The Dissertation Committee for Jonas Reinhardt Michel
certifies that this is the approved version of the following dissertation:

Supporting Device-to-Device Search and Sharing of Hyper-Localized Data

Committee:

Christine Julien, Supervisor

Vijay Garg

Simon Lam

Gustavo de Veciana

Sriram Vishwanath

Supporting Device-to-Device Search and Sharing of Hyper-Localized Data

by

Jonas Reinhardt Michel, B.S.E.E.; M.S.E.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2015

For Amanda

Acknowledgments

An undertaking that requires as much time, effort, and perseverance as a PhD cannot be accomplished without a significant amount of help, support, and guidance from a great many people, to all of whom I am indebted. I would especially like to thank my advisor, Dr. Christine Julien, whose enthusiasm and discipline are an inspiration. Without a doubt, she is the best advisor a graduate student can have. Very patiently, Christine guided me along the path to becoming an independent researcher giving me freedom and encouragement at every step to explore new directions. Her suggestions and feedback were always insightful and I never felt that she was trying to make my life difficult, though I suspect the converse is not true. I absolutely must thank her for being such a tenacious grant writer. I have traveled to many places thanks to the financial support garnered from those grants and her gracious letters of recommendation. I would also like to thank my dissertation committee, especially Dr. Vijay Garg—his Distributed Systems course was quite possibly the most challenging and most rewarding course I have ever taken. I refer to Vijay's textbook nearly every week. Over the course of my graduate student career I was fortunate enough to collaborate with and learn from some very bright individuals. Dr. Jamie Payton and Dr. Catalin Roman in particular contributed significant direction during the formative years of my doctoral research. Our weekly Skype calls were invaluable and helped me find my voice as a researcher in the field of pervasive computing.

Graduate school is full of chutes and ladders. Fortunately I had the luck to go through it alongside some savvy peers whose candid advice, both solicited and unsolicited, helped me learn the ropes and maintain perspective. Tony Petz and Vidur Bhargava were the first two graduate students that engaged me before I began taking courses—they volunteered (or perhaps *were* volunteered) to take me out to lunch one scalding June day. Over the next years, many crucial experiments would not have been possible without Tony's technical guidance. And my countless whiteboard brainstorming sessions with Vidur have been the source of some of my most enlightening thoughts. Sharing lab time with Seth Holloway and Evan Grim was always a treat. They were each quick to provide much needed laughs in dire straits and words of wisdom about everything from debugging multi-machine computing jobs to writing conference papers to the best pubs near campus.

It goes without saying that I would not have survived the PhD without the support of my family and close friends. My parents have each been my greatest source of encouragement, and my years in this program were no exception. My wife, and often project manager, Amanda stuck by my side for almost every hour. She was there to celebrate with me during the highs, console me during the lows, and keep me on task, well, always. There are, of course, certain advantages to being married to a professional statistician. I am extremely grateful for Amanda's many hours of pro bono "consulting" and brainstorming about measurement approaches. Amanda's parents, Dan and Pam, have likewise been a great source of encouragement along the whole PhD roller coaster ride. They have become my family away from my family and I am very grateful for their enduring support.

I would also like to thank RoseAnna Goewey and Melanie Gulick for being such outstanding administrative staff members. Melanie was always quick to answer questions and supply forms, instructions, and need-to-know information about daunting administrative responsibilities; RoseAnna ensured these responsibilities were the least of my concerns. RoseAnna in particular provided lots of assistance pointing me to resources and filling out paperwork before and after international travel.

Finally, I would like to thank a few professors from my undergraduate years who in large paved my way to graduate school and the PhD. Dr. Shwetak Patel and Dr. Juris Vagners were gracious enough to welcome me into their research groups and to assign me interesting, challenging, and meaningful research projects. Dr. James Peckol taught me what it means to be an engineer and the blessings (and curses) of design freedom. My many conversations with each of them helped shaped my direction and ultimately my decision to pursue graduate studies.

I am deeply appreciative of everyone that helped me along the way, I couldn't have done it without you.

Supporting Device-to-Device Search and Sharing of Hyper-Localized Data

Publication No. _____

Jonas Reinhardt Michel, Ph.D.
The University of Texas at Austin, 2015

Supervisor: Christine Julien

Supporting emerging mobile applications in densely populated environments requires connecting mobile users and their devices with the surrounding digital landscape. Specifically, the volume of digitally-available data in such computing spaces presents an imminent need for expressive mechanisms that enable humans and applications to share and search for relevant information within their digitally accessible physical surroundings. Device-to-device communications will play a critical role in facilitating transparent access to proximate digital resources. A wide variety of approaches exist that support device-to-device dissemination and query-driven data access. Very few, however, capitalize on the *contextual history* of the shared data itself to distribute additional data or to guide queries. This dissertation presents Gander, an application substrate and mobile middleware designed to ease the burden associated with creating applications that require support for sharing and searching of hyper-localized data *in situ*. Gander employs a novel trajectory-driven model of

spatiotemporal provenance that enriches shared data with its contextual history—annotations that capture data's geospatial and causal history across a lifetime of device-to-device propagation. We demonstrate the value of spatiotemporal data provenance as both a tool for improving ad hoc routing performance and for driving complex application behavior. This dissertation discusses the design and implementation of Gander's middleware model, which abstracts away tedious implementation details by enabling developers to write high-level rules that govern when, where, and how data is distributed and to execute expressive queries across proximate digital resources. We evaluate Gander within several simulated large-scale environments and one real-world deployment on the UT Austin campus. The goal of this research is to provide formal constructs realized within a software framework that ease the software engineering challenges encountered during the design and deployment of several applications in emerging mobile environments.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In pervasive computing spaces, people and devices are integrated with the surrounding physical environment; wireless network 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 has the potential to generate large volumes of spatiotemporal data that can be exploited by pervasive computing applications of the future. In these settings, users and applications are interested in what is happening around them, right here and right now.

Consider the following motivating examples. At an outdoor music festival with thousands of attendees, festival-goers may use a pervasive computing application to find nearby mobile vendors carrying particular food items, be alerted when friends are near, discover popular photos of recent performances, receive coupons from merchandise vendors, and be alerted about special events and limited product offerings at festival sponsors' tents. Vendors at the festival may likewise use the application to disseminate special offerings and advertisements to attract customers. Attendees at a large business convention may use a pervasive computing application to connect with nearby colleagues and organization representatives or receive

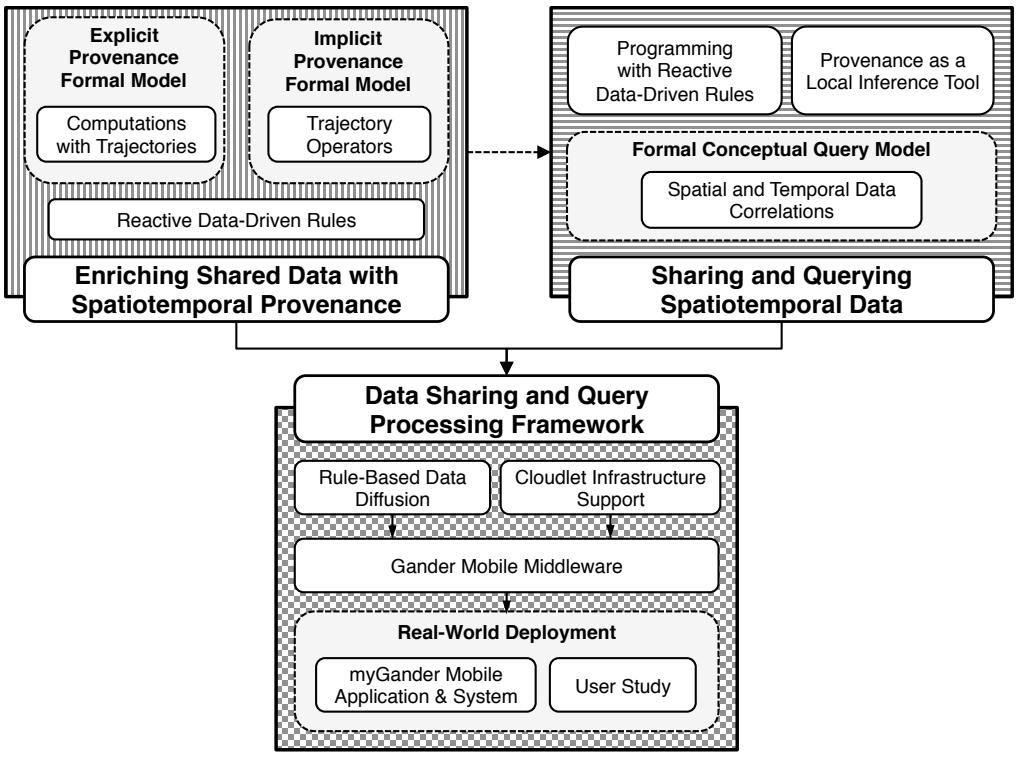


Figure 1.1: Graphical overview of the research contributions. This figure will be used as a pictorial guide throughout this dissertation.

alerts about spontaneous events. A parade goer may be aided by an application that recommends the closest available viewing spot in a shady area, helps users to avoid footpaths that are congested, or finds live streaming video clips of a particular parade attraction that are being captured by a nearby spectator.

To realize these interactions, it is essential to help users and applications *share*, *reason about*, and *search* for *hyper-localized* data as they move through a densely populated and rapidly changing information space. For many reasons, it is becoming apparent that device-to-device interactions will play a pivotal role in connecting users' devices with this hyper-localized information: the demand for fixed network infrastructure resources (e.g., cellular upload bandwidth) could greatly ex-

ceed available capacity; round trip times to the "cloud" may be too slow [152]; and the battery constrained nature of users' devices may make short-range network interfaces far more advantageous than long-range ones in terms of energy/bandwidth consumption [157]. This dissertation addresses formal and practical challenges associated with sharing, reasoning about, and querying hyper-localized data via device-to-device interactions.

The goal of this dissertation is *Gander*—a pervasive computing application substrate with three dovetailing components (depicted in Figure 1.1): (*i*) expressive and flexible models of spatiotemporally-enriched data (the vertical-striped box in the upper left corner of Figure 1.1); (*ii*) distributed query and data dissemination constructs that support a search engine for pervasive computing spaces (the horizontal-striped box in the upper right); and (*iii*) the systems contribution of an application development framework that combines the two (the checkered box on the bottom). Each objective presents unique challenges in pervasive computing spaces. Next we enumerate some salient characteristics associated with these environments.

1.1 Characteristics of Pervasive Computing Spaces

The characteristics of the data in pervasive computing spaces present a set of interesting challenges:

- *Overwhelming Ratio of Data Available to Data Used*: While the data in these spaces is generated by humans and sensors at a unprecedented scale, only a small fraction of it will be exploited by pervasive computing applications and their users.

- *Ephemeral Data*: The information generated by people and sensors is spatiotemporally relevant and as such may be useful only for a short period of time; as time passes, devices and users move, physical phenomena evolve, and social interactions shift, and a datum may no longer reflect the here and now. Existing systems for acquiring localized information enable access to relatively static data instead of the inherently ephemeral data of pervasive computing spaces, which cannot be easily indexed outside of the here and now (e.g., in the Internet).
- *Data Sensitivity*: In addition, much of the data generated in these spaces captures information about people and their movements, interactions, and intentions, which raises concerns about privacy. These concerns impact how people are willing to share their data; studies have shown that people may be willing to share with other co-located users (even unknown ones) what they would not share publicly (i.e., on the Internet) [76, 109].
- *Heterogeneous Networks*: Finally, people have differing degrees of access to fixed network infrastructure over time, even in developed areas. While many commuters in large metropolitan areas carry mobile phones, for example, it is often difficult to connect to the Internet via the cellular network while riding the subway. Supporting sharing and search of information that represents the here and now requires a new perspective that takes advantage of both opportunistic interactions between peer pervasive computing devices and the emerging availability of localized infrastructure in the form of cloudlets [152].

Spaces exhibiting these types of characteristics are increasingly common place, given the widespread embedding of computational, sensing, and communicating technologies in our physical environments. Pervasive computing spaces can and will generate vast amounts of spatiotemporally relevant data. As in the Internet, large volumes of data motivate the need for expressive and efficient search mechanisms to provide access to information relevant to a user and his needs. Clearly, such mechanisms must be capable of leveraging the contextual dependencies of data. On the other hand, to enable complex application-level reasoning, data must be able to express its relationship to the context in which it was created. In this dissertation, we will look at both sides of this two-sided coin. Crucial to this goal is a general-purpose data model with spatiotemporal underpinnings aimed at capturing the ever-changing state of pervasive computing environments.

1.2 Enriching Shared Data with Spatiotemporal Provenance

Users and applications in pervasive computing environments rely on their perceptions of the surrounding world to make decisions and adapt their behavior. A key challenge lies in capturing and managing information that is subject to the high levels of dynamics that characterize pervasive computing environments: time passes, devices and users are constantly in motion, social patterns evolve, data is moved, and information expires. At the same time, the data used by pervasive computing applications is inherently *spatiotemporal*, i.e., its *relevance* to a particular user or application is parameterized by both space and time. There are numerous mechanisms designed to support the nuts and bolts of distributing messages in a device-to-device

fashion, even over multiple hops, and even some of these mechanisms are "content aware" (i.e., they distribute the data based on its own semantics). However, few approaches tap into the *contextual history* of the shared data to distribute additional information. In this dissertation, we propose an approach whose key tenet is that knowing the context in which a piece of data was created and the contexts in which it has been shared over its lifetime can help determine the future contexts in which that data might be relevant.

We present two variants of a model that both enrich shared data with its own *spatiotemporal provenance*—trajectory-based annotations that capture data objects' contextual history over a lifetime of device-to-device propagation. The spatiotemporal trajectory data type [44] uses trips and modes of travel to represent the spatiotemporal path of a data item. Our approach is similarly motivated, but targets a completely distributed approach, where the trajectory itself is defined by properties of the space and time in which data exists and is shared. The first model variant (shown as the box labeled *Explicit Provenance Formal Model* within the vertical-striped box in Figure 1.1) employs an *explicit* coordinate system and annotates data objects with timestamped spatial points that capture objects' trajectories (within the coordinate system) over time. We further extend this model to allow the explicit spatiotemporal trajectories to be aggregated, fused, and queried on-the-fly. The second variant (the box labeled *Implicit Provenance Formal Model*) considers an *implicit* form of space that captures the causal propagation of data objects through a time-varying network of mobile nodes. We describe spatiotemporal operators for comparing implicit trajectories and drawing local inferences about the distributed

state of knowledge.

Associating meta-information with data objects regarding the objects' use and evolution is sometimes referred to as *data provenance* [133], which has received attention in both sensor networks [7] and cloud computing applications [111]; our trajectory-based spatiotemporal annotations are a direct form of data provenance. To our knowledge, this dissertation contains the first investigation into the design and use of *spatiotemporal* data provenance for mobile and pervasive computing applications.

Pervasive computing applications inherently deal with data about real-world phenomena. Accordingly, a significant portion of the development effort in creating such applications typically deals with implementing data-driven behavior: for example, creating structured digital data items from sensors, moving data items between nodes, raising alerts when conditions are met, combining different types of data to form higher level aggregate views, and eventually deleting stored information when it is no longer needed. Rather than embedding such data-dependent logic throughout an application, we describe how this behavior may be expressed as reactive rules driven by data and its spatiotemporal provenance (the *Reactive Data-Driven Rules* shown in Figure 1.1). These rules may even be attached to and transmitted alongside shared data, which enables data to completely dictate when and how it should be replicated, moved, and eventually deleted in a distributed environment. In this dissertation, we show how such data-driven rules may be used as functional building blocks to express and implement distributed data dissemination and query protocols.

1.3 Sharing and Querying Spatiotemporal Data

In emerging pervasive computing spaces, users and applications require transparent access to digital data that exist directly within the proximate surroundings. Data is disseminated and queried by leveraging proximally available resources in the user's or application's immediate physical environment, via a localized cloudlet infrastructure or dynamically formed wireless ad hoc networks. Given the spatiotemporal nature of data in pervasive computing spaces, data's spatiotemporal provenance is exceptionally valuable for both driving data-dependent application behavior and for informing ad hoc routing protocols.

This dissertation demonstrates the utility of historical contextual data annotations from two perspectives. First, from a software engineering perspective we show how data-dependent application logic may be expressed as reactive rules (the box labeled *Programming with Reactive Data-Driven Rules* within the horizontal-striped box in Figure 1.1). These rules encapsulate logic that may otherwise be embedded throughout the application, greatly simplifying application design. We demonstrate how such rules may be parameterized by characteristics of spatiotemporal annotations and benchmark the overhead of our explicit provenance metadata in a simulated pervasive computing environment. Second, we focus on a particular task that is essential in pervasive computing applications: device-to-device routing.

A wide spectrum of approaches have been proposed that aim to provide support for distributing data in a device-to-device fashion. Some of these solutions are even "content aware" and exploit the semantics of data's content to guide its propagation. Of these many approaches, few leverage the *contextual history* of data

itself to drive routing decisions and formulate higher-level conclusions. Spatiotemporal provenance provides a window into data's contextual past—it reveals precisely where data has traveled and when. Using empirical data sets of human proximity we demonstrate the practical utility of implicit spatiotemporal annotations for making bandwidth-saving routing decisions (*Provenance as a Local Inference Tool* in Figure 1.1): first, as a means for making local inferences about other nodes' knowledge and second, for identifying commonly co-located data-data and data-device pairs, which may be used as substitute routing targets. Such spatiotemporally-informed ad hoc routing strategies can also support in-network query protocols.

Existing systems that have enabled query-driven access to localized data provide only capabilities for searching relatively static data instead of the inherently ephemeral data of pervasive computing spaces, which cannot be easily indexed outside of the here and now (e.g., in the cloud). Supporting the execution of queries over data that represents the here and now requires a new perspective on the design of the search engine architecture that relies on search execution capabilities that take advantage of both opportunistic interactions between peer pervasive computing devices and the emerging availability of localized infrastructure in the form of cloudlets. Enabling expressive search over dynamic data in the here and now also requires understanding and efficiently collecting and representing the *context* of that data in a pervasive computing 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 a search engine for pervasive computing spaces. This relevance is further influenced by intrinsic characteristics of data that populates pervasive com-

puting spaces; 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.

To begin to address these needs, we introduced the Gander conceptual model [101] (the *Formal Conceptual Query Model* within the horizontal-striped box in Figure 1.1), which provides a foundation for precisely defining and reasoning about search of the here and now, in the here and now. In this dissertation, we use this conceptual model to explore the impact of the intrinsic spatial and temporal correlations of data in pervasive computing spaces on the performance of *in situ* query processing. To reify these goals and to shield application developers from the complexities of deploying their applications in pervasive computing spaces, we introduce a data sharing and query processing software framework, which aims to provide an implementation of the spatiotemporal data models and distributed query processing mechanisms.

1.4 Data Sharing and Query Processing Framework

A comprehensive implementation and evaluation of the Gander application substrate is a key research component in this project, constitutes a major systems contribution, and sets the stage for technology transfer. Key to our approach are real-world application inputs in the form of data, mobility, search content, and network traffic. This dissertation presents the Gander application framework (the checkered box of Figure 1.1), which provides infrastructure support to enable spatiotemporal data generation and expressive search in pervasive computing spaces.

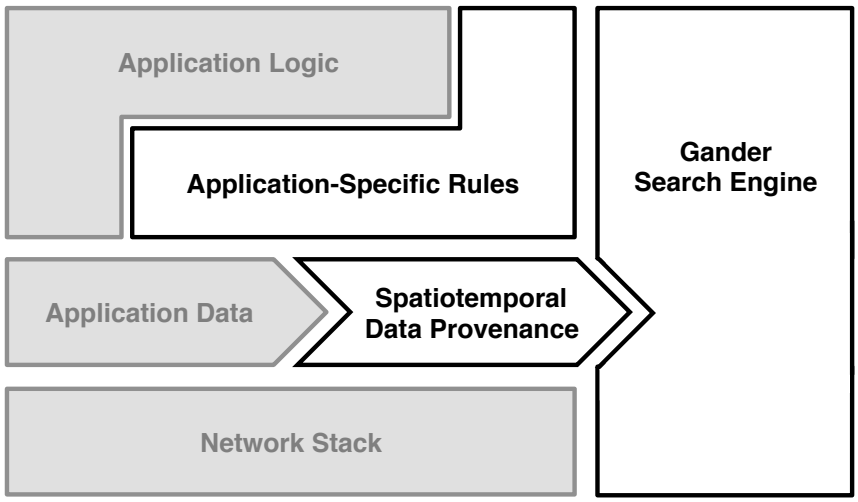


Figure 1.2: A high-level view of the Gander application framework (white objects).

This third and final aim addresses the concrete systems directions necessary to evaluate the formal foundations proposed by the first two aims and to deploy Gander within real world pervasive computing applications.

Figure 1.2 illustrates the core features of the Gander application framework (shown as white objects) and where they sit in the application stack (shown as gray objects). Each piece of application data has associated with it spatiotemporal data provenance (metadata), which captures both the (expected) dynamics of the real-world phenomenon the datum represents and the (actual) dynamics of the datum. This historical metadata enables complex reasoning about the *context* of data and also provides a foundation for expressing how an application should *react* to the state of its environment. Rather than embedding data-dependent logic throughout an application, this behavior may be expressed as reactive rules over data's content and spatiotemporal provenance (e.g., when and how data should be created, replicated,

moved, and eventually deleted). Gander's query processing strategies are provided as a middleware layer whose distributed protocols are implemented using reactive rules that may be parameterized by data content, provenance metadata, or complex combinations of the two. We evaluate the system performance and user-perceived utility of Gander's search strategies through a real-world deployment of a mobile application on the UT Austin campus. The rest of this dissertation will study the need for and realization of these components in detail.

1.5 Research Contributions

Parts of this dissertation have been published in peer-reviewed conferences and journals [99–101, 104] It addresses challenges related to sharing and querying spatiotemporal data in pervasive computing spaces by making the following contributions:

Enriching Shared Data with Spatiotemporal Provenance (Figure 1.1, vertical-striped box)

Research Contribution 1: We provide a formal model of explicit *spatiotemporal data provenance* for pervasive computing applications (Section 2.2). The model is founded on the spatiotemporal trajectory data type, which captures the dynamics of both digital data and the physical phenomena they represent. Spatiotemporal provenance metadata may be attached to application data to enable complex reasoning *about* data and the computation of its past and present context.

Research Contribution 2: We extend the model of explicit spatiotemporal provenance and introduce a formal model of implicit spatiotemporal data provenance (Section 2.3). Implicit data provenances employs a more contextual form of space that captures the causal propagation of data through a time-varying network. Our implicit model provides two complementary views on data provenance: a data-dependent view (i.e., what data is about) and a data-agnostic view (i.e., how data moves). By enhancing transmitted data with some degree of its causal history of propagation, these two views enable a mobile application to gain local insight into the overall spreading behavior of a piece of information, which devices have received it, what other information it is commonly associated with, and the time-varying topology of the network. We describe formal operators for comparing trajectories in meaningful ways.

Research Contribution 3: We describe how pervasive computing application development can be simplified by expressing data-dependent application behavior as reactive rules (Section 2.2.4). Rather than embed data-dependent logic throughout an application, a developer may use these rules to reason about and interact with spatiotemporal data in a general-purpose way, providing a nice separation of concerns. Our rule-based programming approach provides functional building blocks for the implementation of spatiotemporally-informed data dissemination and query processing mechanisms.

Sharing and Querying Spatiotemporal Data (Figure 1.1, horizontal-striped box)

Research Contribution 4: We create developer tools that enable the formulation of reactive rules governing data creation, sharing, and deletion (Section 3.1). Using these software components we benchmark the overhead of explicit spatiotemporal data provenance with varying trajectory resolution in a simulated pervasive computing environment (Section 3.1.4). Different pervasive computing applications will likely require different granularities of spatiotemporal data history. To gain a deeper quantitative understanding of how trajectory resolution impacts system performance we evaluate the system-level cost of explicit provenance annotations in terms of bandwidth and storage using a geospatial coordinate system.

Research Contribution 5: Leveraging the formal operators introduced in *Research Contribution 2*, we demonstrate the practical utility of implicit spatiotemporal data provenance. Using real-world data sets of human proximity, we benchmark the performance of causal data annotations as a means of locally estimating global network characteristics and data spreading dynamics (Section 3.2.1). Next, we showcase the inferential power of causal provenance for making bandwidth-saving routing decisions (Section 3.2.2): first, as a tool for detecting and eliminating redundant device-to-device transmissions and second, for identifying commonly co-located data-data and data-device pairs, which may be used as substitute routing targets within a targeted data dissemination protocol.

Research Contribution 6: We describe the Gander conceptual model of

search for pervasive computing environments, which supports queries over data and its spatiotemporal metadata. The relevance of real-world information is inherently parameterized by both space and time. Therefore, we assess the impact of data correlations in space and time on Gander's performance in simulation (Section 3.3). This task sheds light on how the Gander query processing protocols perform when the degree of these data correlations are varied; that is, we ask whether different degrees of correlations impact the *quality* of the results achieved by a Gander search. These results further validate the utility of spatiotemporal provenance and aid in the design and implementation of developer tool support for spatiotemporal trajectories and query processing protocols.

Data Sharing and Query Processing Framework (Figure 1.1, checkered box)

Research Contribution 7: We introduce a concrete mobile software framework that provides high-level programming constructs for expressing data-dependent application behavior in the form of reactive rules (Section 4.1). As a use case, we present the design and implementation of a mobile application created using the framework that distributes user content in a device-to-device fashion across multiple network hops. The framework abstracts away low-level implementation details regarding network communication and data serialization enabling developers to focus on the data-driven behavior unique to their application.

Research Contribution 8: We create and benchmark a resource discovery service for cloudlet-supported pervasive computing environments (Sec-

tion 4.2). Similar to a cloud computing resource, a cloudlet [152] hosts services that perform the expensive computations required by a mobile application. Unlike the cloud, however, cloudlets are inherently within close physical proximity to the mobile devices that utilize their resources. We leverage cloudlets to implement proximal discovery for applications targeting densely populated and highly mobile physical spaces.

Research Contribution 9: We provide the Gander Framework, an extensible software framework to support data sharing and query processing in pervasive computing environments (Section 4.3). This framework reifies the formal foundations of spatiotemporal data provenance, Gander's data model, and search protocols and provides the systems support necessary to evaluate and deploy Gander within real world pervasive computing applications.

Research Contribution 10: We evaluate the system- and user-level utility of the Gander search engine in a real-world deployment (Section 4.4). Our ultimate goal is to provide expressive support for real applications to share and search for data in pervasive computing spaces. This task aims to explore the impact of our approaches under realistic conditions that reflect the complexity of the rapidly changing physical environment at a large scale, which would otherwise be difficult to observe in controlled simulated settings.

The body of this dissertation is organized as follows. Chapter 2 lays the foundation for models of spatiotemporally rich data generation. Building on this

foundation, Chapter 3 describes formal methods and practical programming approaches for sharing and querying spatiotemporally-enhanced data. We introduce the Gander conceptual model and describe distributed methods for querying pervasive spatiotemporal data. Chapter 4 outlines the Gander application framework, which reifies the methods and mechanisms introduced in Chapters 2 and 3 within an extensible software framework. We evaluate the system performance and user-perceived utility of Gander's constructs within a real-world deployment on the UT Austin campus. Finally, Chapter 5 concludes and discusses directions of future work that build on the contributions of this dissertation.

Chapter 2

Enriching Shared Data with Spatiotemporal Provenance

In this chapter, we demonstrate the need for a general-purpose model for metadata capable of capturing the spatial and temporal dynamics of ephemeral data in pervasive computing spaces. We introduce two variants of such a model, both which enrich shared data with *spatiotemporal provenance*—metadata annotations that express data objects' contextual history across a lifetime of device-to-device propagation. The first model variant employs an *explicit* coordinate system to define space; the second variant uses an *implcit* form of space that captures data objects' causal propagation through a time-varying network. This chapter focuses on the items illustrated within the vertical-striped box shown in the upper left of Figure 1.1. Our proposed data models are founded on the *spatiotemporal trajectory* data type, which enables applications to reason about relationships between real world phenomena and the digital data that represents those phenomena.

Consider an example in which a boat collects observations of oil in coastal waters. These observations are represented as digital data carried by a device on the boat. Each of these *datums* moves in physical space as it is carried by the boat and is therefore subject to the boats movement patterns; any one of these datums could

Portions of this chapter appear in [104] for which coauthors Christine Julien, Jamie Payton, and Gruia-Catalin Roman provided advising and editing.

also move by being passed from one device to another. The physical phenomenon of interest also often exhibits spatiotemporal dynamics after its observation; in this example, the observations are of a physical phenomenon subject to dynamics in both space and time as coastal currents cause the oil to move and dissipate. A datums spatiotemporal trajectory records these dynamics of the observation and the physical phenomenon of interest, allowing applications within the pervasive computing network to create rules governing how datums capturing the phenomenon should move, change, live, and die. Spatiotemporal trajectories also provide a foundation for applications to reason about the impact of spatiotemporal dynamics on the use of a datum.

In this chapter, after expositing necessary components of a spatiotemporal-aware model for metadata annotations, we present two variants of such a model that employ different definitions of space. We further demonstrate the utility of our approach from an engineering perspective through a set of use cases that demonstrate how data-driven application behavior may be expressed as reactive rules triggered by application data and its spatiotemporal annotations.

2.1 Motivating Spatiotemporal Data Provenance

Much attention has been paid to capturing context, enabling resource discovery, delivering relevant information, and handling dynamics in pervasive computing environments; far less has been devoted to composing and organizing such systems [53]. Ultimately, resources and services will need to interoperate in real-world deployments. Applications may need to seamlessly move tasks among envi-

ronments [159], users may wish to aggregate information across resources [137], a service might need to collaborate with others [18, 108]. A common glue, or language, is needed to facilitate interoperability, enable composition, and maximize resource use in pervasive computing networks. This underlying uniformity will be best satisfied by embedding metadata within the data already communicated among applications. A shared model of such metadata would greatly simplify application development and support capture of richer contexts, enabling new classes of applications.

Context-awareness is a major theme in pervasive computing. From defining it to leveraging it, context determines the *modi operandi* of most pervasive computing applications. A common approach is to explicitly define an application-specific notion of context; determining what data is "relevant" rests solely with the application. This is sufficient when data exists only instantaneously and is not shared among applications. However, this is not the case in many pervasive computing environments, which significantly increases the amount of data available. A far better approach is to enable shared data itself to articulate the context in which it was acquired to provide clues as to contexts in which it might be relevant. A metadata model designed in this way would enable its associated data to exploit its contextual dependencies, providing a separation of concerns and greatly easing development burdens.

Traditional software models offer little support for the challenges that arise due to the inherent dynamics of pervasive computing environments [43], and developers are often forced to address these facets at the application level. A general-

purpose model capable of independently exploiting contextual dependencies that could be shared across services would enormously simplify and facilitate pervasive computing application development. Rather than impose a totalitarian framework or middleware, we believe that these facilities are best addressed by and within the data already used by applications.

User and application needs in pervasive computing are driven by real-world phenomena; they are inevitably a product of the environment of the users' interactions [109]. This is evident in the popularity of location-based services. A general-purpose metadata model should promote the separation of concerns called for in pervasive computing and emphasize attributes present in all real-world phenomena, namely, space and time.

Every action and event that captures one's existence has *both* spatial and temporal attributes, not merely one or the other [62, 134]. Users, devices, and data "move" through time. As time passes, they may move through space. Space and time may be treated independently, but are inseparable as correlated attributes of real-world phenomena. This is intuitive; the passage of time is naturally understood in terms of perceived changes to objects in space [126]. We argue that space and time are necessary (but not sufficient) to express context. Thus, a general-purpose metadata model for pervasive computing must account for the inherent spatiotemporal characteristics of real-world phenomena. More to the point, space and time must be first-class citizens of such a model.

We now introduce a formal model for spatiotemporal metadata that uses an explicit coordinate system to define space. In Section 2.3 we introduce an alternate

variant of this model that employs a implicit form of space.

2.2 A Model of Explicit Spatiotemporal Metadata

Existing work in pervasive computing focuses almost exclusively on how to *query* data, presupposing an existing data-rich environment and neglecting questions related to how data is created, replicated, moved around, stored, and destroyed.

Space and time must be first class citizens of our data model since pervasive computing intrinsically entails interaction with the physical world, and phenomena in this real world are inherently spatiotemporal. To bridge from the physical world into the digital one, we distinguish *reality* from our *perception* of it. Reality is defined by physical *phenomena* that can be sensed by the digital world (e.g., environmental conditions, availability of resources, presence of humans). An *observation* is made when a phenomenon is sensed. The observation is a digital representation of the phenomenon as it is instantaneously captured through the sensing process and is associated with an immutable spatiotemporal stamp. A *datum* is a digital representation of some *quantum* of knowledge about an observation. We want to expose the relation between a datum and an associated phenomenon (especially as data moves and time passes, i.e., as the datum experiences *spatiotemporal dynamics*). To facilitate this, we augment each datum with spatiotemporal metadata that represents the dynamics of the datum and the associated phenomenon in space and time.

Associating historical metadata with data objects is often referred to as *data provenance*, which has received recent attention in both distributed file systems [133] and large-scale cloud stores [111] for tracking the *logical* history and evolution of data

objects (e.g., modifications, domains of ownership, sources of contributed content, etc.). In these application domains, provenance metadata is maintained to help verify the authenticity and integrity of anonymously shared data. Our proposed model is a direct form of provenance, but aims to capture the *physical* contextual history of data objects as they are generated and exchanged between proximal hosts and applications in pervasive computing spaces. Similar to existing provenance aware storage systems [111], our *spatiotemporal data provenance* metadata may be used trace back information to its origins and aid in deciding how much to trust it [7]. However, we specifically target applications that deal with data about real world phenomena and operate in inherently distributed environments with no central provenance authority. To our knowledge, this dissertation represents the first investigation into the use of data provenance for mobile and pervasive applications.

We employ a *spatiotemporal trajectory* data type for spatiotemporal data provenance, which can capture both the (expected) dynamics of the phenomenon and the (actual) dynamics of the datum. Applications that create and use the data can define *rules* that use the trajectories to determine how data moves, changes, lives, and dies. These rules are not part of our data model, but Section 2.2.4 gives examples of rules that are obviously useful for spatiotemporal data in pervasive computing. In a simple sense, one can relate a trajectory to *spatiotemporal decay*, or the notion that the further a datum gets in space and time from its "genesis" the "weaker" it is.

2.2.1 From Physical Phenomenon to Digital Datum

The first challenge is the jump from the physical to the digital. A datum represents an observation of a phenomenon digitally. We store datums as semi-structured data [3, 47] that is self-descriptive in the sense that it consists of a set of name-value pairs. We assume an underlying vocabulary for data that is shared among all participants in the digital world.

Every phenomenon is associated with a "place" and "time." The place is a description of the spatial *influence* of the phenomenon; depending on a phenomenon's type, the shape and scope of its place may differ dramatically. Space can be physical, logical, or even contextual (see Section 2.3). In this section, we assume traditional two-dimensional physical spaces. A phenomenon's time is a (potentially open) range with a discrete beginning.

An observation logs a phenomenon and generates a datum. In the simplest sense, a datum exists at exactly the phenomenon's place for exactly the phenomenon's lifetime. This may not be possible or desirable for several reasons: (*i*) the observation may not happen at the exact location of the phenomenon; (*ii*) there may not be a digital device at the phenomenon's place; (*iii*) the phenomenon's place may be larger than a single device; (*iv*) devices that store data may be dynamic or unreliable; or (*v*) we may want to disseminate the datum more widely than the phenomenon's associated place.

In our model, each datum has a spatiotemporal trajectory that captures the initial relationship between the datum and the phenomenon. Over time, the

trajectory should also capture the evolution of this relationship by capturing the evolution of the datum and the phenomenon in both space and time.

