Gander: Mobile, Pervasive Search *of* the Here and Now *in* the Here and Now

Jonas Michel, *Member, IEEE,* Christine Julien, *Senior Member, IEEE* and Jamie Payton, *Member, IEEE*

***Abstract*—The vision of the Internet of Things will enable networked environments populated with vast amounts of data that can be exploited by humans. The volume of digitally available data in such emerging computing spaces presents an imminent need for search mechanisms that enable humans and applications to find relevant information within their digitally accessible physical surroundings. This paper presents Gander, a search engine for these pervasive computing spaces enabled by the Internet of Things and characterized by large volumes of highly transient data. Gander is founded on a novel conceptual model of search that resolves queries *about* a user's here and now by leveraging proximally-available resources *in* the here and now. We formally describe the model underlying Gander, describe the networking protocols that enable Gander's search, and provide a realization of Gander via an extensible framework. Employing this Gander framework, we describe a concrete middleware implementation for wirelessly networked environments. We evaluate this implementation of Gander through a user study that examines the perceived utility of myGander, a real-world mobile application enabled by the Gander middleware, and we benchmark the performance of Gander in large pervasive computing spaces through network simulation.**

I. Introduction

THE same fundamental need to share and discover information that fuels the widespread use of existing Internet search engines is also an essential requirement for in the emerging Internet of Things (IoT). Visions of the IoT include massive numbers of distributed and digitally accessible objects that represent the state of the world and its inhabitants. In these settings, wireless connections support opportunistic interactions between humans, the devices they wear and carry, and intelligent sensors embedded in everyday objects and natural landscapes. This tight integration of sensing, computation, and communication with the physical and social environment results in large volumes of spatiotemporal data generated at rapid rates. These pervasive computing spaces are an essential component of the Internet of Things; as these spaces increasingly become a reality, it becomes essential to provide approaches to help users find the information they need—in a way that reflects what is around them, right here and now—as they move through a densely populated and rapidly changing information space.

Consider the following situation, which contrasts the information needs in using traditional Internet-based information retrieval versus searching for information about the here and now in rapidly changing pervasive computing spaces. A traditional Internet search engine may be used to follow news updates and social feeds about a popular parade. While at the parade, a user might wish to know which areas are the least crowded and provide the best views for children right now, or what are the best foot traffic patterns for navigating to see a particular parade attraction. A police officer may want to monitor patterns of movement or nearby unruly crowds. In this situation, data is generated and shared by spectators, parade participants (e.g., marching bands), objects in the environment (e.g., parade floats or city infrastructure) or police or other officials. Other examples similarly highlight the contrast in Internet search needs versus those found in spaces that contain spatiotemporally relevant data. Using the Internet, one may find available train routes and timetables, whereas a traveler hurrying to board a crowded train may need to search for the closest available second class seat. The traveler's search may be supported by devices in the station, embedded in the train, or carried by other passengers. When planning a trip to an amusement park, one may find directions and hours of operations using the Internet; visitors at the park may wish to know which rides have the shortest wait right now, where their friends are, or which nearby vendor is currently making a fresh batch of funnel cakes.

Problem statement: A key challenge to realizing our vision of search is that the information available in these spaces is subject to high levels of dynamics; time passes, devices and users are constantly in motion, social patterns evolve, data is moved, and information expires. In addition, the ratio of data used to data generated is quite low. Existing systems that have enabled access to localized data provide only capabilities for searching relatively static data instead of the inherently ephemeral data that will characterize the Internet of Things; much of this data cannot be easily indexed outside of the here and now. Supporting the execution of queries over data that represents the here and now requires a new perspective on the search engine architecture that relies on search execution capabilities that take advantage of both opportunistic interactions between peer computing devices and the emerging availability of localized infrastructure in the form of cloudlets [1]. Enabling expressive search over dynamic data also requires understanding and efficiently collecting and representing the *context* of that data in a pervasive computing

J. Michel and C. Julien are with the Department of Electrical and Computer Engineering at the University of Texas at Austin, e-mail: {jonasrmichel, c.julien}@mail.utexas.edu

J. Payton is with the University of North Carolina at Charlotte, e-mail: payton@uncc.edu

A previous version of this paper appeared in Proceedings of the 8^{th} International ICST Conference on Mobile and Ubiquitous Systems (MobiQuitous), 2011

space. That context has a significant impact on the *relevance* of a particular data item to a particular search, which must be able to be captured in the search engine. This relevance is further influenced by intrinsic characteristics of data that arises when one speaks of an Internet of Things; elements of this data are inherently correlated with each other across space and time, and those correlations (and their dynamics) can impact the ability to resolve queries for that data.

Contributions: As a starting point to address these needs, we have previously introduced the Gander conceptual model [2] for expressive search in the here and now. This prior work formalized the Gander model and the search algorithms that underpin the Gander search engine. In this paper, we extend the Gander conceptual model with an expressive graph-based model of the data that is searchable by a Gander query (Section IV-B). Specifically, through the definition of two graph-based data structures, the Gander middleware provides an abstraction of the globally available datums in a pervasive computing space and expressively captures the *context* of those data items, including the relationships of data items to their environments and to other data items in the space. This paper also introduces a REST-ful implementation of the Gander search engine middleware that simplifies application development (Section IV-A) and demonstrates the use of the interface to build a case study application (Section V). Finally, we provide an evaluation of Gander through both a user study on our real world application and through performance benchmarks in a large scale simulation (Section VI).

II. Related Work

Search has become one of the most popular services on the Web and, in many cases, defines how users interact with the World Wide Web on a daily basis. Just as users today rely on Web search to find documents online, users in the future will demand on-demand access to real world information generated by their surroundings as objects in those surroundings become increasingly digitally accessible. The Gander search engine aims to provide the necessary support for IoT applications requiring relevant information in rapidly changing, data-rich, networked spaces. Discovery, acquisition, and administration of dynamic information produced in these environments is by no means a novel goal. Indeed, managing and coordinating access to transiently available data and resources will inherently characterize most, if not all, IoT applications. Gander, however, targets a specific type of scenario, where many common assumptions about the network, data, and node behavior simply do not hold. In this section, we overview prominent systems that, either explicitly or implicitly, support the search for real-world information (i.e., information generated by mobile devices, sensors, human users, etc.). Some have been designed with search as a first class citizen; however, others provide mechanisms that could facilitate search.

A large body of work of is concerned with efficient data acquisition in wireless sensor networks (WSNs), where severely resource-constrained sensors deployed for environmental monitoring, surveillance, phenomena tracking, etc. generate huge volumes of data. Data stream systems treat the sensory data collected by a WSN as a set of continuous streams and provide distributed query processing mechanisms to resolve queries in an energy-aware fashion. TinyDB [3] supplies a toolbox of data stream-based query processing techniques that provide programmers with on-demand access to the WSN through a SQL-like interface. Each TinyDB query processing technique is characterized by power- and bandwidth-aware heuristics that dictate where, when, and how data is physically acquired from sensors. The Regiment macroprogramming system [4] enables developers to program a WSN at the global level by specifying *region streams*, or representations of spatiotemporally varying collections of node state, at compile time to access collective groups of data from groups of sensors sharing geographic, topological, or logical relationships. Similarly, logical neighborhoods [5] provide access to dynamically formed groups of sensor nodes satisfying a set of logical constraints (e.g., communication costs and node characteristics) as a single *virtual* node. These and similar stream-based systems provide the style of resource-aware data access we desire, but they presuppose a relatively static network of nodes formed by a *known* set of sensors affixed in a physical space and "rooted" at a base station (the network sink). In our target environments, networks are formed opportunistically and comprise both stationary (e.g., objects embedded in the environment) and mobile (e.g., smartphones carried by human users) devices. In other words, neither the participating devices nor their network topology may be known ahead of time.

In both the Internet of Things (IoT) [6] and Web of Things (WoT) [7], ordinary objects are imbued with sensing, networking, and otherwise "smart" capabilities enabling their access via a network connection. Given such a world of digitally accessible "things," many recent systems have attempted to address *entity discovery*[1]. Unlike data stream query processing where queries are resolved over a given set of devices, entity discovery is concerned with searching for real-world entities (i.e., people, places, things) and their representative sensors, potentially in a desired state. Generally speaking, these systems support keyword search over static or pseudo-static sensor metadata (e.g., Snoogle/Microsearch [9], [10], MAX [11], SenseWeb [12]) or additionally over dynamic sensor states and location (e.g., Dyser [13], IoT-SVK [14], CASSARAM [15]). Entity discovery shares the same fundamental motivation as Gander, but each of these approaches presume searchable sensor resources are accessible via a reliable Internet connection and therefore employ centralized resources to intelligently index sensors' metadata, location, and changing state (e.g., using prediction models [13]). No such assumptions can be made about device-to-device connectivity in our target environments—Internet connectivity may be impractical, infeasible, or simply undesirable. Therefore, Gander must operate in a purely distributed fashion without assuming complete reliance on centralized resources.

