\boldsymbol{a}_i$ in time, after other planned chunks
        if $\boldsymbol{u}_j$ is not null then
            $b_i \leftarrow j$; W $\leftarrow$ S + W; $\mathbf{j} \leftarrow \mathbf{b}$; break; // Successfully planned $\boldsymbol{a}_i$
        else
            $\boldsymbol{a}_k \leftarrow$ A planned chunk with least importance level to sacrifice;
            if $\boldsymbol{a}_k$ is not null then
                totalSac $\leftarrow$ totalSac + $f\big(\Delta_e(\boldsymbol{a}_k, \{\boldsymbol{a}\}, \{\boldsymbol{u}\}, v_e, \lambda, l)\big)$;
            end
            if $totalSec > p(\boldsymbol{a}_i)$ then
                $\boldsymbol{u}_j \leftarrow$ Fastest $\boldsymbol{u}$ to upload $\boldsymbol{a}_i$ after other planned chunks;
                $b_i \leftarrow j$; $\mathbf{j} \leftarrow \mathbf{b}$; break; // Too much sacrifice, give up $\boldsymbol{a}_i$
            else
                S $\leftarrow \{\boldsymbol{a}_k\}$ + S; $b_k \leftarrow -1$; W $\leftarrow \{\boldsymbol{a}_i\}$ + W; // Sacrificed $\boldsymbol{a}_k$
            end
        end
    end
end
return $\mathbf{j}$;
```

Table 1: Specifications of small, medium, and large test sets.

Scenario Characteristics		**Small**	**Medium**	**Large**
MDC	Constant Speed V/ms^{-1}	5 [a]		
	Est. Conn. Time T_c/s	12		
Path	Length Exp. $\mathbf{E}[L]/km$	12	24	200
Data Sites and Chunks	Bandwidth $R_a/Mbps$	12		
	Total Number N	120	120	1,000
	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. $\Delta d/s$	$U(-40, 200)$		
Oppor-tunities	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 probabilities 0.6, 0.3, 0.1, respectively.

[c] The minimum valid bandwidth is 200 Kbps, and the same to other test cases.