2.2.2 Spatiotemporal Trajectories

A datum's spatiotemporal trajectory may evolve for many reasons: the phenomenon may be expected to be dynamic, the device carrying the datum may move, or the datum may be passed through the network. A datum's spatiotemporal trajectory should react to these dynamics and update itself. We separate *how* data items are communicated, stored, or moved from the fact that, by existing and moving in pervasive computing networks, they generate "fields of influence" given by their spatiotemporal availability. We begin with a simple yet expressive model of spatiotemporal trajectories and give insights into how these trajectories can be used to support a variety of expressive pervasive computing applications.

A datum is associated with a space-time stamp that marks where and when the observation was made. Each datum's *trajectory* is a sequence of vectors indicating the movement of the datum in space and time relative to its phenomenon. If the datum is carried by a mobile device, the vectors are defined by the path the device takes. If the datum is communicated from one device to another, then the trajectory contains a vector that traverses the distance between the devices at a velocity defined by the time the exchange requires.

Consider a data item about an exhibit in a museum collected and carried by the mobile device of a visitor:

Phenomenon. the Mona Lisa is in room 6
Observation. (at location $[x, y]$ and time t) the Mona Lisa is observed
Datum. $\langle painting = ML, (loc = [x, y], time = t)\rangle$
Trajectory. vectors of the visitor's path in the museum

An application relying on this information to share information with visitors about objects nearby them in the museum may use this trajectory to implement a form of data decay in space. This datum may *decay* with space as the distance from the painting grows. This notion of decay is similar to that captured by some existing middleware for pervasive computing [93]. The information may not decay in time since the painting is not expected to move or change. One could do something similar with a datum that decays only in time but not in space (i.e., anything that is true in all places but not at all times).

As another example, consider observations of the air quality (AQI) at a particular place and time collected by users' smartphones and shared via peer-to-peer interactions:

Phenomenon. the particulate concentration is $40.5\mu g/m^3$
Observation. (at location $[x, y]$ and time t) the AQI is 101
Datum. $\langle AQI = 101, (loc = [x, y], time = t)\rangle$
Trajectory. vectors of the movement of device(s) carrying the datum and passing of the datum among devices

An application that uses these observations to provide the user a dynamic and localized view of the air quality may degrade datums that are further in both space

and time from their observations. Note that these first two simple examples only employ the the observation's space-time stamp and a representation of the "here and now." Our model is not limited to these situations, and our later examples will show how the trajectory itself may be essential to the application.

A phenomenon itself may have some space-time behavior that must also be associated with the datum. Consider the following, in which the presence of oil in water is detected by a mobile collection point:

Phenomenon. oil in water moving at $32cm/s$ to the northwest
Observation. (at location $[x, y]$ and time t) there is oil in water whose current is $32cm/s$ to the northwest
Datum. $\langle oil = \text{true}, speed = .32m/s, direction = 135°, (loc = [x, y], time = t)\rangle$
Trajectory. vectors of the movement of the mobile collection device and the (expected) movement of the phenomenon

An application can use this information to, for example, make predictions about the current location of the oil. The complete trajectory can also give information about where the oil has been, enabling direction of cleanup efforts.

In this final example, the spatiotemporal dynamics of the phenomenon (the oil in the water) are captured (as the current's speed and direction) in a very application-specific way. We have limited ourselves for now to a simple model of two dimensional space; we also simplify our representation of a phenomenon's spatiotemporal dynamics as a single vector, whose starting point is given by *loc* and *time*, and whose (expected) speed and direction are represented as part of the da-

tum. In general, we can associate each phenomenon with its own trajectory that starts from the observation (as measured by *loc* and *time*). This trajectory could be simply a single vector as in the example, a series of vectors, or even some function of time and context whose value is a series of vectors.

2.2.3 Computing with Trajectories

Associating a spatiotemporal trajectory with an inherently spatiotemporal object is intuitive, but it is also extremely powerful. In the next section, we describe ways that this information can be used to enable applications to reason about and interact with the ephemeral spatiotemporal data that characterizes their environments. First, we give a taste of the computations that can be done on our spatiotemporal trajectories to enable additional application semantics. Ultimately, the ability to perform these types of computations will be provided as part of a set of development tools for spatiotemporal data provenance (as illustrated in Figure 1.1).

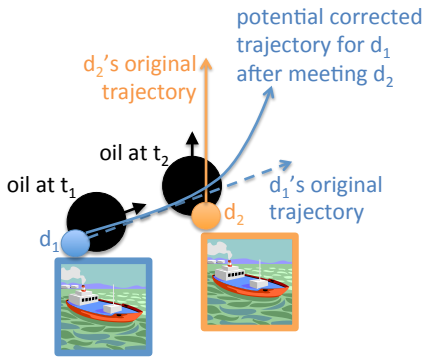


Figure 2.1: Sample trajectory computation

Smoothing Trajectories. Spatiotemporal trajectories associated with datums that live and move for long periods of time or with high degrees of dynamics may grow very long. Using vector addition, we can "smooth" trajectories, reducing the resolution of the information about the datum's spatiotemporal dynamics, at the benefit of decreased size of the datum.

Computations on Trajectories. Applications can also do relatively simple calculations using a datum's spatiotemporal trajectory. For example, an application could compute a *decay* value that numerically represents how "far" a datum is in space and time from its phenomenon. Applications can also perform computations over multiple trajectories. For example, if a datum encounters another datum that represents either the same or a different phenomenon, their trajectories can both positively and negatively reinforce each other.

Consider our example of oil movement and the trajectory that captures the spatiotemporal dynamics of the phenomenon (i.e., the speed and direction of the current). If a datum d_1 capturing an observation at location $[x_1, y_1]$ and time t_1 encounters a second datum d_2 of the same type (i.e., oil in the water) whose location $[x_2, y_2]$ and time t_2 lie on the trajectory defined by d_1's phenomenon's trajectory, then the trajectory in d_1 can be updated to reflect new observations of the current. Such a situation is shown in Figure 2.1, in which the trajectories (arrows) have been simplified to depict only the movement of the phenomenon (omitting the potential movement of the datums).

Aggregations of Observations. We can also extend our base model to allow a datum to be an aggregation of one or more observations. Consider the following example, where observations are made by vehicles moving in an urban area:

Phenomenon. there is an available parking space on the south side of 3^{rd} Street between Pine Street and Oak Street

Observation. (at location $[x, y]$ and time t) there is an available parking space
Datum. $\langle parking = 1, (loc = [x, y], time = t)\rangle$
Trajectory. vectors of the vehicle's movement

As the vehicle carrying the datum moves, it may observe additional available parking. It may aggregate these observations into the original datum, incrementing the counter *parking* and expanding the location and time. An application can use the associated trajectory to discover a path along which available parking spaces lie. Alternatively, the datum carried by one vehicle may encounter a datum from a different vehicle measuring parking availability along a different trajectory. These datums can be aggregated together in a more complex way to give a sense of overall parking availability in a general area (e.g., by computing the bounding box of the combination of the trajectories) or by representing the aggregate trajectory as a set of trajectories, giving a web of spatiotemporal information.

2.2.4 Expressing Data-Dependent Behavior using Rules

Our spatiotemporal model for ephemeral data enables pervasive computing applications to reason about the data they rely on, providing various ways to judge the (spatiotemporal) *quality* of data. In this section, we give a handful of examples of uses of this spatiotemporal model. Our model is not limited to these few uses; instead we intend to give a flavor of the variety of possibilities that exist. Effectively, each example is a way in which a pervasive computing application deployment provides a set of *rules* that dictate how the spatiotemporal trajectories associated with datums can be used to determine how data in is used, moved, stored, changed, and

destroyed. As part of the development tools for spatiotemporal data provenance (Figure 1.1) we will provide support for specifying and executing such rules.

2.2.4.1 Data Death

The most obvious (and likely common) thing to do is to define a very thin layer on the data model to control when datums are deleted. Such a layer should be parameterized by both space and time; i.e., when a datum gets a certain distance in space and time from its observation, it should be deleted. Consider the following simple rule that a data death layer could use to periodically delete any data item d that originated more than *threshold* units of distance away:

$$delete(d) \text{ if } dist(myLoc, d.loc) > threshold$$

Similar rules could account for time or for location and time jointly. Rules can also be defined over the entire trajectory; for example, if the sum of the trajectory's vectors indicates that the datum has *traveled* a certain distance, it could be deleted:

$$delete(d) \text{ if } \Big(\sum_{v \in d.traj} v.length \Big) > threshold$$

These definitions are not themselves part of the data model but instead are enabled by the availability of the spatiotemporal information the data model provides. In

fact, the definitions are highly application-dependent. Both thresholds on data death and the definitions' use of locations and trajectories will depend highly on the application and its operating conditions.

2.2.4.2 Data Persistence

Many applications generate data that is relevant in a particular physical space with the desire that the data stays in that space, even if the digital devices that inhabit that space move (e.g., [120] and [177]). A carrier of a datum may desire to *transfer custody* of the datum to another device if that device is closer to a target location or has a higher degree of location stability (e.g., as computed based on local context). For example, the rule:

$$transfer(d, h) \text{ if } dist(h.loc, d.loc) < dist(myLoc, d.loc)$$

would transfer the custody of datum d to the device h if h is closer to d's initial location than the current custodian. More complicated rules could also use the trajectory to attempt to make a datum's trajectory approximate the expected trajectory of a phenomenon (e.g., in the oil example described above).

2.2.4.3 Supporting Fidelity Estimation

In addition to using the spatiotemporal trajectories to move, replicate, and delete data, we can also use them to post-process data and potentially reason about the quality of queries or applications it supports. We can define *quality metrics*

that associate values with a datum, where the quality is determined based on space and time. For example, data that moves quickly may be determined to have a high fidelity (and a high positive potential impact on query resolution). The following rule uses the trajectory to compute an average velocity of a datum based on the velocities of the component vectors:

$$velocity(d) = \frac{\sum_{i=1}^{|d.traj|} \frac{d.traj[i].loc - d.traj[i-1].loc}{d.traj[i].time - d.traj[i-1].time}}{|d.traj| - 1}$$

Measures of fidelity could be based on how fast (or slow) data moves, how much it moves, or even how widely it moves; the appropriate fidelity metric is clearly application-dependent.

2.2.4.4 Directing Data and Queries

Using datums that have associated notions of spatiotemporal decay, we can think of the digital world as having *gradients* defined by the movement of data through space and time [93]. We can use these gradients to direct data, queries searching for relevant data, and physical entities that traverse the space. For example, given the (aggregate) datum that represents parking availability in an urban area, an application on an automobile could send a reservation for the parking space in the reverse direction of the datum's trajectory.

Considering a datum that indicates the potential propagation of a plume of oil in a large body of water, subsequent queries about water temperature or the concentration of important ocean flora can follow the gradients to the potentially highly impacted areas. Figure 2.2 shows a simple such situation in which a single datum generates a reverse query path; clearly multiple datums representing observations of the same or similar phenomena may generate more complex resulting query paths.

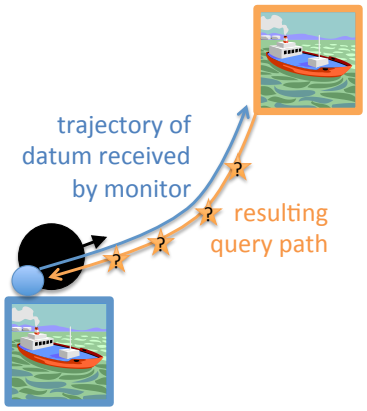


Figure 2.2: Query path defined by gradient

2.2.4.5 Trust and Security

Space and time also have significant potential impacts on trust, privacy, and security. For example, given that we can often control access to physical spaces, even if only for brief periods of time, we can compute a spatiotemporal bounding box to determine whether a particular datum has been *compromised* by exiting some controlled space and time. The following rule defines a data time d as "*safe*" if its entire trajectory is contained within $locThresh$ distance of $targetL$ and occurred within $timeThresh$ time of $targetT$:

$$safe(d) \text{ if } \forall \tau \in d.traj : dist(\tau.loc, targetL) < locThresh$$
$$\wedge |\tau.time - targetT| < timeThresh$$

2.2.5 Related Work

Modeling spatiotemporal data, both in theory and in practice, is not new. In fact, spatiotemporal data models have received prolific attention in the database and Geographic Information System (GIS) communities. Within pervasive computing applications, frameworks, and mechanisms, specifications pertaining to the creation, storage, and death of data are either embedded throughout the application or neglected altogether; an information-rich environment is typically presupposed.

The advent of Database Management Systems (DBMS) and, shortly thereafter, positioning and tracking technology (e.g., GPS) led to a boom of conceptual and practical spatiotemporal modeling efforts, resulting in a plethora of spatiotemporal data models [52,55,60,123,170,172], Moving Object Databases (MOD) [37,39], data stream management systems [1], spatiotemporal access methods [38], and ranking and indexing techniques [11,82,95,164]. Detailed summaries of these and similar approaches can be found in [70] and [126]. Recent work has focused on continued development of spatiotemporal query languages [90], the analysis of movement patterns [56], On-Line Analytical Processing [49], and even mobile extensions [109]. While these approaches have become well-accepted and have even produced new standards in GIS and DBMS [118], they all rely on centralized resources (e.g., a database) to catalog and intelligently index information. Valuable lessons can be learned from these well-developed mechanisms, but unprecedented challenges arise when an equivalent degree of spatiotemporal analysis must be performed in a distributed fashion on highly dynamic data with limited resources.

Pervasive computing applications are characterized by a desire to explicitly

associate physical space with virtually accessible resources and information through a window of context. DataSpace [71] is an envisioned spatially-addressed 3D global network in which physical space is modeled as a collection of 80-bit addressed *datacubes*. Querying and monitoring objects and *dataflocks* (cohesive groups of objects) in DataSpace is then spatially driven and constrained. In the same vein, EnviroTrack [2] is an extensive middleware in which events in an external environment are the addressable entities. Context-specific computation and actuation tasks (e.g., recording, signaling other devices, etc.) can then be "attached" to event signatures agnostic of where they are in physical space. Similarly, CASAMAS [17] is a model for cooperative pervasive computing environments that makes cohesive cooperative groups called *communities* first-class entities. Here, software agents within a community may remotely interact when they are "sensitive" to the *fields* emitted by other agents. Field intensity is modulated by space according to a em diffusion function and similarly, each agent type is characterized by a *sensitivity function*. In addition to spatial locality, other approaches seek to leverage logical [108, 114, 169], temporal [97, 153], and even social [27] locality to facilitate efficient resource access. These approaches are indeed driven by a common need for resource access parameterized by a notion of locality, be it space, time, or some product of the two. However, they each address that need by embedding a model or mechanism into the application itself. This is largely the case in existing pervasive computing applications.

Discovery and use of embedded resources is paramount in pervasive computing applications. Countless mechanisms have been developed to facilitate effi-

cient and effective resource discovery in large-scale and dynamic networks requiring decentralized control (e.g., peer-to-peer (P2P) networks, mobile ad hoc networks (MANETs), etc.). Distributed Hash Table approaches [85, 145, 161, 175] provide a distributed storage mechanism for structured spatiotemporal data enabling fast resource lookup in P2P networks. Similarly, query-access mechanisms for spatiotemporal data in MANET networks [107, 137, 177] typically maintain a virtual overlay network to handle dynamics. Rule and policy-based approaches, like TOTA [93], TOTAM [156], and xDUCON [147], enable autonomous opportunistic data propagation in MANET and P2P networks following a provided set of application-specific rules or policies. In recent years, the publish/subscribe paradigm has enjoyed copious attention in pervasive computing applications [13, 28, 29, 40, 43]. Publish/subscribe provides a high degree of decoupling, flexibility, and scalability while enabling efficient event distribution making it fit content-driven P2P and MANET applications very well. This wide variety of techniques and models heavily reflects a fundamental need to discover and share pertinent information in large-scale and dynamic networks. These and the many similar approaches all focus almost exclusively on the retrieval, or querying, of information, presupposing a data-rich environment and neglecting (or simply ignoring) questions related to how data is created, replicated, stored, and destroyed. We believe these questions and reiterated needs necessitate a flexible general-purpose metadata model that can be shared across services and tailored to meet individual application requirements.

Our coordinate-based notion of space is useful for supporting pervasive computing applications that rely on an explicit coordinate reference system, namely

geospatial. Coordinate-based models of spatiotemporal trajectories have mainly been developed to store and index the motion of objects in spatial databases [127], but have also been employed to coordinate efficient broadcast-based search protocols in wireless sensor networks [94] and to route messages in dense mobile ad hoc networks relative to a predefined Cartesian path [116] or along the motion path of a particular node [89]. On the other hand, for many applications, especially those involving battery-operated devices, localizing participants within an absolute coordinate system may be beyond the capabilities of device hardware, require too much energy, or simply be unnecessary. Therefore, in the next section, we introduce a variant of our model of spatiotemporal data provenance that employs an *implicit* form of space based on proximal network contacts rather than an explicit coordinate system. The use of an implicit view of space is motivated by practicalities: it is more general, it can support a wider range of resolutions, it does not require potentially energy-intensive sensing, and it allows analysis to be restricted to a temporal domain. We are also motivated by potential use cases: we target supporting opportunistic routing protocols, which are often contact-driven.

2.3 A Model of Implicit Spatiotemporal Metadata

The sheer density of connected devices in users' everyday environments is rapidly growing. The Internet of Things (IoT), for example, envisions that every "thing" we interact with will possess digital capabilities. Supporting emerging mobile applications in these very dense deployments requires connecting devices and their users to *hyper*-localized information. For many reasons, it is becoming ap-

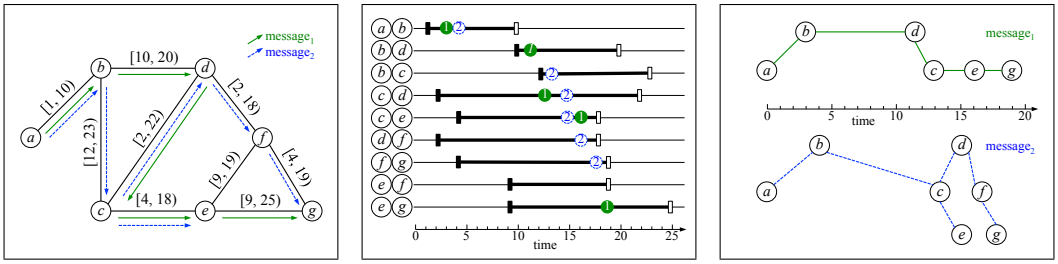
parent that device-to-device interactions will play a pivotal role in getting users connected to this data. There are numerous mechanisms designed to support the nuts and bolts of distributing messages in a device-to-device fashion, even over multiple hops, and even some of these mechanisms are "content aware" (i.e., they distribute the data based on its own semantics). However, few approaches tap into the *contextual history* of the shared data to distribute additional information. In this paper, we propose an approach whose key tenet is that knowing the context in which a piece of data was created and the contexts in which it has been shared over its lifetime can help determine the future contexts in which that data might be relevant.

Consider the following IoT-style scenario that illustrates the utility of messages' contextual history. At an outdoor music festival with thousands of attendees, festival-goers may wish to find nearby mobile vendors carrying particular food items, be alerted when friends are near, discover popular photos of recent performances, receive coupons from merchandise vendors, and be alerted about special events and limited product offerings at festival sponsors' tents. In such a densely populated environment, connecting attendees' devices with this hyper-localized information may require device-to-device interactions: the demand for fixed network infrastructure resources (e.g., cellular upload bandwidth) could greatly exceed available capacity; round trip times to the "cloud" may be too slow [152]; and the battery constrained nature of users' devices may make short-range network interfaces far more advantageous than long-range ones in terms of energy/bandwidth consumption [157]. As a message is opportunistically propagated between festival goers' devices it accu-

mulates a contextual history comprising causal trajectories of the device-to-device paths it traverses over space and time. For example, as a user's query for "*vendors selling craft beer*" is disseminated using an epidemic protocol, we can represent the query's complete contextual history as a partially ordered list of the timestamped pairs of devices that capture a transmission of the query. Knowing this history for one or more messages not only gives insight into the evolution of the network's time-varying topology (e.g., which may reveal the network diameter [22]), but also indicates *how* those messages have moved through the network. If many users are querying about craft beer, for example, the trajectories of previous queries may help guide future queries to relevant results more efficiently. Likewise, vendors carrying craft beer may use these trajectories to pinpoint demand and distribute digital coupons more effectively.

Similar scenarios are readily found in other domains. For example, cars in vehicular networks [86] communicate in a device-to-device fashion with one another and stationary roadside units to facilitate cooperative traffic monitoring, control the flow of traffic, and generate alternate routes on-demand based on ground-level traffic conditions. Alternatively, in a smart city [25] vehicles may communicate with "smart" municipal entities like parking meters to guide cars to available parking spaces; pedestrians may browse or search for hyper-localized information about urban spaces (e.g., a coffee shop with the shortest queue length) shared by passerbys' devices [120].

The primary design challenge for device-to-device mobile applications (e.g., mobile ad hoc networks [146], delay tolerant networks [132], and mobile social net-



(a) (b) (c)

Figure 2.3: Transmission graphs. Panel (a) shows an interval graph representation of an opportunistic network of mobile nodes where each solid black edge indicates 1-hop network proximity and is labeled with the time interval in which the contact occurred. Solid green and dashed blue arrows indicate transmission of message_1 and message_2 (respectively) over a connection. Panel (b) gives a slightly more temporally-emphasized picture of the system, where a line corresponds to an edge in (a) and the active interval is indicated. The position of a numbered circle along the x-axis on an edge indicates the reception time of the corresponding messages over that connection. The transmission networks of the two messages are illustrated in (c) where a node's position along the x-axis indicates the reception time.

works [80]) is operating under the intermittency of connections that result from device mobility. Formal and empirical studies (e.g., [21, 121]) of human mobility have driven the development of a wide spectrum of protocols that attempt to provide reliable and efficient dissemination of information within a dynamic network topology. While most of these approaches focus on human mobility (and the mobility of the devices the humans carry), we instead focus on *data mobility*, i.e., the time-varying spatial and temporal properties that characterize the spreading of a piece of data within an opportunistic network of mobile nodes. Real-world human trajectory data sets and analysis have been exceptionally useful for understanding the spatiotemporal dynamics of how humans move and interact. Data trajectories, on the other hand, are subject to an entirely different set of barriers (e.g., physical, social, security) and can be very different from human paths. For example, consider Figure 2.3, which visualizes the device-to-device propagation of two mes-

sages (indicated by the solid green and dashed blue lines) through a transmission graph representation [141] of an opportunistic network of mobile devices. The time-varying proximities of the devices, shown in Figure 2.3(a) and 2.3(b), are a product of (among other things) human mobility; as the humans carrying these devices move, the devices come into contact with one another. However, the causal trajectories of the two messages, shown in Figure 2.3(c), while enabled by device proximity, extend beyond the bounds of any single device's mobility—data mobility is rather a product of device *interactions*. In Figure 2.3, for example, nodes a and g never come into physical contact (e.g., as a result of physical, social, or temporal boundaries), however both messages originating at a are eventually delivered to g. Harnessing the inferential value of a message's contextual lifetime directly within a highly mobile environment requires a distributed approach that can express the causal structure of message propagation.

In this section, we extend our model of spatiotemporal data provenance and introduce a variant model that captures the *implicit* contextual history of device-to-device propagated messages. Our model provides two complementary views on data mobility: a data-dependent view (i.e., what data is about) and a data-agnostic view (i.e., how data moves). For example, the transmission network in Figure 2.3 may be viewed in a data-dependent fashion in terms of message_1's trajectory (solid green lines), message_2's trajectory (dashed blue lines), or a union of both. Alternatively, the network's time-varying edges may be considered in a data-agnostic view irrespective of the data that has traversed across them. By enhancing transmitted data with some degree of its contextual history, these two views enable a mobile

application to gain local insight into the overall spreading behavior of a piece of information, which devices have received it, what other information it is commonly associated with, and the time-varying topology of the network. We describe spatiotemporal operators for comparing trajectories and drawing inferences about the state of knowledge.

2.3.1 Model & Terminology

Our model is based on the *time-varying graph* (TVG) formalism [20], which we extend to support spatiotemporal trajectories. Whereas a graph is a natural fit for representing a fixed network, a *time-varying* graph is a natural means for representing the kind of highly-dynamic and infrastructure-less networks we target. The TVG formalism is advantageous for our purposes because it offers an *edge-centric* analytical perspective on the evolution of the graph that enables, for example, focusing on the dynamics of a single edge independent of the dynamics of the entire graph. In this section, we overview the relevant concepts of the TVG formalism. The core of the formalism is the notion of a *time-varying graph*, which enhances a static *underlying graph* with time-dependent dynamics. Data propagated between nodes in the graph follows *journeys*, a form of temporal reachability. We represent the set of all journeys that a piece of data takes as a *transmission network*. Acquiring a global view of a transmission network may be impractical in an opportunistically-connected network. In our model, along side each piece of transmitted data nodes share a *spatiotemporal trajectory*, a subset of the transmission

network that captures the propagation of data along a causal path[1]. The extent of each spatiotemporal trajectory's coverage of the transmission network is determined by an *update strategy*.

Time-Varying Graph. We represent an opportunistic network of mobile nodes as a *time-varying graph* (TVG) $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta, M)$ as follows. V is a set of mobile nodes making contact with each other over the *lifetime* ($\mathcal{T} \subseteq \mathbb{T}$) of the network. Here the temporal domain $\mathbb{T}$ corresponds to $\mathbb{R}^+$ (continuous-time). $E \subseteq V^2$ is the set of intermittently available undirected edges defined by the contact between nodes; $(x, y) \in E \Leftrightarrow x$ and y are in contact at least once in $\mathcal{T}$. For simplification, we use undirected edges under the assumption that two nodes in contact can both transmit to and receive from one another. In a real world system this assumption may not hold (e.g., communication capability may not be symmetric, some devices may be strictly transmitters or receivers, and not all devices may advertise their identity). Our model easily generalizes to a directed graph; the temporal domain creates its own level of direction. The *presence function*, $\rho : E \times \mathcal{T} \to \{0, 1\}$, indicates whether a given edge is available at a given time, and the *latency function*, $\zeta : E \times \mathcal{T} \to \mathbb{T}$, indicates the time it takes to propagate a message over a given edge at a given time. For simplicity of presentation, we assume the latency function to be fixed for all edges and times and denote it as a constant value ζ, which we assume to be known by all nodes. The duration of edge presence, however, can be arbitrarily long. To signify the time-dependent availability of an edge e, we use the

[1] We assume that mobile nodes can marshall spatiotemporal trajectories in and out of a serialized form.

notation $\rho_{[t_1,t_2)}(e) = 1$, which indicates that $\forall t \in [t_1, t_2), \rho(e, t) = 1$. For example, in the TVG shown in Figure 2.4 $\rho_{[1,3)}(ab) = 1$. So far, this is the same definition of TVG as in [20]. We enhance the TVG framework with M, a set containing triples of the form (m, e, t), where each triple signifies the transmission of a message m on edge $e \in E$ at time t; to account for transmission times, all $(m, e, t) \in M$ must satisfy $\rho_{[t,t+\zeta)}(e) = 1$.

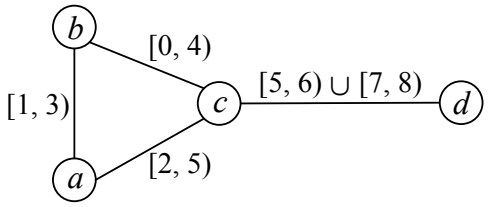


Figure 2.4: A TVG $\mathcal{G}$ [20]. Edge labels represent the time intervals during which those edges are available, i.e., $\cup(t \in \mathcal{T} : \rho(e, t,) = 1)$.

We assume that nodes have unique identifiers and can detect the appearance and disappearance of an incident edge instantaneously. In our model, each node maintains a local TVG in which the node's proximate neighborhood and the duration of each neighbor's contact are stored as *first-hand* knowledge. Each transmitted message carries also the message's contextual history, which is extracted and locally stored by a receiving node as *second-hand* knowledge. Local TVG maintenance is driven by dedicated operations triggered by the appearance and disappearance of incident edges and the reception of messages containing contextual history metadata (see Algorithm 1). We take an entirely distributed approach; the global TVG $\mathcal{G}$ only exists in a virtual sense (i.e., it is a global virtual data structure [130]) and is a union of the local TVGs maintained by the nodes in the network: $\mathcal{G} = \cup_{v \in V} \mathcal{G}_v$.

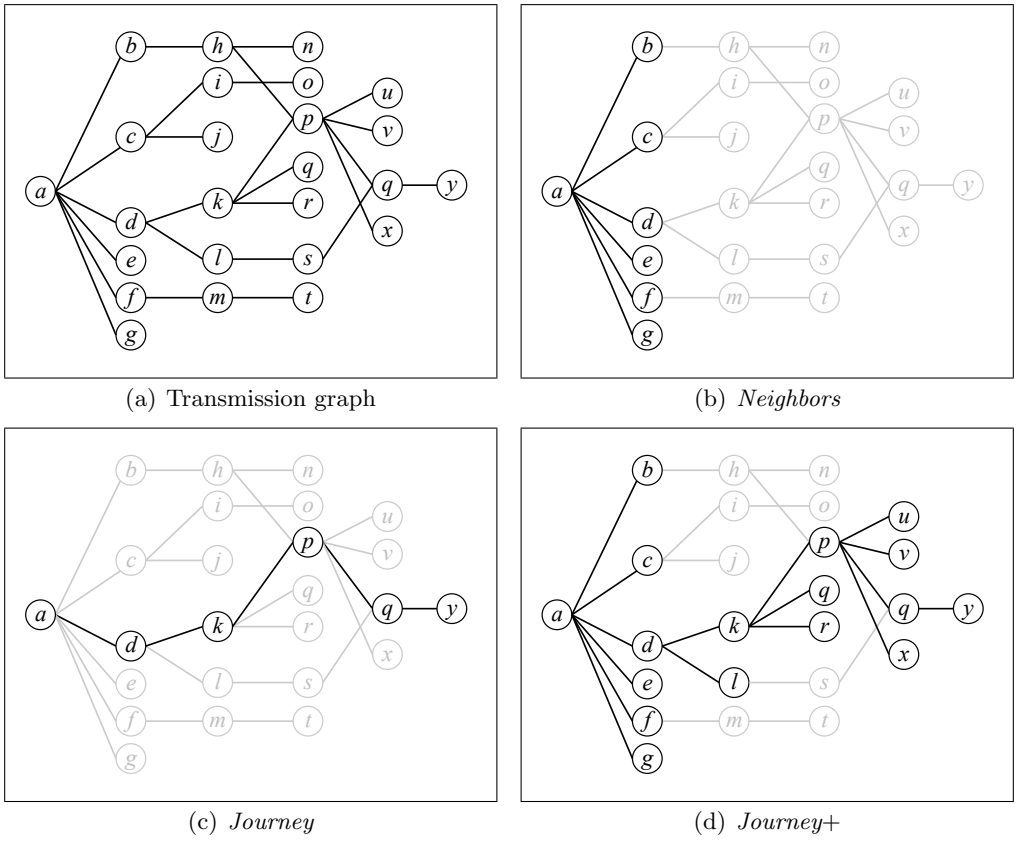
Underlying Graph. We define $G = (V, E)$ as the static counterpart of $\mathcal{G}$ called the *underlying graph*. G can be thought of as the "footprint" of $\mathcal{G}$ if the time

dimension were flattened.

Journey. We capture the notion of temporal reachability [168] through a *journey*. A sequence of tuples $\mathcal{J} = ((e_1, t_1), (e_2, t_2), \ldots, (e_k, t_k))$, where $e_1, e_2, \ldots, e_k$ is a walk in G and $t_i + \zeta \leq t_{i+1}$ for $1 \leq i < k$, is a journey in $\mathcal{G}$ if and only if $\rho_{[t_i, t_{i+\zeta}]} = 1$. $\mathcal{J}_{(u,v)}$ is a path over time from node u to node v. We denote $\mathcal{J}^*_{\mathcal{G}}$ as the set of all journeys in $\mathcal{G}$ and $\mathcal{J}^*_{(u,v)} \subseteq \mathcal{J}^*_{\mathcal{G}}$ as those journeys starting at node u and ending at node v. In other words, $\mathcal{J}^*_{(u,v)}$ is the set of all time-dependent paths along which a message could travel from u to v. If a journey exists from u to v, i.e., if $\mathcal{J}^*_{(u,v)} \neq \emptyset$, then we say u can reach v, which we denote as $u \rightsquigarrow v$. However, this relationship is not symmetric. For example, $a \rightsquigarrow d$ via the journey $\mathcal{J}_{(a,d)} = ((ab, 1), (bc, 3), (cd, 7))$ in Figure 2.4; however, no valid journey exists from d to a.

Transmission Network. A *transmission network* represents the diffusion of a piece of information through the opportunistic network, i.e., the set of journeys that define the causal propagation of a message within $\mathcal{G}$. The notion of a transmission network was introduced in [73] to represent the causal structure of an infection's propagation, but the concept was not formalized. Here, we borrow the concept and formally define the transmission network of m as $\mathcal{J}^*_{\mathcal{G}}(m) \subseteq \mathcal{J}^*_{\mathcal{G}}$, the set of all journeys along which m traveled. While a single journey represents a time-dependent path a message could take, a transmission network represents the set of all journeys a message has actually taken.

Spatiotemporal Trajectory. To enable local insight into the contextual history of a message, we introduce a novel *spatiotemporal trajectory* data type. In



(a) Transmission graph (b) *Neighbors*

(c) *Journey* (d) *Journey+*

Figure 2.5: Spatiotemporal trajectory update strategies. Panel (a) shows a complete transmission graph. Panels (b)–(d) illustrate a sample trajectory's contents (darkened nodes and edges) when updated per each of our three strategies.

our model, an outgoing message's spatiotemporal trajectory $T(m) \subseteq \mathcal{J}^*_{\mathcal{G}}(m)$ is updated based on a *strategy* before its transmission and attached as metadata to the message. We define three fundamental update strategies below; additional strategies could be derived and integrated based on application or domain requirements. A node receiving m extracts $T(m)$ and inserts the contents into the local TVG (see Algorithm 1). Conceptually, a spatiotemporal trajectory serves two purposes. First, $T(m)$ reveals a subset of nodes that have received m and when, which can be useful input for an opportunistic routing protocol. Second, $T(m)$ provides a data-

dependent view on the evolution of $\mathcal{G}$ through m's time-varying *vicinity* [128]. These two uses are complementary because our model enables the causal propagation of m to be investigated in terms of the context of $\mathcal{G}$ or vice-versa. As trajectories are aggregated together within a node's local TVG they may be used collectively to make remote inferences about encountered nodes' knowledge. For example, upon coming into contact with node v for the first time, node u may inspect its local TVG for trajectories containing v, which provides u with a glimpse of the kind(s) of data v possesses and the topological context under which v sent or received that data. A local collection of spatiotemporal trajectories may also indicate what types of messages are commonly associated with other types of messages and nodes, which can be used to determine viable "proxies" for particular types of messages. Messages about "*craft beer*," for example, may frequently occur close in space and time to messages about "*pretzels*" or perhaps node v may exist in many trajectories of messages about "*craft beer*," indicating that node u may use "*pretzels*" messages and v as "proxies" for "*craft beer*." We explore these two use cases in Section 3.2.

Update Strategies. Alongside each transmitted message a node appends a spatiotemporal trajectory as metadata. The attached trajectory includes the node's local view of the message's transmission network (the message's causal past) as well as a snapshot of the current state of the network from the node's perspective. Exactly what is included in this snapshot is determined by a trajectory update strategy.

We introduce three strategies for updating an outgoing message's trajectory based on properties of the transmission network. These strategies capture the

coverage of the message's transmission network to varying degrees. Figure 2.5 illustrates the differences between each of these strategies and their coverage of the transmission network.

***Neighbors*:** Before transmitting m at time t to k proximate neighboring nodes, the transmitting node v_{TX} generates a single new spatiotemporal trajectory $T(m)$ containing edges between itself and all k neighbors that will receive m, i.e., $T(m) = \cup_{i \in k}((v_{TX}, v_{RX_i}), t)$. The updated $T(m)$ is then attached to m and transmitted to each of the k proximate neighbors. This strategy produces a $T(m)$ strictly representing the first hand receivers (i.e., the "witnesses") of m at t (see Figure 2.5(b)).