Decentralization offers an inherent scalability and robustness to failures, making it a natural design foundation for large-scale applications that target dynamic and unreli-

[1]We refer the reader to [8] for an excellent survey of entity discovery systems.

able networks. Data space and coordination models targeting distributed systems (e.g., tuple spaces [16], distributed databases [17], and distributed hash tables [18]) attempt to abstract away the distributed and disconnected nature of resources (e.g., mobile devices) by providing access to them as if they were a holistic global virtual data structure [19]. Similarly, distributed routing and location systems (e.g., Tapestry [20], Pastry [21], and CAN [22]) emerged to tackle the challenge of routing requests to pertinent content within large scale networks where failures and periods of heavy loads are the norm. Members of this family of systems typically employ statistical- or hash-based data replication to mitigate failures and facilitate low latency interactions. These abstractions enable convenient and simplified interfaces for an otherwise complex system but often require significant overhead to accurately maintain distributed data structures, routing tables, roles, and schemas. The high degrees of network churn induced by real-world dynamics and the sheer rate and volume of generated data in our target environments renders such approaches ineffective. Nevertheless, a key commonality among these approaches is the implicit or explicit parameterization of interaction based on locality (e.g., logical or physical).

In many pervasive computing systems, interaction is parameterized by some notion of *context* in an effort to facilitate low latency data access, coordination, communication, etc. Such *context-aware* systems may impose application-level overlays to, for example, keep data about events close to where it will likely be spatiotemporally relevant [23]. Alternatively, some systems enable developers to specify application-specific patterns that dictate how and when data is moved and shared (e.g., TOTA [24]), relieving applications from decisions regarding the physical transport of data. The publish/subscribe paradigm has received much attention within the context-aware domain due to its high degrees of decoupling and flexibility. Existing publish/subscribe frameworks parameterize event distribution by social metrics [25], physical proximity [26], contextual relations [27], temporal properties [28], degree of interest matching [29], and even complex formulations of context [30], [31]. The ability to represent, infer, then leverage some notion of context has proved to be extremely beneficial in real-world applications and remains an active area of research. While not always the case, a critical assumption made by existing context-aware systems is that generated events and data will, in general, be consumed by "interested" parties. In our targeted environments, no such assumption can be made; there is no guarantee that data generated by a human user or some smart object or embedded device will be "of interest" to an application or another user. Indeed, the amount of data potentially produced by real-world events vastly outweighs the amount of data consumed.

In summary, our target environments represent some of the most challenging conditions found in computing today—large scale heterogeneous networks, high degrees of mobility and network churn, large volumes of information generated at rapid rates, and a small ratio of data consumed to data generated. We position Gander as a first cut search engine specifically designed to operate within in this emerging cyber-physical space.

III. Background: The Gander Conceptual Model

Gander performs queries about the here and now in the here and now, using locally available capabilities without reliance on a globally accessible index. In this section, we review the Gander conceptual model [2] and introduce a structural data model for pervasive computing spaces, which defines the substrate over which queries execute. We give brief examples of datums and queries over them; our examples are taken from a pervasive computing application that enables a user carrying a mobile device at a parade to search for information related to his experience. The model is essential to a rigorous understanding of the quality with which protocols resolve users' searches, and we use it to develop the Gander search engine; Section IV describes how Gander's design and implementation reifies the conceptual model presented in this section. This implementation relies on *sampling* as a fundamental component of search; this sampling is formalized in our conceptual model and driven by the modes of query processing described in this section.

A. A Model of Queries in Pervasive Computing Spaces

We assume that nodes (e.g., mobile devices carried by human users or smart objects embedded in the environment) issue queries that are evaluated using information provided by other nodes. As an example, a parade-goer may issue a query to discover nearby shady benches that are close to funnel cake vendors. A *datum* provides information about the here and now (e.g., a measure of some condition of the environment) and is associated with meta-data that describes its *context* (e.g., the device(s) that generated it, the location, a timestamp, or even the data's volatility or freshness). Continuing our example, a pervasive computing space may include datums generated by food vendors indicating their menus, prices, and locations; datums may be generated by "things" such as park benches, providing static information such as location and dynamic information such as the current shade conditions. We have formalized the Gander query model previously [2]; here we provide a comprehensive summary of that model. Our model relies on *partial functions*, which are not required to be defined for every element of their domains. Partial functions naturally lend themselves to *incremental* query processing, which can consider additional datums as they are discovered.

Gander Queries. A Gander query is a partial function $G_h : D \rightarrow \Phi$; D contains all datums in the entire pervasive computing space, Φ is the domain of relevance, and h is the node issuing the query. A *datum* is a pair, (ν, d); ν is the data value, and d is the value's meta-data. The meta-data captures the *context* of the datum (e.g., its relationship to the space and potentially to other datums that inhabit that space).

Abstractly (and from a global perspective), the Gander query function applies a sequence of filters to all of the available datums and returns a list of datums, sorted by increasing relevance. Every returned datum must be *reachable* (from a networking perspective), either because the peer device owning the datum is connected via a mobile ad hoc network or because the datum is stored in a locally available cloudlet. Every valid result must "match" the search, which we refer to as *query*

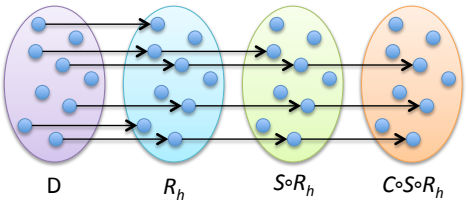


Fig. 1. Query processing functions

resolution. A query can also include one or more *constraints*. While query resolution focuses on the content of the datum (i.e., ν), evaluating constraints may rely on context captured in the meta-data, d. For example, query resolution may identify datums indicating nearby park benches; constraints ensure that benches discovered are in the shade. A *relevance metric* compares valid results to each other, using information from both ν and d. A search for a shady bench could favor closer benches or benches close to funnel cake vendors. A query can use multiple relevance metrics evaluated independently or using weighted statistics.

Practically, a gander query is implemented using a multihop *query processing protocol*, $\mathrm{QP_h}$, that distributes a query across the network. Informally, $\mathrm{QP_h}((\nu, \mathrm{d})) = (\nu, \mathrm{d})$ if $(\nu, d) \in D$ is a "valid" result; otherwise $\mathrm{QP_h}((\nu, \mathrm{d}))$ is undefined. More concretely, $\mathrm{QP_h}$ is a *filter* on D with three pieces:

Reachability. The partial function $\mathcal{R}_h : D \to D$ expresses whether (ν, d) is *reachable* from h; if not, $\mathcal{R}_h((\nu, d))$ is not defined. We focus on *query* reachability, the ability of node h to send a query to some other node h' *and* receive a response [32]; $\mathcal{R}$ depends on actual communication capabilities and the protocols used.

Query Resolution. The partial function $\mathcal{S} : D \to D$ is defined for each $(\nu, d) \in D$ that matches the search.

Query Constraint. The partial function $\mathcal{C} : D \to D$ is defined for each $(\nu, d) \in D$ that satisfies the query constraints; $\mathcal{C}$'s resolution may rely on the datum's *meta-data* (d).

These functions filter D to the subset of reachable datums that satisfy the search string and constraints; Fig. 1 shows the composition, $\mathrm{QP_h} = \mathcal{C} \circ \mathcal{S} \circ \mathcal{R}_\mathrm{h}$, assuming a snapshot of all available datums. Conceptually, a Gander query should return *all* matching datums that are reachable. Practically, of course, many aspects of real IoT environments limit these capabilities. One possibility, in Internet connected scenarios, is to use a centralized monitor (as is done in systems like Snoogle [10] and MAX [11]) to verify that all reachable datums are returned. We do not include such a monitor in the Gander conceptual model, as we focus on a device-to-device style of query processing. The cloudlet-based query resolution process we describe as part of the realization of the Gander search engine in Section IV-C does, however, lend itself naturally to the integration of a centralized monitor, which we use in our implementation to evaluate the Gander search engine's ability to achieve the ideal (i.e., returning all matching reachable datums) when using a device-to-device approach to query processing.

Relevance. A *relevance metric*, $\mathcal{M}_i : D \to \phi_i$ gives the distance of a datum (ν, d) from an ideal. A Gander query may entail more than one relevance metric; a Gander query, $G_h = \mathcal{K}_{\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n\}} \circ \mathcal{C} \circ \mathcal{S} \circ \mathcal{R}_h$, is therefore a partial function maps valid results on to the multidimensional space $\Phi = \phi_1 \times \phi_2 \times \cdots \times \phi_n$. $G_h((\nu, d))$ is not defined if (ν, d) is not reachable or does not satisfy the search string or constraints; otherwise $G_h((\nu, d))$ is an n-tuple, where field i has the value $\mathcal{M}_i((\nu, d))$. We can plot each valid result in a multidimensional space, where each axis represents one of the n metrics[2]. We can apply different distance measures from a point in this space to the origin. For example, datums can be ranked using a primary metric, and later relevance metrics can break ties. We can also map each n-tuple to a single value (e.g., $m : \Phi \to \mathbb{R}$) using different weights for different metrics.

B. A Model of Data in Pervasive Computing Spaces

From a query's perspective, D is a *global virtual data structure* (GVDS) [19]; resolving a concrete Gander query requires accessing components of this global structure that are distributed in a dynamic and unpredictable network. Practically, D is not constructed centrally; instead datums in D are generated by and stored at devices distributed in the pervasive computing space. In a most basic sense, these storage location are simply peer devices; Gander can also support *cloudlet*-style storage locations [33], which allow lightweight, highly localized pieces of infrastructure to support pervasive computing applications in the here and now.

Gander assumes that each node (both peers and local cloudlet storage providers) implements a *local tuple space* containing semi-structured data [34]. A datum's ν is a tuple that consists of an unordered set of name/value pairs. Meta-data, d, is treated similarly, but the tuple is constructed on-the-fly by assessing the instantaneous context. Queries, constraints, and relevance are represented as *patterns* that restrict a matching tuple's fields and values or compute across multiple fields or tuples. Data can be generated, destroyed, changed, and moved arbitrarily; Gander's query model is independent of these processes, however, we assume data is generated and stored close in space and time to the phenomena it describes.