***Journey*:** Before transmitting m at time t to a single proximate neighboring node v_{RX}, the transmitting node v_{TX} updates m's trajectory $T(m)$ (or generates a new one if $T(m) = \emptyset$) by appending an edge between itself and the receiver, i.e., $T'(m) = T(m) \cup ((v_{TX}, v_{RX}), t)$. This strategy produces a trajectory that is precisely a journey of m in $\mathcal{G}$ (see Figure 2.5(c)).

***Journey+*:** This strategy effectively combines the first two. Before transmitting m at time t to k proximate neighboring nodes, the transmitting node v_{TX} updates m's trajectory $T(m)$ (or generates a new one if $T(m) = \emptyset$) by appending an edge between itself and each of the k receivers, i.e., $T'(m) = T(m) \cup_{i \in k} ((v_{TX}, v_{RX_i}), t)$. This strategy produces a journey of m enhanced with the receivers of m at every transmission along the journey's causal chain in $\mathcal{G}$. The result is that each node has a different (but potentially overlapping) view of the trajectory (see Fig. 2.5(d)).

When trajectories meet one another in the field (i.e., when a node receives multiple copies of the same message via different causal paths) their knowledge may be *merged* at the receiving node, producing a more complete local view of the entire transmission network. For example, in Figure 2.5(d) when node y receives the message via the journey that passes through node h, y will receive the edges $\{(b,h),(h,n),(h,p)\}$, which are merged into y's view of the message's transmission network. Each of these update strategies produce trajectories that vary in their coverage of the global transmission network. The *Neighbors* strategy provides good spatial coverage, but only at an instant in time. On the other hand, the *Journey* strategy provides good temporal coverage, but only captures the propagation along a single causal path. The *Journey+* strategy combines the coverage strengths of both the previous strategies but at the cost of added overhead. Applications must take these tradeoffs into consideration when choosing a trajectory update strategy.

In Algorithm 1 we provide the dedicated operations used by a node u to maintain its local TVG $\mathcal{G}_u$. Initially, $\mathcal{G}_u$ only contains a single node u (itself); E_u, ρ_u, and M_u are each empty. When u comes into contact with a node v, the *onContact*() operation is triggered, which first ensures that $v \in V_u$ and that $e = (u,v) \in E_u$ and finally updates ρ_u to indicate the presence of e. Upon node v moving out of contact range the *onTimeout*() operation is triggered, which updates ρ_u to indicate that e is no longer present. Both the *onContact*() and *onTimeout*() operations alter $\mathcal{G}_u$ based on u's first hand knowledge. The injection of second hand knowledge is performed by the *onReception*() operation, which is triggered whenever a message m is received. This operation extracts m's spatiotemporal trajectory $T(m)$ and inserts

Algorithm 1: Local time-varying graph operations at mobile node u.
All operations are performed on u's local TVG instance $\mathcal{G}_u$.

$\mathcal{G}_u \leftarrow (\{u\}, \emptyset, [now(), \infty), \emptyset, \zeta, \emptyset)$

onContact with a neighbor v at time t:
$e = (u, v)$
$V_u \leftarrow V_u \cup v$
$E_u \leftarrow E_u \cup e$
$\rho_{u,[t,\infty)}(e) \leftarrow 1$

onTimeout of a neighbor v at time t:
$e = (u, v)$
$\rho_{u,[t,\infty)}(e) \leftarrow 0$

onReception of a message m:
extract $T(m)$ from m
foreach *tuple* $(e = (i, j), t)$ *in* $T(m)$ **do**
    $V_u \leftarrow V_u \cup \{i, j\}$
    $E_u \leftarrow E_u \cup e$
    $M_u \leftarrow M_u \cup (m, e, t)$
    $\rho_{u,[t,t+\zeta)}(e) \leftarrow 1$

getTrajectory of a message m in window $[t, t')$:
return $\cup\{w \in M : w.m = m \wedge t \leq w.t \leq (t' + \zeta)\}$

the trajectory's contents into $\mathcal{G}_u$ tuple-by-tuple. Given trajectory tuples of the form $(e = (i, j), t)$, each tuple's nodes and edge are added to V_u and E_u (respectively), ρ_u is updated to indicate the presence of e for the duration of the transmission $[t, t+\zeta)$, and a tuple (m, e, t) is added to M_u indicating the transmission of message m on edge e at time t. Finally, u may retrieve the spatiotemporal trajectory of message m within a time window using the *getTrajectory*() operation.

2.3.2 Operating on Trajectories

So far we have presented the formal constructs of our model and how a node uses those constructs to maintain its local TVG instance. At a local level, a node's TVG contains first hand knowledge of the nodes it encounters and second hand knowledge of other nodes' encounters through received messages' trajectories. Though not practically obtainable, a system's global TVG may be represented by the union of all nodes' local TVGs. This section describes practical operators that are useful for comparing trajectories and drawing inferences about the state of knowledge, both at a local and global level. In Section 3.2, we then use these operators to benchmark the performance of our trajectory update strategies at the global level using two real world data sets of human proximity.

The mobile environments we target are densely populated with sensing and computing resources that disseminate large amounts of spatiotemporal information *about* the environment. Given that such digitally-accessible spaces are rich with data and their contextual histories, valuable inferences can be drawn not only from individual trajectories, but also from the comparison of trajectories.

Consider, for example, two devices at the crowded outdoor music festival that periodically broadcast digital special offers: a stationary device at a sponsor's merchandise tent and a mobile device carried by a beverage vendor. A mobile node (e.g., an attendee's smartphone) receiving one of these offers may inspect its trajectory, which reveals, to an extent, *where*, *when*, and *with whom* the offer has travelled; these are indicators of the offer's spatiotemporal "about-ness" and the time-varying state of knowledge about the offer across the network. Since both

offers are propagated close in space and time, their contextual histories will likely share a large overlap. A mobile node receiving both offers may infer from this overlap that the offers pertain to a similar place and time. From a routing perspective, this is useful information both for identifying where and when more of such offers may be available and for propagating the offers towards contexts that they have not yet visited. And yet, this information may be misleading due to the mobility of the beverage vendor who is only transiently nearby the merchandise tent. Comparing spatiotemporal trajectories in meaningful ways requires structured operators that can be used in well-defined ways to draw accurate inferences. Here we describe binary trajectory operators that may be used as inferential building blocks.

Since trajectories carry both spatial (structural) and temporal information, our operators must be applicable along both these dimensions and also across the combination of the two. Our representation of spatiotemporal trajectories as causally-ordered sets of tuples naturally enables the use of set operations. Given two trajectories $T(m_1)$ and $T(m_2)$, from here on shortened to T_1 and T_2 respectively, we can apply any set operation $\otimes$ (e.g., union $\cup$, intersection $\cap$, complement $\setminus$, and symmetric difference $\triangle$) across both the spatial and temporal dimensions as $T_1 \otimes T_2$. Alternatively, we can focus only on the structure of the trajectories by restricting $\otimes$ to the spatial dimension:

$$T_1 \otimes_s T_2 := \otimes\{e : e \in T_1 \otimes T_2\} \tag{2.1}$$

A spatial trajectory operator $\otimes_s$ flattens the temporal domain and produces a static graph in G. Likewise, we can ignore the causal relationships embedded in the tra-

jectories and restrict $\otimes$ to the temporal dimension:

$$T_1 \otimes_t T_2 := \otimes\{[t, t+\zeta) : t \in T_1 \otimes T_2\} \tag{2.2}$$

Temporal operators $\otimes_t$ produce the kinds of timing information commonly used within explanatory analysis techniques in the field of information diffusion [50]. These trajectory operators are binary but can easily be extended to *weighted* versions akin to weighted set similarity metrics (e.g., Tanimoto coefficient).

Comparing trajectory similarity across the spatial and temporal dimensions may be particularly useful input for opportunistic routing protocols. For example, a protocol may favor data that is spatiotemporally diverse (about different places and times) and choose to propagate data whose trajectories have small spatiotemporal overlap with local knowledge at each mobile node. At the crowded outdoor music festival this kind of routing strategy may be useful for eliminating redundant information and freeing up the wireless spectrum. We can measure spatiotemporal similarity using the Jaccard similarity coefficient:

$$J(T_1, T_2) = \frac{|T_1 \cap T_2|}{|T_1 \cup T_2|} \tag{2.3}$$

Completely independent of message contents (*what* their data is about), trajectory similarity indicates the relationship between disparate messages' contexts (*where* and *when* their data is about). As an example, consider a message m that arrives at a festival attendee's mobile device u through a chain of device-to-device propagations. The device u may inspect its local TVG $\mathcal{G}_u$ and retrieve all other received messages $M_{sim} \subseteq M$ whose trajectories share high spatiotemporal similarity

with m's trajectory $T(m)$. The similarity between these messages provides a local window into remote devices' probable knowledge. For example, when u comes into contact with another device v, u may use $T(m)$ and the trajectories of all other spatiotemporally similar messages $T_{M_{sim}} = \cup\{T(m_{sim}) : m_{sim} \in M_{sim}\}$ to decide whether or not to propagate m to v. Based on u's knowledge of v's spatiotemporal proximity to any node in $T(m) \cup T_{M_{sim}}$ u may infer the likelihood that v has already received m. If u determines that v very likely has received m then there is no reason to waste energy and network resources sharing the redundant information.

So far these operations have been data-agnostic. A useful feature of our model is the ability to additionally investigate spatiotemporal phenomena in a data-dependent fashion. We can extend our spatiotemporal operators to a third dimension to express data-dependent measurements:

$$T_1 \otimes_d T_2 := \otimes\{m : m \in T_1 \otimes T_2\} \tag{2.4}$$

In a dynamic and distributed environment the data-dependent dimension becomes more meaningful when associated with one or both of the previous two dimensions. For example, a device may monitor the content of received messages whose trajectories frequently share high spatiotemporal similarity and maintain sets of such contextually similar data in a "contextual cache." At the crowded festival, perhaps an attendee's mobile device detects that messages about "*craft beer*" commonly propagate at along similar causal paths and shortly after messages about "*hotdogs*" (e.g., because the beer vendor is following the hotdog vendor). An opportunistic query protocol may leverage these contextual data-dependent similarities as

substitutable routing targets. At the festival a user's query for "*hotdogs*," for example, may be automatically extended to "*hotdogs* or *craft beer*" given their frequent spatiotemporal similarity. We explore both of these use cases in Section 3.2.

2.3.3 Related Work

Emerging mobile applications increasingly require transparent access to digital resources in a user's physical proximity. For example, Apple's iBeacon technology provides mobile users with hyper-localized alerts about nearby locations (e.g., special offers from a nearby store in the mall); Google's Physical Web project[2] aims to provide "walk up" mobile access to smart objects. Extending beyond just single hop device-to-device connectivity, OpenGarden's FireChat[3] application constructs a multi-hop mesh infrastructure to facilitate chat rooms between physically proximate mobile users without the need for an Internet connection. More generally, these technologies and applications point towards the grand vision of the Internet of Things [4] and Web of Things [58] wherein commonplace entities, from kitchen appliances to vending machines to bus stops to billboards, are imbued with digital capabilities that are seamlessly accessible through a user's mobile device. The role of device-to-device interactions (i.e., via short-range wireless protocols) becomes crucial in supporting mobile users' hyper-localized information needs in such densely populated environments of networked devices, particularly in scenarios where an Internet connection is unavailable, infrastructure network resources are saturated, global-accessibility is undesired, or use of a short-range network interface (e.g., Bluetooth Low Energy)

[2] `http://google.github.io/physical-web/`
[3] `http://opengarden.com/firechat`

requires less overall energy than of a long-range network interface (e.g., cellular). Multi-hop device-to-device routing mechanisms, which are still largely relegated to the research domain, are necessary to unlock mobile users' vicinity [129] and support information needs that span both space and time.

Numerous approaches have been developed to support the mechanics of disseminating data in a device-to-device fashion within dynamic networks of mobile nodes. Epidemic routing schemes [32] mimic the spread of infectious agents (data packets) to susceptible hosts (mobile nodes). Many publish/subscribe mechanisms have been proposed and evaluated in the literature [27, 117, 158] and some are even "content aware," routing messages based on their own semantics [28, 43]. Other dissemination approaches aim to provide high availability of data where it is spatiotemporally relevant [120]. The TOTA middleware [93] gives an application developer great flexibility by providing adaptive context-aware programming constructs that can be used to define reactive rules that govern how data diffuses. Predictive mechanisms [174] leverage the cyclical nature of human movements by analyzing contact (duration of an encounter between a pair of nodes) and inter-contact (duration between consecutive encounters of the same pair of nodes) times. Still other methods attempt to exploit the interplay between social networks, data diffusion, and mobility patterns by routing messages to nodes with high social centrality [69] or regularity [98]. Despite this wide spectrum of device-to-device communication mechanisms, none directly capitalize on the contextual history (i.e., the causal structure) of transmitted data to distribute additional data. Likewise, most analytical frameworks for data dissemination [54] employ stochastic models that compute mes-

sage delay distributions (e.g., based on parameters describing inter-contact time and transmission range) and do not specifically address the causal structure of message propagation.

Studying and modeling the role of causality in the spreading of information is of prime interest within online social networks like Twitter, Facebook, Reddit, and Digg. These mediums play a huge role in how news stories, media, and information in general spread at a global scale across the Internet, therefore there is great interest in learning *how* particular information spreads. The field of information diffusion [57] focuses on understanding and modeling how information "diffuses" within these networks with the purpose of, for example, anticipating popular topics, identifying influential information spreaders, and optimizing social marketing campaigns. Explanatory approaches [50] attempt to infer the underlying tree of influence representing who transmitted a piece of information to whom given only a set of nodes ordered by the times at which those nodes "learned" the information. On the other hand, predictive approaches [48] try to anticipate how a specific diffusion process will unfold on a given network by learning from past diffusion traces. Valuable lessons can be learned from these models. However, many of the techniques developed assume the underlying social network is globally accessible and either static or only minimally dynamic. Neither of these assumptions holds in the highly mobile and intermittently connected environments that we target in our work.

In the field of epidemiology, models that capture how diseases spread through networks of human contacts [64, 113] also have a significant potential relationship to data movement in device-to-device communication networks. These models make no

assumption about any underlying social network and have found many uses outside of the study of diseases. For example, disease spreading models have been used to probe the causal structure of data dissemination in networks of human proximity with the motivation of uncovering human behavioral patterns that will aid in better defining data routing strategies for device-to-device communication [73, 121]. The motivation of these works is very similar to ours; however, their focus is on post-hoc analysis over a global view of data's spreading history. In this work, we introduce a distributed model in which local information alone may be used to make *in situ* inferences about data's causal history. In Section 3.2 we demonstrate the utility of the kinds of insights that may be derived from our model for informing routing protocols at runtime.

The crux of our model is a *spatiotemporal trajectory* data type, which captures the time-dependent device-to-device propagation of a piece of information. Our spatiotemporal trajectory is similar in spirit to the *gradient* abstraction developed to guide queries and data in wireless sensor networks. In the directed diffusion [72] paradigm, for example, a sink node conducts a sensing task by propagating *interest* messages that establish gradients along which relevant *events* flow from source nodes towards the sink. Whereas a complete gradient is defined at the network level by a scalar field held at each node, an instance of our spatiotemporal trajectory directly stores the time-dependent view of a network from the perspective of a single piece of transmitted data, making a portion of the network-level knowledge locally available to a receiving node.

The T-CLOCKS model [19] is a distributed tool that provides devices in

delay tolerant networks with *temporal views* of other network participants (i.e., a measure of the temporal and topological distance between nodes). Our trajectory data type may be thought of as a topological enrichment of the T-CLOCKS temporal view construct—a spatiotemporal trajectory expresses a data-dependent temporal view in addition to the causal path of device-to-device propagation taken by a piece of data. Similar to the T-CLOCKS tool, the *change awareness* measure [59] quantifies information freshness in opportunistic networks. This metric captures how much the latest information received from a node differs (in time) from the most up-to-date information being propagated by that node. Computing change awareness requires global knowledge of the network topology. Our model operates in a distributed fashion and captures the time-dependent paths pieces of information take through the network, providing greater local insight into the contextual history of that information and the characteristics of the network as a whole.

2.4 Research Contributions

This chapter makes the following research contributions:

Research Contribution 1: We provide a formal model of explicit *spatiotemporal data provenance* for pervasive computing applications (Section 2.2). The model is founded on the spatiotemporal trajectory data type, which captures the dynamics of both digital data and the physical phenomena they represent. Spatiotemporal provenance metadata may be attached to application data to enable complex reasoning *about* data and the computation of its past and present context.

Research Contribution 2: We extend the model of explicit spatiotemporal provenance and introduce a formal model of implicit spatiotemporal data provenance (Section 2.3). Implicit data provenances employs a more contextual form of space that captures the causal propagation of data through a time-varying network. Our implicit model provides two complementary views on data provenance: a data-dependent view (i.e., what data is about) and a data-agnostic view (i.e., how data moves). By enhancing transmitted data with some degree of its causal history of propagation, these two views enable a mobile application to gain local insight into the overall spreading behavior of a piece of information, which devices have received it, what other information it is commonly associated with, and the time-varying topology of the network. We describe formal operators for comparing trajectories in meaningful ways.

Research Contribution 3: We describe how pervasive computing application development can be simplified by expressing data-dependent application behavior as reactive rules (Section 2.2.4). Rather than embed data-dependent logic throughout an application, a developer may use these rules to reason about and interact with spatiotemporal data in a general-purpose way, providing a nice separation of concerns. Our rule-based programming approach provides functional building blocks for the implementation of spatiotemporally-informed data dissemination and query processing mechanisms.

2.5 Impact

Our work in this chapter is, to our knowledge, the first to conceptualize and formalize models of *spatiotemporal data provenance* for pervasive computing applications. Traditionally, provenance metadata is maintained to verify the authenticity and integrity of data objects as they are shared, accessed, and edited by many participants in large scale systems (e.g., distributed file systems [133] and cloud data stores [111]). Our models of spatiotemporal provenance capture the *contextual* history of data objects across their lifetime as they are generated, replicated, and exchanged between proximal hosts and applications in physical spaces.

2.6 Chapter Summary

This chapter motivated the need for a general-purpose model of metadata to capture the spatiotemporal dynamics of shared application data in pervasive computing spaces. We presented two variants of a model that annotates data objects with *spatiotemporal trajectories*, which provide the contextual history of objects across a lifetime of device-to-device propagation. The first model variant employs an *explicit* coordinate system to define space, which may be integrated into applications that rely on an explicit form of space (e.g., geospatial coordinates). Applications may then define expressive computations and behavior rules over data's spatiotemporal provenance. The second model variant uses an *implicit* form of space that captures data objects' causal relationship to their genesis within a time-varying network of mobile nodes. Our implicit model provides two complementary views on data mobility: a data-dependent view (i.e., what data is about) and a data-agnostic view

(i.e., how data moves). We described spatiotemporal operators within our implicit model for comparing trajectories in meaningful ways across spatial, temporal, and data-dependent dimensions. In Chapter 3, we build on the formal foundations introduced in this chapter and demonstrate the practical utility of our models' constructs for driving data-dependent application behavior and for improving ad hoc routing performance. The next chapter also introduces a conceptual model for querying and accessing short-lived data in opportunistic networks, which we use to directly investigate the impacts of spatial and temporal data correlations on distributed query protocols.

Chapter 3

Sharing and Querying Spatiotemporal Data

In pervasive computing spaces, users and applications share and search for information about *me*, *here*, *now*. These fundamental tasks are resolved by interactions that leverage proximally available resources directly within the immediate surroundings, via a localized cloudlet infrastructure or dynamically formed wireless ad hoc networks. Many emerging mobile applications require support for such interactions; however, the entry point for creating mobile applications that communicate with dynamic sets of peers (e.g., proximate devices) is still quite high and often requires specialized knowledge of a particular wireless protocol or platform-dependent networking libraries. In an effort to lower that development barrier, this chapter presents the design and implementation of software tools and programming constructs that aid in the development of mobile applications driven by heavily data-dependent behavior and hyper-localized information needs. Using the models introduced in Chapter 2 as a foundation, we introduce formal and practical constructs that aim to lower the programming barrier for applications that require transparent access to hyper-localized digital data and resources. In the next chapter (Chapter 4), we describe a concrete software framework that provides these

Portions of this chapter appear in [100] and [101] for which coauthors Christine Julien, Jamie Payton, and Gruia-Catalin Roman provided advising and editing.

constructs as extensible building blocks for application developers. This chapter focuses on the research items depicted within the horizontal-striped box in the upper right of Figure 1.1.

Using the *explicit* model of spatiotemporal data provenance (Section 2.2), we elaborate on the rule-based programming approach described in Section 2.2.4 and concretely demonstrate how data-dependent application logic may be expressed as reactive rules. This aim is motivated by software engineering principles—encapsulating data-dependent behavior provides a separation of concerns and makes that behavior more portable. We present the design and implementation of a middleware that realizes our model of explicit spatiotemporal metadata and provides a rule-based programming interface. The middleware may either be used to directly store application data and automatically track data's spatiotemporal provenance or to supplement an existing data store, enriching application data with spatiotemporal annotations. We conduct a case study using the middleware aimed at assessing the tradeoffs between the overhead of spatiotemporal annotations and their effectiveness at representing the changing state of a simulated environment. The case study presents rules that govern data generation and the maintenance of data's spatiotemporal provenance. Rules may also express how and when data moves within a dynamic network.

Next, we turn our attention to the *implicit* model of spatiotemporal provenance introduced in Section 2.3 and use the formal operators presented in Section 2.3.2 to demonstrate the practical utility of implicit provenance for making inference-based routing decisions that significantly improve the performance of op-

portunistic data dissemination. This task advances theoretical understanding of time-varying graphs [20] through the lens of real-world uses cases. Our use cases further shed light on the impact of spatial and temporal dynamics on the performance of ad hoc routing protocols. Searching for data in dynamic networks requires query mechanisms that are sensitive to the intrinsic spatiotemporal dynamics of these systems.

Finally, we present the Gander conceptual model [101] for expressive search in pervasive computing spaces. Gander supports queries with constraints specified over data content, data's spatiotemporal provenance, or combinations of the two. A Gander query is resolved entirely *in situ* using distributed query mechanisms that employ ad hoc network routing protocols. A number of query protocols may be able to resolve a query for spatiotemporal data. However, the execution of a query should reflect the user's requirements in terms of the expected quality of query results and the associated cost of query processing under the current operating conditions. In an effort to guide developers in selecting appropriate search protocols and settings for an operating environment, we study the impact of spatial and temporal correlations within sensed data on Gander's distributed query mechanisms. This aim provides an experimental baseline for the system-level performance of Gander's formal constructs.

3.1 Programming with Reactive Data-Driven Rules

Creating pervasive computing applications requires a developer to exploit real-world data by defining how that data is stored, manipulated, shared, and

queried within the application. To alleviate the developer's burden this section presents the design and implementation of software tools to aid in the development of applications with significant data-dependent behavior. Specifically, we present the design and implementation of the explicit spatiotemporal metadata model overviewed in Section 2.2. The model itself has been implemented within a graph database framework and packaged into a Java middleware layer that can be used to directly store application data and its spatiotemporal provenance or simply to enrich existing application data with spatiotemporal metadata. Finally, the middleware's API enables a developer to define data-dependent rules (as described in Section 2.2.4) to govern how, when, and where data and its metadata is created, moved, updated, and destroyed.

3.1.1 Explicit Spatiotemporal Metadata Model Implementation

We provide an implementation of our explicit spatiotemporal provenance model within the TinkerPop Blueprints graph database framework[1]. The choices of a graph-based solution and this particular framework were the result of a few key constraints and requirements. First, graphs are an exceptionally flexible abstraction tool for data modeling. Second, our *implicit* model of spatiotemporal provenance is graph-based; implementing our *explicit* model within a graph database facilitates reuse of high-level components. Finally, there are a plethora of open source graph databases currently available, each with their own set of tradeoffs; our goal was to avoid marrying our implementation to a specific backend as much as possible.

[1] `http://www.tinkerpop.com/`

The Blueprints framework satisfies each of these requirements and provides a highly portable foundation on which to build any graph database application.

Blueprints is a *property graph model* interface that provides a whole stack of useful graph database tools (e.g., for dataflow processing, a traversal query language, POJO-to-graph object mapper) and is entirely backend-agnostic. A property graph is a special type of graph whose (*i*) vertices and edges may possess any number key-value pairs (properties), (*ii*) edges are directed, and (*iii*) different edges may represent different relations between vertices (i.e., the graph is multi-relational). Any graph database that implements the Blueprints interfaces automatically supports any Blueprints-enabled application. In other words, a graph database application implemented within the Blueprints interfaces can plug-and-play different graph database backends without any code changes.

Abstractly, our model of data and its associated spatiotemporal metadata can be defined as a graph with two logical partitions: one partition comprises application data (datums), the other comprises spatiotemporal data provenance (spatiotemporal trajectories).

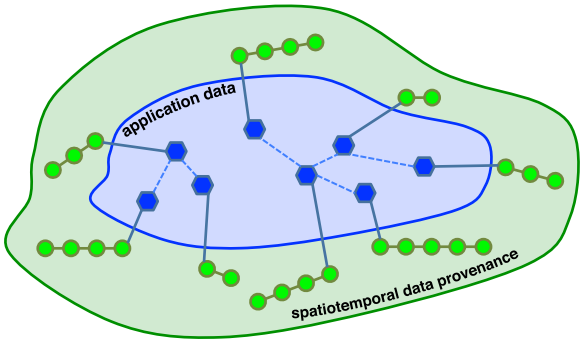


Figure 3.1: The application data (datum) and spatiotemporal metadata (provenance) partitions.

This abstraction is illustrated in Figure 3.1. Concretely, our graph is defined as $G = (V, E)$, where each vertex $v \in V$ is either a datum vertex (the inner hexagonal vertices in Figure 3.1) and is in the data partition ($v_d \in V_d$) or is a spatiotemporal

metadata vertex (the outer circular vertices in Figure 3.1) and is in the metadata partition ($v_m \in V_m$). An edge connecting two datum vertices $e = (v_{d1}, v_{d2})$ may represent an arbitrary application-dependent data relation. Edges between metadata vertices $e = (v_{m1}, v_{m2})$ form our spatiotemporal trajectory data type by defining a doubly-linked list of space-time positions (this is discussed in greater detail next). Finally, an edge that crosses the boundary between these two partitions $e = (v_{d1}, v_{m1})$ associates a spatiotemporal trajectory with a particular datum. In our model, each datum v_d may only have one associated spatiotemporal trajectory $\{v_{m1}, v_{m2}, \cdots, v_{mn}\}$.

We reify the explicit spatiotemporal trajectory data type as a doubly-linked list of *space-time position* (i.e., spatiotemporal metadata) vertices. Each space-time position vertex is a tuple $v_m = \langle(\textit{latitude}, \textit{longitude}), \textit{time}\rangle$. This implementation uses a two-dimensional geospatial coordinate system for sake of concreteness. However, a *space-time position* vertex could easily be extended to represent a point within any explicit coordinate system (e.g., three-dimensional geospatial by adding a *altitude*). To map from graph objects to Java objects, and vice versa, Blueprints provides Frames[2], which enable a developer to "frame" a graph element in terms of a particular Java interface, exposing the graph as a collection of interrelated well-defined domain objects rather than a collection of raw key-value pairs. Listing 3.1 provides the pertinent elements of the framed space-time position vertex definition. Each space-time position vertex possesses a *location* and a *time* property (indicated by the `@Property` annotations) and directed references to a *previous* and a *next*

[2] `https://github.com/tinkerpop/frames/wiki`

Listing 3.1: The Blueprints Frame definition of a space-time position vertex.

```
public interface SpaceTimePosition extends VertexFrame {
  @Property("location")
  public float[] getLocation();

  @Property("location")
  public void setLocation(float[] latlon);

  @Property("time")
  public long getTime();

  @Property("time")
  public void setTime(long time);

  @Adjacency(label = "next", direction = Direction.OUT)
  public SpaceTimePosition getNext();

  @Adjacency(label = "next", direction = Direction.OUT)
  public void setNext(SpaceTimePosition pos);

  @Adjacency(label = "previous", direction = Direction.IN)
  public SpaceTimePosition getPrevious();

  @Adjacency(label = "previous", direction = Direction.IN)
  public void setPrevious(SpaceTimePosition pos);
}
```

Listing 3.2: The Blueprints Frame definition of a datum vertex.

```
public interface Datum extends VertexFrame {
  @Adjacency(label = "trajectory-head")
  public SpaceTimePosition getTrajectoryHead();

  @Adjacency(label = "trajectory-head")
  public void setTrajectoryHead(SpaceTimePosition pos);
}
```

`SpaceTimePosition` (indicated by the `@Adjacency` annotations). The Blueprints Frames library takes care of type checking and conversion between raw graph vertex objects and framed `SpaceTimePosition` objects. Not shown in the listing, for space considerations, each space-time position also possesses a reference to its associated datum vertex.

The definition of a datum vertex's properties is left entirely to the application (e.g., phenomenon location, time of creation, etc.). Instead, we simply require that it possesses a reference to its spatiotemporal trajectory as shown in Listing 3.2. An application may explicitly define required datum properties and relations (edges) by extending the framed datum interface or simply by adding properties to datum vertices and creating edges between them on-the-fly at runtime (a mechanism to do so is detailed in Section 3.1.2). For example, the air quality example described in Section 2.2.2 may wish to define datum properties to capture the measured concentrations of various airborne chemicals, sensed wind speed and direction, air temperature, etc. Though not shown in Listing 3.2, the `Datum` interface also provides a convenience method for pushing new `SpaceTimePosition`s onto the datum's spatiotemporal trajectory. Blueprint's Frames enable `VertexFrame` interfaces to imple-

ment any of their methods annotated with `@JavaHandler` within a nested abstract class, which is used to handle frame method calls.

Now that we have defined the constructs of our spatiotemporal metadata model, it would be helpful if there were software components and tools to instantiate and manage them. The next section describes a Java middleware layer created to relieve applications from the responsibilities of generating, managing, and updating application data and spatiotemporal metadata by exposing a rule-based programming interface.

3.1.2 Middleware Architecture

To enable ease of use and deployment within an application we provide a middleware layer that encapsulates the metadata model implementation described in Section 3.1.1. The layer is implemented entirely in Java and may be used to directly store application data (as framed `Datum` vertices) or simply to maintain spatiotemporal provenance for existing application data. This middleware is publicly available on GitHub[3].

Figure 3.2 illustrates the middleware's architecture. At a high level, the middleware comprises our graph database implementation of the spatiotemporal metadata model (the dashed box in Figure 3.2) and a family of interfaces to interact with the database (the solid trapezoidal boxes surrounding the graph database).

At the core of the graph database itself is a `BaseGraph` interface, which is the lowest level Blueprints interface on top of a concrete graph database backend.

[3] `https://github.com/jonasrmichel/spatiotemporal-data`

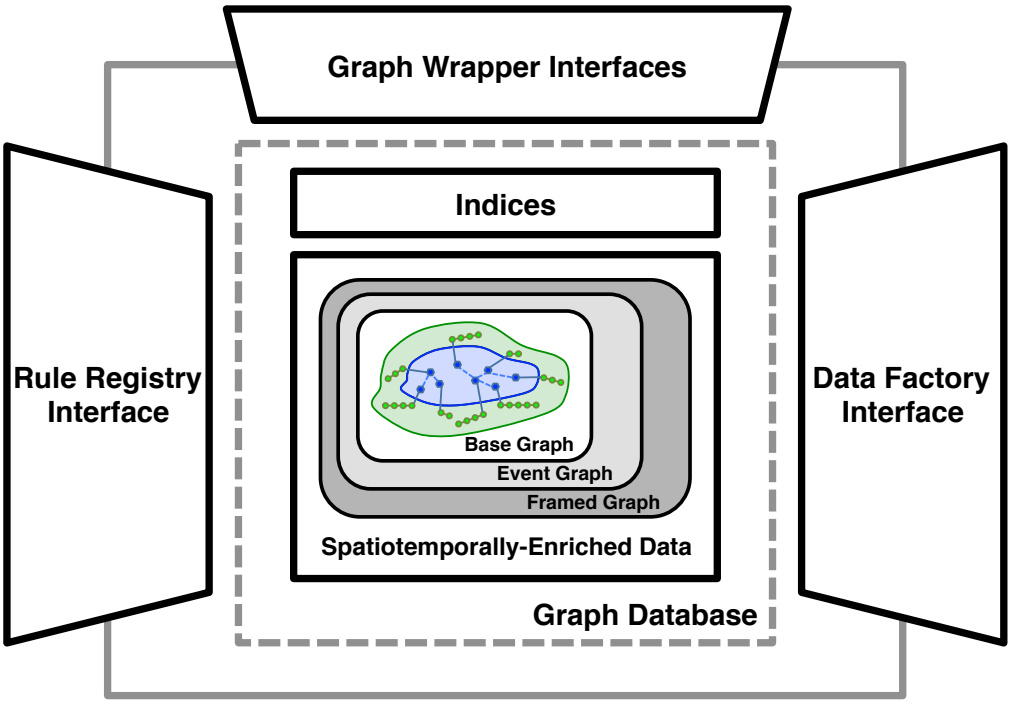


Figure 3.2: Graph database middleware architecture.

The `BaseGraph` is wrapped by two Blueprints `WrapperGraphs`: an `EventGraph` and a `FramedGraph`. The `EventGraph` wrapper enables listeners to be attached to particular graph events (e.g., when vertices are added or properties change), which is useful for triggering reactive behavior (e.g., data-driven rules). The `FramedGraph` wrapper enables framed vertices (e.g., `SpaceTimePoints` and `Datums`) to be created, inserted into, and extracted from the `BaseGraph`. The graph database also supports pluggable *external* indices (e.g., Lucene[4] and Elastic Search[5]). These may be used to index data's properties or spatiotemporal metadata's geospatial locations to enable

[4] `https://lucene.apache.org/`
[5] `http://www.elasticsearch.org/`

Listing 3.3: The middleware's *Datum Factory* interface.

```
public interface IDatumFactory {

public Datum addDatum(float[] location, long time,
                      Map<String, Object> properties,
                      List<Rule> rules);

}
```

faster vertex lookups.

An application may wish to interact directly with the database's graph or either of its wrappers (e.g., for querying). The middleware exposes each of these interfaces, shown in Figure 3.2 as the top-most trapezoid labeled *Graph Wrapper Interfaces*. On the other hand, to simplify the creation and storage of datums and their spatiotemporal metadata, the middleware exposes a *Data Factory* interface (the trapezoidal box on the far right of Figure 3.2), which provides a single method shown in Listing 3.3. The `addDatum()` method creates and inserts a new framed `Datum` vertex populated with any key-value pairs provided in the `properties` parameter. This method also initializes the datum's spatiotemporal trajectory with a new framed `SpaceTimePosition` vertex using the `location` and `time` parameters. In many circumstances, an application may require that datums' spatiotemporal trajectories be updated periodically, either per some time interval (e.g., every 10 minutes) or per some spatial interval, in other words, after the device running the application has traveled a certain distance (e.g., every 100 meters). To relieve applications of this responsibility, our middleware enables *rules* to be defined and attached to individual datums either using the final parameter of the `addDatum()`

method or using the *Rule Registry* interface (the trapezoidal box on the far left of Figure 3.2). These rules are described in detail in the next section.

3.1.3 Defining Data-Dependent Rules

Rather than create, update, maintain, and destroy data and its spatiotemporal provenance directly, the middleware layer enables an application to define *rules* that implement this otherwise completely data-dependent logic. From an engineering perspective, these rules provide a convenient separation of concerns, fully encapsulating logic that might otherwise be scattered throughout an application.