C. Processing Gander Queries

Acquiring a global view is infeasible in pervasive computing spaces; protocols must instead operate only over locally available data. We relate query processing to formal definitions of *sampling* the available data, which enables reasoning about results' quality. We resolve constraints and relevance metrics by inspecting a result's *context*, accessible through the metadata. We describe the concrete data structures we use to achieve this in Section IV. Gander's partial functions lend themselves to *incremental* protocols, which gradually fill in the various filters defining G_h. These protocols sample the information space to incrementally build a query result Q that represents the desired result G_h. A Gander *query processing protocol* distributes a query to other nodes in the space, including both peer devices and cloudlet-based storage locations. Gander can

[2]The origin could itself be relative to the results; this causes a translation of the axes of the multidimensional space; the same distance functions apply.

use the query's contents to direct how and to which other nodes a query is distributed; ultimately the goal is to efficiently collect and present only data that is most relevant.

Gander exploits the relationship between query processing protocols and their spatiotemporal sampling for search processing. Gander query protocols must provide temporally-sensitive sampling, by processing queries *on-demand*, and spatially-sensitive sampling, determined by the protocol that selects the space to sample. A query is distributed to any node (peer or cloudlet device) located in or responsible for the selected space at the time the query is issued. We quantify spatial quality through *coverage*, which measures how much of the target space the query sampled, and *distribution*, which measures how evenly the query sampled the space. Gander provides four sampling styles, which trade quality for cost, measured in terms of latency and network overhead. Fig. 2 shows the styles and their relationships to spatial sampling.

Flooding. These protocols attempt to reach every node and examine every datum belonging to the space, i.e., they attempt to resolve the function G_h exactly. However, because of the potential scale of pervasive computing spaces, Gander employs *constrained flooding*, in which propagation is limited by network hops (Fig. 2(a)). Flooding attempts high *coverage* and reflects the actual evenness (or unevenness) of the nodes' *distribution*.

Random. Random sampling protocols propagate queries similarly to flooding but reduce responses. In random sampling (Fig. 2(b)), the likelihood of responding is parameterizable; the goal is for Q (the query result) to approximate G_h with an even *distribution* across the pervasive computing space but a reduced *coverage* (i.e., Q is smaller is total size than G_h but covers the same overall space).

Probabilistic. In probabilistic sampling (Fig. 2(c)), each node that receives a query probabilistically forwards it to peers; this reduces the overhead, but nodes closer to the issuer are more likely to receive queries [35]. These protocols trade cost for coverage at the edges of the target area, while maintaining even distribution near the query issuer. This style is similar to approaches used for sampling in geographic information systems [36].

Greedy Gossip. In greedy gossip (Fig. 2(d)), nodes receiving a query first evaluate it then retransmit it with a probability dependent on the local result. The intuition is that real-world events are tied to space and time [23], and therefore effort should be spent "accelerating" a query towards spaces with more relevant data and "decelerating" it when little relevant data is present. As such, this strategy seeks to provide good *coverage* and *distribution* only where necessary. In relation to the formal Gander query G_h, greedy gossip attempts to intentionally avoid collecting the pieces of G_h that would be rated lower with respect to the query's relevance metric.

Section VI valuates these protocols' costs, the quality of their results, and their relation to end user satisfaction.

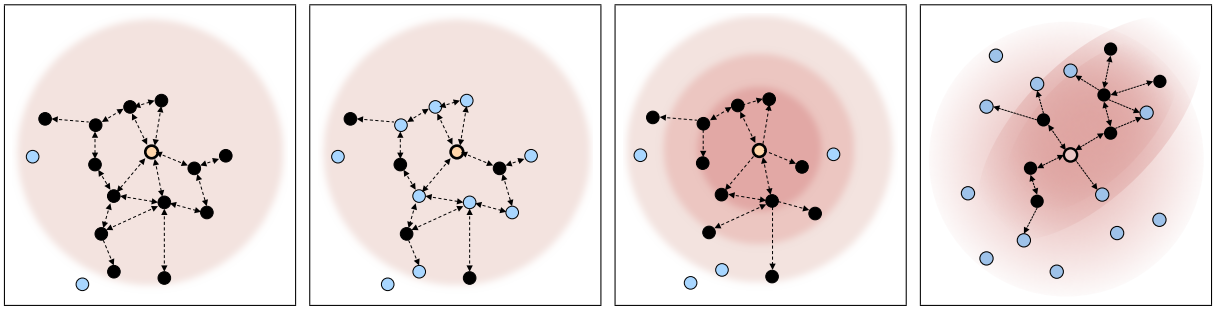
IV. The Gander Search Engine

We have developed the *Gander framework* as a Java library that embodies the Gander conceptual model described in Section III. Our framework makes no assumptions about the underlying network(s) and transport mechanisms responsible for connecting Gander devices as they move through digitally accessible environments. Instead, the framework provides interfaces and abstract implementations as cues for developers to create concrete network-specific implementations where necessary. In this paper, we present a concrete middleware implementation of the Gander framework for mobile computing environments that offer a wired/wireless Internet connection via a wired or wireless Internet connection (e.g., a university campus, corporate office, conference venue) and use this middleware implementation to study the utility of Gander's spatiotemporal sampling in a real-world scenario. To facilitate Gander's query processing amongst proximal devices, our distributed Gander middleware is supplemented by a cloudlet-based *Proximal Discovery Service* (*PDS*) [33], which enables the formation of *virtual* opportunistic networks.

Ultimately, Gander is intended to operate in heterogeneous environments where physically proximal devices (e.g., mobile devices, smart objects, environmental sensors) interact locally using ad hoc network connections. These peer-to-peer connections may be augmented by localized *cloudlets*, which are responsible for maintaining knowledge about the digital resources (e.g., peer devices and datums) available in a specific region of space [1]. Our implementation of the Gander framework relies entirely on Internet-connected environments for two reasons. First, for pragmatic reasons—current off-the-shelf mobile operating systems' support for localized device-to-device interaction is weak, often requiring a dedicated radio interface (e.g., Bluetooth) or administrative device privileges to establish ad hoc networks, which is unreasonable for average users. Second, for evaluation purposes—though not as scalable as purely ad hoc communication, leveraging a centralized resource (i.e., the cloudlet-based *PDS*) to administer *virtual* ad hoc communication enables us to log queries, responses, and the use of the application over time to better measure system performance. From the application's perspective, this middleware-level implementation detail is transparent; the application is unchanged whether the networking capabilities are filled in by this cloudlet-supported infrastructure or by a pure peer-to-peer networking implementation.

A. System Architecture

Fig. 3 shows the Gander system architecture. The *Gander Middleware*, our implementation of the Gander framework, is responsible for the creation and storage of data and the distributed resolution and propagation of queries and their results. An instance of the *Gander Middleware* runs on each node and exposes a minimal REST API to enable other proximal devices to query it. Remotely initiated queries, received by the *Query Server*, and locally created queries are managed by the *Query Processor*, which delegates the query to a dedicated *Query Handler Thread*. Each thread attempts to resolve its designated query using locally-available knowledge about the hosting node's current context (available within the *Context Attribute Graph*) or previously acquired information (stored in the *Tuple Space*). Once a query has been locally resolved, it



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

Fig. 2. Query processing styles and sampling. Dashed lines are sent messages. Darkened nodes respond to a given query. (a) Flooding. Every node in a given range (3 hops) retransmits the query; the target area is the shaded region. (b) Random. A receiving node responds to the query with a given probability; a high quality search evenly samples the shaded space. (c) Probabilistic. Every node that receives the query retransmits it with a given probability; the likelihood of reception drops with distance from the query issuer. (d) Greedy Gossip. Every node that receives the query retransmits it with a probability dependent on the quality of its own local resolution of the query; a high quality local resolution of the query results in a higher probability of its retransmission.

is propagated to other nearby (i.e., reachable) devices hosting their own instances of the Gander Middleware; this forwarding is accomplished via a *Network Handle* provided by the *PDS*. In a purely ad hoc networked environment, detecting proximal devices and acquiring network handles would be handled by the traditional network stack. Later in this section, we discuss how an instance of the Gander Middleware uses the *PDS* to acquire "reachable" devices' *Network Handles* to fulfill a distributed query processing protocol. First, we discuss the implementation of the Gander data model.

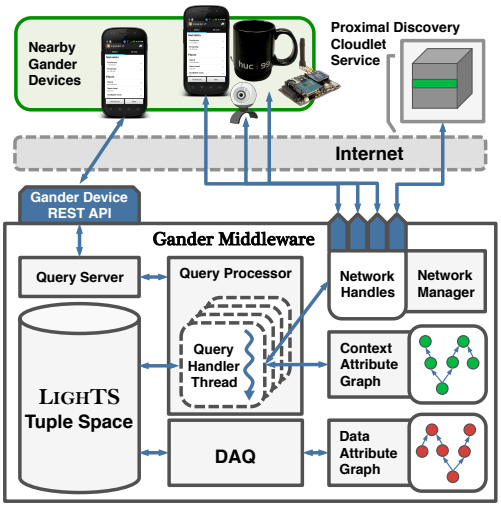


Fig. 3. The Gander system architecture.

B. Data Model

The foundation of the Gander data model implementation is provided by the LIGHTS tuple space framework [34]; this tuple space holds the concrete versions of the datums, i.e., (ν, d) from the conceptual model presented in Section III. LIGHTS not only provides a flexible means of storing both the application data (ν) and the contextual meta-data (d), but also the expressive matching semantics necessary for Gander's query resolution (the function $\mathcal{S}$ in Section III).

A common design task in pervasive computing applications requires composing raw sensor data into higher-level semantic abstractions of context values. For example, the occupancy of a room (a high-level context value) may be inferred from infrared snapshots and noise level readings (raw sensor data). The Gander framework aids in the generation of such structured semantic data by providing mechanisms to configure reusable and composable application-specific data hooks. These mechanisms act as datum marshalers, taking unstructured raw data, potentially from multiple sources, and fusing that data into a structured datum, possibly composing the raw data into higher level information in the process. Essentially, these mechanisms act as datum marshalers; rather than requiring a developer to parcel raw data into datums, she can configure and connect these mechanisms to incoming sensory data streams so that a datum is automatically generated with the appropriate structure and contextual meta-data each time that a new piece of raw data arrives.

Concretely, the Gander framework provides two *semantically-related tuple graphs*, one for representing an application's contextual hooks (the *Context Attribute Graph*) and another for application data hooks (the *Data Attribute Graph*), each potentially capturing different relational semantics. The vertices of each graph are `Attributes`, an extension of a LIGHTS tuple that also specifies how the attribute tuple is composed, potentially using sub-attributes. A directed edge from attribute a_1 to a_2 means that a_1 is a sub-attribute of a_2, or that that a_1's fields may be used to compose a_2's tuple. An attribute representing a room's noise level, for example, may possess multiple sub-attributes representing ambient noise level sensor readings, which may be composed (e.g., aggregated) and mapped to a semantic value (e.g., "quiet" or "low hum"). These constructs are useful for defining reusable contextual and application data building blocks, which may be combined to form larger representations of information (e.g., a floor attribute may comprise its rooms' attributes; a building attribute could be composed from its floors' attributes). It may also be desirable to associate particular contextual information with application data to describe the data's *situation*. Developers may also define inter-*graph* relations to attach contextual attributes to application data attributes, effectively connecting a Gander data value (ν) with its associated meta-data (d).

Fig. 4 illustrates how the Context and Data Attribute Graphs manage the generation of structured *datums*. An attribute's

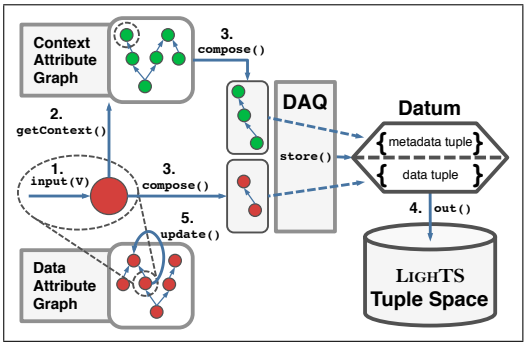


Fig. 4. The Gander data model.

state is modified when the associated underlying sensory data changes, triggering the attribute's `input(V)` method. In a Context Attribute Graph, when an attribute value changes, the graph's local state is simply updated. In a Data Attribute Graph, a change in a data attribute triggers the generation of a new *datum* formed by (*i*) composition actions on the Data Attribute Graph to generate a new value, ν, and (*ii*) the acquisition of the associated meta-data, d, from the Context Attribute Graph, which is retrieved by calling `getContext()` for each of the relations between the Data Attribute Graph and the Context Attribute Graph. The *Data Acquisition Interface* (*DAQ*) generates the final complete datum and stores it in the Gander middleware's local tuple space. This process applies to the update of a single attribute value; since other attributes in the Context or Data Attribute Graph may be dependent on this updated value, their `update()` methods are triggered, and these attributes ultimately generate their own updated datums via the same process.

C. Executing Queries

A `GanderQuery` is an extension of a LIGHTS `BooleanTuple`, which enables pattern matching given arbitrary logical expressions over a tuple's fields. Additionally, a `GanderQuery` is defined by a maximum number k of desired results (datums), a network hop limit *ttl*, the maximum time t a query may be executed in the network, an ordered list of `RelevanceMetrics`, which each implement a datum comparator, and a `QueryProtocol`. Each of the four Gander `QueryProtocols` is defined by its implementation of (*i*) a *sampling filter*, which dictates if and when a participating device should return its local results for the query, and (*ii*) a *forwarding filter*, which determines if, when, and how to propagate the query to other reachable devices. Gander's *probabilistic* protocol, for example, implements a sampling filter that always returns a device's local results, but a forwarding filter that propagates a query only with some probability p.

Gander targets environments where networked objects may form direct localized connections opportunistically. The Gander framework is network-agnostic; the framework is independent of the underlying communication network and protocols. In this work, we opt to implement an Internet (i.e., HTTP) specific middleware to (*i*) deploy the Gander search engine on off-the-shelf mobile devices without requiring the use of a dedicated radio interface (e.g., Bluetooth) or special administrative privileges and (*ii*) to more easily measure system performance using a centralized resource that mediates the establishment of peer-to-peer connections. To accomplish this goal, we extend the cloudlet-based *Proximal Discovery Service* (*PDS*) introduced in [33] to facilitate the discovery of nearby Gander devices and realize *virtual* opportunistic networks. In a nutshell, a *PDS* instance is a web service that acts as a hyper-localized lookup mechanism for proximal peers; given a request for peers parameterized by some distance d, a *PDS* instance will respond with a list of the network handles of peers physically within d from the requesting device. We extend the *PDS* platform to emulate the ad hoc machine-to-machine interaction expected in Gander's target environments.

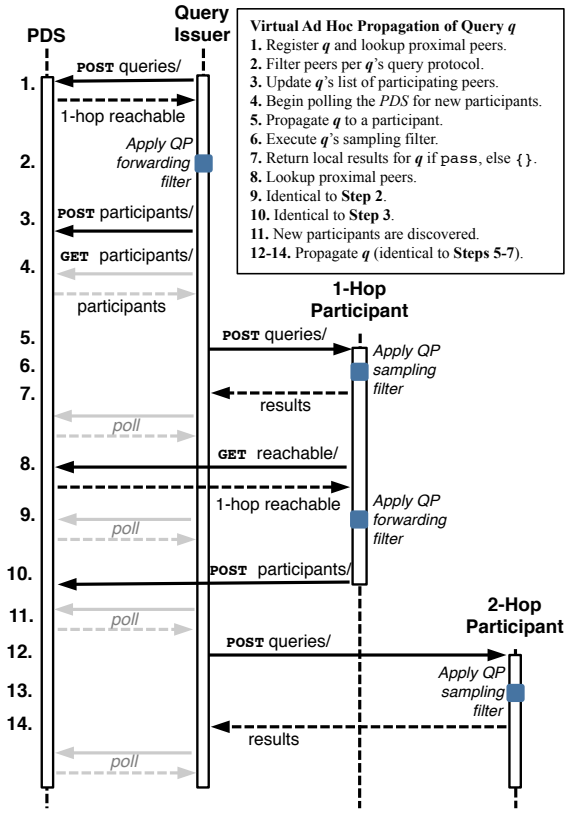


Fig. 5. Issuing and propagating a Gander query.

Fig. 5 illustrates the sequence of events for propagating and resolving a query supplemented by a *PDS* cloudlet instance. A *query issuer* first creates a `GanderQuery` q and retrieves any results available in its local tuple space (via a `rdg(q)`). The query issuer then issues a `POST queries/` to the *PDS*, which responds with a list of network handles for any devices that are directly reachable (via one hop) from the query issuer (Step 1 in Fig. 5). Next (Step 2), the query issuer applies the sampling filter associated with the query's processing protocol to determine which of the proximal peers should receive the query and stores these potential participants at the *PDS* with a `POST participants/` request (Step 3). The query issuer then begins periodically issuing a `GET participants/` request to the *PDS* to check if any new peers have been added to q's participant set (Step 4) and simultaneously issues

a `POST queries/` to each of the filtered reachable peer network handles (Step 5). A receiving device executes q's protocol's sampling filter (Step 6) and responds with up to k local datums matching q (as determined by applying the query resolution function $\mathcal{S}$) and sorted by q's relevance metrics if the sampling filter passes or an empty list if the sampling filter fails (Step 7). Upon receiving participants' results, the query issuer applies q's relevance metrics locally, only retaining the top-k datums across all of the results received from all of the participants. In Steps $8-10$, the query participant device "propagates" q by requesting a list of its own 1-hop reachable peers from the *PDS*, applying q's protocol's forwarding filter to the returned list of device network handles, and storing the filtered reachable devices as part of q's reachable set with a `POST participants/`. Finally (Step 11), the query issuer's periodic `GET participants/` requests initiated in Step 4 begin returning the discovered 2-hop participants as they are identified by the 1-hop participants. Steps $12-14$ are identical to Steps $5-7$. The query q continues propagating in this query issuer-driven fashion until ttl hops have been reached or q's maximum in-network execution time t is reached. The sole purpose of the *PDS* is to enable a device to "discover" directly reachable devices (via one hop) providing their network handles. In our current implementation, the Gander middleware determines the hop distance using a pre-specified parameter; in a deployment that relies only on ad hoc networking, this distance would be determined by RF connectivity. A `GanderQuery` continues propagating until ttl hops have been reached or the maximum in-network execution time t is reached.