Within the middleware, an abstract `Rule` class is provided that possesses references to each of the graph wrapper interfaces, enabling the underlying graph structures to be modified based on whatever necessary application criteria. Of particular import to mobile and pervasive applications is the ability to update data's spatiotemporal provenance automatically (e.g., periodically per some time or distance traveled interval). To support this sort of reactive behavior, the middleware provides an abstract `GraphChangedRule`, which extends the `Rule` class and, when created, will register itself as a Blueprints `GraphChangedListener` with the graph database's `EventGraph` interface. A `GraphChangedRule` can trigger data-dependent behavior whenever the underlying graph's structure or properties are altered (e.g., when a new vertex or edge is added or removed or when a vertex or edge property is modified).

As a concrete example, consider a mobile application that requires all stored datum's spatiotemporal trajectories be updated with the operating device's current

Listing 3.4: An example rule that updates spatiotemporal provenance whenever the user has traveled more than 100 meters.

```
public class SpatialUpdateRule extends GraphChangedRule {
  float[] refLoc;

  // constructor omitted for space

  @Override
  public void vertexPropertyChanged(Vertex vertex,
              String key, Object oldValue,
              Object setValue) {
    if (!(vertex.hasProperty("special-key") &&
              key.equals("location"))) {
      // this is not the "special" location vertex
      return;
    }

    float[] myLoc = (float[]) setValue;

    if (dist(myLoc, refLoc) > 100) {
      for (Datum d : delegate.getGoverns()) {
        // convenience method provided by the datum
        // interface to update a datum's trajectory
        d.add(myLoc);
      }
      refLoc = myLoc;
    }

  private float dist(float[] loc1, float[] loc2) {
    // returns the distance in meters between
    // two geographic coordinates
  }
}
```

location whenever the device moves more than 100 meters (e.g., because the device is carried by a human user). We could express this logic formally with the following rule: $\forall v_d \in V_d, v_d.add(v_m(myLoc))$ if $dist(myLoc, refLoc) > 100m$. To define this rule within the the middleware we would first need to create and insert a "special" vertex into the graph to represent the user's current location (e.g., a vertex $v_{location} = \langle$*special-key*, *location*$\rangle$ where the *location* property's value is a (*latitude*, *longitude*) pair). Next, we would define the rule class shown in Listing 3.4. Once instantiated, the above rule class will automatically maintain data's spatiotemporal provenance, appending a new `SpaceTimePosition` to each datum's trajectory whenever the device travels more than 100 meters.

An important feature to point out in the `SpatialUpdateRule` class is the `delegate` variable that is referenced to obtain an iterator of associated `Datum`s. To activate a rule within the middleware it must be "registered" with the graph database through the *Rule Registry* interface (Figure 3.2)[6]. When a rule is registered it is actually stored in the graph database within a special `RuleContainer` framed vertex. Each `RuleContainer` vertex possesses a reference (edge) to any `Datum` that it "governs." This enables a rule to quickly access the datums and spatiotemporal trajectories it affects or is affected by. So, to complete the example, whenever a new datum is created using the *Datum Interface*'s `addDatum()` method (Listing 3.3), we would simply provide an instance of the `SpatialUpdateRule` within the method's `rules` parameter. The created `Datum` will then be autonomously "governed" by our

[6]Rule registration is automatically performed for rules provided to the *Datum Factory* interface's `addDatum()` method.

`SpatialUpdateRule`'s behavior. We now employ our spatiotemporal provenance middleware to conduct a case study that provides concrete examples of data-driven rules and explores the accumulated storage overhead of spatiotemporal metadata within a simulated pervasive computing application.

3.1.4 Case Study

In an effort to demonstrate the utility of spatiotemporal data provenance and to conduct an assessment of the tradeoffs between spatiotemporal metadata's "cost" and "quality" this section presents a case study that directly employs the middleware described in Section 3.1.1. Given that many pervasive computing applications must operate without reliance on an Internet connection and cloud resources, a pervasive or mobile application interested in maintaining spatiotemporal provenance will likely be constrained by devices' physical storage capacity. Therefore, when creating an application crucial design decisions include defining precisely *how much* spatiotemporal metadata to maintain and for how long it should be stored. To make these decisions, an application developer needs to know how much metadata is "good enough" for the application and at what point data is no longer of any use (i.e., when it becomes spatiotemporally irrelevant). In this section, we present a case study that aims to provide evidence to assist a developer in making these decisions by using data's spatiotemporal provenance to evaluate the spatial and temporal "decay" of knowledge in a simulated mobile environment as the environment's mobility and metadata spatiotemporal resolutions are varied. We demonstrate how notions of information decay may be realized as concrete computations performed directly

on data's spatiotemporal trajectories as discussed in Section 2.2.3

3.1.4.1 Spatiotemporal Decay & Metadata Overhead

In pervasive computing spaces data is ephemeral and may have a very short lifespan. Once a physical phenomenon is sensed by a particular device, both the phenomenon and the representative digital datum may undergo dynamics—the phenomenon may move or evolve, the device carrying the datum may move, the datum may be transferred between devices. As these dynamics occur the datum's quality (i.e., how accurately it represents reality) may degrade or *decay* in both space and time as the datum travels farther from its space-time point of genesis. The rate at which the datum decays depends on both the degree of these dynamics and the resolution of the datum's spatiotemporal metadata.

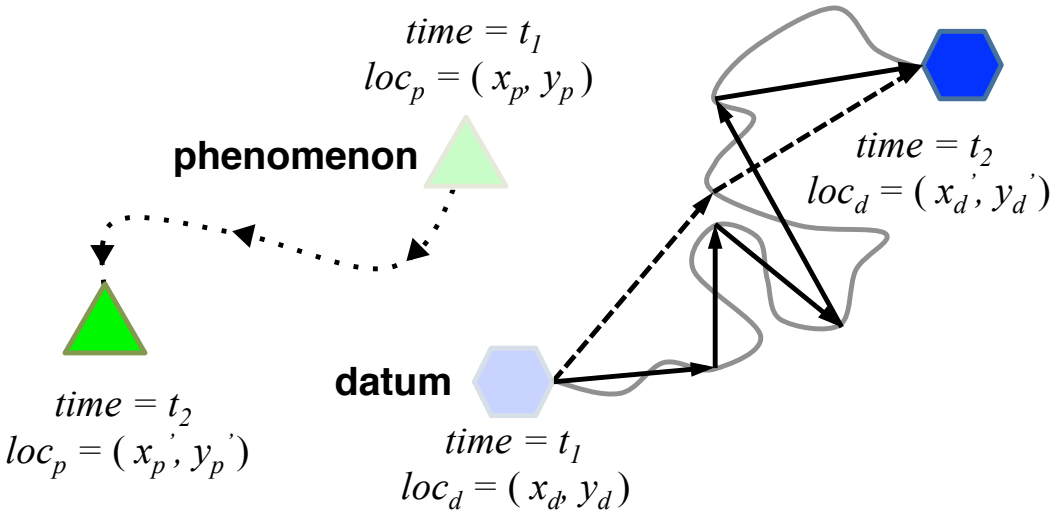


Figure 3.3: Spatial and temporal decay with varying trajectory resolution.

To illustrate the notion of decay more clearly, consider the scenario shown in

Figure 3.3. At time t_1 a phenomenon located at (x_p, y_p) (the triangle) is sensed by some device located at (x_d, y_d), which generates a datum (the hexagon) representing this phenomenon and its observed location at time t. At some later time t_2 both the phenomenon and datum have both moved—the phenomenon to (x'_p, y'_p) along the dotted black path and the datum to (x'_d, y'_d) along the solid gray path. In this scenario, we can compute the *temporal decay* of the datum as the delta between the phenomenon's observed location at time t_1 and its actual location at time t_2 as a function of time:

$$decay(t_1, t_2) = distance\Big(\big(x_p, y_p\big), \big(x'_p, y'_p\big)\Big) \tag{3.1}$$

Similarly, we can compute the datum's *spatial decay* using the same distance delta, but instead parametrized by two notions of space. First, we can compute a datum's spatial decay as the distance between the phenomenon's observed and actuals locations as a function of the *absolute* distance traveled by the datum:

$$distance\Big(\big(x_p, y_p\big), \big(x_p(A_d), y_p(A_d)\big)\Big), \quad \text{where } A_d = distance\Big(\big(x_d, y_d\big), \big(x_d(t), y_d(t)\big)\Big) \tag{3.2}$$

We can also compute the datum's spatial decay as a function of the *cumulative* distance traveled by the datum, given by the sum of the distances between each of its n space-time positions:

$$distance\Big(\big(x_p, y_p\big), \big(x_p(C_d), y_p(C_d)\big)\Big), \quad \text{where } C_d = \sum_{i=1}^{n} distance\Big(\big(x_d(i), y_d(i)\big), \big(x_d(i-1), y_d(i-1)\big)\Big) \tag{3.3}$$

Figure 3.3 includes two trajectories of different resolutions that attempt to capture the datum's dynamics (the solid gray path). The higher resolution trajectory (shown with solid black arrows) more accurately represents the datum's actual spatiotemporal dynamics and will experience a lower rate of decay between each of its space-time positions; however, the higher resolution comes at a higher storage cost (six space-time positions). On the other hand, the lower resolution trajectory (depicted with dashed black arrows) will experience more significant decay between each of its space-time positions, but consumes half as much storage space.

In an effort to illuminate these tradeoffs, we next present experimental results from a simulated pervasive computing environment of mobile hosts (representing user-carried smartphones, sensing devices, "smart" objects, etc.) and moving phenomena (discrete entities that may be "sensed"). Each mobile host runs an instance of our spatiotemporal metadata middleware and periodically "senses" proximate mobile phenomena, which triggers the creation and storage of representative datums and their associated spatiotemporal provenance. We investigate tradeoffs between the rate of trajectory decay and its additional storage overhead while varying metadata resolution and the mobility of hosts and phenomena.

3.1.4.2 Experimental Setup

The simulated scenario comprises 50 mobile hosts and 375 mobile phenomena moving around a $0.4km^2$ space (roughly the size of the University of Texas campus) for 60 simulated minutes. Hosts and phenomena each follow their own synthetic mobility trace generated using the MobiSim mobility trace generator [110] per the

Levy-walk [140] mobility model[7]. Every five minutes each host "senses" all phenomena that are within 30 meters[8], creating a new digital datum for each that records the phenomenon's observed location at the current time. A "phenomenon" here is simply an abstract placeholder for a moving object. For example, this scenario could represent a mobile social application that periodically senses the presence of nearby friends (e.g., FourSquare Swarm and Facebook's "nearby friends" features), a participatory application where users take pictures of animals or phenomena of interest [16], or a wildlife research application where researchers in the wild periodically scan for nearby tagged animals (e.g., the Starkey Project[9]).

Each host runs an instance of our middleware on top of a Titan graph database backend[10], which implements the Blueprints interface stack (and hence, all of our middleware's constructs). Datums created by a host are stored indefinitely in its graph database instance. As a host moves about the simulated space it updates its collected datums' spatiotemporal metadata per a spatially- or temporally-modulated rule (similar to the rule shown in Listing 3.4). A *spatially*-modulated rule updates datums' spatiotemporal metadata when the host has traveled more than a parameterized physical distance; a *temporally*-modulated rule updates metadata after a parameterized time interval.

In an effort to tease out the tradeoffs between rate of decay and metadata overhead we vary the degree of host and phenomenon mobility as well as the spatial

[7]Levy-walk mobility is considered to closely mimic human mobility.

[8]A sensing range of 30 meters was chosen to represent a typical outdoor Wi-Fi communication range.

[9]`http://www.fs.fed.us/pnw/starkey/index.shtml`

[10]`http://thinkaurelius.github.io/titan/`

Table 3.1: Simulated Independent Variable Values

Variable	Values	Description
Host mobility (m/s) / *Phenomenon mobility* (m/s)	*slow*: 1-5 *medium*: 5-10 *fast*: 10-15	Ranges correspond to the *minSpeed* and *maxSpeed* parameters of the Levy-walk mobility model. *Slow*, *medium*, and *fast* ranges correspond to *foot*, *bicycle*, and *automobile* speeds respectively.
Spatially-modulated update resolutions (meters)	*low*: 50 *medium*: 25 *high*: 10 *very high*: 5	After a datum has traveled *resolution* meters a new space-time position with the current time and its current location will be added to its spatiotemporal trajectory.
Temporally-modulated update resolutions (seconds)	*low*: 600 *medium*: 300 *high*: 60 *very high*: 10	Every *resolution* seconds a new space-time position with the current time and the datums's current location will be added to its spatiotemporal trajectory.

and temporal resolutions of the metadata update rules. The simulated values of these variables are shown in Table 3.1. In all cases, a variable's *medium* value is used when it is considered "fixed."

3.1.4.3 Simulation Results

Spatiotemporal Metadata Overhead: We first investigate the additional overhead induced by our spatiotemporal provenance under varying degrees of host mobility and trajectory update resolutions. Figure 3.4 shows the storage overhead rate (space-time positions per second) as the degree of host mobility is varied and all other variables are fixed. The results illustrated by Figure 3.4 are quite obvious, but are quantified here both as a sanity check and to help guide a pervasive computing application developer in fine-tuning our middleware to meet his needs. We see that higher host mobility has no effect on temporally-modulated spatiotemporal metadata (i.e., trajectories that are updated per a time interval), but is directly proportional to the amount of spatially-modulated metadata produced (i.e., trajectories

that are updated whenever their datum has traveled a certain distance). In other words, the more mobility hosts exhibit, the more often their spatially-modulated update rules will be triggered.

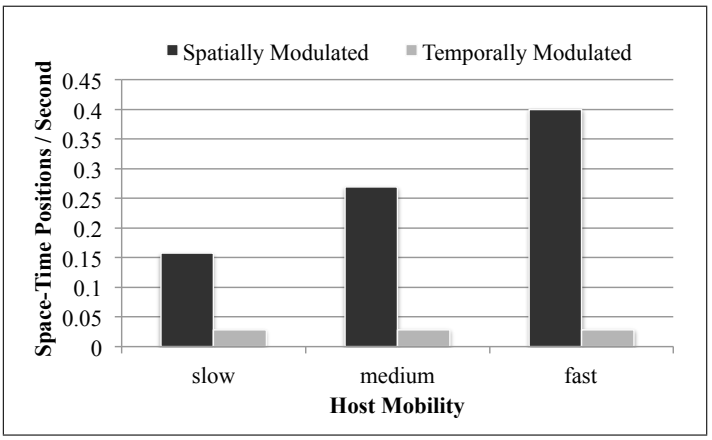


Figure 3.4: Mean overhead of spatiotemporal data provenance as the degree of host mobility varies.

To associate actual byte values with these overhead results, we assume that a single space-time position consumes 128 bytes (two 32-bit floating-point decimals for location and a 64-bit long integer for time). Then, the spatially-modulated metadata at *fast* host mobility generating 0.4 space-time positions per second shown in Figure 3.4 would consume 320 bytes per second or roughly 19 kilobytes per minute.

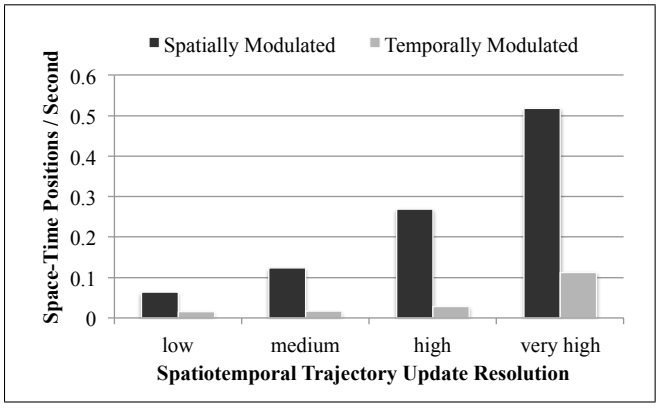


Figure 3.5: Mean overhead of spatiotemporal data provenance as metadata update resolution is varied.

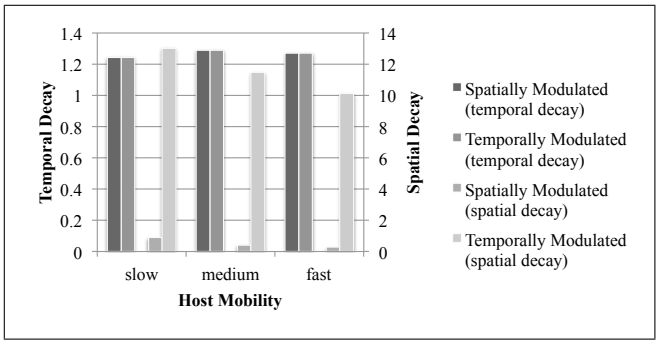
Next, Figure 3.5 shows the effects of varying metadata spatiotemporal update resolution with host and phenomenon mobilities fixed. At least for the spatiotemporal resolution settings used in this case study,

spatially-modulated metadata is much more sensitive to adjustments than temporally-modulated metadata. The takeaway from Figure 3.5 is that spatially-modulated metadata update rules should be used with caution since they can induce significantly more overhead than equivalent temporally-modulated update rules. However, an application may have certain requirements for the spatial resolution and decay of data that are only satisfied by maintaining higher resolutions of spatiotemporal data provenance.

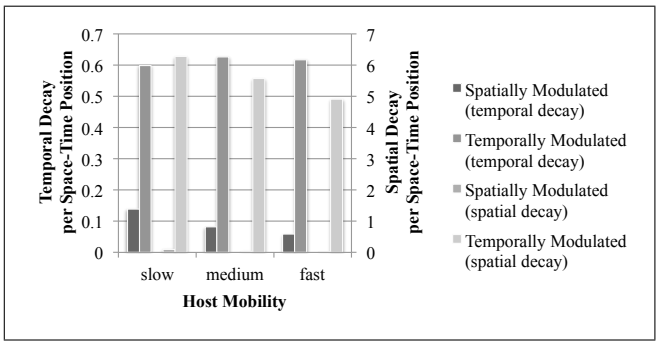
Spatial & Temporal Decay: We next explore how data *decays* in both space and time within our simulated scenario. Figure 3.6 illustrates the effects of host mobility on spatial and temporal decay (with fixed phenomenon mobility and spatiotemporal update resolutions). Here (and in following the following charts), temporal decay captures the mean distance of a datum's representative phenomenon from its initially observed location as a function of time (per equation (3.1)) normalized by trajectory age—i.e., meters of decay per second. Spatial decay captures the mean distance of a datum's phenomenon from its observed location as a function of cumulative trajectory length (per equation (3.3))—i.e., meters of decay per cumulative meter of datum travel. Figure 3.6(a) shows that data's temporal decay is relatively unaffected by host mobility, which is intuitive because host and phenomenon mobility are independent in our simulated scenario. On the other hand, data decays less in space as host mobility increases since hosts' data are able to travel farther relative to phenomena.

Not surprisingly, spatially-modulated metadata results in less spatial decay than temporally- modulated metadata since it generates space-time positions at a

higher rate than its temporally-modulated counterpart (as observed in Figure 3.5). This observation is corroborated by Figure 3.6(b), which shows Figure 3.6(a)'s results normalized by trajectory size (number of space-time positions). This is effectively the rate of decay per space-time position. Figure 3.6(b) quantifies the "bang for the buck" achieved by our metadata update rules. In general, spatially- modulated metadata, though much more costly, exhibits significantly lower spatiotemporal decay than temporally-modulated metadata (about two orders of magnitude less).



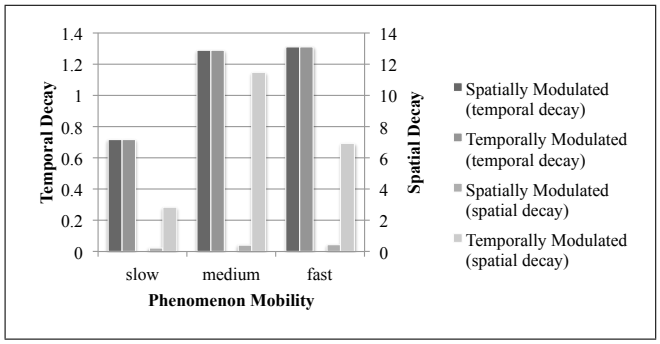
(a)



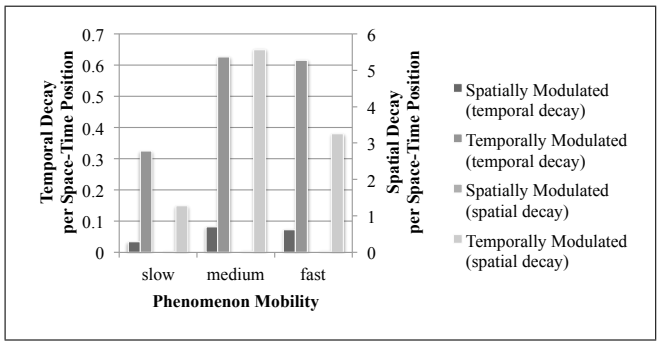
(b)

Figure 3.6: (a) Mean spatial and temporal decay as host mobility is varied. (b) Mean spatial and temporal decay *per space-time position* as host mobility is varied.

Next, we investigate spatiotemporal decay as phenomenon mobility is varied, holding host mobility and spatiotemporal update resolutions fixed (Figure 3.7). As expected, Figure 3.7(a) shows an increase in temporal decay as phenomena become increasingly mobile, which indicates that, in general, phenomena move farther away from their points of observation as time progresses. However, data experiences a



(a)

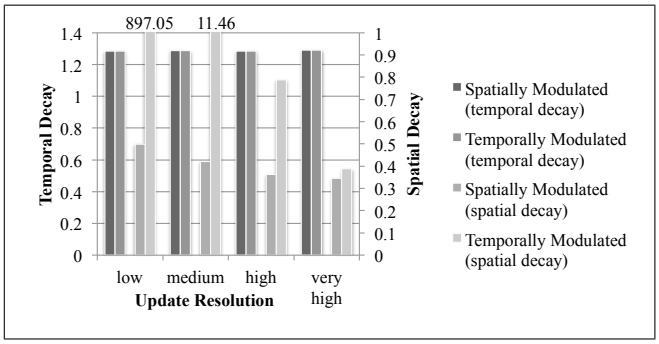


(b)

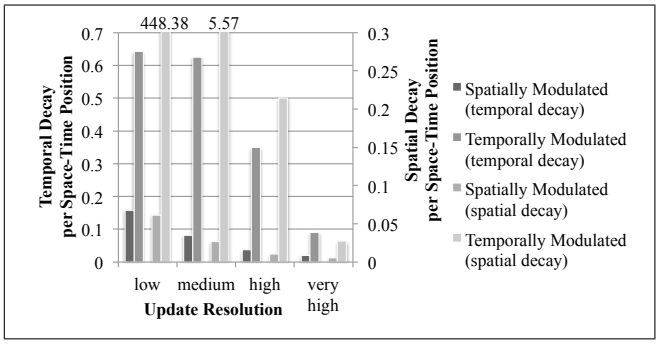
Figure 3.7: (a) Mean spatial and temporal decay as phenomenon mobility is varied. (b) Mean spatial and temporal decay *per space-time position* as phenomenon mobility is varied.

drop in the spatial decay observed by both spatially- and temporally-modulated metadata when phenomenon mobility is *fast*. This is a product of the Levy-walk mobility model in which objects make many short movements in small geographic regions and occasional long "leaps" to new geographic regions. So, for this particular combination of host and phenomenon mobility (or rather their relative difference), as a digital datum undergoes dynamics it may not always be the case that it is decaying in space since its representative phenomenon may just be "hovering" around a particular geographic region. The takeaway here is that higher degrees of phenomena mobility do not strictly necessitate higher resolution metadata to mitigate spatiotemporal decay. This is crucial evidence for pervasive computing application developers. Given the developer knows something about the relative difference between host and phenomenon mobilities, he may be able to satisfy his application's decay requirements with less metadata than intuitively expected.

Finally, we fix host and phenomenon mobility and observe spatiotemporal decay as our metadata update resolutions are varied (Figure 3.8). Something very interesting happens here: at *low* and *medium* update resolutions temporally-modulated metadata spatial decay is literally off the charts in Figure 3.8(a) and (b) and substantially drops as temporal resolution becomes higher. This is simply a tale of caution to pervasive computing application developers—there is clearly a "sweet spot" of temporal resolution for this scenario and tuning temporal update resolution poorly could have drastic impacts on data decay. Interestingly, at *very high* resolution spatially- and temporally- modulated metadata achieve very similar levels of spatial decay (Figure 3.8(a)). However, the spatial and temporal rate of decay (Figure 3.8(b)) is still much higher for temporally-modulated metadata (about 5x higher).



(a)



(b)

Figure 3.8: (a) Mean spatial and temporal decay as metadata spatiotemporal update resolution is varied. (b) Mean spatial and temporal decay *per space-time position* as spatiotemporal update is varied.

The middleware presented in this section provides concrete developer tools

and constructs that simplify the implementation of pervasive computing applications. Rather than embed data-driven logic throughout an application, the middleware's programming interfaces enable a developer to define rules that autonomously react to data-dependent conditions. The case study demonstrates how such rules may be used to govern the maintenance of explicit spatiotemporal data provenance and provides developers with quantifiable insights to guide decisions regarding provenance resolution. In summary, this section focused on development tools that leverage spatiotemporal data provenance, which are useful from an engineering perspective. In the next section, we turn our attention to the direct utility of spatiotemporal provenance from a performance perspective and demonstrate how locally-available provenance can be exploited to enhance distributed protocols that support sharing and querying spatiotemporally-enhanced data.

3.2 Leveraging Spatiotemporal Provenance in Opportunistic Networks

In this section we step down a layer and shift our focus from simplifying application development to enhancing routing protocols that support sharing and querying of spatiotemporally-enhanced data. We demonstrate the practical utility of spatiotemporal data provenance for making local inferences about global network characteristics and remote nodes' knowledge. Employing our *implicit* model of data provenance (Section 2.3) we first benchmark the effectiveness of the model as a tool for estimating characteristics of data objects' propagation patterns using only locally-available spatiotemporal provenance. We conduct performance analyses driven by real-world data sets of human proximity that benchmark the effective-

ness of provenance-based estimation and quantify the degree of distributed shared knowledge supported by each of the implicit model's update strategies under varying network connectivity and dynamics. Motivated by results from these experiments, we next demonstrate how nodes' overlapping views of datas' causal history (i.e., implicit spatiotemporal provenance) may be exploited to make powerful inferences. Through two use cases, we demonstrate the practical utility of our model's constructs for making intelligent routing decisions: first, as a means for making local inferences about other nodes' knowledge and second, for identifying commonly co-located data-data and data-device pairs, which may be used as substitute routing targets. Both use cases are directly applicable to ad hoc routing protocols that support data sharing and querying in pervasive computing environments.

3.2.1 Benchmarking Trajectory Performance

Recall from the model of *implicit* spatiotemporal data provenance introduced in Section 2.3.1, a spatiotemporal trajectory $T(m)$ captures, to an extent, the spatial and temporal history of m. A message's complete spatiotemporal history is given by its global transmission network $\mathcal{J}^*_{\mathcal{G}}(m)$, of which $T(m)$ is a subset. A node u that receives m initially has a local view of the global transmission network $\mathcal{J}^*_{\mathcal{G}_u}(m) = T(m)$ However, u's local view may expand to include more of $\mathcal{J}^*_{\mathcal{G}}(m)$ if more copies of m (via different causal paths) are received. In Section 2.3.1 we presented three trajectory update strategies that each produce trajectories with varying degrees of coverage of their transmission network. Here, we aim to quantify the coverage each update strategy achieves and the impacts of that coverage.

Given that a node u receiving m may not have complete knowledge of m's global transmission network $\mathcal{J}^*_{\mathcal{G}}(m)$, we would like to measure how well structural and temporal characteristics of $\mathcal{J}^*_{\mathcal{G}}(m)$ can be estimated using locally-available $\mathcal{J}^*_{\mathcal{G}_u}(m)$. Specifically, we compare the *receivers per broadcast*, *transmission network depth*, *transmissions per second*, *hops per second*, and *inter-transmission delay* measured on local transmission networks to that of the same metric as measured on the corresponding global transmission network of a given message. From a data-agnostic perspective, these measurable characteristics shed light on network dynamics and connectivity (e.g., density and contact frequency). Taking message content into account, these characteristics reveal *how* particular data propagate through the network. Both views on these metrics (data-agnostic and -dependent) may be useful runtime input for an opportunistic data dissemination protocol. In addition to being indicators of data's spatiotemporal "about-ness," trajectories also describe the distributed state of knowledge about the time-varying state of the network. Two nodes u and v, for example, may respectively receive trajectories $T(m)$ and $T(m')$ where $T(m) \cap T(m') \neq \emptyset$ and may therefore be able to make similar inferences (possibly that they each have received m or m'). Our second series of benchmarks aims to quantify the degree of data-agnostic knowledge convergence supported by each of our update strategies.

3.2.1.1 Experimental Setup

We target dynamic operating environments densely populated with wirelessly communicating mobile devices. Events that draw large spontaneous crowds

of people (e.g., music festivals, conferences, parades, etc.) represent an immediate scenario where human users typically have hyper-localized information needs. However, cellular infrastructures often become saturated in these scenarios due to the large burst of competition for network resources. In such spontaneously crowded environments device-to-device communications may connect users and their devices to digital data that exists within users' surroundings.

To better understand how our model and its constructs perform in networks of wireless devices carried by human users we employ two real world contact trace data sets of human proximity collected by the SocioPatterns project[11]. The SocioPatterns sensing platform captures face-to-face human proximity through exchange of radio packets between RFID devices embedded in badges; a "contact" occurs when two people are at close range (1-1.5 m) and facing each other. We use proximity data collected from two SocioPatterns deployments, both of which are described in detail and analyzed in [73] and [121]. The first deployment took place at the Science Gallery (SG) in Dublin[12] and lasted almost three months (we use data from three days). The second deployment tracked human proximity of about 100 volunteers at the Hypertext 2009 (HT09) conference in Turin[13], which lasted about three days. The details of each of these data sets is listed in Table 4.1. Each of these deployments vary in density and represent very different kinds of human behavior. In a museum visitors spend a limited time on-site and generally follow a pre-defined path. At a conference, on the other hand, attendees stay on-site for

[11] http://www.sociopatterns.org/
[12] http://dublin.sciencegallery.com/infectious/
[13] http://www.ht2009.org/

most of the day and move at will between different parts of the conference venue. We chose these data sets because of their density, scale, and focus on human interactions. Ultimately, the networks we target are by no means limited to face-to-face proximity. Wireless protocols like Bluetooth, Wi-Fi, and LTE Direct enable device-to-device communications across ranges of tens to hundreds of meters. However, the fine-grained level of human behavior and interactions captured by the SocioPatterns deployments, which is the key driver of our evaluations here, would be present in a network of human-carried devices connected by any wireless protocol.

Deployment	Date	Nodes	Contacts	Duration (h)
HT09	*June 29*	100	6,922	15.96
	June 30	102	7,134	17.87
	July 1	97	6,792	10.81
SG	*May 19*	99	3,421	7.64
	May 20	188	8,416	7.98
	July 14	279	12,330	8.03

Table 3.2: SocioPatterns data set deployment details.

We simulate device-to-device propagation of messages and their spatiotemporal trajectories following an epidemic routing protocol. Given a contact trace, we assume that each node in the trace represents a wireless device carried by a human user and require that each node maintain its own local TVG instance. For each simulation, we select a random subset of the nodes to be *seeds*, which periodically inject data into the network. Once a seed becomes active (i.e., after it appears in the network for the first time) every 60 seconds it transmits both a message and the message's spatiotemporal history to all other nodes the seed is in contact with at transmit time (i.e., the message is flooded). Prior to transmitting the message, the message's spatiotemporal history is updated per a trajectory update strategy.

For simplicity, we assume the latency of all transmissions to be one second ($\zeta = 1$ s)[14]. Any node that receives a message first stores the message's contents and all of its attached contextual history in the local TVG then performs the same steps as the seed: the receiver updates the spatiotemporal history according to the update strategy, then propagates the message and its updated history to all nodes the receiver is currently in contact with. Intermediate receivers do not buffer messages; if a receiving node does not have any active contacts at receive time, then it does not propagate a received message. We do not impose any limit on the number of hops a message can take – a message's device-to-device diffusion continues until no receivers having contacts at reception time remain.

3.2.1.2 Trajectories as an Estimation Tool

In a distributed deployment, it would be impractical to obtain a message's complete contextual history (i.e., its global transmission network). Therefore, our first set of benchmark measurements investigates how well nodes' partial (local) knowledge of messages' complete contextual history functions as a tool for estimating global spreading dynamics and network characteristics. We use five spatiotemporal metrics to measure messages' propagation characteristics: *receivers per broadcast*, *transmission network depth*, *transmissions per second*, *hops per second*, and *inter-transmission delay*. When a message m is received by a node u at time t, for each of our five metrics we compute the error between the metric as measured

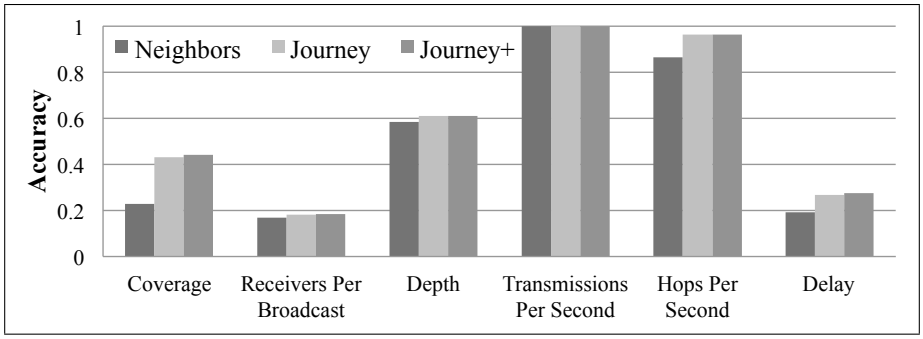
[14]In a real world deployment, transmission time depends on payload size, the wireless protocol, available network bandwidth, etc. These realities are clearly non-negligible; however, in this work our focus is on the utility of spatiotemporal trajectories at the application layer. We leave investigations of the impacts of trajectory and data size as future work.

on $\mathcal{J}^*_{\mathcal{G}_u}(m,t)$ (u's local view of m's transmission network at t) and $\mathcal{J}^*_{\mathcal{G}}(m,t)$ (m's global transmission network at t). We also measure how well u's local view of m's transmission network covers the global transmission network using our Jaccard spatiotemporal similarity operator from Section 2.3.2:

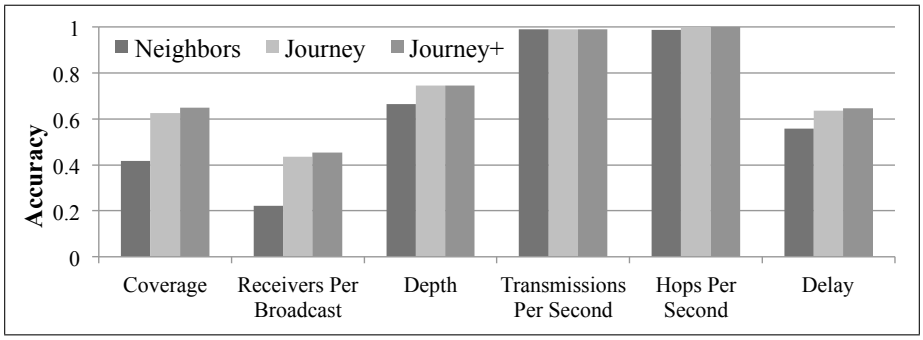
$$coverage(m,u,t) = \frac{|\mathcal{J}^*_{\mathcal{G}_u}(m,t) \cap \mathcal{J}^*_{\mathcal{G}}(m,t)|}{|\mathcal{J}^*_{\mathcal{G}_u}(m,t) \cup \mathcal{J}^*_{\mathcal{G}}(m,t)|}$$

We first investigate how well each of our update strategies supports local estimation of global network characteristics across deployments. Figure 3.9 shows the mean per-message average of the coverage and estimation accuracy ($1 - error$) achieved by each of the update strategies measured on one day of each SocioPatterns deployment. We use the mean per-message averages to account for varying transmission network size. In both simulations, half of the nodes act as data-generating seeds.