V. Case Study: The myGander Mobile Application

Motivated by the desire to evaluate the utility of the Gander search engine within real-world scenarios, we use the Gander system described in Section IV to create *myGander*[3], a mobile application for Android that enables students to search for *live* information about *people*, *places*, and *services* around an engineering building on the UT Austin campus. A *myGander* user can pose queries like, *"Which of my classmates are nearby?"*, *"How long is the queue for coffee?"*, and *"Is there an available seat in the quietest part of the study lounge?"*. We have deployed this application and made it publicly and freely available to members of the UT Austin community. After 15 weeks of uptime, myGander was downloaded a total of 124 times and processed 705 queries issued by 63 devices.

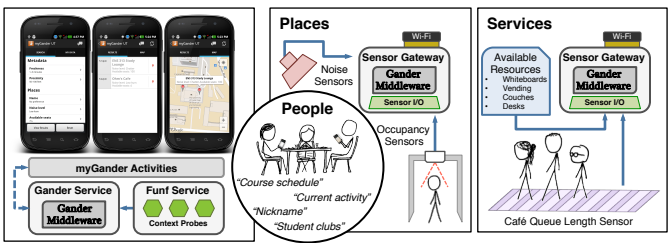


Fig. 6. The myGander mobile application.

[3]http://mpc.ece.utexas.edu/mygander

Fig. 6 shows the core features of *myGander*. The Android application hosts an instance of the Gander Middleware within an Android *service*, which runs in the background even when the application is not in use; this enables the device to participate in responses to other users' queries even with the device's user is not actively querying. The *myGander* search interface resembles a *faceted browser*—a user poses a query by selecting options from a list to place constraints on fields (as opposed to creating a free-form query, as in [13])—enabling a straightforward mapping of search terms and constraints onto the constructs of a `GanderQuery`. As such, this deployment of *myGander* is tailored to the particular application of it (i.e., local search on a university campus); in this sense *myGander* is an application layer on top of the Gander system described previously.

Searching for people. *myGander* users can input information about themselves (e.g., course schedule, current activity, nickname, student club affiliations). This user input generates locally-stored datums created by pre-configured *Data Attribute Graph* hooks. We employ the Funf Open Sensing Framework[4] to attach contextual information (e.g., device location and Wi-Fi fingerprint) to each user-input datum via *Context Attribute Graph* hooks, creating connections between the Context and Data Attribute Graphs as described in Section IV. This data represents the dynamic collection of searchable *people* data.

Searching for places and services. In pervasive computing spaces, data is generated by humans as well as sensors embedded in the environment. In an effort to investigate Gander's performance in digitally accessible environments, we have instrumented two heavily frequented spaces in a university engineering building with various sensors. Specifically, in a coffee shop and a study lounge, we have deployed various sensors that allow us to sense noise level, occupancy, queue length, available resources and to make this sensed information available to Gander queries. Because many of our deployed sensors are severely energy and memory constrained, they must report their sensory data to *Sensor Gateways* (similar to [13]), each of which host an instance of the Gander Middleware and are configured with contextual and application data hooks to aggregate and convert raw sensor readings (e.g. `noise=156`) into structured datums with semantic values (e.g., `noiseLevel=chatter`).

VI. Evaluation

Prior to this work, we have studied the ability of Gander's spatiotemporal sampling methods (i.e., the protocols depicted in Fig. 2) to support search of pervasive computing spaces [2]. This evaluation was done through a realistic simulation involving 20,000 mobile visitors emulated in a day-long visit to an amusement park, in which the visitors issued queries about the wait times for rides using constraints based on distance from the querier and relative to other venues (e.g., other attractions or food vendors). The key highlights of this evaluation were that: (*i*) Gander queries, based on existing query protocol methods, were able to pretty reliably reflect the *ground truth* of the sought information while substantially reducing the

[4]http://code.google.com/p/funf-open-sensing-framework

communication overhead associated with collecting the query results; (*ii*) that, in collecting and presenting query results, the specificity of the relevance metric selected has a significant impact on the quality of the returned results; and that (*iii*) the ability of query protocols to *reflect* on their own behavior and to operate *incrementally* can improve Gander's query performance in terms of the speed with which Gander can return results and the communication overhead associated with collecting those results. This last point motivates a key piece of future work: the development of tailored query processing protocols that can use Gander query constraint and relevance information, as well as information about the data that may be available in the pervasive computing spaces, to *direct* query processing protocol behavior.

In this paper, our evaluation of Gander is in three parts. First, we benchmark the performance of three of the Gander query protocols in a very large scale pervasive computing environment that has a very high density of data and a very high density of mobile devices. Second, we continue the thread of evaluating through simulation, using the *myGander* application and all four query processing protocols to assess how Gander performs when the correlations of the pervasive computing environment's data in space and time are varied. The third thread of evaluation performs an in-depth study on a targeted subset of the users of the *myGander* app deployed on the UT campus.

A. Benchmarking Query Processing

We use *myGander* with an amusement park scenario and real data collected about Disney World's Magic Kingdom, including dynamic wait time information for rides in the park[5] and locations of attractions and amenities. Our scenario includes 30 attractions, 12 restaurants, and 8 restrooms. We populated the park with 1000 users of the *myGander* app (i.e., visitors who store data and participate in query resolution)[6]; users move along park paths following randomly generated routes at an average speed of 0.5 m/s. Simulated users' devices collect and carry timestamped data about attractions and amenities that they have recently been near (i.e., within ∼20m). In a real deployment, this could correspond to a sensing device embedded in the environment pushing data to a user's device or the user making an observation and creating a piece of human-generated data. Collected data has a 15 minute lifetime, after which it is deleted from the device's data space.

We integrated *myGander* and the Gander search engine with the OMNeT++ network simulator [38]. We used OMNeT++'s INET framework [39] for networking and SUMO [40] for node mobility. Each simulated node executes Gander's query processing logic, which interacts directly with modified versions of INET's MANET routing capabilities [41] to implement three of our four sampling approaches (i.e., flooding, random, and probabilistic). The simulated users' devices issue queries, which are distributed across the dynamic topology formed by the users' devices using the protocol implementations. By default, all three protocols use a hop constraint of 9 hops, and the random and probabilistic response and forward probabilities are both set to 0.5. We use queries for a "thrill ride" with a constraint of "with wait time less than 20 minutes." Each query issuer issues one query every minute; we present averages over all of the queries issued in four hours. 20% of the nodes (i.e., 200 users) are designated query issuers; all 1000 nodes serve as data providers and routers.

We do not evaluate the greedy gossip protocol in this first step. Greedy gossip is fueled by relevance: a host receiving a query and possessing more relevant results is more likely to retransmit the query. In this first setting, the world is not very "data rich" (i.e., there is not a lot of data per device); in this situation, the greedy gossip protocol does not shine, but the performance is a product less of the inherent nature of the protocol and more a product of the lack of data. That is, the results for running the greedy gossip protocol are effectively the same as running the probabilistic protocol with a very low probability of retransmission, so including them does not add to the discussion here. In the next set of evaluations, we add in the greedy gossip protocol as we investigate an environment that is much more data rich.

In these first evaluations, we benchmark the query processing styles in terms of query latency and participation (Fig. 7). We report the average minimum latency (the time to receive the first result), the average latency of all received results, the average maximum latency (the time to receive the last result), and the average unique number of nodes that participate in a query[7]. Perhaps unexpectedly, flooding has the lowest latencies of the three styles. Intuitively, one would expect this style to produce the largest average and maximum latencies since it stipulates the sampling of all devices in the sample space. These abnormally low latencies can be attributed to link contention. For a flooding query, all nodes receiving a query immediately send replies, which interferes with the continuing query propagation and causes congestion, necessitating retransmissions. These low flooding latencies reflect a query's inability to propagate more than a few hops in the sample space. This conclusion is corroborated by the relatively low number of query participants for flooding, which one would expect to be higher than both random and probabilistic.

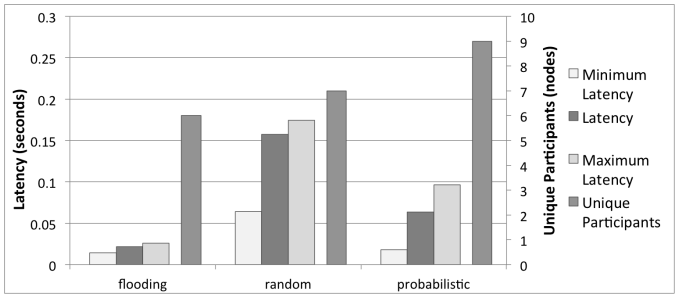


Fig. 7. Query Latency and Participation

Next, we benchmark the sampling styles in terms of query

[5]We use ride wait times published by Lines [37]; we collected wait time data every 60 seconds over full day of the park's operation (i.e., 16 hours).

[6]This is roughly 2% of the park's visitors, based on our collected data.

[7]Unless otherwise stated, we use medians for averages, which helps in identifying trends when there are a few significant outliers

overhead and bandwidth (Fig. 8). We report the average per-query overhead of distributing a query, successfully and unsuccessfully, and of sending the responses, measured in number of messages and bytes. Flooding requires dramatically lower overhead than random and probabilistic (one would expect the opposite), supporting the conclusion that congestion has limited the propagation of the flooded messages.

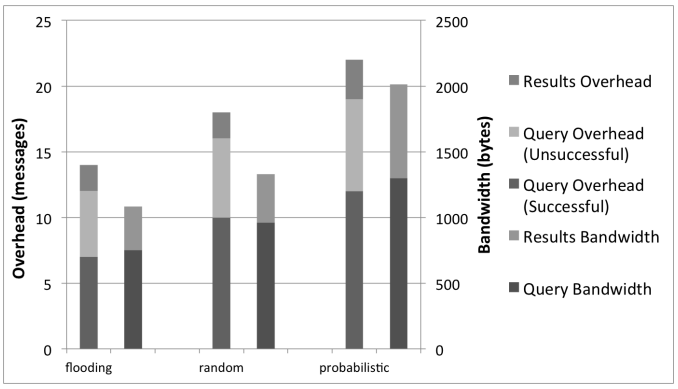


Fig. 8. Query Overhead and Bandwidth

The quality of Gander queries is obviously important. We compute *coverage* of both the data returned and the nodes that responded. We compute the target area as a circle centered at the query issuer with a radius equal to the protocol's hop constraint multiplied by 20m, our effective wireless range. We associate each data item (or responder) with a circular area of radius 20m and define coverage as the percentage of overlap between the areas associated with the data items returned as a result (or the areas associated with the responding nodes) and the target area. We compute *distribution* using an upper quartile distribution uniformity (UQDU) test [42]. This test divides the target area into equal-sized bins, counts the nodes in each bin, and divides the density of the 75th percentile of bins by the expected density for the target area. A uniform distribution results in a value of 1; the further from 1 the UQDU varies, the less uniform a distribution is. Fig. 9 plots the coverage and distribution (i.e., distance from 1) achieved by the five sampling styles; a lower UQDU indicates a more uniform distribution.

The supposed congestion induced by the flooding protocol is further corroborated by the poor coverage it is shown to achieve here. Random and probabilistic achieve nearly identical coverage, which makes sense as they share identical sampling spaces, but the random style samples in a more evenly distributed manner.

B. The Impact of Data Correlations in Space and Time

The relevance of real-world information is inherently parameterized by both space and time. If the temperature is 7°C in Austin, it is likely that it will be very nearly 7°C in Dallas. Further, if the temperature is 7°C now, it will very likely be 7°C in five minutes. In other words, temperature is a highly spatially and temporally correlated phenomenon. Sound, on the other hand, is not; it is rapidly attenuated over short distances and changes on a much shorter time scale. In

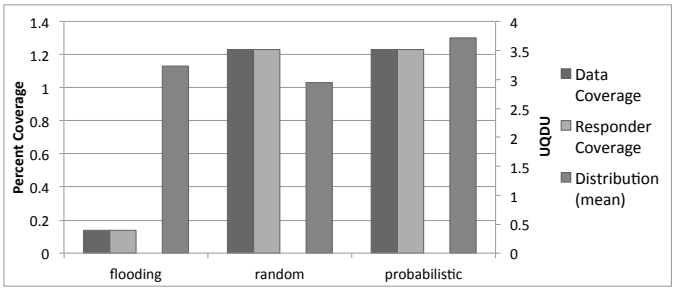


Fig. 9. Query Coverage and Distribution

this thread of our evaluation, we aimed to investigate how the Gander query processing protocols performed when the degree of these data correlations was varied; that is, we ask whether different degrees of correlations impact the *quality* of the results achieved by a Gander search.

To measure the quality, we use *discounted cumulative gain* (DCG) [43], which compares how useful a result is to the user and its ranked position in the result set. We also quantify the *completeness* of a Gander result with respect to the ground truth using the Jaccard coefficient. In both cases, we compare the result of executing a Gander query with the *ground truth*. We compute the DCG for a list of search results as:

$$DCG = \sum_{i=1}^{p} \frac{rel_{d_i}}{\log_2(i+1)},$$

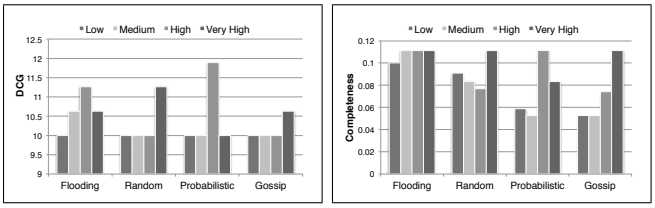
where p is the size of the set of results returned by the Gander query, and rel_{d_i} is an integer that reflects a given result's position in the ground truth's rankings. A query result d returned with ranking i is graded on a scale from 1 (least relevant) to n, where n is the number of items in the set of results in the ground truth. The value of rel_{d_i} is determined by the ranked position j of the item in the set of ground truth results: $rel_{d_i} = n/j$.

The Jaccard coefficient, on the other hand, allows us to measure the *completeness* of the Gander query in comparison to the ground truth. Specifically, the Jaccard coefficient measures the similarity between the Gander query result and the ground truth. We compute completeness as:

$$completeness = \frac{|G_h \cap Q|}{|G_h \cup Q|},$$

where G_h is the ground truth and Q is the Gander query result.

In this evaluation, we assess Gander's performance at a large scale by simulating 400 myGander users in a $650m^2$ space (roughly the size of the UT Austin campus) and varying the degrees to which available data is correlated in space and time. Our simulations employ the software components described in Section IV—each simulated Gander device runs an instance of the *Gander Middleware* and leverages an instance of the *PDS* to discover other nearby peer devices. We use the MobiSim mobility framework [44] to drive simulated myGander users' movement per Levy-walk [45] mobility and populate our simulated environment with synthetically generated data [46], which is "sensed" and locally stored for one minute by simulated users as they move about the environment. For the duration of a simulation (15 minutes) exactly one half (i.e.,



(a) DCG. (b) Completeness.

Fig. 10. The effects of spatial correlations on query protocol performance.

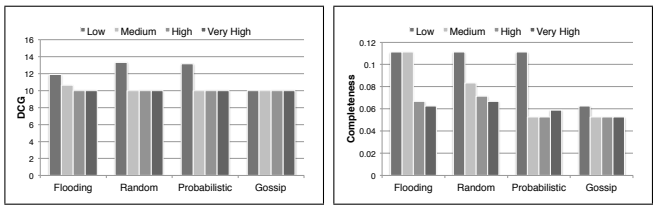
200) of the simulated users' devices issue a query once every 30 seconds for *"Data within 5 units of my* (the query issuer's) *most recently sensed synthetic data value (v)."* A query is processed using one of Gander's four distributed processing protocols (*flooding*, *random*, *probabilistic*, *gossip*) configured using the parameters as above. However, for flooding, random, and probabilistic, we reduce the hop constraint to 5 hops to adjust for the smaller space and fewer users. The greedy gossip's forward probability is set to be $1 - avg\,\%\,error$. Query results are ranked in ascending order of their data values' absolute difference from v, i.e., more similar results are deemed more relevant.

Data Correlation in Space. To draw out the effects of spatial correlations on Gander's performance, we use a synthetic data generation tool [46], varying parameter β, to produce four levels of spatially correlated data—low ($\beta = 0.33$), medium ($\beta = 0.18$), high ($\beta = 0.08$), and very high ($\beta = 0.01$)[8]—and run one simulation per setting (which corresponds to approximately 6000 unique queries). The variations in search results' DCG and completeness are shown in Fig. 10(a,b).

In general, each processing style achieves better DCG as data becomes more spatially correlated (Fig. 10(a)). This is intuitive, since the query is interested in data similar to a recently sensed piece of data. The flooding and probabilistic protocols, however, experience a drop in DCG when synthetic data is *very* highly correlated. This is explained in conjunction with the trends depicted in Fig. 10(b), which shows that when the data's spatial correlation becomes very high, the completeness of Gander queries is unchanged for flooding, but drops for probabilistic. The constant nature of the flooded queries' completeness suggests that flooding gathered an equal amount of results, but they were not necessarily the *most* relevant results. On the other hand, the drop in the completeness of the probabilistic query results reveals that this style actually gathers fewer relevant results when data is highly spatially correlated, indicating that the *most* relevant data existed at the outskirts of query issuers' target spaces where this protocol is less likely to reach. Here, the gossip protocol leverages spatial relationships between datums—as the degree of spatial correlation increases, gossip's greedy heuristic not only finds better data (Fig. 10(a)), but also more of it (Fig. 10(b)).

This knowledge, coupled with application-level knowledge about the data available in the particular type of deployment, can guide the proper selection and tuning of Gander query processing protocols on a per-deployment or even per-query basis. That is, if the Gander system has knowledge about the expected spatial correlation of the data, it can select, even at runtime, the best suited protocol settings to use to process a given query.

Data Correlation in Time. To investigate the impact of temporal correlations on Gander's processing styles, we modulated synthetic data values using a Perlin noise function [47] parameterized by simulation time. Perlin noise is widely used in computer graphics to generate "natural" (i.e., random) looking surfaces and textures,. Thus, simulations can be repeated with the same time-parameterized "random" noise, allowing for comparability. We simulate each of the sampling styles at four levels of data noise: low (approximately a 10% per-minute rate of change), medium (15%), high (20%), and very high (25%). Fig. 11(a,b) shows the results of these experiments.



(a) DCG. (b) Completeness.

Fig. 11. The effects of temporal correlations on query protocol performance.

Both DCG and completeness drop as data becomes more temporally volatile. Recall that the simulated query targets values within a certain range of the value sensed by the query issuer at a particular point and time. As data becomes more short-lived, it becomes increasingly difficult to gather and deliver relevant information before it evolves. Impressively, the random protocol yields the most correct results when data evolves very rapidly (Fig. 11(b)), which are approximately on par with the results for the probabilistic query processing protocol. Recall that the random protocol also achieves a high level of *coverage*, which likely bolsters its correctness; we will see evidence of this in the next study as well.