Not surprisingly, the *Neighbors* strategy achieves the poorest coverage of the global transmission network since it only captures 1-hop contextual history. The *Journey* and *Journey+* strategies achieve much better coverage, and typically yield more accurate estimations, since they both capture an entire causal chain of propagation from seed to destination. Interestingly, spatiotemporal trajectories generally produced more accurate estimations within the SG deployment (Figure 3.9(b)) than within the HT09 deployment (Figure 3.9(a)). We can attribute this performance difference to the nature of these two deployments. According to the temporal investigations of [73], nodes in the SG deployment (museum visitors) typically move through the venue within 35 minutes and rarely interact with other visitors entering



(a) HT09 June 30



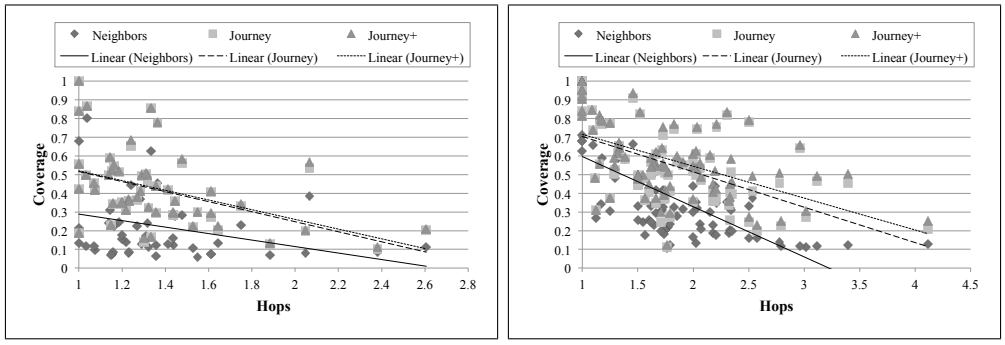
(b) SG May 20

Figure 3.9: Mean per-message average transmission network coverage and estimation accuracy of spatiotemporal metrics measured on nodes' partial transmission networks.

the venue more than an hour after them. Moreover, SG visitors generally spend *more* time in contact with *few* people. Message propagation in the SG deployment is therefore restricted to groups of nodes that enter the venue around the same time, which gives nodes in these cohesive groups more chances to acquire a message's spatiotemporal history. Contact activity in the HT09 deployment, on the other hand, is extremely bursty [121] with spikes of activity occurring during the conference lunch and coffee breaks. HT09 attendees do not follow a pre-defined path, but rather move around the conference venue freely, spending *less* time interacting with *more* people [73]. This diversity of node interaction means that messages do not have the opportunity to propagate many hops in the HT09 deployment and therefore pos-

sess less contextual history than in the SG deployment. Next, we investigate these differences closer.

Figure 3.10 plots the coverage achieved by each update strategy as a function of the topological distance a message has travelled. Each point represents the average coverage of the global transmission network of a message's attached spatiotemporal history as a function of the average number of hops from seed to receiver aggregated across all receptions of the message. These figures corroborate our earlier conclusions. Because of the contact diversity and burstiness messages in the HT09 deployment generally don't propagate more than one or two hops from a seed (Figure 3.10(a)) and on average achieve a lower coverage per-hop of the global transmission network than in the SG deployment (Figure 3.10(b)) where nodes travel along similar spatiotemporal paths in cohesive groups. In this paper we define space in a relative way. A message's spatial context is based on a perception of the surroundings—if that perception does not change, then the message's "space" does not change. The spatiotemporal cohesiveness of the SG network means



(a) HT09 June 30 (b) SG May 20

Figure 3.10: Local transmission network coverage of messages' global transmission networks as a function of topological distance from the seed.

that, while nodes may be moving through physical space, their *context* changes at a much slower rate, which is why we observe better estimation accuracy in SG than in HG09. Next we explore how network dynamics and density within one of these deployments affect nodes' second hand perception of their context.

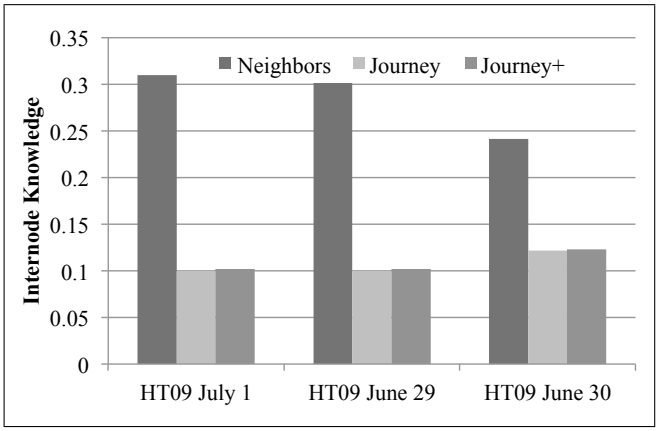
3.2.1.3 Knowledge Convergence

Completely ignoring a message's content, its spatiotemporal trajectory provides a historical window into the time-varying topology of the network along the message's path of propagation. Depending on the trajectory update strategy in use and the amount of shared data being generated, nodes may possess varying degrees of "overlapping" historical knowledge about the structure of the network. Our second set of benchmark measurements explores the degree to which nodes' knowledge of the network's past states *converges* using each of our update strategies under varying network conditions and volumes of shared data.

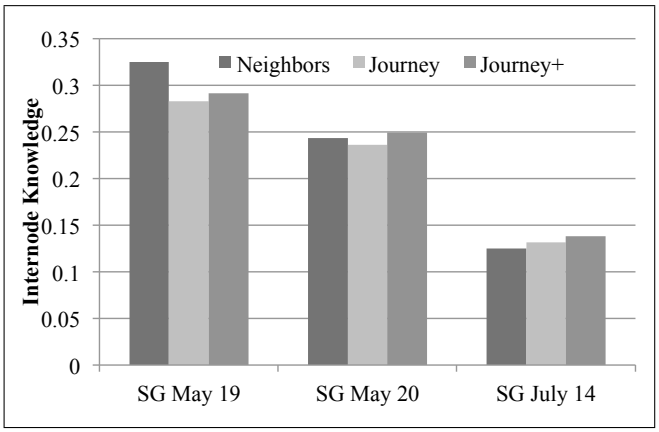
We investigate distributed knowledge convergence within windows of time across our simulations. For a given simulation, we separate its duration into consecutive non-overlapping five minute time windows. Within each time window, we compare the local knowledge of the network's topological states possessed by each node that was "active" (i.e., sent or received a message) in the window. A node u's local knowledge in a time window $[t, t')$ comprises all of the edges it learned about (either directly as a contact or indirectly through a spatiotemporal trajectory) in $[t, t')$. We quantify knowledge convergence in a time window by measuring *inter-node knowledge*, which captures the average similarity of local knowledge measured

between every unique pair of active nodes in the window.

Figure 3.11 illustrates the degrees of internode knowledge convergence measured across each day of the two SocioPatterns deployments. From left to right, the days are ordered in increasing order of number of nodes and contacts. The size and dynamics of the network do not vary drastically between each day of the HT09 deployment (Figure 3.11(a)). On the other hand, the SG deployment (Figure 3.11(b)) exhibits an increase of about 100 nodes and 4k contacts each successive day. We observe an inverse relationship between internode knowledge and network size. This is intuitive; the more topological knowledge there is the overlap of it there is between nodes. Interestingly, within the SG deployment the



(a) HT09



(b) SG

Figure 3.11: Knowledge convergence under varying network size and dynamics. From left to right, deployment days are ordered in increasing order of number of nodes and contacts (refer to Table 4.1).

Neighbors update strategy yields the highest internode knowledge at low degrees of network size and dynamics (May 19), but in the largest network the *Journey+* strategy prevails (July 14). This occurs because at low network densities where messages propagate few hops the *Neighbors* strategy provides the most complete contextual snapshot of the network state in a discrete window of time. In the HT09 deployment (Figure 3.11(a)), where the network is as dense but about twice as active as SG May 19, this effect is amplified. Here the *Neighbors* strategy yields much higher degrees of internode knowledge overlap than the other two strategies. However at higher densities (e.g., SG May 20 and July 14) messages are able to propagate farther and the *Journey* and *Journey+* strategies are able to capture more complete contextual snapshots than *Neighbors*.

These benchmark evaluations illustrate the extent to which each trajectory update strategy captures a complete transmission network and how network size, density, and dynamics affect messages' contextual history, which, from a routing perspective, are useful for making distributed decisions. Given that nodes' knowledge of data's causal propagation overlaps to an extent in both SocioPattern deployments, we wish to exploit this implicitly shared knowledge to make data-dependent and data-agnostic inferences. Next, we concretely demonstrate how our model and its constructs may be used to improve the performance of opportunistic data dissemination by making inference-based decisions.

3.2.2 Use Cases

This section demonstrates how spatiotemporal trajectories can be leveraged to draw concrete higher level inferences at a local level through two practical use cases. The second hand information carried by a trajectory directly indicates which mobile nodes know (have received) a piece of data. Our first use case illustrates how trajectories may be "stitched" together in a data-agnostic fashion to make indirect data-dependent inferences about remote nodes' knowledge with respect to particular data. We demonstrate how knowledge inferences may be used to eliminate transmission of redundant (already received) information within an opportunistic data dissemination protocol. In the networks we target, the data generated and exchanged between devices is highly contextual—it is a product of the place and time in which it exists. Human mobility is no exception to this assumption; it is highly spatiotemporally correlated. Our second use case showcases how such spatiotemporal regularities may be exploited to direct data's propagation. Given a piece of target data, we first demonstrate how trajectories may be used to detect other types of data and nodes that are commonly co-located with the target data. In the event that knowledge about the target data is lacking, frequently co-located data or nodes may be used as logical routing substitutes. We demonstrate the effectiveness of such routing substitutes for carrying out targeted data dissemination.

3.2.2.1 Knowledge Inference

In addition to indicating where and when data is spatiotemporally "about," trajectories also provide a view into the network-wide state of knowledge of partic-

ular data. Concretely, a trajectory explicitly details when and from whom a piece of information was adopted along the causal chain of its device-to-device propagation. Given an application knows something about the underlying routing protocol responsible for disseminating data (e.g., its probability of propagation), a partially-complete trajectory can be inferentially supplemented with second hand knowledge from other data's trajectories. For example, an epidemic protocol may only propagate messages to a subset of a node's neighbors, modulo some probabilistic parameter. Alternatively, the wireless channel may be unreliable and drop a predictable proportion of transmitted packets.

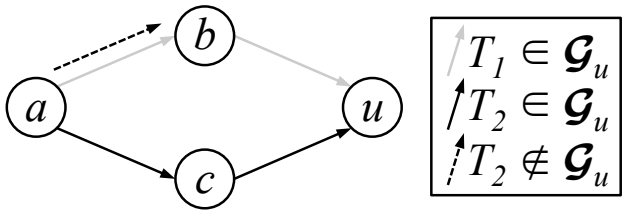


Figure 3.12: Knowledge inference. Solid gray and black arrows respectively indicate trajectory T_1 and T_2's elements that are known by node u (i.e., in u's local TVG). The dashed black arrow represents an element of T_2 that is *not* known by u (e.g., due to a non-flooding routing protocol, lossy wireless medium, etc.), but which u can probabilistically infer by "stitching" together T_1 and T_2.

As an example, consider the simplified scenario shown in Figure 3.12. The figure shows two trajectories, T_1 and T_2 (shorthand for $T(m_1)$ and $T(m_2)$) that are received via different causal paths at node u. Ignoring the temporal domain, from u's perspective T_1 (solid gray arrows) traverses the upper walk *(a, b, u)*, and T_2 (solid black arrows) traverses the lower walk *(a, c, u)*. Node b receives both m_1 and m_2 from a. However, b only propagates m_1 to u, due to a global propagation probability $P_p < 1$ for example, so the walk *(a, b, u)* is not included in T_1.

Assuming node u knows P_p it can probabilistically infer the likelihood that

b is in trajectory T_2 by supplementing with spatiotemporal information from T_1. Since node a is in the intersection of T_1 and T_2, u knows that both m_1 and m_2 "passed through" a. If m_2 arrived at a before m_1 (or if a has no predecessor in T_1 or T_2), which u can directly infer by comparing the temporal entries of T_1 and T_2 containing a, then u knows there was a chance that a propagated m_2 to b. Under these circumstances u can derive the likelihood that a propagated m_2 to b by inspecting $\mathcal{J}^*_{(a,b)}$, the set of journeys from a to b. From u's perspective $P(b \in T_2)$, the probability that b received m_2, is equal to the probabilistic union of P_p times the topological length each of $\mathcal{J}_{(a,b)} \in \mathcal{J}^*_{(a,b)}$ such that each $\mathcal{J}_{(a,b)} \in \mathcal{G}_u$ (i.e., the journey exists in u's local TVG). For the example shown in Figure 3.12 there is only a single journey from a to b known by u (via T_1), so $P(b \in T_2) = P_p$. The general case of this claim is formalized in Claim 3.2.1.

Claim 3.2.1 *Let u be a node, $\mathcal{J}^*_{\mathcal{G}_u}(m_1)$ and $\mathcal{J}^*_{\mathcal{G}_u}(m_2)$ be the transmission networks of messages m_1 and m_2, respectively, and the global probability of message propagation $P_p < 1$. If there exists a node n such that $n \in \mathcal{J}^*_{\mathcal{G}_u}(m_1) \cap \mathcal{J}^*_{\mathcal{G}_u}(m_2)$ and there exists a node n' such that $n' \in \mathcal{J}^*_{\mathcal{G}_u}(m_1) \setminus \mathcal{J}^*_{\mathcal{G}_u}(m_2)$ and $arrival_n(\mathcal{J}^*_{\mathcal{G}_u}(m_2)) \leq arrival_n(\mathcal{J}^*_{\mathcal{G}_u}(m_1))$ or predecesor(n) in either $\mathcal{J}^*_{\mathcal{G}_u}(m_1)$ or $\mathcal{J}^*_{\mathcal{G}_u}(m_2)$ is $\perp$, then $P(n' \in \mathcal{J}^*_{\mathcal{G}_u}(m_2)) = \cup\{P_p \times |\mathcal{J}_{\mathcal{G}_u}(n, n')|_h : departure_n(\mathcal{J}_{\mathcal{G}_u}(n, n')) \geq arrival_n(\mathcal{J}^*_{\mathcal{G}_u}(m_2))\}$.*

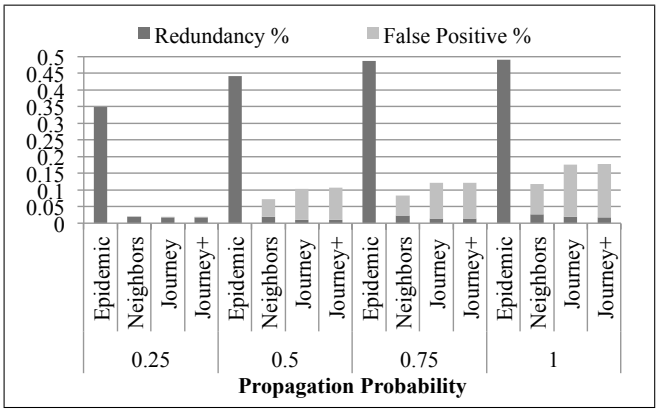
Here, we show how such probabilistically-guided local inferences of remote nodes' knowledge can be exploited to eliminate redundant overhead in an opportunistic data dissemination protocol. We run our device-to-device message propagation simulations like before, but require that before transmitting a message m the

source node u compute its confidence c that the destination v has already received m (using local knowledge only). If c exceeds a confidence threshold c_{thresh}, then u infers v has already received m and will *not* transmit m to v. Otherwise, u transmits m to v like normal and m's propagation continues at v. We measure the proportion of redundant transmissions ($\frac{|redundant|}{|total|}$) produced by an epidemic protocol and by knowledge inference filtering supported by each update strategy. Additionally, for knowledge inference, we measure the achieved false positive rate (i.e., the proportion of incorrectly eliminated messages, $\frac{|incorrectly\ eliminated|}{|total|}$).

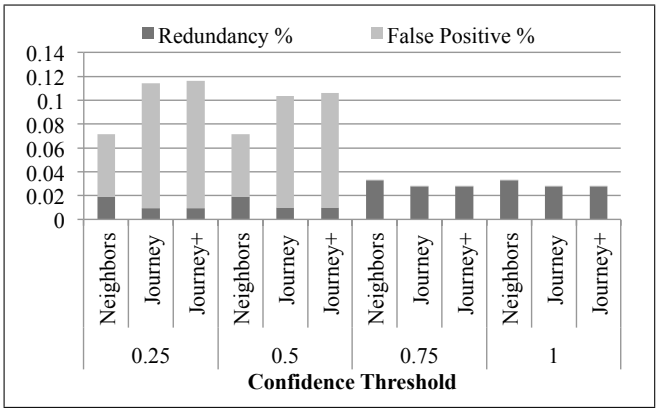
We investigate the reduction of routing redundancy under varying global propagation probability (P_p) and knowledge inference confidence threshold (c_{thresh}). Figure 3.13(a) illustrates the impact of varying P_p between 0.25 and 1 while keeping c_{thresh} fixed at 0.5. Across the board, knowledge inference supported by each of our update strategies eliminates on average 96% of redundant transmissions relative to the epidemic protocol. However, as P_p increases, the false positive rate (the proportion of non-redundant transmissions eliminated) also increases, which means that our c_{thresh} may be too low resulting in overly-aggressive inference. This could be a problem for opportunistic network applications that require high delivery fidelity (e.g., because messages need to be delivered to as many nodes as possible). However, for applications that can stand to sacrifice some portion of message deliveries or that operate in networks with limited bandwidth, a lower c_{thresh} setting could be a huge boon to performance.

Figure 3.13(b) investigates the effects of varying c_{thresh} with P_p fixed at 0.5 (i.e., at each hop a message is only propagated to half of the proximate receivers).

Though not shown, since P_p is fixed the un-assisted epidemic protocol's redundancy is the same (44%) for each c_{thresh} setting. As suspected, increasing c_{thresh} nearly eliminates false positives altogether, but at the expense of some additional redundancy when $c_{thresh} > 0.5$. Not much variation exists between knowledge inference supported by the three trajectory update strategies. However, the drastic difference in false positive rates between $c_{thresh} \leq 0.5$ and $c_{thresh} > 0.5$ stands out. In fact, though not shown, this jump in false positives occurs precisely when c_{thresh} exceeds P_p. This is an incredibly important feature of our knowledge inference mechanism – given that an application knows the global P_p it can pick an appropriate c_{thresh} depending on how many false positives the application can tolerate.



(a) Percentage routing redundancy under varying propagation probability settings. The confidence threshold (c_{thresh}) is set at 0.5.



(b) Percentage routing redundancy under varying confidence threshold settings. The propagation probability (P_p) is set at 0.5. Epidemic redundancy is 44%.

Figure 3.13: Reduction of routing redundancies in opportunistic data dissemination using knowledge inference in HT09 June 30. These same trends are present in the SG deployment.

The next and final use case extends our knowledge inference routing assistance and demonstrates how local contextual history can be used to detect data that frequently propagate in a similar spatiotemporal fashion, which can function as logical substitutes in a targeted data dissemination application.

3.2.2.2 Routing Substitute Inference

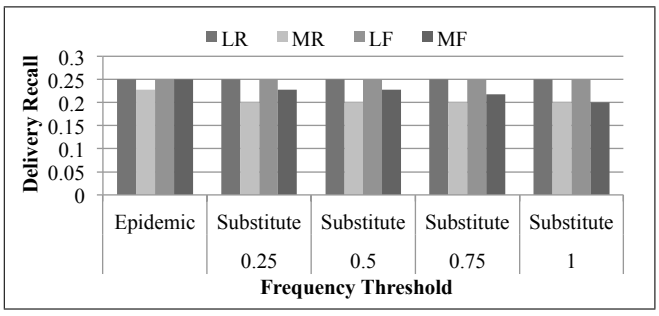
Under the assumption that data generated in similar spatiotemporal contexts will propagate along similar trajectories, we can detect data that are commonly co-located by measuring the degree to which their locally-available trajectories overlap. We refer to such data as *data substitutes* for reasons that will become clear in the following sentences. Similarly, many mobile social network routing mechanisms [80] leverage the heuristic that socially-related people tend to be regularly co-located [27]. Either a byproduct of regular co-location or of user interest, particular data may commonly occur at particular nodes. In a similar fashion to detecting frequently co-located types of data, we can use locally-available contextual history to detect data that commonly occur at particular mobile nodes (*node substitutes*).

In a distributed environment where global knowledge is not available, predictable regularities like these are extremely valuable. Here, we demonstrate how such regularities may logically extend an opportunistic dissemination protocol by functioning as substitutable routing targets. A targeted data diffusion or query protocol may wish to deliver a message m to nodes possessing a particular type of target data d_{target}. For example, at the music festival a mobile beverage vendor may wish to deliver a digital coupon to attendees who have purchased salty snacks. Con-

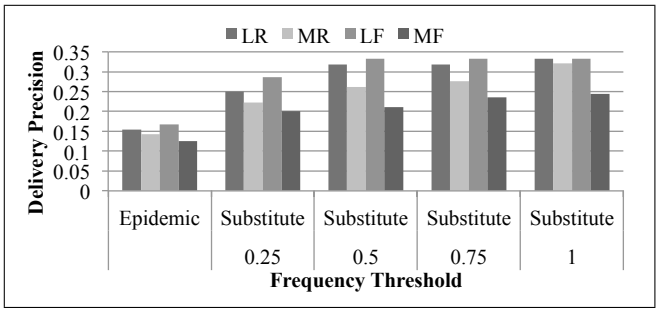
versely, a thirsty attendee may want to query for digital data about nearby vendors selling cold drinks. Before propagating m, we can compute data and nodes that are frequently co-located with d_{target} (i.e., its *data* and *node substitutes*, respectively). During m's diffusion, if m reaches a node where sufficient knowledge about d_{target} is lacking, we can instead use one or more of its substitutes to direct m's continued propagation.

Our final use case demonstrates the effectiveness of routing substitutes for carrying out targeted data diffusion. We use four target data selection policies: most and least *recently* learned (M/LR) and most and least *frequently* learned (M/LF). The first two selection policies (recentness) target data that are different spatiotemporal distances from the diffusing node; the last two selection policies (frequency) target data that differ in availability. We simulate targeted data diffusion using a greedy protocol that uses *only* substitutes to direct propagation. Initially, a seed is responsible for choosing the target data d_{target} per one of the selection policies. The goal is to deliver a message m to as many nodes possessing d_{target} as possible while minimizing the number of transmissions. Given d_{target}, a source node u will compute d_{target}'s data D_{sub} and node N_{sub} substitutes and their co-location frequencies from available local knowledge at diffusion time t. The node u will then propagate m using only the substitutes with co-location frequencies exceeding a threshold f_{thresh} as routing targets ($D'_{sub} \subseteq D_{sub}$ and $N'_{sub} \subseteq N_{sub}$). When data substitutes are being used, a source node u will only transmit m to a destination node v if u's confidence that v has received at least one $d_{sub} \in D'_{sub}$ exceeds a confidence threshold c_{thresh} (confidence is computed using knowledge inference). Likewise, for node substitutes,

u will only transmit m to v if u's confidence that v has received any message from at least one $n_{sub} \in N'_{sub}$ exceeds c_{thresh}.



(a) Delivery "recall" measures the proportion of deliveries made to nodes possessing d_{target} out of all nodes possessing d_{target}.



(b) Delivery "precision" captures the proportion of deliveries to nodes possessing d_{target} out of all diffusion deliveries made.

Figure 3.14: Mean per-message average delivery "recall" and "precision" for targeted data diffusion using strictly *data substitutes* supported by the *Journey+* update strategy within the HT09 June 30 deployment.

We simulate our greedy substitute-guided data diffusion alongside an unguided epidemic protocol with P_p and c_{thresh} both fixed at 0.5 and report results for *data substitutes* supported by the *Journey+* update strategy on a single day of the HT09 deployment [15]. To quantify routing performance we compute two metrics: delivery *recall* measures the proportion of nodes possessing d_{target} that successfully received m; delivery *precision* captures the proportion of nodes possessing d_{target} that received m out of *all* the nodes that received m. We vary f_{thresh} from 0.25 to 1, which is interpreted as a percentile threshold. For example, using *data*

[15] Performance results between *data* and *node substitutes* differed by no more than 5%. Trends were similar between all three trajectory update strategies and were consistent between the HT09 and the SG deployments.

substitutes and an $f_{thresh} = 0.5$, only the top 50% most commonly co-located data would be used as routing substitutes for d_{target}. An $f_{thresh} = 1$ is equivalent to our previous knowledge inference use case (i.e., no substitutes). Figure 3.14 shows our results. Again we report the mean per-message average to weight our computed metrics by transmission network size.

Figure 3.14(a) illustrates the delivery recall achieved under varying settings of f_{thresh}. Here, the un-guided epidemic protocol acts as a baseline, which is less than 1 simply because not all target nodes (i.e., nodes possessing d_{target}) are reachable. Impressively, substitutes achieve nearly the same delivery recall as the epidemic protocol, which means that using strictly substitutes our greedy protocol is able to deliver a message to almost as many target nodes as are physically possible. Delivery recall is unaffected by f_{thresh} under all d_{target} selection policies except for the *most frequent* policy, which decreases as f_{thresh} increases. This means that, for these settings, regardless of the *quality* of substitutes used the same proportion of the target nodes is reached.

We next examine the delivery precision achieved by substitute-guided routing (Figure 3.14(b)). Routing precision captures the overhead required to deliver a message to a set of target nodes (i.e., it expresses the proportion of successful target deliveries out of all deliveries). Under all values of f_{thresh}, substitute-guided routing achieves better precision than un-guided epidemic routing, meaning substitute-guided routing is more accurate per-transmission than epidemic routing. This is a byproduct of extending our knowledge inference mechanism, which we demonstrated as being very effective in these deployments at "guessing" if an encountered node

has received a piece of data or not. We observe that delivery precision increases with f_{thresh}, which is intuitive. When inferential decisions are based on higher quality substitutes, more accurate inferences can be made.

Applications in emerging pervasive computing spaces, for example the crowded music festival scenario used throughout this section, require practical support for sharing and accessing hyper-localized data. These two fundamental tasks will play huge roles in how users interact with their digital landscapes as heavily networked environments become ubiquitous. This section presented experimental results that showcased the practical utility of spatiotemporal data provenance for improving the performance of distributed routing protocols that support data dissemination and querying in pervasive computing spaces. In the next section, we focus entirely on the task of *searching* for data in dynamic and opportunistically-connected environments. We introduce a formal conceptual model that directly employs the epidemic-style protocols studied in this section and use that model to explore the effects of spatial and temporal correlations within searched data.

3.3 Conceptual Model of Search in Pervasive Computing Environments

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 emerging pervasive computing environments. Visions of the Internet of Things, for example, 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.

A key challenge to realizing this 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 such 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 design of 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 [152]. Enabling expressive search over dynamic data in the here and now also requires understanding and efficiently collecting and representing the *context* of that data in a pervasive computing 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.

As a starting point to address these needs, we have previously introduced the Gander conceptual model [101] 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 section, we first review the Gander conceptual model, which is essential to a rigorous understanding of the quality with which protocols resolve users' searches, and use it to evaluate the impacts of spatial and temporal correlations within shared data on Gander's distributed search mechanisms.

3.3.1 A Model of Queries in Pervasive Computing Spaces

We assume that hosts (e.g., mobile devices carried by users or smart objects embedded in the environment) issue queries that are evaluated using information provided by other hosts. A *datum* provides information about the here and now (e.g., a measure of some condition of the environment) and is associated with metadata that describes its *context* (e.g., the device(s) that generated it, the location, a timestamp, or even the data's volatility or freshness). We have formalized the Gander query model previously [101]; 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 $G_h : D \to \Phi$; D contains all datums in the entire pervasive computing space, Φ is the domain of relevance, and h is the host issuing the query. A *datum* is a pair, (ν, d); ν is the data value, and d is the value's metadata. The metadata captures the *context* of the datum (e.g., its relationship to the space and potentially to other datums that inhabit that space). Though not an explicit requirement of our model, d could be a spatiotemporal trajectory capturing the datum's spatiotemporal provenance per one of the metadata models introduced in Chapter 2.

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 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 metadata, 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 distributed, multihop

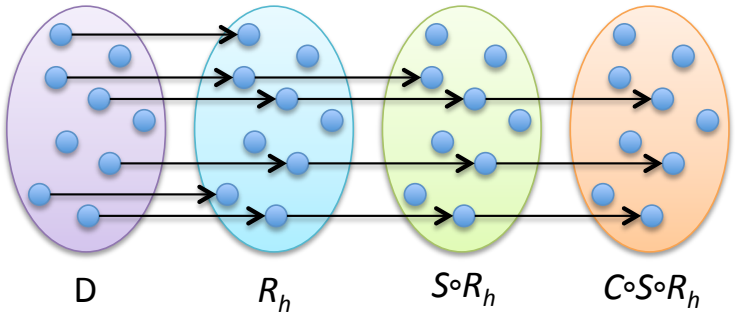


Figure 3.15: Query processing functions

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 host h to send a query to some other host h' *and* receive a response [136]; $\mathcal{R}$ 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 *metadata* (d).

These functions filter D to the subset of reachable datums that satisfy the search string and constraints; Figure 3.15 shows the composition, $\mathrm{QP_h} = \mathcal{C} \circ \mathcal{S} \circ \mathcal{R}_\mathrm{h}$, assuming a snapshot of all available datums.

Relevance. A *relevance metric*, $\mathcal{M}_i : D \rightarrow \phi_i$ 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[16]. 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., $m : \Phi \rightarrow \mathbb{R}$) using different weights for different metrics.

3.3.2 A Model of Data in Pervasive Computing Spaces

From a query's perspective, D is a *global virtual data structure* (GVDS) [130]; 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 [99], which allow lightweight, highly localized pieces of infrastructure to support pervasive computing applications in the here and now.

[16]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.

Gander assumes that each host (both peers and local cloudlet storage providers) implements a *local tuple space* containing semi-structured data [8]. A datum's ν is a tuple that consists of an unordered set of name/value pairs. Metadata, 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.

3.3.3 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. 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 hosts in the space, including both peer devices and cloudlet-based storage locations. Gander can use the query's contents to direct how and to which other hosts a query is distributed; ultimately the goal is to efficiently collect and present only data that is most relevant.

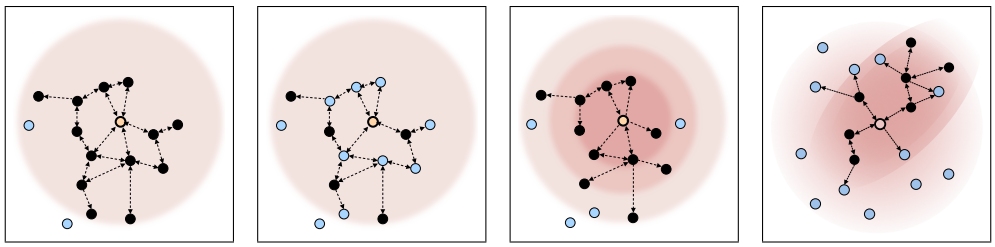


Figure 3.16: Query processing styles and sampling. Dashed lines are sent messages. Darkened hosts respond to a given query. (a) Flooding. Every host in a given range (3 hops) retransmits the query; the target area is the shaded region. (b) Random. A receiving host responds to the query with a given probability; a high quality search evenly samples the shaded space. (c) Probabilistic. Every host 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 host 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.

Gander exploits the relationship between existing 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 host (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 styles of sampling, which trade quality for cost, measured in terms of latency of search processing and network overhead. Figure 3.16 shows the styles and their relationships to spatial sampling.

Flooding. These protocols attempt to reach every host 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 (Figure 3.16(a)). Flooding attempts high *coverage* and reflects the actual evenness (or unevenness) of the hosts' *distribution*.

Random. Random sampling protocols propagate queries similarly to flooding but reduce responses. In random sampling (Figure 3.16(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 (Figure 3.16(c)), each host that receives a query probabilistically forwards it to peers; this reduces the overhead, but hosts closer to the issuer are more likely to receive queries [135]. 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 [30].

Greedy Gossip. In greedy gossip (Figure 3.16(d)), hosts 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 [177], 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.

Prior to this work, we have studied the ability of Gander's spatiotemporal sampling methods (i.e., the protocols depicted in Figure 3.16) to support search of pervasive computing spaces [101]. 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 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 the remainder of this section, we continue the thread of evaluating

through simulation, using 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.

3.3.4 Effects of Spatial & Temporal Correlations on Pervasive Search

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 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) [74], 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:

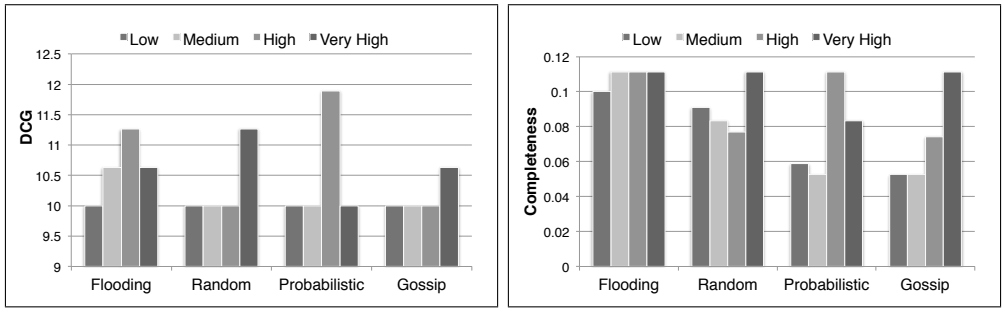
$$completeness = \frac{|G_h \cap Q|}{|G_h \cup Q|},$$

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 mobile 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. We use the MobiSim mobility framework [110] to drive simulated users' movement per Levy-walk [140] mobility and populate our simulated environment with synthetically generated data [75], 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., 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 vary a parameter (β) in the synthetic data generation tool we use [75] to produce four levels of spatially correlated data—low ($\beta = 0.33$), medium ($\beta = 0.18$), high ($\beta = 0.08$), and very high ($\beta = 0.01$)[17]—and run one simulation per setting (which corresponds to approximately 6000 unique queries). The variations in search results' DCG and completeness are shown in Figure 3.17(a,b).



(a) DCG. (b) Completeness.

Figure 3.17: The effects of spatial correlations on query protocol performance.

In general, each processing style achieves better DCG as data becomes more spatially correlated (Figure 3.17(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 cor-

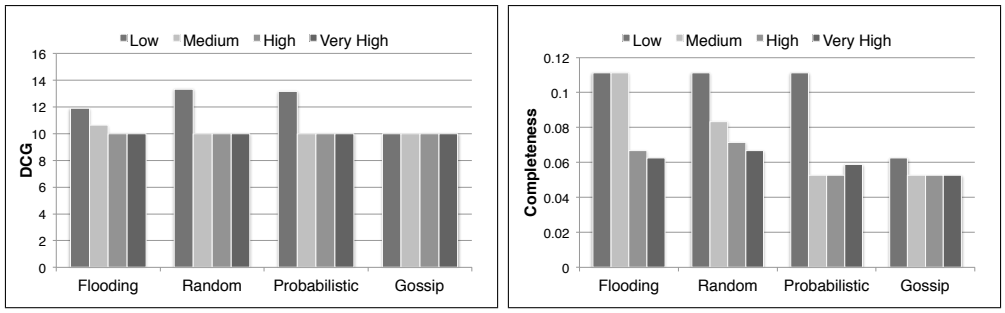
[17]The settings for different degrees of data correlation were chosen based on guidance from the tool paper.

related. This is explained in conjunction with the trends depicted in Figure 3.17(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 (Figure 3.17(a)), but also more of it (Figure 3.17(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 [35] 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%). Figure 3.18(a,b) shows the results of these experiments.