C. User Study

To assess the user-perceived quality of Gander's spatiotemporal sampling methods, we conducted a user study on the UT Austin campus using the myGander mobile application. Our study involved 88 participants; 29 were compensated with credit to an online store. The compensated group of users, who are responsible for the bulk of our reported results, consisted of 5 females and 24 males; 17 undergraduate and 12 graduate students. We report results from two weeks of use.

We educated our group of compensated users on the type of information available in *myGander* and then encouraged them to integrate the app into their activities on campus. When created, each *myGander* query was assigned a query protocol chosen uniformly at random[9]. To assess the user-perceived quality of a Gander query processed with a particular

[8]The settings for different degrees of data correlation were chosen based on guidance from the tool paper.

[9]We do not report user study results for the gossip protocol since our scenario did not elicit dense enough data for the gossip protocol's greedy heuristic to leverage spatiotemporal correlations.

protocol, users were prompted with an in-application pop-up upon performing a search and receiving results[10]. Specifically, the user was presented with the Gander results for his query alongside the ground truth results in a separate tab and asked to rate the Gander results in terms of three user experience (UX) metrics: (*i*) their *utility*, (*ii*) *confidence* in the Gander results, and (*iii*) overall *satisfaction* with the Gander results using the following prompts:

(*i*) How relevant, useful, and of interest are these results?
(*ii*) How confident are you in these results' accuracy?
(*iii*) Overall, how satisfied are you with these results?

Rating each metric constituted labeling it as: `significant`, `some`, `none`, `don't-know`, `service-error`. Users were also encouraged to provide an explanation of their ratings in a comment field. During two weeks of use, 404 myGander queries were issued, 42 of which were rated by the user (2 by non-compensated users). With respect to the manner of query processing, of the rated queries, 16 were issued with the flooding protocol, 14 with the random protocol, and 12 with probabilistic.

To compute the overall user-perceived performance of Gander with respect to our UX metrics M, where M is the set {*utility, confidence, satisfaction*}, we adapt an averaging function introduced in [48] originally used to compute the expected utility of Web search result social annotations. Let U be the set of all user-rated queries and U_p be a subset of queries $u \in U$ issued with query protocol $p \in P$ (P is the set {`flooding, random, probabilistic`}). We define $R_m(U_p)$ as the average rating of each query in U issued with query protocol p and rated per metric $m \in M$ as:

$$R_m(U_p) = \frac{\sum_{u \in U_p} \omega_m(u)}{|U_p|}$$

where $\omega : \mathbb{J} \rightarrow \mathbb{R}$ maps ratings $\mathbb{J}$ to $[0, 1]$. We employ the same ω variants introduced in [48], listed in Table I. $R_{m,Rel}(U_p)$ is a *relevance*[11] function that assigns a *graded* score to each rating for metric m. $R_{m,Prec}(U_p)$ uses a *binary* scale and is a measure of *precision* for m. For our evaluation we omit the 21.4% of `don't-know` and `service-error` ratings.

TABLE I
METRICS OF UX RATINGS.

$R(U_p)$	ω **function definition**		
$R_{m,Rel}(U_p)$	$\omega_{m,Rel} =$	`significant` :	1
		`some` :	0.5
		`none` :	0
$R_{m,Prec}(U_p)$	$\omega_{m,Prec} =$	`significant` :	1
		`some` :	1
		`none` :	0

Fig. 12 shows the overall graded relevance ($\omega_{m,Rel}(U_p)$) and binary precision ($\omega_{m,Prec}(U_p)$) scored UX metrics per query processing protocol. Overall, our users labeled their Gander search results with a *utility* relevance of 0.548, a *confidence* relevance of 0.536, and a *satisfaction* relevance of 0.476, indicating that they found their results slightly useful and were marginally confident in them, but not entirely satisfied. However, each individual query protocol influences *utility*, *confidence*, and *satisfaction* in very different ways.

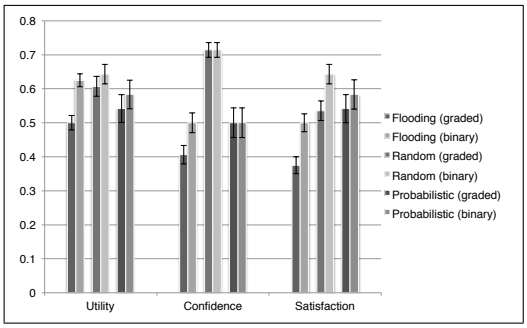


Fig. 12. myGander user experience metrics per query protocol.

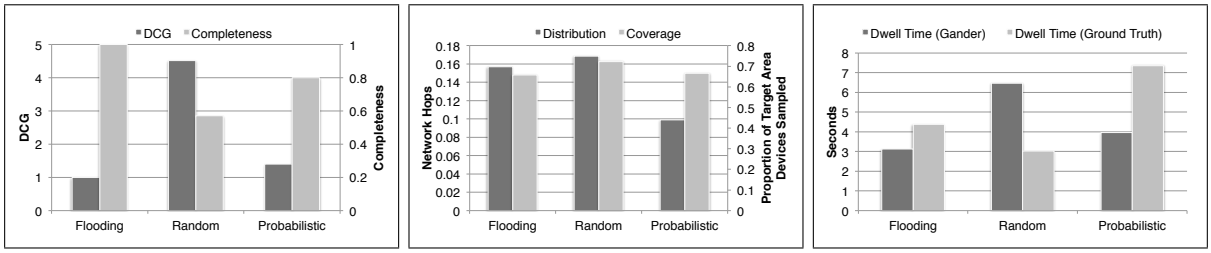
Fig. 13 reveals system level performance metrics elicited in our user study. We use the same DCG and completeness metrics defined in the previous section. We also measure a Gander query's *coverage* as the proportion of target area devices that the query reached and *distribution* as the number of network hops from the query issuer the query traveled. Because our deployment network is relatively sparse, many queries travel very low numbers of hops (often 0). Finally, the *dwell time* is a measure of the amount of time the user spends observing a set of results.

Fig. 12 shows that users expressed significantly more confidence in results gathered using the `random` protocol. Due to the relatively small scale of our university campus deployment, none of the processing styles varied significantly in terms of network-imposed overhead; the larger scale performance evaluation in [49] provides a more detailed study of performance metrics at scale. However, even in this small scale study, for approximately the same cost, the `random` protocol achieved the highest *distribution* and *coverage* (Fig. 13(b)), yielding results with the greatest DCG (Fig. 13(a)). This higher DCG is a reflection of the greater number of results gathered by the `random` style. Interestingly, `random` queries produced the least *complete* results, but users spent longer looking at these search results on average (Fig. 13(c)). In other words, though `random` did not acquire the *best* results, users were more confident (and marginally more satisfied) being given *more* results.

Another trend evident in Fig. 12 is that the `flooding` protocol consistently elicits the greatest difference between the graded ($\omega_{m,Rel}(Q_p)$) and binary ($\omega_{m,Precl}(Q_p)$) averaging metrics, indicating that users were more "lukewarm" about `flooding` search results in comparison to results from the `random` or `probabilistic` protocols. This same disparity is mirrored in the `flooding` search results' measured DCG and completeness (Fig. 13(a))—`flooding` achieved the most complete results with respect to the ground truth, but not the greatest number of results, further indicating that users' opinions become more ossified when presented with *more* results, but not necessarily *better* results.

[10]A user is prompted for ratings only when ground truth is available.

[11]Note that the use of the term "relevance" here is different from the *relevance metric* defined for a Gander query.



(a) Average result DCG and completeness. (b) Average query distribution and coverage. (c) Average myGander result set dwell time.

Fig. 13. myGander query and result set system metrics.

VII. Lessons Learned

A significant amount of effort went into building, deploying, and servicing the case study application described in Section V and used in Section VI. This undertaking involved creating and deploying a non-trivial array of sensors, creating a usable application, and recruiting and maintaining a user base. We encountered (and survived) many of the well-known pitfalls of such a deployment, including live bug resolution, maintaining user attention (e.g., through gift card competitions), etc. In addition to these, our study comes with several additional lessons more specific to this style of pervasive search in the IoT.

It is well known that having a critical mass of users is important, if only for generating "buzz" about the application or service being evaluated. In the specific context of the Gander search engine, we found this to be even more important, since the users themselves are the generators of much of the data (e.g., "who is around me, now" is an important aspect to many of the Gander queries). By using a controlled space, we aided the Gander search engine with some pre-installed sensors to measure things like the occupancy of the study lounge instead of relying on users' devices for this information. Also on the user side, we discovered that asking a user about his perception of the quality of a Gander search result required communicating the "ground truth" to the user for comparison to the Gander query result. To achieve this, we had to develop a back-end monitor that was omniscient with respect to all of the data in a Gander deployment. This is counter to the general Gander philosophy that users are in control of their own data and data is not stored centrally; however, it was necessary for a user-centered evaluation.

On a related note, we did address users' desires to control their own data and not release potentially private information to be stored centrally. This results in the need for local data storage at each device. We addressed this through a combination of the tuple space abstraction (which has been widely used in previous work) and our expressive graph based data structures, which enable drawing relationships between datums even when those datums are not co-located.