(a) DCG. (b) Completeness.

Figure 3.18: 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 data values within a certain range of the data 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 (Figure 3.18(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 chapter as well (Section 4.4.2).

3.4 Related Work

Sharing, computing on, and discovering information about the surrounding environment are the fundamental activities of almost every pervasive computing application. Therefore, a wide spectrum of mechanisms, systems, and middleware have been proposed to support and facilitate the development and execution of distributed applications that target dynamic environments. The constructs described in this chapter build on previously proposed approaches (e.g., rule-based programming [93]), but are differentiated by their general-purpose-ness and explicit sensitivity to spatial and temporal dynamics. This section highlights key areas of related work in wireless sensor networks and context-aware applications.

Developer Support for Distributed Pervasive Applications. The rule-based programming paradigm is by no means a novel concept. Several key rule-based systems have been proposed for pervasive computing applications that enable data-dependent application behavior to be encoded as reactive *patterns* over data. Very similar to our rule-based programming approach, the Tuples On The Air (TOTA) middleware infrastructure [93] aims to reduce the complexity of distributed applications through patterns defined over spatially-distributed tuples. A developer may define application-specific patterns parameterized by externally-sensed properties and data content that dictate how tuples move (i.e., are routed) within a dynamic network. Similarly, the evolving tuples model [162] embeds this data-dependent behavior directly within communicated data by attaching rules to tuples that express how those tuples propagate and evolve over time. These and similar tuple space infrastructures are extremely flexible, but implicitly require use of a

tuple space data model [112]. Our rule-based programming constructs are truly general-purpose and make no such assumptions about an application's underlying data model. We instead provide decoupled interfaces that explicitly separate *what* data is about from *how* it is used.

The publish/subscribe paradigm has received significant attention in the pervasive computing domain due to its high degrees of decoupling and flexibility. Publish/subscribe can be viewed as a rule-based mechanism with an extremely limited selection of rules, in fact just two: *publish* and *subscribe*. Context-aware extensions have been proposed that enable these mechanisms to be parameterized by complex formulations of context. For example, the SPCF protocol [28] supports subscriptions to remotely sensed environmental conditions (e.g., a fire extinguisher unit may subscribe to carbon monoxide levels sensed by a smoke detector unit). The model proposed in [43] enables both publications and subscriptions to target spatiotemporally defined regions (e.g., a vehicle may subscribe to traffic events published by vehicles that are 10 minutes down its path of travel). Space and time are likewise first class citizens in our programming constructs and are equally capable of supporting context-aware parameterizations. However, we do not restrict applications to any mechanisms or require developers extend only a particular set of rules. Although, publish/subscribe mechanisms could easily be implemented within our rule middleware.

General-purpose approaches that facilitate the sharing and discovery of digital data and sensed environmental conditions are useful for creating applications that must satisfy a wide variety of requirements. However, searching proximately-

available resources for dynamic information calls for tailored approaches specifically designed with query-driven data access in mind. We investigate search-driven work next.

Resource Discovery and In-Network Search. 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, we argue that users in the future will demand this same sort of 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 host 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 [91] supplies a tool bag 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 [115] 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 [108] 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) [4] and Web of Things (WoT) [58], ordinary objects are imbued with sensing, networking, and otherwise "smart" ca-

pabilities enabling their access via a network connection. Given such a world of digitally accessible "things," many recent systems have attempted to address *entity discovery*[18]. 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 [163, 167], MAX [77], SenseWeb [79]) or additionally over dynamic sensor states and location (e.g., Dyser [119], IoT-SVK [34], CASSARAM [125]). 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 [119]). 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 unreliable networks. Data space and coordination models targeting distributed systems (e.g., tuple spaces [112], distributed databases [15], and distributed hash tables [160]) attempt to abstract away the distributed and disconnected na-

[18]We refer the reader to [144] for an excellent survey of entity discovery systems.

ture of resources (e.g., mobile devices) by providing access to them as if they were a holistic global virtual data structure [130]. Similarly, distributed routing and location systems (e.g., Tapestry [176], Pastry [145], and CAN [138]) 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 [177]. As described earlier, some systems enable developers to specify application-specific patterns that dictate how and when data is moved and shared (e.g., TOTA [93]), relieving applications from decisions regarding the physical transport of data. Similarly, existing publish/subscribe frameworks parameterize event distribution by social metrics [27], physical proximity [41], contextual relations [13],

temporal properties [158], degree of interest matching [117], and even complex formulations of context [28, 43]. 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.

3.5 Research Contributions

This chapter makes the following research contributions:

Research Task 4: We create developer tools that enable the formulation of reactive rules governing data creation, sharing, and deletion (Section 3.1). Using these software components we benchmark the overhead of explicit spatiotemporal data provenance with varying trajectory resolution in a simulated per-

vasive computing environment (Section 3.1.4). Different pervasive computing applications will likely require different granularities of spatiotemporal data history. To gain a deeper quantitative understanding of how trajectory resolution impacts system performance we evaluate the system-level cost of explicit provenance annotations in terms of bandwidth and storage using a geospatial coordinate system.

Research Task 5: Leveraging the formal operators introduced in *Research Contribution 2*, we demonstrate the practical utility of implicit spatiotemporal data provenance. Using real-world data sets of human proximity, we benchmark the performance of causal data annotations as a means of locally estimating global network characteristics and data spreading dynamics (Section 3.2.1). Next, we showcase the inferential power of causal provenance for making bandwidth-saving routing decisions (Section 3.2.2): first, as a tool for detecting and eliminating redundant device-to-device transmissions and second, for identifying commonly co-located data-data and data-device pairs, which may be used as substitute routing targets within a targeted data dissemination protocol.

Research Task 6: We describe the Gander conceptual model of search for pervasive computing environments, which supports queries over data and its spatiotemporal metadata. The relevance of real-world information is inherently parameterized by both space and time. Therefore, we assess the impact of data correlations in space and time on Gander's performance in simulation (Section 3.3). This task sheds light on how the Gander query processing protocols perform when the degree of these data correlations are varied; that is,

we ask whether different degrees of correlations impact the *quality* of the results achieved by a Gander search. These results further validate the utility of spatiotemporal provenance and aid in the design and implementation of developer tool support for spatiotemporal trajectories and query processing protocols.

3.6 Impact

The spatiotemporal provenance middleware layer presented in Section 3.1 was successfully extended in [78] to assist in the implementation of a router that provides geo-source routing of DTN bundles. The our knowledge, the experiments in this chapter represent the first evaluation of historical spatiotemporal metadata as a general-purpose routing assistance tool. The Gander conceptual model is the first to formally integrate the notion of user-defied relevance directly with query processing functions supported by ad hoc routing protocols. Gander's search mechanisms are inherently sensitive to the spatial and temporal dynamics inherent in pervasive computing spaces. Our investigations of these spatiotemporal sensitivities in this chapter are the first to benchmark the performance of ad hoc routing protocols with respect to user-oriented relevance metics traditionally employed in information retrieval.

3.7 Chapter Summary

In this chapter, we presented the design and implementation of our *explicit* model of spatiotemporal data provenance within an extensible middleware. The

middleware provides programming interfaces that enable a developer to encapsulate data-dependent logic within reactive "rules," which may be parameterized based on data or spatiotemporal metadata conditions. We showcased the middleware and its programming constructs through a use case that benchmarked the tradeoffs between storage space and the "decay" rate of geospatial data provenance. In this chapter we also demonstrated the value of *implicit* model of spatiotemporal provenance as a practical tool for estimating global network and data spreading characteristics using only locally-available information. Next, we showed the inferential power inherent in spatiotemporal provenance for making distributed routing decisions at run-time that significantly improve the performance of opportunistic data dissemination protocols. Our uses cases demonstrated direct applications of spatiotemporal annotations for reducing redundant transmissions and directing targeted data diffusion, for example, a distributed query protocol. Finally, we overviewed the Gander conceptual model of search in pervasive computing spaces, which we previously introduced in [101]. Our general approach is to treat search as a *sampling* task using spatially and temporally sensitive distributed query processing protocols. We used the model to explore the effects of spatial and temporal correlations in sensed data on Gander's search protocols. In the next chapter, we discuss the Gander application framework, which unifies the formal foundations provided in Chapter 2 and the sharing and search constructs described in this chapter within an extensible mobile software framework. We use this framework to create a distributed search engine that supports expressive user-input searches over data and spatiotemporal metadata generated by real-world entities.

Chapter 4

Data Sharing and Query Processing Framework

In addition to traditional approaches to benchmarking performance of algorithms through simulation, a key challenge is to explore the impact of these approaches under realistic conditions that reflect the complexity of the rapidly changing physical environment at a large scale. Therefore, a major research component of this dissertation is a comprehensive implementation and evaluation of the Gander application substrate. Central to this effort is the creation of the *myGander* mobile system, which allows users to create and control data and to pose queries and view their results.

In this chapter, we discuss the final research aim of this dissertation, which is shown as the checkered box at the bottom of Figure 1.1. We introduce the Gander application framework, which reifies the conceptual model overviewed in Chapter 3 and allows hyper-localized data sharing and search resolution across both mobile ad hoc network interactions and through localized cloudlets. In other words, Chapters 2 and 3 provide the formal foundations necessary to realize the Gander search engine; in this chapter, we describe the systems directions and support necessary to evaluate and deploy Gander within real world pervasive computing applications. We use the

Portions of this chapter appear in [99] and [100] for which coauthors Christine Julien and Jamie Payton provided advising and editing.

Gander framework to evaluate our distributed search mechanisms in real world settings though the design and deployment of the myGander mobile system on The University of Texas campus. The myGander system leverages localized cloudlets to enable proximal mobile devices to form peer-to-peer connections.

4.1 A Rule-Based Data Sharing Middleware

Applications targeting emerging mobile environments require support for hyper-localized data access through device-to-device interactions. For example, Apple's iBeacon technology facilitates push notifications triggered by entities (e.g., a store at a shopping center, a restaurant, a piece of art at a museum, etc.) on a proximate users' mobile device. Google's Physical Web project[1] aims to provide "walk up" mobile access to smart objects. For example, a user may approach a vending machine and purchase a snack using his smartphone through a website severed by the vending machine via a direct Bluetooth connection. Extending beyond just single hop device-to-device connectivity, OpenGarden's FireChat[2] application constructs a multi-hop mesh infrastructure to facilitate chat rooms between physically proximate mobile users without the need for an Internet connection. More generally, these technologies and applications point towards the grand vision of the Internet of Things [4] and Web of Things [58] wherein commonplace entities, from kitchen appliances to vending machines to bus stops to billboards, are imbued with digital capabilities that are seamlessly accessible through a user's mobile device.

[1] `http://google.github.io/physical-web/`
[2] `http://opengarden.com/firechat`

The role of device-to-device interactions (i.e., via short-range wireless protocols) becomes crucial in supporting mobile users' hyper-localized information needs in densely populated environments of networked devices, for example, at a crowded outdoor festival. Support for device-to-device data sharing becomes particularly critical in scenarios where an Internet connection is unavailable, infrastructure network resources are saturated, global-accessibility is undesired, or use of a short-range network interface (e.g., Bluetooth Low Energy) requires less overall energy than of a long-range network interface (e.g., cellular).

Motivated by these fundamental needs, we extend the data provenance middleware described in Section 3.1 to implement a real-world data sharing middleware. Our data sharing middleware leverages ad hoc wireless protocols to "passively" diffuse application data between proximate devices. We target mobile applications; therefore, a key requirement of our middleware is that it support native mobile applications with minimal setup and configuration. This section presents the architecture and implementation of our mobile middleware and concludes with several real-world application examples.

4.1.1 Middleware Architecture & Implementation

In emerging mobile environments data is "everywhere;" users' physical environments are a rich and dynamic landscape of digital information. Here, we aim to provide practical support that enables real-world mobile applications to tap into users' digital landscape. We introduce a middleware that extends our implementation of the *explicit* model of spatiotemporal provenance (Section 3.1). Next, we

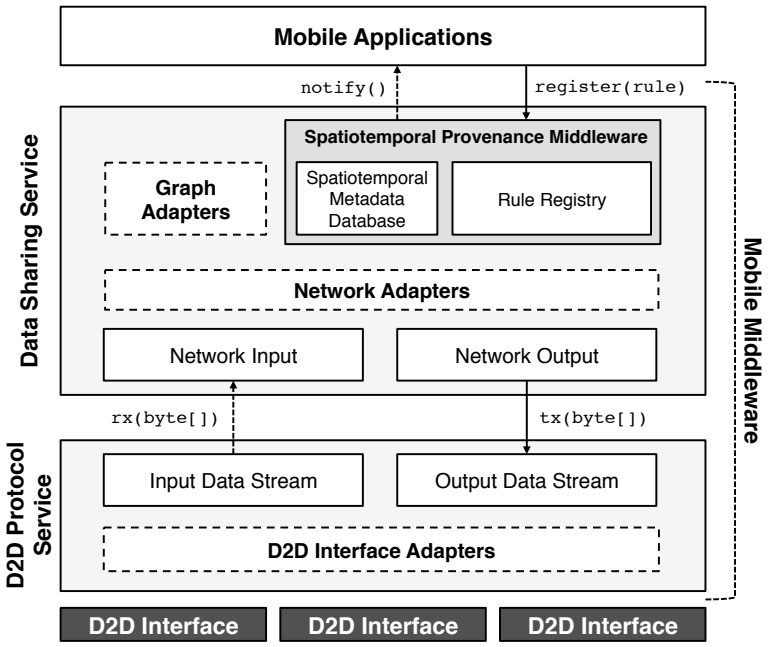


Figure 4.1: Mobile device-to-device data sharing middleware architecture.

describe the architecture and implementation of our mobile middleware.

Figure 4.1 illustrates the high level design our middleware, which we have implemented for the Android platform. The key components of the middleware are two Android `Service`s, both which may either be run on a device as standalone applications or embedded within a mobile application. These services separate *what* data is being shared from *how*. The dashed boxes with bolded text represent abstract implementations of components that require application and network protocol - specific implementations.

The *Data Sharing Service* (the second gray box from the top in Figure 4.1) employs an instance of our spatiotemporal data provenance middleware (the darkened gray box) and facilitates proximate data access via that middleware's rule-

based programming interface. Application-defined rules govern precisely what data is exposed to proximate devices, when, and how often. For example, a native mobile application may define a rule that periodically "sends" (we will define how *sending* tasks are executed shortly) a user-input status update to all proximate devices. This rule may be *registered* with the *Data Sharing Service*, which will autonomously execute the rule's behavior whenever its conditions are satisfied, for example, after 10 minutes have transpired since the last execution. We only require that an application define two *adapters* that define how the application's rule-governed data are marshaled in and out of graph- and network-readable formats. If an application wishes to store structured data within the middleware (e.g., so it may be sent periodically) it must provide a *Graph Adapter*, which translates application data to and from a graph-based format. To serialize and de-serialize application data to a byte format, an application must also provide a *Network Adapter*. An application data object is sent and received via the *Network Output* and *Network Input* interfaces, respectfully, which employ the object's corresponding *Network Adapter* to convert between the application object and a serialized byte array. We provide abstract implementations of these adapters, which an application may easily extend. Within our middleware, JSON (JavaScript Object Notation[3]) is the default serialization strategy.

The *D2D Protocol Service* (the second gray box from the bottom in Figure 4.1) defines *how* digital data is shared between proximate devices. Essentially, this service acts as a bridge between the *Data Sharing Service* and one or more on-

[3] `http://json.org/`

board device-to-device network interfaces (shown as dark gray boxes at the bottom of the figure). To send and receive data via a particular device-to-device network interface (e.g., Bluetooth, WiFi Direct, LTE Direct, NFC, etc.) we simply require an application provide a concrete implementation of an abstract *D2D Interface Adapter*, which dictates exactly how bytes are sent and received via that interface.

For the sake of demonstration, we implement a *D2D Interface Adapter* for a custom WiFi-based device-to-device communication technology[4] to share serialized data between proximate devices. The architecture of this implementation is illustrated in Figure 4.2 within the *Beacon Interface Adapter* dashed box. Interactions with a device's WiFi interface are managed by the *WiFi Beacon Service* shown as the bottom-most box in Figure 4.2. The *WiFi Beacon Service* employs *beacon-stuffing* [23] to communicate small data frames to proximate (i.e. in WiFi range) devices without establishing explicit connections. Beacon-stuffing is a strategy that exploits vendor-specific *information elements* (IEs) in IEEE 802.11 beacon packets. An IE (the grayed field in Figure 4.3) is a 256 byte block of "extra" space in the beacon packets that are periodically broadcast by a WiFi access point (AP). Provided enough unused space exists in a packet's IE, arbitrary bytes can be "stuffed" into the IE, which are

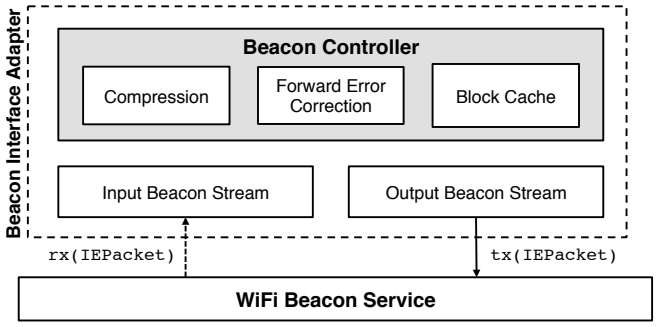


Figure 4.2: An implementation of a *D2D Interface Adapter* for WiFi beacon stuffing.

[4] `http://www.m-87.com/`

received by all client devices in range. In this way, beacon-stuffing operates as a low bandwidth wireless communication protocol. This is precisely how we use the *WiFi Beacon Service* to facilitate "passive" (connection-free) data sharing. To transmit data, the *WiFi Beacon Service* places the transmitting device into AP-mode, forcing the device to periodically broadcast beacon packets. A *Network Interface Adapter* (i.e., our *Beacon Interface Adapter*) may then provide the *WiFi Beacon Service* with "chunks" of bytes (32-256 bytes in size), which are each placed into a beacon packet. To receive data, *WiFi Beacon Service* simply listens on a particular WiFi channel and strips the IE contents from detected beacon packets. The remaining elements of the *Beacon Interface Adapter* coordinate the decomposition of blocks of application data into IE-sized chunks and the correct composition of IE-bytes back into their application data block. We briefly describe these components next.

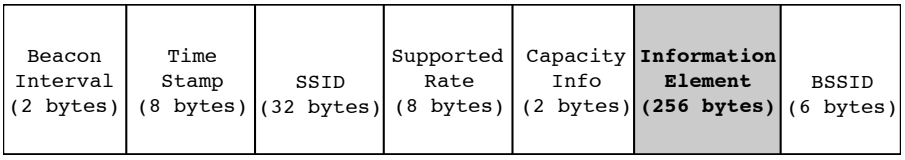


Figure 4.3: Some fields in the IEEE 802.11 beacon packet [23].

The *Beacon Interface Adapter*'s *Beacon Controller* provides functional components that compress application data blocks, support forward error correction, and cache received IE byte chunks until all block chunks have been received. The successive use of these components to transmit a piece of application data via the *WiFi Beacon Service* is shown in Figure 4.4. We now use this figure to walk through the entire sequence of steps involved in transmitting a piece of application data. First, a piece of application data is serialized to a byte array by its application-provided *Network Adapter* (within the *Data Sharing Service* shown in Figure 4.1).

Our middleware supports one or more associated rules to be optionally attached to and transmitted alongside data (this functionality will be explained shortly). The byte array is transferred to one or more *Network Interface Adapters* (the *Beacon Interface Adapter* in our case) via an exchange between the *Network Output* and *Output Data Stream*. The *Output Data Stream* sends the byte array to the *Beacon Interface Adapter*'s *Beacon Controller*, which compresses the bytes then applies forward error correction[5] to mitigate dropped beacon packets in the event of a lossy channel. The forward error correction step breaks the compressed data up into *blocks* comprised of fixed-sized *chunks* of bytes. Each *chunk* is queued by the *Output Beacon Stream* and sent to the *WiFi Beacon Service*, which places each *chunk* into the IE of an outgoing bea-

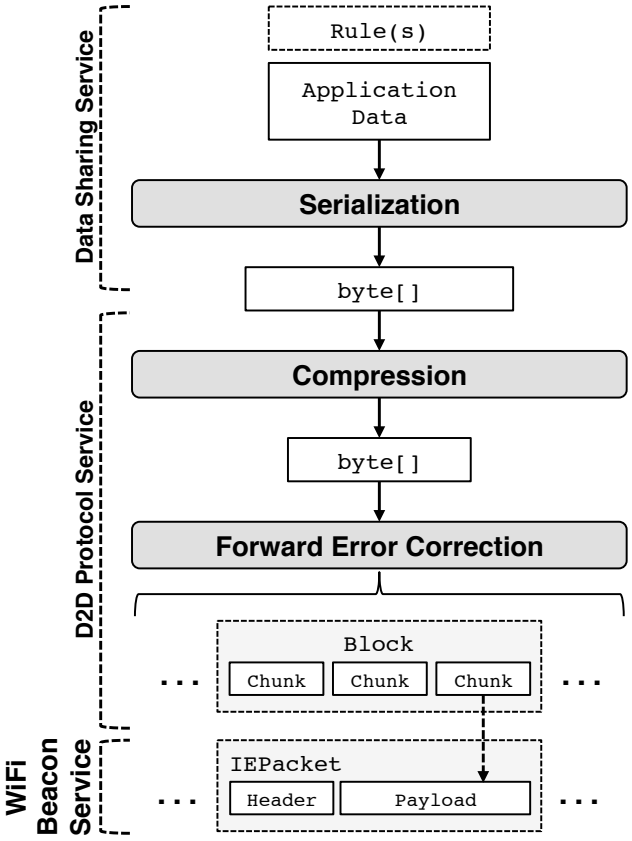


Figure 4.4: The functional sequence of steps comprising the transmission of application data using the data sharing middleware.

[5]We employ OpenQ, an open source implementation of the RaptorQ digital fountain erasure code (`http://www.lasige.di.fc.ul.pt/openrq/`).

con packet. Information indicating a *chunk*'s *block*, sequence number, and data type are encoded within a header also placed with the IE. The receiving process is essentially the reverse of these steps, except that received *chunks* are cached within the *Block Cache* (Figure 4.1) until enough are present to successfully decode an entire piece of application data.

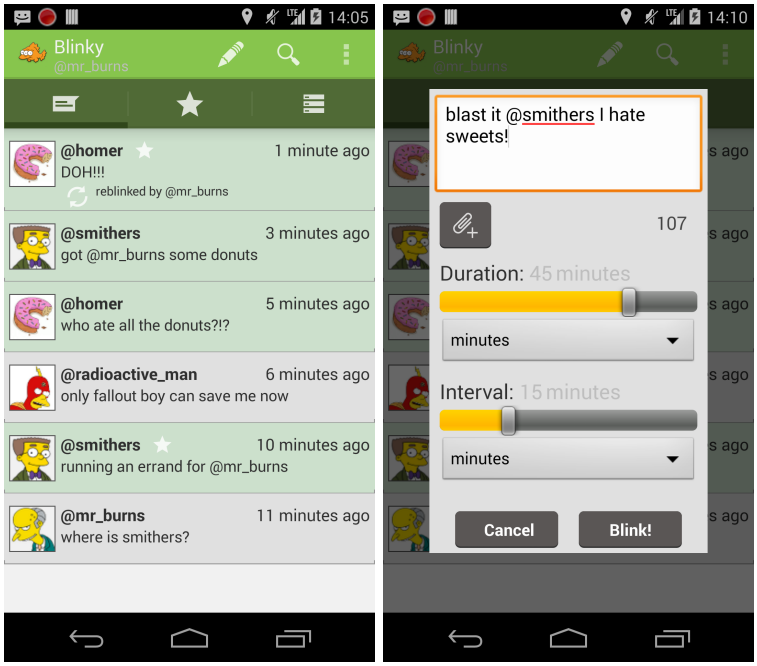
As mentioned earlier, we enable rules to be optionally transmitted alongside application data. Any rules attached to received application data objects are immediately registered in the receiver's *Data Sharing Service* and executed, from the receiver's perspective, when their conditions are satisfied. Transmitting rules with their data is useful, for example, for implementing a distributed routing protocol without explicitly deploying the protocol's code *a-priori*. In other words, transmitting rules with their data enables data-dependent behavior to be deployed and executed on-the-fly. We now describe real-world use cases of our data sharing middleware.

4.1.2 Application Examples

Our middleware supports emerging use cases that connect users and their devices with the surrounding digital landscape. This section briefly describes two real-world applications of our proximate data sharing middleware.

4.1.2.1 Hyper-Localized Social Application

Often, users that are congregated in the same place share similar interests and information needs. For example, at a large business convention users may be



(a) A user's feed of received "blinks."

(b) The user may parametrize the spatiotemporal device-to-device propagation of a "blink."

Figure 4.5: The *Blinky* hyper-localized mobile social application.

interested in connecting with nearby colleagues and organization representatives or receiving alerts about spontaneous events. At a crowded music festival, festival participants may likewise be interested in connecting face-to-face with their friends and sharing photos. Vendors at the festival may wish to disseminate special offerings and advertisements to attract customers. These example are perfect use cases for a social application that exploits users' hyper-localized information needs. To that end, we created a small demo application on top of our data sharing middleware called *Blinky*. Blinky enables users to compose "blinks," short text or picture messages, and broadcast them to proximate devices, possibly across multiple network hops. Blinks that are received over the proximate network are displayed in the user's feed

(Figure 4.5(a)). A user may optionally tailor the spatial and temporal characteristics of how a blink is propagated between devices (Figure 4.5(b)). Specifically, the user can parameterize any of the following: the temporal duration of a blink's propagation, the temporal interval at which the blink is transmitted and re-transmitted, the maximum geospatial distance from the location of creation the blink can propagate, the geospatial distance of travel that will trigger a re-transmission of the blink, and the maximum number of network hops from the creator the blink may propagate. These parameters are reified within rules, which are registered with our middleware and attached to transmitted blinks. We envision socially-driven mobile applications like Blinky to become ubiquitous as developer constructs for device-to-device protocols (e.g., Bluetooth Low Energy, LTE Direct) on mobile devices continue to receive popular attention.

4.1.2.2 Efficiently Accessing Dynamic Data

In the previous application example, users broadcast static information. However, an application may need to make dynamic information available, for example, the sensed state of a real-world entity (e.g., the queue length at a cafe, a parking space's availability, the ambient noise level and available seating in a university study lounge). LTE Direct[6] (LTED) is an emerging communication technology that enables device-to-device discovery and communication over long ranges (up to 500 kilometers). LTED employs a cloud-based entity to store the actual data to be communicated; devices instead exchange hash codes (that map to that data) over

[6] `http://ltedirect.qualcomm.com/`

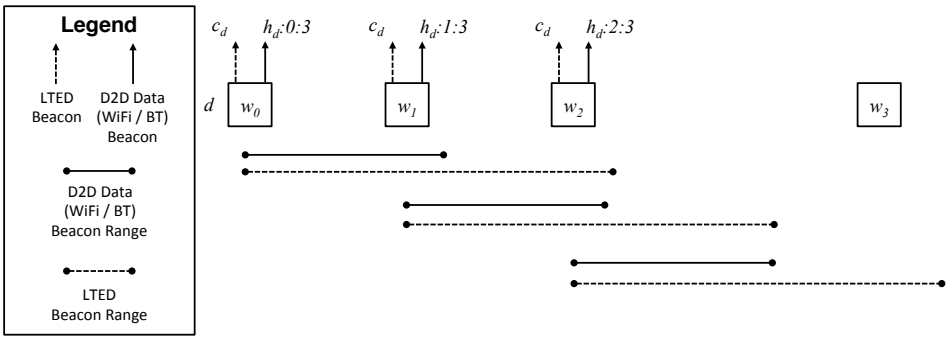


Figure 4.6: Dynamic data access strategy in LTED-enabled networks.

the air in a device-to-device fashion. A device receiving one of these hash codes must resolve the code's data at the cloud-based entity over the Internet. While this approach is useful for static data, communicating time-varying data over an LTED network can become expensive—whenever the data changes it must be re-written at the cloud entity, producing a new hash code, which must be re-resolved to the changed data by devices receiving the new hash code. We propose a better approach: the advertisement of static information (e.g., a cafe's location, its menu, etc.) may remain advertised over the LTED network; however, dynamic information (e.g., the cafe's current queue length) may be made available through our middleware.

As an example consider the scenario shown in Figure 4.6. The figure shows four devices (w_0—w_3) that each possess both an LTED network interface and a short-range device-to-device network interface (e.g., WiFi, Bluetooth). In this example, device w_0 is the provider of a time-varying data value d. For example, perhaps w_0 is a queue length sensing device at the cafe described earlier and d is the current number of people waiting in line. Rather than advertising d over the LTED network, which would require a multiple device-to-cloud transactions per proximate device (i.e., w_1 and $w2$) every time d changed, w_0 may advertise d over a short-range

device-to-device protocol (e.g., Bluetooth, WiFi). In Figure 4.6, c_d represents an LTED-assigned hash code that maps to a static description of d (e.g., "*live queue length*") and a secondary hash code h_d that also maps to d. In other words, a device receiving c_d may resolve it to h_d at the LTED cloud-based entity. In our scenario, the device w_0 advertises c_d over LTED and h_d over a short-range device-to-device protocol. In Figure 4.6, a hop count and hop limit are also transmitted alongside h_d. Consider the device w_1 that is in LTED-range receiving c_d. Upon receiving c_d, w_1 may resolve c_d to h_d at the LTED cloud entity and listen on its short-range interface for h_d. If h_d is received on the short-range interface, w_1 may use the associated hop count information to access d over the same short-range interface (e.g., by issuing a request). The device w_1 may re-advertise both c_d (via LTED) and h_d (via the short-range interface, modifying the hop count), creating a daisy-chained route to w_0 and d. Devices farther down the chain (e.g., w_2) may access d by issuing requests that follow the *gradient* defined by decreasing hop counts to d. Devices that receive c_d, but not h_d (e.g., w_3) must give up or employ another cloud-based entity to retrieve d over the Internet (e.g., via a peer-to-peer connection to w_0). Accessing time-varying data d in this fashion can greatly reduce the transactional load at the LTED cloud-based entity.

This section presented the design and implementation of a middleware for connecting users and their devices to hyper-localized digital data via device-to-device interactions. It may not always be the case that device and application support for ad hoc network communication exist. For example, LTE Direct is not a common mobile technology yet; our *WiFi Beacon Service* requires root-level permissions to

be fully functional. Given these limitations, the next section introduces a hyper-localized discovery service that uses cloudlets to facilitate proximate device discovery and communication.

4.2 Device-to-Device Interaction via Localized Cloudlets

Pervasive computing demands the ability to discover locally available resources, whether data shared by nearby users or digital capabilities in the surroundings. More specifically, applications require a location-based discovery service that, when provided a user's or device's location, can "look up" nearby (digital) resources. In this section, we focus specifically on the need to discover resources relevant in a particular user's space and time; we are interested in finding digital resources that are available in the user's *here* and *now*.

Here, we introduce a hyper-localized discovery service that creates *cloudlets* that are responsible for maintaining knowledge about the digital resources available in specific regions of space. Ultimately, this discovery service provides systems support for the query processing proposed in Chapter 3. This point will be discussed in detail in Section 4.2.2. A cloudlet is a trusted computing resource with good Internet connectivity that is available for use by physically nearby mobile devices over a wireless LAN [152]. Essentially, a cloudlet has the same responsibilities as the cloud—it hosts a service that performs the significant computation required by a mobile application while enabling the mobile device to act as a thin client with respect to the service. Cloudlets differ from the cloud in that they are inherently within close physical proximity to the mobile devices that utilize their

resources, which reduces network latency and jitter and mitigates peak bandwidth demands [152]. Consequently, cloudlets service fewer users and keep data close to where it originates.

We leverage cloudlets to implement proximal discovery for pervasive computing applications that operate in densely populated and highly mobile physical spaces. A cloudlet-based approach for co-located peer discovery becomes seriously advantageous, perhaps even necessary, when applications need to operate in spaces with hundreds of thousands of devices per square mile (e.g., as in the Internet of Things [4]).

We present the architecture, implementation, and API for a cloudlet-based proximal discovery service that leverages a physically dispersed infrastructure of cloudlets, each of which provides the service for a specific geographic region. The distributed nature of a cloudlet infrastructure mitigates the congestion and resource contention of a centralized cloud-based mechanism. Moreover, the physical and logical proximity of cloudlets to the clients they serve results in low one-hop network latency. These factors combined provide the support required by mobile pervasive applications targeting densely populated physical spaces. A key technical challenge in the design of a cloudlet-based proximal discovery service is the synchronization of information at the boundaries of bordering or overlapping cloudlet administrative regions. As users and devices move around, they inevitably migrate between cloudlets' administrative regions. An application running on a device within the fringe of one of these ambiguous spaces may request information that exists *beyond* the administrative boundary of its respective cloudlet's administrative region,

though the geographic relevance of that information may be physically close. We evaluate our service's performance in terms of the system-level *cost* and the *quality* with which it performs proximal neighbor discovery under various bordering cloudlet synchronization strategies.

4.2.1 Architecture & Implementation

Under the assumption that cloudlets will be deployed within an existing Internet infrastructure [152], we implement our proximal discovery service as a Web service intended to be hosted on a cloudlet computing resource and made available through a RESTful API over HTTP. In Section 4.3.3 we describe how the discovery may be used to support the query processing protocols discussed in Chapter 3. Here, we focus on the implementation of the proximal discovery service itself.

4.2.1.1 The Proximal Discovery Service

Our proximal discovery service's architecture is shown in Figure 4.7. The service has two components that reside on a cloudlet: a spatial index to keep track of resources' locations and a REST API that enables remote storage and retrieval of these resources via an Internet connection. A "resource" is anything that a particular application wishes to associate with a geographic location—e.g., a user's mobile device, a sensor embedded in the environment, a location-dependent message, or a mobile software agent that migrates between machines. We implement the spatial index using a PostgreSQL[7] database with PostGIS[8] geospatial database extensions.

[7] `http://postgresql.org`
[8] `http://postgis.net`

PostGIS employs an R-Tree [61] to intelligently index and efficiently query geospatial data. Our service's REST API is implemented in Node.js[9], which is a particularly good fit for our use case because of its small footprint and ability to efficiently handle many simultaneous connections and requests.

Our service exposes a minimalist REST API with the following resource "routes." We aim solely to facilitate discovery of physically nearby resources, leaving other application-specific functions to applications themselves.

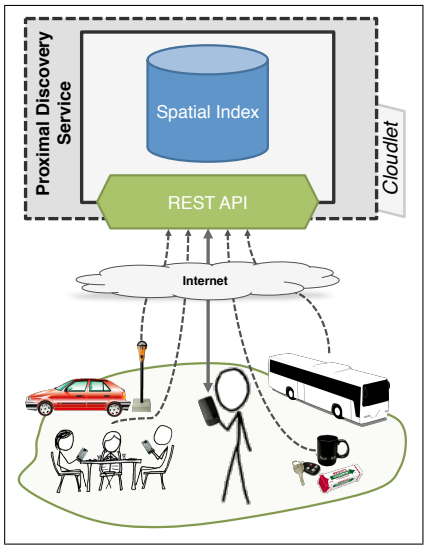


Figure 4.7: The proximal discovery service architecture. Dashed arrows are `POST /devices` requests; the solid arrow is a `GET /neighbors` request.

`POST /devices` A client (mobile device, embedded sensor, software agent) issues this request to an instance of the discovery service to create or update the resource representing the client on the cloudlet.

`GET /neighbors` A client issues this request to query a cloudlet for resources within its vicinity. We parameterize the request with a *range* to limit the query's geographic search space. An application could further impose additional application-specific parameters. Upon receiving this request, the cloudlet responds with a list of resources whose locations are believed to be within the request's range of the client.

[9] `http://nodejs.org`

Our approach only provides an advantage over a centralized cloud-based LBS when many independent instances are distributed across a geographic region, where each instance assumes responsibility for a particular sub-region. However, important challenges arise when clients physically move between these sub-regions or issue queries that cross the boundaries of these sub-regions.

4.2.1.2 Distributed Cloudlet Synchronization

Our proximal discovery service supports pervasive computing applications in densely populated physical spaces. To fully utilize the potential of a cloudlet infrastructure and support large numbers of anticipated clients, many instances of our service must be deployed in a geographic region, with each instance responsible for a particular sub-region. These spaces are highly mobile—people and their devices move and inevitably migrate between adjacent cloudlet-administered sub-regions; our proximal discovery service must address the distributed synchronization of cloudlets governing these adjacent sub-regions.

Consider the scenario in Figure 4.8. A space is decomposed into nine rectangular *administrative regions* (AR). One cloudlet exists in each AR and is responsible for responding to cloudlet-bound requests for that region. The solid dot represents a mobile client that has traveled along the dashed path over a period of time. As the client moves, it periodically is-

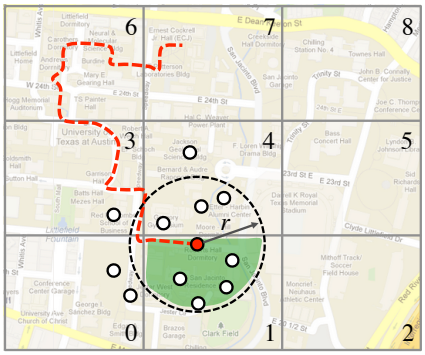
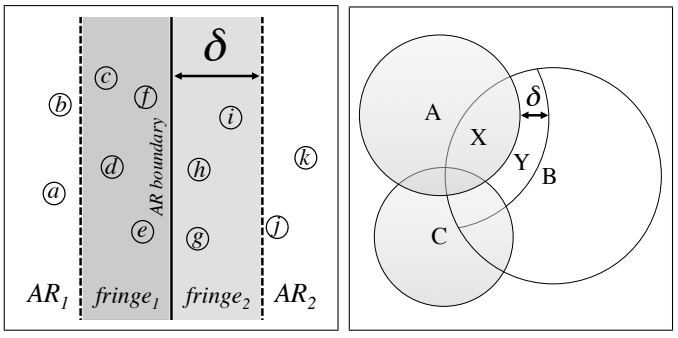


Figure 4.8: A query for resources within a range of r from a mobile client.

sues a `POST /devices` with its current location to the cloudlet responsible for the AR it occupies. At the point in time shown, this client wishes to discover nearby resources within a range of r, so it issues a `GET /neighbors` query to $cloudlet_1$. However, the query's range extends beyond the borders of $cloudlet_1$'s AR and into those of $cloudlet_0$, $cloudlet_3$, and $cloudlet_4$. Without some means of synchronization, $cloudlet_1$ can only respond with the four resources in the shaded region of the query's target space (i.e., the resources within its AR). To facilitate this synchronization, our proximal discovery service implements a third resource route:

`POST /fringes` A neighboring cloudlet issues this request to communicate knowledge of resources that exist at or near the border of an adjacent AR.

To both capture and quantify "at or near the border," we introduce the concept of a *fringe*, illustrated in Figure 4.9. A fringe is a sliver of space of width δ on the interior of an AR's border. For simplicity, the ARs shown in Figure 4.9(a) are rectangular, and therefore the computed fringes are also rectangular. In actuality, ARs can be of any shape and even overlapping. AR_i's fringe with AR_j consists simply of any point within AR_i that is within a specified distance (δ) of AR_j. Consider the more complex pair of ARs in Figure 4.9(b). AR_A overlaps AR_B; any resource



(a) (b)

Figure 4.9: Computing administrative region fringes.

within the area marked X can report to either AR_A or AR_B (or both of them). The fringe that AR_B computes to create a digest for AR_A contains all of the resources it knows about located in Y *and* all of the resources it knows about located in X (because they may not have also reported their locations to A). Fringes can also overlap; consider the adjacency of AR_B and AR_C; the fringe of AR_B with respect to AR_C (not depicted) will include not only their overlapping area but also a sliver adjacent to this area that also overlaps Y.

Periodically, our service computes each of its fringes' *digest*—a snapshot of the fringe's resources—which it sends to the appropriate adjacent cloudlet via a `POST /fringes`. In Figure 4.9(a), for example, $cloudlet_1$ computes AR_1's fringe digest to be (c, d, e, f), which it `POST`s to $cloudlet_2$.

Our service entails three variable deployment parameters, each which must be tuned for a particular application's needs:

μ The number of administrative regions (and correspondingly, cloudlets) that a geographic region is decomposed into.

δ The width of administrative regions' fringes.

T The interval of time between fringe digest computations.

A particular combination of parameters, represented by the tuple (μ, δ, T), is a *synchronization strategy*. In Section 4.2.3 we evaluate our service's performance under various synchronization strategies in terms of the *quality* with which proximal discovery queries are satisfied and the system-level *cost* of the strategy.

4.2.1.3 Operating Assumptions

We assume clients can localize themselves and have a labeled map of cloudlets and their ARs. Localization may be too expensive a task for severely resource constrained devices (e.g., battery operated sensors); however, lightweight localization methods are currently a very active area of research [9, 26, 105, 124]. Alternatively, resource-constrained devices could communicate with cloudlets via less resource-constrained *gateways* [58]. We also assume that cloudlets are aware of neighboring cloudlets; in our implementation, clients and cloudlets each possess a copy of the same labeled map.

4.2.2 Application Examples

We next provide some concrete examples of mobile and pervasive machine-to-machine (M2M) applications that are enabled by our proximal discovery service, how they can be built with our service, and the inherent advantages of doing so.

Location Based Services. Perhaps the most obvious application of our service is to enhance location based services with hyper-localized capabilities (e.g., location-based social networking services, location sharing services, friend finders, and spatial crowdsourcing [81]). Using our cloudlet-based service for localized discovery, such services could target small and heavily crowded regions (e.g., music festivals, parades, theme parks, university campuses, etc.) much more efficiently. Instead of shipping application requests to a (physically and logically) distant central cloud index, requests *and* users' location information would be localized to the geographic areas where they are inherently relevant, providing low-latency responses,

lighter server-side loads, and privacy gains through the distributed cloudlets [173].

P2P Network Overlays and Routing. Using a P2P routing and location infrastructure like Tapestry [176] or Chord [160] to implement proximal resource discovery requires new network participants to have advanced knowledge of at least one peer already in the P2P overlay. This crucial discovery step could be performed with our proximal discovery service, where the resource of interest is any peer already in the overlay. The use of a cloudlet infrastructure, in this case, encourages and facilitates P2P interaction between devices that are *physically* near one another. Peer physical proximity is not a requirement for these systems but can be advantageous if the P2P overlay exists to store spatio-temporal events [177].

Mobile Ad Hoc and Opportunistic Networks. Our proximal discovery service could also be used to construct *virtual* mobile ad hoc network (MANETs) and opportunistic networks. In fact, this is precisely how the discovery service is utilized by the Gander Middleware (Section 4.3.1). Beyond the ability to implement MANET-style routing algorithms without a separate dedicated radio interface, virtualization of MANETs has added security benefits [67]. Virtual MANETs could be used to form on-demand mobile clouds [149], execute geographic routing [31], implement location-based publish-subscribe [42], or enable wireless sensor network (WSN) query-access mechanisms [45, 91, 108] across smartphones. The realization of cloudlet-based virtual MANETs for mobile computing could conceivably lead to a renaissance of techniques formerly developed for MANETs and WSNs.

Next, we introduce the Gander application framework, which leverages the proximal discovery service in cloudlet-supported environments to create virtual

MANETs and execute the distributed query processing protocols introduced in Chapter 3.

4.2.3 Cost and Quality of Cloudlet Peer Discovery

We next benchmark the performance of our discovery service in terms of its system-level cost and the quality with which the service performs proximal resource discovery under various combinations of (μ, δ, T) (number of administrative regions, fringe width, fringe digest computation period), in an effort to guide application developers in tuning our service's parameters to meet application requirements. We simulate mobile clients using two real-world and one simulated mobility trace data sets (Table 4.1). Our framework is implemented in Python.

Table 4.1: Mobility Trace Data Sets

Data Set	Geographic Region	Description
UT-Real	$640m^2$ UT Austin campus	24 hours of location information from 18 users of a mobile application deployed on the UT Austin campus in June 2013
UT-Sim	$640m^2$ UT Austin campus	6 hours of simulated location information for 200 nodes generated using MobiSim [110] and Levy-walk [140] mobility
Cabspotting [131]	$10km^2$ downtown San Francisco	6 hours of location information for ∼500 taxi cabs in downtown San Francisco, USA

We decompose a square geographic region into μ administrative regions (ARs), assign each AR one instance of our discovery service, and generate a labeled map of these regions and their ARs; the clients and the μ service instances each receive a copy of this map. Each simulated client moves about the region and issues a `POST /devices` with its current location at most once per minute to

the cloudlet-hosted service responsible for the AR it occupies. Every T seconds, each of the μ instances of the discovery service computes the digest[10] for each of its fringes. The service instance then issues a `POST /fringes` containing the digest to the respective neighboring service instance. During each simulation, we perform periodic proximal resource discovery queries (i.e., cloudlet-bound `GET /neighbors` requests): at 10 randomly chosen simulation times we randomly select 25 simulated clients[11] and issue three queries (r =["near", "medium", "far"])[12]. For each data set, we also generate an *Oracle*, which is effectively a central (cloud) server to which every client posts its location once per minute.

Table 4.2: Evaluation Parameters

Parameter	**Value(s)**
μ: *number of cloudlets*	$4, 9, 16, (25), 36, 49, 64, 81, 100$
δ: *fringe width (fraction of width of an AR)*	$\frac{1}{10}, \frac{1}{9}, \frac{1}{8}, \frac{1}{7}, \frac{1}{6}, (\frac{1}{5}), \frac{1}{4}, \frac{1}{3}, \frac{1}{2}$
T: *digest update period (in seconds)*	$60, 300, (900), 1800, 3600, 7200, 18000$
t: *client location update period (in seconds)*	60[13]
u: *location update size (in bytes)*	80

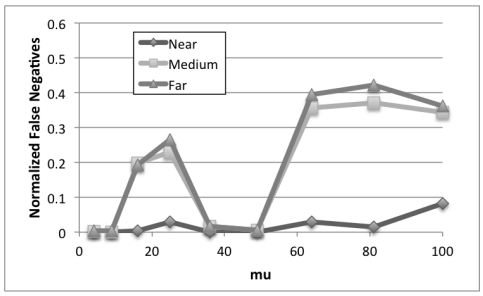
Table 4.2 shows our evaluation parameters; in cases where we explored multiple values, the value in parentheses is the default. We first evaluate the *quality* of our service in terms of the mean number of *false positives* (resources *in* a cloudlet result and *not in* the corresponding *Oracle* result) and *false negatives* (resources *not in* a cloudlet result, but *in* the corresponding *Oracle* result) produced per query under various synchronization strategies. In both cases, we normalize the false pos-

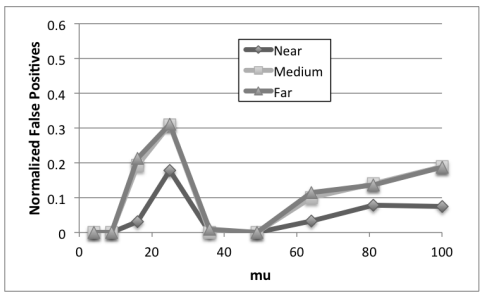
[10]To prevent "stale" resources, we restrict digests to contain only resources that issued a `POST /devices` within the last T seconds (i.e., since the last fringe digest).

[11]Only 18 clients were available in the *UT-Real* data set.

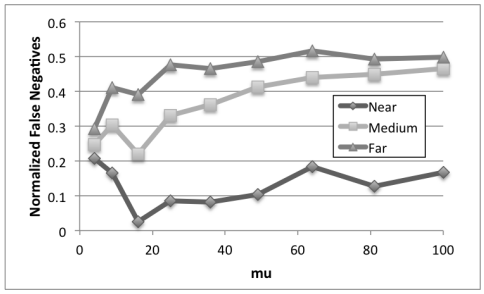
[12]We define the ["near", "medium", "far"] ranges as $(25m, 50m, 75m)$ for the *UT-Real* and *UT-Sim* data sets and $(100m, 500m, 1000m)$ for the *Cabspotting* data set.

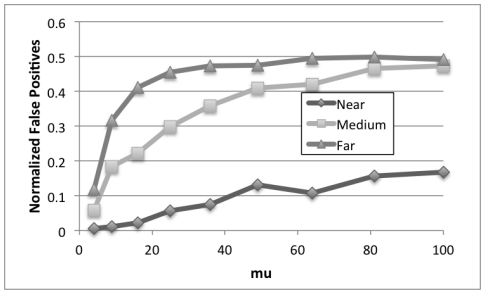
itives and false negatives to the size of the *Oracle* result set. That is, a value of 0.1 for a normalized false negative score indicates that for every 10 items in the *Oracle* result, there was one result missing from the cloudlet result.





(a) *UT-Real*





(b) *Cabspotting*

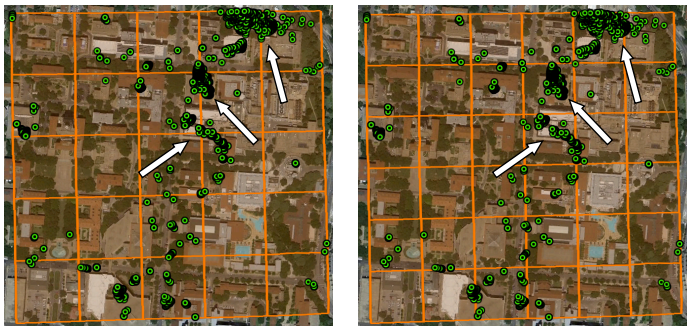
Figure 4.10: Impact of varying the number of cloudlets.

Figure 4.10 shows the false negative and false positive rates for varying the number of cloudlets for the *UT-Real* and *Cabspotting* traces; we omit the results for *UT-Sim* as they are very similar to the results for *Cabspotting*. Two observations are relatively consistent across all data sets. First, the errors show a slight upward trend as the number of cloudlets increases. As the cell sizes decrease, queries are increasingly likely to rely on the digest from neighboring cloudlets for correct information, and this information is more out of date than the local cloudlet's information. Sec-

[13]The client update period was set to exactly 60 seconds for *UT-Sim*. For *UT-Real* and *Cabspotting*, the period was determined by the data set but was close 60 seconds.

ond, the false positive and false negative rates for "near" queries are significantly better than for "medium" or "far." This demonstrates (and begins to benchmark) that the cloudlet-based approach is more suited for highly localized search and is not as well suited to searching for information that is located further afield. Consider a query in the *UT-Real* when μ is 64, where each cloudlet is responsible for a 10 x $10m^2$ area. A "medium" query, looking for resources within $50m$, may match resources that are not even located in an adjacent administrative region and will be impossible to find using our service.

Figure 4.10(a) also shows an interplay between the behavior of real users and the definition of cloudlets. For μ values of 36 and 49, the upward trend of the error rates is violated. The reason can be identified by examining the location traces of the users, shown in Figure 4.11. In the case of $\mu = 36$, the common areas where users cluster (in this case, three buildings on the university campus) happen to lie entirely within single administrative regions. In the case of $\mu = 25$, these clusters cross the boundaries of administrative regions. Users' queries therefore also often cross these boundaries, meaning that they increasingly rely on digests for correct query resolution. The conclusion from this observation is that defining administrative regions should account for user mobility patterns and should not create artificial boundaries

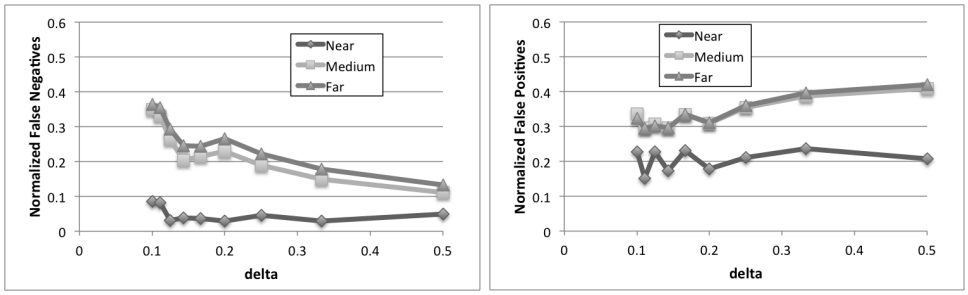


(a) $\mu = 25$ (b) $\mu = 36$

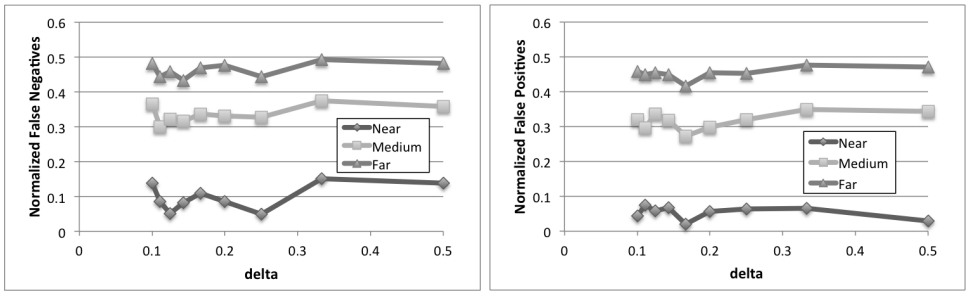
Figure 4.11: A "poor" choice of μ may split clusters.

to separate users within natural congregation areas.

We next examine the impact of varying the fringe width. Figure 4.12 shows the false positive and false negative rates for the *UT-Real* and *Cabspotting* traces; we again omit the results for *UT-Sim*, which are again very similar to the results for *Cabspotting*. Except within the (small) *UT-Real* data set, there is little relationship between changing δ and false positives and false negatives. These figures again show the significant difference in quality for "near," "medium," and "far" queries. While the error (as measured by the false positive or false positive rate) is routinely below 10% for "near" queries, it grows up to nearly 50% for "far" queries. This again demonstrates that the cloudlet based approach is particularly suited to very local queries of the immediate surroundings.



(a) *UT-Real*



(b) *Cabspotting*

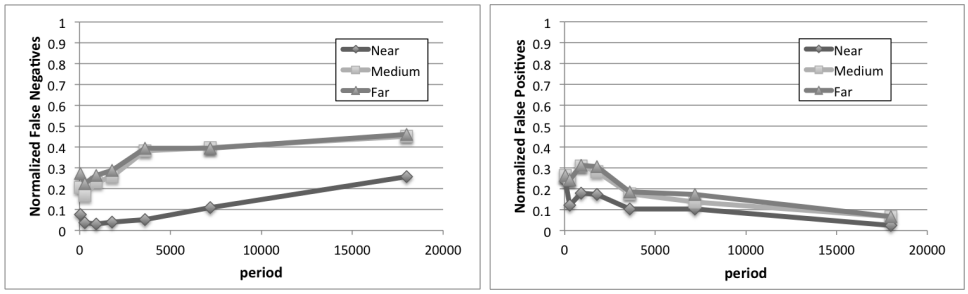
Figure 4.12: Impact of varying the fringe width.

Finally, we examine the quality of our cloudlet based discovery service with respect to the update interval for fringe digests in Figure 4.13. As T increases up to 50 minutes, quality degrades substantially. In the limit, the false positive rates also decrease; this is a result of the fact that the digests are simply not updated well, so they do not contain extra (irrelevant) information. However, we can also discern that it is acceptable for the update interval to be larger than the frequency with which the clients update their own cloudlets; specifically, any setting of T under 900 seconds has only a marginal impact on false negatives.

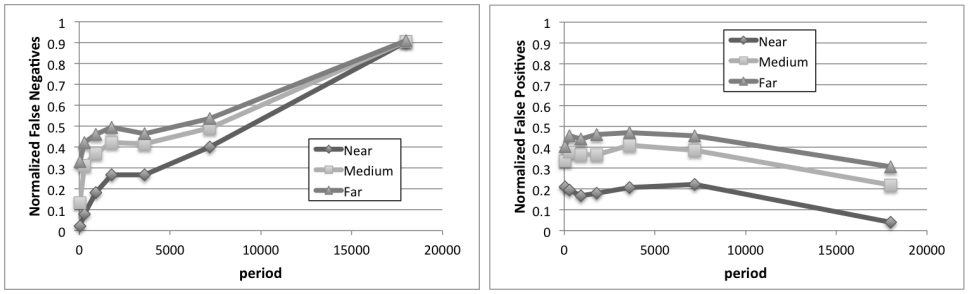
We also benchmark the system-level *cost* of a synchronization strategy in terms of the mean overhead (in KB) for sending both individual location updates to the local cloudlet and the cost of sending the fringes according to the update period. Because individual location updates are sent every minute, the cost, per minute, of sending individual location updates to the local cloudlet is computed as: $n \times 80\text{B} \times 1\,\text{hop} \times \frac{1\text{KB}}{1024\text{B}}$, where n is the number of clients sending location updates, and 80 bytes is the size of a client location update (from Table 4.2). We assume that each client is located within one network hop of the cloudlet server. We compute the cost of sending the digests between cloudlet servers as:

$$4(\mu + \sqrt{\mu}) \times \overline{size_{digest}}\text{B} \times 1\,\text{hop} \times \frac{1\text{KB}}{1024\text{B}}$$

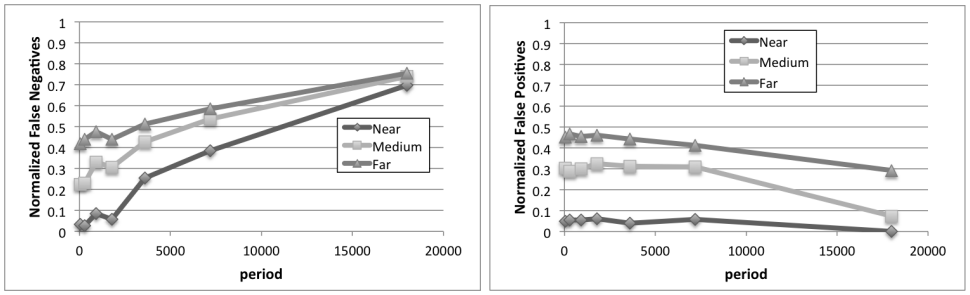
where $4(\mu + \sqrt{\mu})$ is the number of fringes in our square region and $\overline{size_{digest}}$ is measured during execution. We assume a single network hop between adjacent cloudlet servers. The total cost for the cloudlet approach is the sum of these two values. We compare this to the system-level cost of the centralized approach, computed as: $n \times 80\text{B} \times h\,\text{hop} \times \frac{1\text{KB}}{1024\text{B}}$, where the only cost is sending the clients' individual updates



(a) *UT-Real*



(b) *UT-Sim*

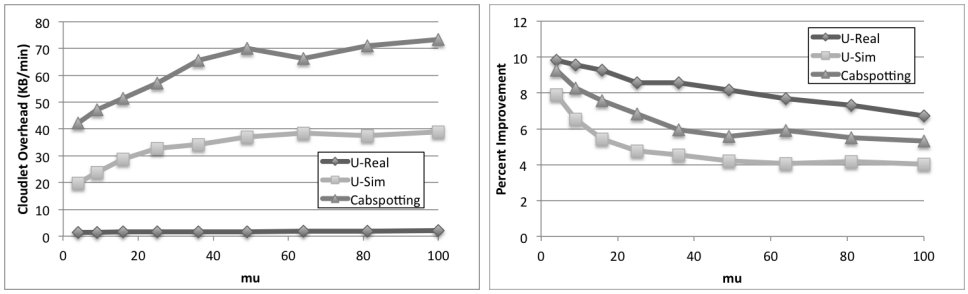


(c) *Cabspotting*

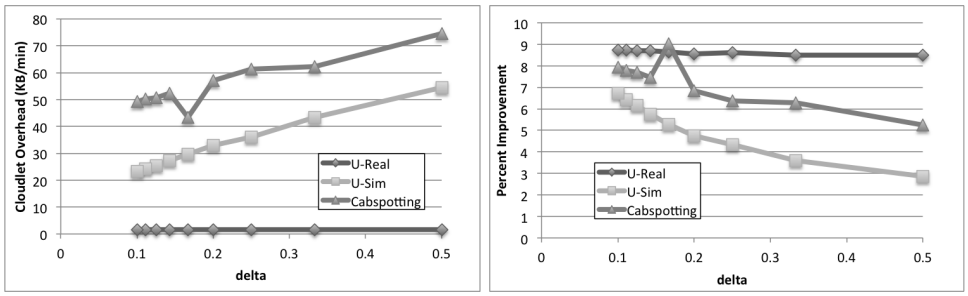
Figure 4.13: Impact of varying the fringe digest update period.

to the central server. In the following results, we used the (conservative) assumption of 10 hops to the central server[14]. Figure 4.14 shows the results, which include both the absolute values of the overhead (measured in KB/min) for the cloudlet based

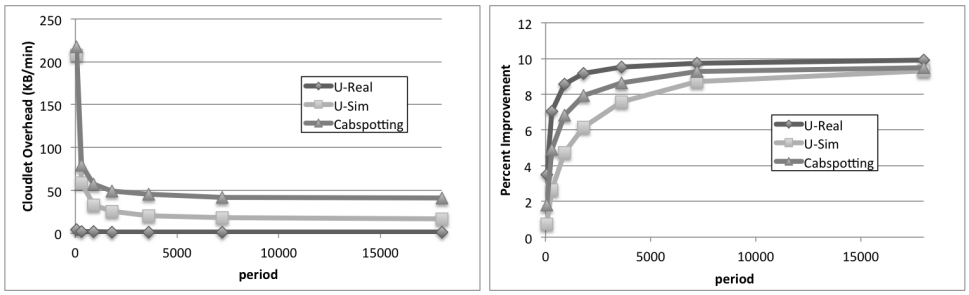
[14]The value of 10 hops is significantly below the values we measured using `traceroute`s to common cloud servers, which were routinely above 15 hops. The number of network hops is not the only (or necessarily "best") measure of cost, but it gives a reasonable measure of the relative costs of these two approaches.



(a) *Varying* μ



(b) *Varying* δ



(c) *Varying* T

Figure 4.14: Cloudlet overhead and comparison to Cloud based approach.

approach and the improvement of the cloudlet based approach over the centralized approach.

Overhead increases with both increasing μ and increasing δ. As μ increases, there are more digests to be sent because there are more fringes. Note, though, that the overhead increases relatively slowly for increasing μ. Overhead increases with δ simply because, as the fringe size increases, there are more resources within the

fringe, so the size of the digest grows. Interestingly, the overhead falls off rapidly with increasing T; for all values of T greater than 60 seconds, we observed relatively low overheads (and therefore significant improvements in comparison to the centralized approach). The plots in Figure 4.14 show one additional interesting phenomenon. The degree of improvement for the *Cabspotting* trace is consistently better than the degree of improvement for the *UT-Sim* data set. In real situations, resources tend to cluster in non-uniform ways, which the cloudlet-based approach is designed to be sensitive to. This is in contrast to the manufactured *UT-Sim* case in which the resources are uniformly distributed with no attention to how real users or resources would be distributed in a real space.

Our evaluation of the cloudlet based proximal resource discovery service has demonstrated the situations in which it can prove beneficial over a more traditional cloud based approach. The cloudlet based approach may not always be the ideal option (e.g., it does not have a high quality of query resolution for "far" queries, or queries about resources that are multiple hops away in the local network). For applications searching for very local resources (i.e., resources in the immediate environs), the cloudlet based proximal resource discovery provides high quality discovery at significantly decreased costs (with respect to the amount of data transmitted within the network in total). The use of digests to aid in discovering resources in neighboring cloudlets can be beneficial, especially when the administrative regions are defined not randomly, but with input about how they match with the real spaces that people inhabit.

4.2.4 Related Work

Existing location-dependent resource discovery approaches can generally be classified into two categories: centralized methods, which rely on a single index to resolve spatial queries, and distributed methods, which can further be decomposed into infrastructure-dependent and infrastructure*less* (i.e., ad hoc) approaches.

Centralized Location-Dependent Discovery. Geographic information systems (GIS) have emerged over the last few decades out of advancements in both database management systems (DBMS) and positioning and tracking systems (e.g., GPS) (see [126] for an excellent survey of this work) and are now an industry standard. A GIS is a collection of software geared at efficiently performing a wide range of operations over geographic data, for example, resolving spatial queries, generating maps, detecting geographic patterns over time, etc. [51]. GIS extensions exist for most modern DBMS and are the enabling technology for location-based services (LBS), which integrate a mobile device's location with other (spatially-dependent) information [154].

Google Latitude[15], Foursquare[16], Facebook Places[17], and Follow Me [173] are cloud-based examples of LBS that enable users to share their location with friends. The NearMe wireless proximity server [84] compares clients' Wi-Fi fingerprints (observed access points and signal strengths) to compute their relative proximity. MoCA (Mobile Collaboration Architecture) [148] is a client-server mid-

[15] `http://latitude.google.com`
[16] `http://foursquare.com`
[17] `http://facebook.com/about/location`

dleware for developing and deploying context-aware applications; MoCA supports mobility by monitoring a mobile client's location and switching to the application proxy (an intermediary between a mobile client and the application server that exists at the edge of a wired network) closest to the user. Similarly, Hydra [150] facilitates mobile pervasive application development by providing mobile agents that follow a user about a pervasive environment and construct a virtual machine that meets a user's current needs based on her location, tasks, number of co-located people, etc.

These approaches employ a single point of lookup to store participants' locations and resolve queries for location-based resources. Ypodimatopoulos and Lippman point out that any system that centralizes user location information by definition compromises user privacy [173], which can have potentially dangerous implications (e.g., burglary[18] and personal information inference [46]). For this reason their Follow Me indoor location sharing service is intended to be implemented at the per-building level instead of at a global level. We further argue that a completely centralized approach fails to meet the requirements of pervasive applications that demand low latency and target densely populated physical spaces where hundreds of thousands of devices may be carried by mobile users (e.g., Body Area Networks [24]) and embedded in the environment (e.g., the Internet of Things [4] and Web of Things [58]). These types of applications are more aptly supported by computing resources that are physically near participating devices and do not rely on a (potentially distant) single globally known resource to manage devices' location information. This flavor of approach both reduces network latency and keeps

[18] `http://pleaserobme.com`

location availability as local as possible.

Distributed Location-Dependent Discovery. The efficient discovery of "nearby" peers is an active area of interest in peer-to-peer (P2P) applications, where nodes interact directly with one another in a decentralized and localized fashion. Zero Configuration Networking (zeroconf)[19] and the serverless messaging extensions of the Extensible Messaging and Presence Protocol[20] (XMPP) both enable automatic discovery of available services on a local area network (LAN). Friends Radar [96] uses XMPP messaging to support P2P location sharing. Likewise, Virtual Cloud [68] employs XMPP messages to facilitate forming on-demand mobile clouds comprising co-located mobile devices. While zeroconf discovery over a wide-area network (WAN) is possible, it requires advanced setup at each client. Our proximal discovery service makes no assumptions about the *type* of network clients are a part of, simply that they are reachable.

Still other P2P applications maintain network overlays and routing tables to leverage *logical* locality. pSense [155] enables discovery of virtually visible peers in position-based massively multiplayer online games (MMOGs) through localized multicast; this eliminates the need to propagate players' location updates to a global resource. Proximal peer discovery could be implemented on top of a decentralized routing and location infrastructure like Tapestry [176] or Chord [160]. These approaches could certainly be applied in scenarios where *physical* locality was the citizen of interest. However, overlay and routing table based approaches require

[19] `http://zeroconf.org`
[20] `http://xmpp.orgextensions/xep-0174.html`

knowledge of at least one peer already in the network who may act as an entrance point for the new peer. In the implementation we present here, we require advanced knowledge of cloudlets and their administrative regions. However, one could envision cloudlet resources existing at the edge of a wired network infrastructure (e.g., in physically dispersed wireless access points) [151], and a device would implicitly interact with the cloudlet resource hosted on the Wi-Fi access point it was currently connected to.

Purely ad hoc approaches are *inherently* localized in both space and time due to the attenuation of radio signals as they propagate. FlashLinQ [171] is a telecommunication technology that enables long-range (approximately two mile) P2P discovery and operates in a licensed 5MHz spectrum. Other existing pieces of work use short range communication to localize (and co-localize) mobile users. Virtual Compass [9], for example, constructs a two dimensional graph of nearby devices via periodic (Wi-Fi or Bluetooth) signal strength measurements. Clearly, ad hoc strategies are advantageous in densely populated spaces as they eliminate the need for a bottleneck centralized resource and keep device interaction entirely local. Nevertheless, the maintenance of accurate routing tables, network overlays, and distributed data structures becomes expensive in scenarios that exhibit heavy churn in formation and high degrees of node mobility.

Distributed Spatial Indexes. Our work aims to combine characteristics from both extremes: we desire the reliability and simplicity of centralized LBS systems and the scalability of entirely distributed approaches. We adopt a similar approach to that of distributed spatial indexing techniques [33, 106], which leverage

the hierarchical nature of a spatial index structure (e.g., an R-Tree [61]) to strategically distribute portions of its hierarchy among networked peers. Rather than distribute portions of a holistic data structure, our proximal discovery service distributes independent computing resources that each employ their own spatial index. Moreover, our discovery service is designed under the assumption that pervasive applications likely require localized device interaction; we dictate that a computing resource monitoring device location information for a spatial region be physically near or within that region. Therefore our discovery service is able to keep location data about proximal peers as local as possible.

4.3 The Gander Application Framework

In this section, we deliver a systems contribution that unifies the goals of the spatiotemporal data provenance implementation (Section 3.1), the rule-based mobile middleware (Section 4.1), and the cloudlet-based discovery service (Section 4.2) within a concrete software framework that enables sharing and querying of spatiotemporally-enriched data via hyper-localized device interactions. We use this framework to implement and deploy a mobile system enabling users to share and search for digital information in their proximate surroundings. In the next section, we conduct a user study to evaluate the system-level performance and user-perceived utility of the framework's distributed search mechanisms.

We have developed the *Gander application framework* as a Java library that embodies the Gander conceptual model described in Chapter 3. Our framework makes no assumptions about the underlying network(s) and transport mechanisms

responsible for connecting Gander devices; instead, it provides interfaces and abstract implementations as cues for developers to create concrete network-specific implementations where necessary. In this section, we present a concrete middleware implementation of the Gander framework for pervasive computing environments that offer a wired/wireless Internet connection. To facilitate Gander's query processing amongst proximal devices in such settings, our distributed Gander middleware is supplemented by the cloudlet-based *Proximal Discovery Service* (*PDS*) [99] introduced in Section 4.2.

We choose to implement the Gander framework for Internet-connected environments in this work 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.

We next describe the design and operation of this specific Gander framework implementation, followed by an assessment of its use in a real-world mobile application (Section 4.4).

4.3.1 System Architecture

Figure 4.15 shows the Gander system architecture. An instance of the *Gander Middleware*, our implementation of the Gander framework, 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 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.

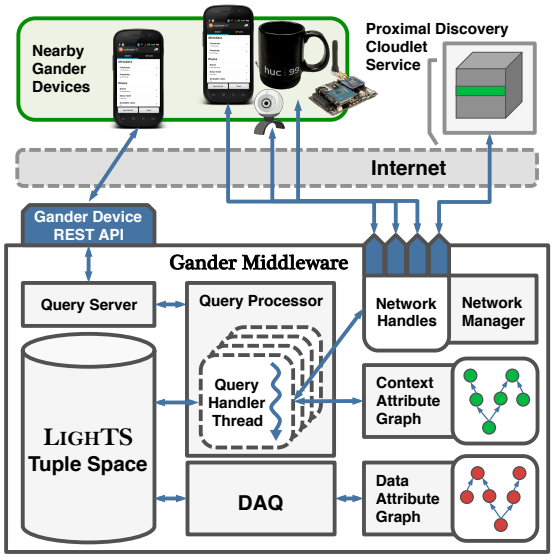


Figure 4.15: The Gander system architecture.