Finally, from the inception of Gander, we have advocated a device-to-device paradigm for query resolution. However, current device capabilities largely prevent these direct device-to-device connections (often for very fundamental and sound security reasons), so the Gander architecture is designed to work around these limitations by utilizing the cloudlet-based proximal discovery service. The use of the cloudlet-based discovery service also enabled a significantly more expressive evaluation, resulting in the strong conclusions we are able to draw about the potential for the Gander search engine to satisfy users' queries of the here and now using data collected from the here and now. We maintained a strict separation of concerns between the query resolution protocol and the query handling mechanisms of the Gander search engine; given this design, we expect that switching between cloudlet-based query resolution and device-to-device query resolution will be simple to support within Gander, particularly given the increasing interest in emerging technologies (e.g., Bluetooth Low Energy and WiFi Direct) that enable direct device-to-device connections.

Acknowledgment

The authors would like to thank Gruia-Catalin Roman for all of his support, guidance, and collaboration on the Gander project. This work was funded, in part, by the National Science Foundation (NSF), Grant #CNS-0844850 and a Google Research Award. The views and conclusions herein are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

References

[1] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE J. of Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[2] J. Michel, C. Julien, J. Payton, and G.-C. Roman, "Gander: Personalizing search of the here and now," in *Proc. of MobiQuitous*, 2011.

[3] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "The design of an acquisitional query processor for sensor networks," in *ACM SIGMOD*, 2003, pp. 491–502.

[4] R. Newton and M. Welsh, "Region streams: functional macroprogramming for sensor networks," in *Proc. of DMSN*, 2004, pp. 78–87.

[5] L. Mottola and G. Picco, "Programming wireless sensor networks with logical neighborhoods," in *Proc. of InterSense*, 2006.

[6] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Net.*, vol. 54, no. 15, pp. 2787–2805, 2010.

[7] D. Guinard and T. Vlad, "Towards the web of things: web mashups for embedded devices," in *Proc. of WWW*, 2009.

[8] K. Romer, B. Ostermaier, F. Mattern, M. Fahrmair, and W. Kellerer, "Real-time search for real-world entities: A survey," *Proc. of the IEEE*, vol. 98, no. 11, pp. 1887–1902, 2010.

[9] C. Tan, B. Sheng, H. Wang, and Q. Li, "Microsearch: When search engines meet small devices," in *Per. Comp.*, ser. LNCS. Springer Berlin / Heidelberg, 2008, vol. 5013, pp. 93–110.

[10] H. Wang, C. C. Tan, and Q. Li, "Snoogle: A search engine for pervasive environments," *IEEE J. on Parallel and Dist. Sys.*, vol. 21, no. 8, pp. 1188–1202, 2010.

[11] K.-K.Yap, V. Srinivasan, and M. Motani, "Max: Human-centric search of the physical world," in *Proc. of SenSys*, 2005, pp. 166–179.

[12] A. Kansal, S. Nath, J. Liu, and F. Zhao, "Senseweb: An infrastructure for shared sensing," *IEEE J. of MultiMedia*, vol. 14, no. 4, pp. 8–13, 2007.

[13] B. Ostermaier, K. Römer, F. Mattern, M. Fahrmair, and W. Kellerer, "A real-time search engine for the web of things," in *Proc. of IOT*, 2010.

[14] Z. Ding, X. Gao, L. Guo, and Q. Yang, "A hybrid search engine framework for the internet of things based on spatial-temporal, value-based, and keyword-based conditions," in *Proc. of GreenCom*, 2012, pp. 17–25.

[15] C. Perera, A. Zaslavsky, P. Christen, M. Compton, and D. Georgakopoulos, "Context-aware sensor search, selection and ranking model for internet of things middleware," in *Proc. MDM*, 2013.

[16] A. L. Murphy, G. P. Picco, and G.-C. Roman, "LIME: A middleware for physical and logical mobility," in *Proc. of ICDCS*, 2001, pp. 524–533.

[17] I. Brunkhorst, H. Dhraief, A. Kemper, W. Nejdl, and C. Wiesner, "Distributed queries and query optimization in schema-based p2p-systems," in *Databases, Inf. Sys., and P2P Comp.*, ser. LNCS. Springer Berlin / Heidelberg, 2004, vol. 2944, pp. 184–199.

[18] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. of SIGCOMM*, 2001.

[19] G. P. Picco, A. L. Murphy, and G.-C. Roman, "On global virtual data structures," *Proc. Coord. and Ubiq. Comp.*, pp. 11–29, 2002.

[20] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," UC Berkeley, Berkeley, CA, USA, Tech. Rep., 2001.

[21] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proc. of Middleware*, 2001.

[22] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. of SIGCOMM*, 2001.

[23] A. Ziotopoulos and G. de Veciana, "P2P network for storage and query of a spatio-temporal flow of events," in *Proc. of PerCom Workshops*, 2011.

[24] M. Mamei, F. Zambonelli, and L. Leonardi, "Tuples on the air: a middleware for context-aware computing in dynamic networks," in *Proc. of ICDCS*, 2003, pp. 342–347.

[25] P. Costa, C. Mascolo, M. Musolesi, and G. Picco, "Socially-aware routing for publish-subscribe in delay-tolerant mobile ad hoc networks," *IEEE J. on Selected Areas in Comm.*, vol. 26, no. 5, 2008.

[26] L. Fiege, F. Gartner, O. Kasten, and A. Zeidler, "Supporting mobility in content-based publish/subscribe middleware," in *Middleware*, ser. LNCS. Springer Berlin / Heidelberg, 2003, vol. 2672, pp. 103–122.

[27] R. Boyer and W. Griswold, "Fulcrum - an open-implementation approach to internet-scale context-aware publish / subscribe," *Hawaii International Conference on System Sciences*, vol. 9, p. 275a, 2005.

[28] G. Sollazzo, M. Musolesi, and G. Mascolo, "Taco-dtn: a time-aware content-based dissemination system for delay tolerant networks," in *Proc. of MobiOpp*, 2007, pp. 83–90.

[29] E. Nordström, P. Gunningberg, , and C. Rohner, "A search-based network architecture for mobile devices," Uppsala University, Tech. Rep. 2009-003, January 2009.

[30] G. Cugola, A. A. Margara, and M. Migliavacca, "Context-aware publish-subscribe: Model, implementation, and evaluation," in *Proc. of ISCC*, 2009.

[31] D. Frey and G. Roman, "Context-aware publish subscribe in mobile ad hoc networks," in *Coord. Models and Lang.*, ser. LNCS. Springer Berlin / Heidelberg, 2007, vol. 4467, pp. 37–55.

[32] V. Rajamani, C. Julien, J. Payton, and G.-C. Roman, "Inquiry and introspection for non-deterministic queries in mobile networks," in *FASE*, 2009, pp. 401–416.

[33] J. Michel and C. Julien, "A cloudlet-based proximal discovery service for machine-to-machine applications," in *Proc. of MobiCASE*, 2013, to appear.

[34] D. Balzarotti, P. Costa, and G. P. Picco, "The lights tuple space framework and its customization for context-aware applications," *J. on Web Intelligence and Agent Sys.*, vol. 5, no. 2, pp. 215–231, 2007.

[35] V. Rajamani and C. Julien, "Adaptive data quality for persistent queries in sensor networks," in *Proc. of QShine*, 2009.

[36] X. Dai, M. L. Yiu, N. Mamoulis, Y. Tao, and M. Vaitis, "Probabilistic spatial queries on existentially uncertain data," in *Proc. of SSTD*, 2005.

[37] "Disney World Lines App (Touring Plans)," http://www.touringplans.com/walt-disney-world-lines.

[38] A. Vargas, "OMNeT++ Web Page," http://www.omnetpp.org, 2008.

[39] "The INET Framework for OMNeT++," http://inet.omnetpp.org/, 2012.

[40] D. Krajzewicz, G. Hertkorn, C. Rössel, and P. Wagner, "SUMO (simulation of urban mobility): An open-source traffic simulation," in *MESM*, 2002.

[41] "The INETMANET Framework for OMNeT++," https://github.com/inetmanet/inetmanet/wiki, 2012.

[42] D. Kieffer and T. O'Connor, "Managing soil moisture on golf greens using a portable wave reflectometer," in *Int. Irrig. Show*, December 2007.

[43] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of IR techniques," *ACM Trans. on Info. Sys.*, vol. 20, pp. 422–446, October 2002.

[44] S. M. Mousavi, H. R. Rabiee, M. Moshref, and A. Dabirmoghaddam, "Mobisim: A framework for simulation of mobility models in mobile ad-hoc networks," in *WiMob*, 2007.

[45] I. Rhee, M. Shin, S. Hong, K. Lee, S. J. Kim, and S. Chong, "On the levy-walk nature of human mobility," *IEEE Trans. Netw.*, vol. 19, no. 3, pp. 630–643, jun 2011.

[46] A. Jindal and K. Psounis, "Modeling spatially correlated data in sensor networks," *ACM Trans. Sen. Netw.*, vol. 2, no. 4, pp. 466–499, nov 2006.

[47] D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, and S. Worley, *Texturing & Modeling A Procedural Approach*. AP Professional, 1998.

[48] P. Pantel, M. Gamon, O. Alonso, and K. Haas, "Social annotations: utility and prediction modeling," in *Proc. of SIGIR*, 2012, pp. 285–294.

[49] J. Michel, C. Julien, J. Payton, and G.-C. Roman, "The gander search engine for personalized networked spaces," The University of Texas at Austin, Tech. Rep. TR-ARiSE-2012-009, November 2012.