4.3.2 Data Model

The foundation of the Gander data model implementation is provided by the LighTS tuple space framework [8]; this tuple space holds the concrete versions of the datums, i.e., (ν, d) from the conceptual model presented in Chapter 3. LighTS not only provides a flexible means of storing both the application data (ν) and the contextual metadata (d), but also the expressive matching semantics necessary for Gander's query resolution (the function $\mathcal{S}$ in Chapter 3).

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.

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*). The vertices of each graph are `Attribute`s, 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

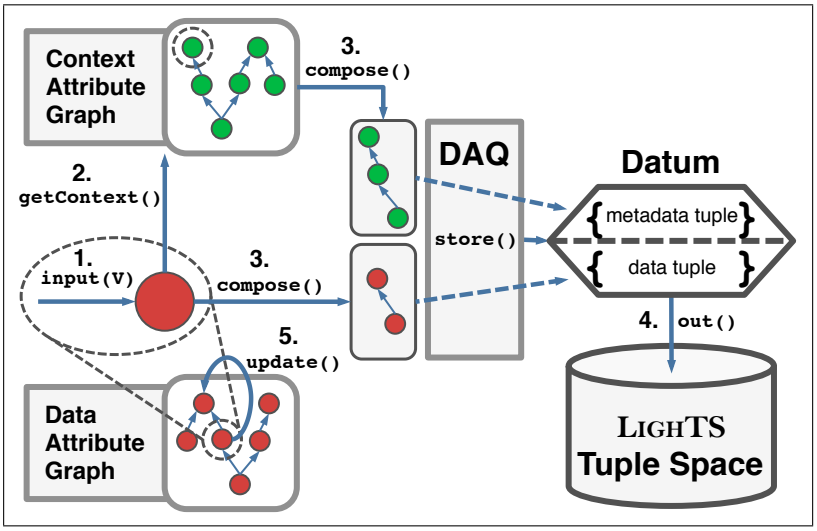


Figure 4.16: The Gander data model.

ambient noise level sensor readings, which may be composed (e.g., aggregated) and mapped to a semantic value (e.g., "quiet" or "low hum"). 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 metadata (d).

Figure 4.16 illustrates how the Context and Data Attribute Graphs manage the generation of structured *datums*. An attribute's state is modified when the associated underlying sensory data changes, triggering the attribute's `input(V)` method. 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 metadata, 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.

4.3.3 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 `RelevanceMetric`s, which each implement a datum comparator, and a `QueryProtocol`. Each Gander `QueryProtocol` 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.

In this work, we opt to implement an Internet (i.e., HTTP) specific middleware. To accomplish this goal we extend the cloudlet-based *Proximal Discovery Service* (*PDS*) introduced in [99] 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.

In our implementation, Gander's spatiotemporal sampling is aided by local instances of the *PDS*. Figure 4.17 illustrates the sequence of events for propagating and resolving a query supplemented by a *PDS* cloudlet instance. 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.

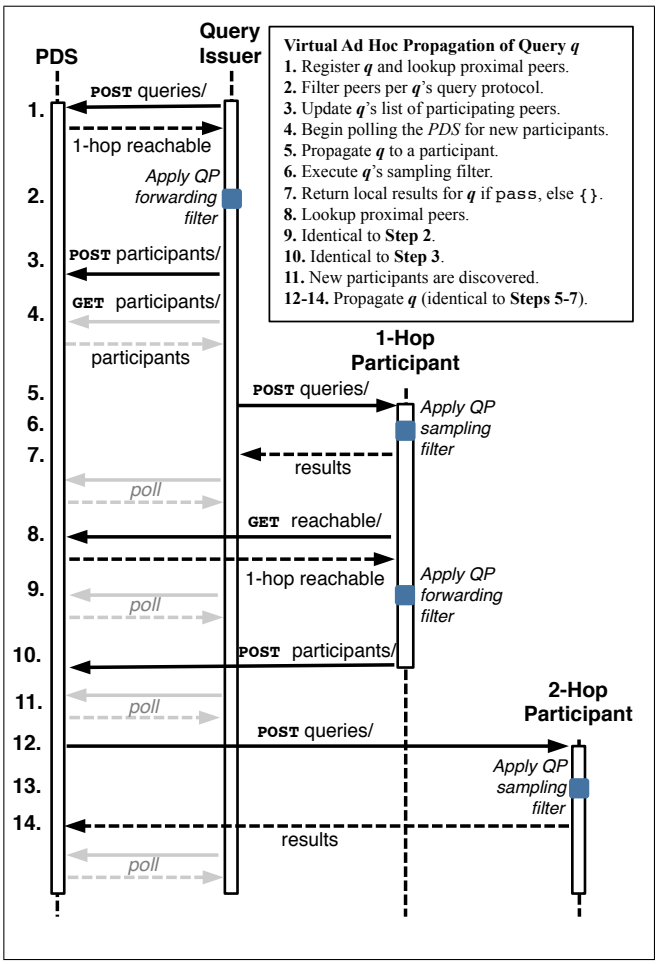


Figure 4.17: Issuing and propagating a Gander query.

Gander targets pervasive computing environments where hundreds, even thousands, of devices exist in very close proximity. Practically, large scale test

beds are challenging and expensive to deploy. Motivated by the desire to assess Gander's performance at a large scale, we next evaluate the performance of the Gander framework and cloudlet-based proximal discovery service in simulated pervasive computing spaces.

4.4 Evaluation through a Real World Deployment

Our ultimate goal is to provide expressive support for real applications to share and search for data in pervasive computing spaces. Therefore, we are targeting large-scale deployments of pervasive computing applications that employ our adaptive approaches to data sharing and search. In previous work [103], we have connected pervasive computing application implementations to the OMNeT++ simulator [165] to enable us to run repeatable experiments using large pervasive computing applications with controlled data generation, search initiation, and mobility patterns. This allowed an interactive user (or users) to be part of the simulation, so search queries could be automatically scripted, provided by the user(s), or both.

Motivated by the desire to evaluate the utility of the Gander search engine within real-world scenarios, we used the results and feedback from our previous study [103] to create *myGander*[21], 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 deployed

[21] `http://mpc.ece.utexas.edu/mygander`

this application for 15 weeks and made it publicly and freely available to members of the UT Austin community. myGander was downloaded a total of 124 times and processed 705 queries issued by 63 devices. Next, we overview the myGander mobile application and use it conduct a user study on the UT Austin campus.

4.4.1 The myGander Mobile Application

Figure 4.18 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 contrainsts on fields (as opposed to creating a freeform query, as in [119])—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[22] to attach contextual information (e.g., device location and Wi-Fi fingerprint) to each user-

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

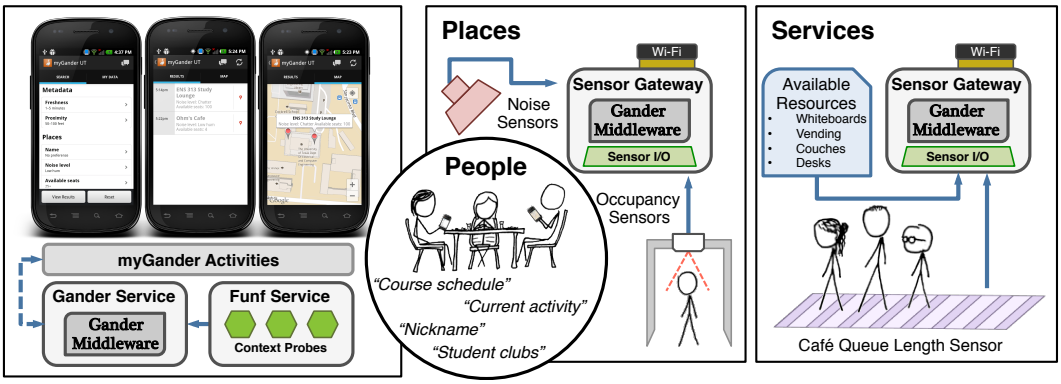


Figure 4.18: The myGander mobile application.

input datum via *Context Attribute Graph* hooks, creating connections between the Context and Data Attribute Graphs as described in Section 4.3.2. 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 an 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 [119]), which each 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`).

4.4.2 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[23]. To assess the user-perceived quality of a Gander query processed with a particular protocol, users were prompted with an in-application pop-up upon performing a search and receiving results[24]. 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?

[23] 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.

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

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.

Table 4.3: 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

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 [122] 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 [122], listed in Table 4.3. $R_{m,Rel}(U_p)$ is a *relevance*[25] 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.

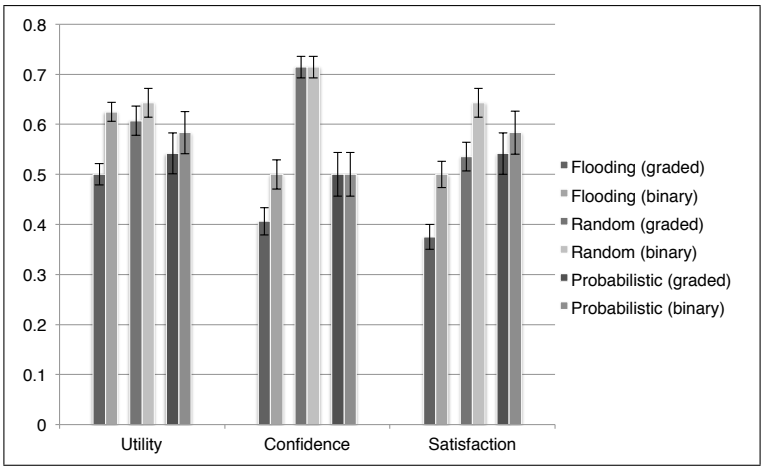


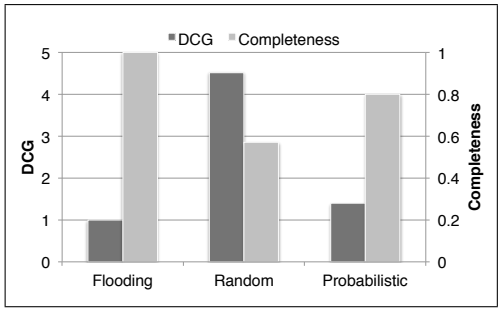
Figure 4.19: myGander user experience metrics per query protocol.

Figure 4.19 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, as Figure 4.19 shows.

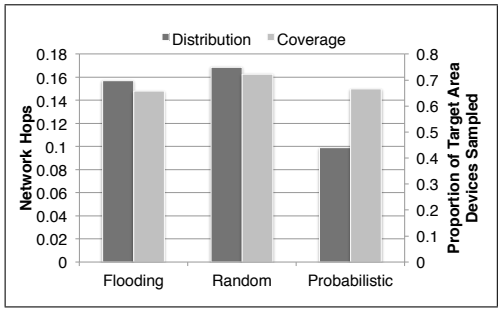
Figure 4.20 reveals system level performance metrics elicited in our user study. We use the same DCG and completeness metrics defined in the previous

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

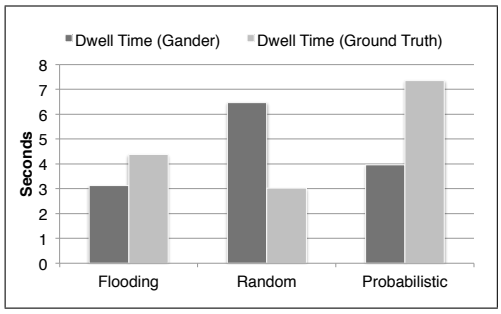
section. We also measure a Gander query's *coverage* as the proportion of target area devices that the query reached and *distribution* as the number of network hops from the query issuer the query traveled. Because our deployment network is relatively sparse, many queries travel very low numbers of hops (often 0). Finally, the *dwell time* is a measure of the amount of time the user spends observing a set of results.



(a) Average result DCG and completeness.



(b) Average query distribution and coverage.



(c) Average myGander result set dwell time.

Figure 4.20: myGander query and result set system metrics.

Figure 4.19 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 [102] 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* (Figure 4.20(b)), yielding results with the greatest DCG (Figure 4.20(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 (Figure 4.20(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 Figure 4.19 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 (Figure 4.20(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.

4.4.3 Lessons Learned

A significant amount of effort went into building, deploying, and servicing the user study application described in Section 4.4.1 and used in Section 4.4.2. 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.

4.4.4 Related Work

Gander diverges from the vision of the Internet of Things [4, 12, 36, 83, 119], which focuses on getting all "things" connected across the globe. Instead, we focus on locally accessible data data that is inherently transient and difficult to centrally index. Furthermore, a key component of Gander is the ability to tailor the application, for example by providing rules that govern the creation and maintenance of spatiotemporal trajectories or by defining the structure of queries posed over the here and now. This style of "user" programming has been explored in similar domains. Extensions to Open Data Kit [63], e.g., ODK Sensors [14], provide program-

ming constructs that simplify access to external sensors connected to smart devices. Dandelion [87] and Gadgeteer [166] have similar motivations. These approaches are at a lower abstraction layer; they represent a way to make data available to the Gander data model. Once that data is available, we focus on how to maintain and move it and how to enable expressive search over it.

The challenges of large-scale deployments of wireless sensor networks and ubiquitous computing systems are frequently cited: these require significant expense and engineering effort. However, as pointed out in [143], real-world deployments and *in situ* evaluations provide context for analysis and uncover design issues; such studies can have a significant impact on usability. To date, most pervasive computing deployments are small-scale, relying on a handful of users in a laboratory setting. Through evaluations of real-world physical deployments of sensor networks, researchers have made several discoveries, including best practices for successful deployment [10, 65], cross-layer approaches to model-driven data acquisition to realize the promise of reduced energy consumption [139], and the use of event-based routing structures that utilize the unpredictable dynamics of the environment [88], and many others. Our final research aim also recognizes the value of large-scale evaluations and deployments to evaluate our data generation and search algorithms and builds upon the lessons learned from large-scale wireless sensor network deployments. We couple these real-world evaluations with a principled use of simulation studies to benchmark, evaluate, and study our approaches to data generation and search in pervasive computing environments.

4.5 Research Contributions

This chapter makes the following contributions:

Research Task 7: We introduce a concrete mobile software framework that provides high-level programming constructs for expressing data-dependent application behavior in the form of reactive rules (Section 4.1). As a use case, we present the design and implementation of a mobile application created using the framework that distributes user content in a device-to-device fashion across multiple network hops. The framework abstracts away low-level implementation details regarding network communication and data serialization enabling developers to focus on the data-driven behavior unique to their application.

Research Task 8: We create and benchmark a resource discovery service for cloudlet-supported pervasive computing environments (Section 4.2). Similar to a cloud computing resource, a cloudlet [152] hosts services that perform the expensive computations required by a mobile application. Unlike the cloud, however, cloudlets are inherently within close physical proximity to the mobile devices that utilize their resources. We leverage cloudlets to implement proximal discovery for applications targeting densely populated and highly mobile physical spaces.

Research Task 9: We provide the Gander Framework, an extensible software framework to support data sharing and query processing in pervasive computing environments (Section 4.3). This framework reifies the formal foundations of spatiotemporal data provenance, Gander's data model, and search protocols

and provides the systems support necessary to evaluate and deploy Gander within real world pervasive computing applications.

Research Task 10: We evaluate the system- and user-level utility of the Gander search engine in a real-world deployment (Section 4.4). Our ultimate goal is to provide expressive support for real applications to share and search for data in pervasive computing spaces. This task aims to explore the impact of our approaches under realistic conditions that reflect the complexity of the rapidly changing physical environment at a large scale, which would otherwise be difficult to observe in controlled simulated settings.

4.6 Chapter Summary

In this chapter, I described the implementations and evaluations of a device-to-device data sharing middleware, a hyper-localized resource discovery service for cloudlet-supported environments, and the Gander middleware, all integral components of the Gander application substrate. Our data sharing middleware directly employs our implementation of *explicit* spatiotemporal data provenance (Section 3.1) and supports data sharing and resource discovery via proximate mobile device interactions. The middleware can easily be deployed on an Android device and integrated into existing mobile applications. Rather than embed data-dependent sharing behavior throughout an application, our middleware simply requires an application define and register *rules* with the middleware that autonomously govern what data is shared, when, and how often. Our resource discovery service targets heavily populated environments possessing an infrastructure of distributed *cloudlets*, dedicated

computing resources that support pervasive computing applications in a nearby physical region. We benchmarked the discovery service's performance varying its operating parameters and evaluated the system-level cost-quality tradeoff versus a cloud-hosted service. Next, I introduced the Gander application framework and described its architecture and operation. The framework's programming interface makes the specifics of the mode of network interaction entirely transparent to the user, allowing search evaluation across both mobile ad hoc network interactions and through localized cloudlets. Finally, we created the Gander middleware, a concrete implementation of the Gander framework, and used it to create and deploy the *myGander* mobile application on the UT Austin campus. We used this mobile application to conduct a user study of 88 participants to assess the performance and user-perceived utility of Gander in a real world setting. Our evaluation demonstrates the feasibility of Gander for spatiotemporal search and explores the relative merits of various Gander query processing protocols with varying spatial and temporal correlation of data and different user requirements. These results provide evidence that enable tailoring of spatiotemporal search deployments to specific data dynamics, application situations, and user preferences.

Chapter 5

Conclusion

In this dissertation, we explored formal and practical techniques with the overarching aim of helping users and applications *share*, *reason about*, and *search* for digital data as they move through a densely populated and rapidly changing information space. In emerging pervasive computing spaces the sheer density of data-sensing devices means that data is "everywhere;" the user's physical surroundings are a rich and dynamic landscape of contextual digital information. There are numerous mechanisms designed to support the nuts and bolts of distributing messages in a device-to-device fashion, even over multiple hops, and even some of these mechanisms are "content aware" (i.e., they distribute the data based on its own semantics). However, few approaches tap into the *contextual history* of the shared data to distribute additional information. We introduced a formal model that enriches application data with its *spatiotemporal provenance*—a history of its explicit (coordinate-based) and implicit (context-based) location over a lifetime of device-to-device propagation. This dissertation presented formal constructs for inferentially deriving global knowledge from spatiotemporal provenance, which we demonstrated may drive runtime decisions that significantly improve routing protocols in dynamic networks. We implemented software components and an extensible middleware that reify these formal mechanisms and enable developers to share and

query spatiotemporally-enriched data using a rule-based programming paradigm.

Sharing and searching for hyper-localized information are fundamental tasks in emerging pervasive computing environments. Existing systems that have enabled query-driven access to localized data provide only capabilities for searching relatively static data instead of the inherently ephemeral data of pervasive computing spaces, which cannot be easily indexed outside of the immediate environment. To address these needs we introduced the Gander conceptual model, which provides a foundation for precisely defining and reasoning about search of the here and now, in the here and now. In this dissertation, we used this conceptual model to explore the impact of the intrinsic spatial and temporal correlations of data in pervasive computing spaces on the performance of *in situ* query processing.

Finally, this dissertation presented the Gander application substrate, which provides a comprehensive implementation of the Gander search engine and both ad hoc and cloudlet-based support for device-to-device interaction. This final aim addresses the concrete systems directions necessary to evaluate the formal foundations proposed by the first two aims and to deploy Gander within real world pervasive computing applications. We presented the Gander mobile middleware, which we employed to create the *myGander* mobile search system and conduct a real-world user study on the UT Austin campus. Our results demonstrate the feasibility of Gander's distributed in-network query processing protocols for spatiotemporal search and provide evidence that will guide development of adaptive query processing mechanisms in future work.

5.1 Future Research Directions

This dissertation opens the door for many exciting future research possibilities. The following is a selection of potential future work enabled by this dissertation.

5.1.1 Distributed Dynamic Network Algorithms and Protocols

This dissertation presented a model of causal data provenance using the time-varying graph formalism [20] for dynamic networks. We used our model to demonstrate methods of augmenting opportunistic routing protocols that reduce transmission redundancy and improve delivery precision. The design and analysis of distributed algorithms and protocols for time-varying graphs is in general an open research area. We envision many more routing-based applications of spatiotemporal provenance. For example, many mobile social network routing mechanisms [80] attempt to exploit regularities in human behavior with predictive mechanisms [174]. Spatiotemporal data provenance could be directly employed to detect such behavioral regularities beyond the scope of mobile nodes' immediate perception (i.e., beyond a single network hop). Our contextual history annotations could also be used to detect nodes with high social centrality [69] or regularity [98], both in a data-dependent (with respect to particular data) or data-agnostic fashion, which have been shown to be effective routing relays. The formal constructs necessary for these mechanisms are already available in our provenance model.

5.1.2 Improving Network Resource Allocation with Contextual History

In this dissertation, we strictly address distributed applications of spatiotemporal data provenance. However, a global (centralized) view of a network's contextual history could be an exceptionally valuable analysis tool. For example, in future cellular networks users' devices may periodically report their (anonymized) collected transmission networks to a centralized resource in the cloud. A network operator may combine devices' partial views to form a historical global view of network state and data propagation, which may be mined for patterns and regularities that assist the operator in more effectively allocating network resources and triggering collaborative device activities (e.g., offloading [142]). From a research perspective, a global view of users' and data's causal history would foster investigations of time-varying human interaction [66] and information diffusion [57] at a scale never before studied. Massive-scale studies like these may be crucial in solving routing and entity discovery challenges [144] in the envisioned Internet of Things.

5.1.3 Adaptive Contextual Query Processing Mechanisms

This dissertation introduced the Gander search engine, whose distributed query mechanisms are sensitive to data's spatial and temporal interdependencies. We envision that data's spatiotemporal annotations may be further exploited to drive adaptive query processing protocols that are able to react on-the-fly to dynamic contextual relations between data. For example, a query should be able to express a festival attendee's search request for a beverage vendor near a food stand with a short line. The constraints expressed within this query represent dynamic real-

world contextual relationships (e.g., a beverage vendor may only be transiently nearby a food stand; food stands' queue lengths fluctuate over time). Altering the order in which the contextual relationships are examined can significantly impact the cost of query resolution within the current operational environment without significantly impacting the intent of the user's query. An exciting research direction extending the Gander search engine would be exploiting spatiotemporally-enriched data to design *adaptive* query processing and optimization techniques for dynamic networks. Similar approaches have been studied in wireless sensor networks [5,6,92], which use heuristics to generate alternative *query plans* and select the lowest-cost plan; query plans are typically represented as graphs that describe ordered sets of relational algebra operations for resolving a query, and structural properties are exploited to optimize the query. However, the problem becomes significantly more challenging in dynamic networks where nodes are mobile. This research direction would directly build on the formal models of spatiotemporal data provenance, which could be married with novel adaptive query mechanisms.

5.2 Dissertation Summary

In summary, this dissertation addresses formal and practical challenges of *sharing*, *reasoning about*, and *searching* spatiotemporal data in emerging pervasive computing environments. We introduce models of historical data provenance, which annotate data with spatial and temporal semantics that illuminate how data moves within dynamic networks. Our solutions represent the first investigation into the design and use of *spatiotemporal* data provenance for mobile and pervasive com-

puting applications. Building on these models we design rule-based programming constructs to lower the development barrier for applications that require transparent access to hyper-localized digital data. We introduce a novel distributed search engine whose query processing mechanisms resolve searches about a user's environment directly with the environment using ad hoc device interactions and cloudlet infrastructures. We provide a concrete software framework that reifies our models of data provenance and pervasive search and conduct a user study that evaluates the effectiveness and perceived utility of our approach. As mobile users' physical spaces become increasingly digitally-accessible, applications will require high-quality support for resolving users' hyper-localized information needs. This dissertation represents a significant step in the evolution of device-to-device networked systems and paves the way for new and exciting research ideas that can be designed leveraging our novel models and built using the Gander framework.

Bibliography

[1] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12:120–139, 2003.

[2] T. Abdelzaher, B. Blum, Q. Cao, Y. Chen, D. Evans, J. George, S. George, L. Gu, T. He, S. Krishnamurthy, L. L. Luo, S. Son, J. Stankovic, R. Stoleru, and A. Wood. Envirotrack: towards an environmental computing paradigm for distributed sensor networks. In *ICDCS*, pages 582–589, 2004.

[3] S. Abiteboul. Querying semi-structured data. In *ICDT*, 1997.

[4] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[5] R. Avnur and J. Hellerstein. Eddies: Continuously adaptive query processing. In *ACM SIGMOD*, 2000.

[6] S. Babu and J. Widom. StreaMon: An adaptive engine for stream query processing. In *ACM SIGMOD*, pages 931–932, 2004.

[7] M. Balazinska, A. Deshpande, M. Franklin, P. Gibbons, J. Gray, S. Nath, M. Hansen, M. Liebhold, A. Szalay, and V. Tao. Data management in the worldwide sensor web. *IEEE Pervasive Computing*, 6(2):30–40, 2007.

[8] 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.*, 5(2):215–231, 2007.

[9] N. Banerjee, S. Agarwal, P. Bahl, R. Chandra, A. Wolman, and M. Corner. Virtual compass: Relative positioning to sense mobile social interactions. In *Proc. of Pervasive*. 2010.

[10] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli. The hitchhiker's guide to successful wireless sensor network deployments. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 43–56, 2008.

[11] K. Beard and V. Sharma. Multidimensional ranking for data in digital spatial libraries. *Int'l. J. on Digital Libraries*, pages 153–160, 1997.

[12] P. Bolliger and B. Ostermaier. Koubachi: A mobile phone widget to enable affective communication with indoor plants. In *Proc. of MIRW*, pages 63–66, 2007.

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

[14] W. Brunette, R. Sodt, R. Chaudhri, M. Goel, M. Falcone, J. Van Orden, and G. Borriello. Open data kit sensors: A sensor integration framework for android at the application level. In *Proceedings of the 10^{th} International*

Conference on Mobile Systems, Applications, and Services, pages 351–364, 2012.

[15] 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.*, volume 2944 of *LNCS*, pages 184–199. Springer Berlin / Heidelberg, 2004.

[16] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. Srivastava. Participatory sensing. In *Proceedings of the* 1^{st} *Workshop on World-Sensor-Web: Mobile Device Centric Sensory Networks and Applications*, 2006.

[17] F. Cabitza, M. Locatelli, and C. Simone. Cooperation and ubiquitous computing: an architecture towards their integration. In *COOP*, 2006.

[18] Q. Cao and T. Abdelzaher. Scalable logical coordinates framework for routing in wireless sensor networks. *ACM Trans. Sen. Netw.*, 2:557–593, 2006.

[19] A. Casteigts, P. Flocchini, B. Mans, and N. Santoro. Measuring temporal lags in delay-tolerant networks. In *IPDPS*, pages 209–218, 2011.

[20] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. In *Ad-hoc, Mobile, and Wireless Networks*, pages 346–359. Springer, 2011.

[21] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott. Impact of human mobility on opportunistic forwarding algorithms. *Mobile Comp.*,

6(6):606–620, 2007.

[22] A. Chaintreau, A. Mtibaa, L. Massoulie, and C. Diot. The diameter of opportunistic mobile networks. In *Proceedings of the 2007 ACM CoNEXT conference*, page 12. ACM, 2007.

[23] R. Chandra, J. Padhye, L. Ravindranath, and A. Wolman. Beacon-stuffing: Wi-fi without associations. In *HotMobile*, pages 53–57, 2007.

[24] M. Chen, S. Gonzalez, A. Vasilakos, H. Cao, and V. Leung. Body area networks: A survey. *Mobile Net. and App.*, 16(2):171–193, 2011.

[25] R.-I. Ciobanu, V. Cristea, C. Dobre, and F. Pop. Big data platforms for the internet of things. In *Big Data and Internet of Things: A Roadmap for Smart Environments*, volume 546 of *Studies in Computational Intelligence*, pages 3–34. Springer International Publishing, 2014.

[26] I. Constandache, X. Bao, M. Azizyan, and R. R. Choudhury. Did you see Bob?: human localization using mobile phones. In *Proc. of MobiCom*, 2010.

[27] 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.*, 26(5), 2008.

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

[29] G. Cugola and G. Picco. Reds: a reconfigurable dispatching system. In *Proc. of the 6th international workshop on Software engineering and middleware*, pages 9–16, 2006.

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

[31] S. Das, H. Pucha, and Y. Hu. Performance comparison of scalable location services for geographic ad hoc routing. In *Proc. of INFOCOM*, 2005.

[32] A. Datta, S. Quarteroni, and K. Aberer. Autonomous gossiping: A self-organizing epidemic algorithm for selective information dissemination in wireless mobile ad-hoc networks. In *Sem. of a Net. World. Sem. for Grid Datab.*, volume 3226 of *Lecture Notes in Computer Science*, pages 126–143. Springer Berlin Heidelberg, 2004.

[33] M. Demirbas and H. Ferhatosmanoglu. Peer-to-peer spatial queries in sensor networks. In *Proc. of P2P*, 2003.

[34] 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*, pages 17–25, 2012.

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

[36] B. M. Elahi, K. Römer, B. Ostermaier, M. Fahrmair, and W. Kellerer. Sensor ranking: A primitive for efficient content-based sensor search. In *Proc. of*

IPSN, pages 217–228, 2009.

[37] M. Erwig, R. Guting, M. Schneider, and M. Vazirgiannis. Spatio-temporal data types: An approach to modeling and querying moving objects in databases. *GeoInform.*, 3:269–296, 1999.

[38] M. Erwig and M. Schneider. Developments in spatio-temporal query languages. In *Proc. of DEXA*, pages 441–449, 1999.

[39] M. Erwig and M. Schneider. Spatio-temporal predicates. *IEEE Trans. on Knowledge and Data Eng.*, 14:881–901, 2002.

[40] P. Eugster, B. Garbinato, and A. Holzer. Location-based publish/subscribe. In *IEEE NCA*, pages 279–282, 2005.

[41] L. Fiege, F. Gartner, O. Kasten, and A. Zeidler. Supporting mobility in content-based publish/subscribe middleware. In *Middleware*, volume 2672 of *LNCS*, pages 103–122. Springer Berlin / Heidelberg, 2003.

[42] L. Fiege, F. C. Gärtner, O. Kasten, and A. Zeidler. Supporting mobility in content-based publish/subscribe middleware. In *Proc. of Middleware*, 2003.

[43] D. Frey and G. Roman. Context-aware publish subscribe in mobile ad hoc networks. In *Coord. Models and Lang.*, volume 4467 of *LNCS*, pages 37–55. Springer Berlin / Heidelberg, 2007.

[44] A. Frihida. Modeling trajectories: A spatio-temporal data type approach. In *Proceedings of the 20th International Workshop on Database and Expert Systems Application*, pages 447–451, 2009.

[45] I. Galpin, C. Brenninkmeijer, A. Gray, F. Jabeen, A. Fernandes, and N. Paton. Snee: a query processor for wireless sensor networks. *Dist. and Parallel Databases*, 29:31–85, 2011.

[46] S. Gambs, M.-O. Killijian, and M. N. del Prado Cortez. Show me how you move and I will tell you who you are. In *Proc. of SPRINGL*, 2010.

[47] D. Gelernter. Generative communication in linda. *ACM Trans. Program. Lang. Syst.*, 7(1), 1985.

[48] J. Goldenberg, B. Libai, and E. Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing letters*, 12(3):211–223, 2001.

[49] L. Gomez, B. Kuijpers, and A. Vaisman. A data model and query language for spatio-temporal decision support. *GeoInform.*, 15:455–496, 2011.

[50] M. Gomez Rodriguez, J. Leskovec, and B. Schölkopf. Structure and dynamics of information pathways in online media. In *WSDM*, pages 23–32. ACM, 2013.

[51] M. Goodchild. *Research Methods in Geography*, pages 376–391. Wiley-Blackwell, third edition, 2010.

[52] T. Griffiths, A. Fernandes, N. Paton, K. Mason, B. Huang, and M. Worboys. Tripod: A comprehensive model for spatial and aspatial historical objects. In *Conceptual Modeling – ER 2001*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2001.

[53] W. Griswold, R. Boyer, S. Brown, and T. Truong. A component architecture for an extensible, highly integrated context-aware computing infrastructure. In *ICSE*, pages 363–372, 2003.

[54] R. Groenevelt, P. Nain, and G. Koole. The message delay in mobile ad hoc networks. *Performance Evaluation*, 62(1):210–228, 2005.

[55] S. Grumbach, P. Rigaux, and L. Segoufin. Spatio-temporal data handling with constraints. In *Proc. of the 6th ACM international symposium on Advances in GIS*, pages 106–111. 1998.

[56] J. Gudmundsson, P. Laube, and T. Wolle. Movement patterns in spatio-temporal data. In *Encyclopedia of GIS*, pages 726–732. Springer-Verlag, 2008.

[57] A. Guille, H. Hacid, C. Favre, and D. A. Zighed. Information diffusion in online social networks: A survey. *SIGMOD*, 42(1):17–28, 2013.

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

[59] X. F. Guo and M. C. Chan. Change awareness in opportunistic networks. In *MASS*, pages 365–373, 2013.

[60] R. Guting, M. Bohlen, M. Erwig, C. Jensen, N. Lorentzos, E. Nardelli, M. Schneider, and J. Viqueira. Chapter 4: Spatio-temporal models and languages: An approach based on data types. In *Spatio-Temporal Databases*, pages 117–176. 2003.

[61] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. of SIGMOD*, 1984.

[62] T. Hägerstrand. Innovation diffusion as a spatial process. 1967.

[63] C. Hartung, A. Lerer, Y. Anokwa, C. Tseng, W. Brunette, and G. Borriello. Open data kit: Tools to build information services for developing regions. In *Proceedings of the 4^{th} ACM/IEEE International Conference on Information and Communication Technologies and Development*, 2010.

[64] H. W. Hethcote. The mathematics of infectious diseases. *SIAM review*, 42(4):599–653, 2000.

[65] T. W. Hnat, V. Srinivasan, J. Lu, T. I. Sookoor, R. Dawson, J. Stankovic, and K. Whitehouse. The hitchhiker's guide to successful residential sensing deployments. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, pages 232–245, 2011.

[66] P. Holme and J. Saramäki. Temporal networks. *Physics reports*, 519(3):97–125, 2012.

[67] D. Huang, X. Zhang, M. Kang, and J. Luo. MobiCloud: Building secure cloud framework for mobile computing and communication. In *Proc. of SOSE*, 2010.

[68] G. Huerta-Canepa and D. Lee. A virtual cloud computing provider for mobile devices. In *Proc. of MCS*, 2010.

[69] P. Hui, J. Crowcroft, and E. Yoneki. Bubble rap: Social-based forwarding in delay-tolerant networks. *Mobile Computing, IEEE Transactions on*, 10(11):1576–1589, 2011.

[70] S. Ilarri, E. Mena, and A. Illarramendi. Location-dependent query processing: Where we are and where we are heading. *ACM Computing Surveys (CSUR)*, 42(3):12, 2010.

[71] T. Imieliński and S. Goel. Dataspace–querying and monitoring deeply networked collections in physical space. In *MobiDE*, pages 44–51, 1999.

[72] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *MobiCom*, pages 56–67, 2000.

[73] L. Isella, J. Stehlé, A. Barrat, C. Cattuto, J.-F. Pinton, and W. Van den Broeck. What's in a crowd? analysis of face-to-face behavioral networks. *Journal of theoretical biology*, 271(1):166–180, 2011.

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

[75] A. Jindal and K. Psounis. Modeling spatially correlated data in sensor networks. *ACM Trans. Sen. Netw.*, 2(4):466–499, nov 2006.

[76] Q. Jones, S. Grandhi, S. Karam, S. Whittaker, C. Zhou, , and L. Terveen. Geographic place and community information preferences. *CSCW*, 17(2–3):137–167, 2008.

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

[78] T. Kalbarczyk, B. Walker&, C. Julien, A. Hennessy&, P. Santacruz, J. Michel, and A. Alford. The breadcrumb router: Bundle trajectory tracking and geographic source routing in dtn. In *Proc. of ExtremeCom*, 2012.

[79] A. Kansal, S. Nath, J. Liu, and F. Zhao. Senseweb: An infrastructure for shared sensing. *IEEE J. of MultiMedia*, 14(4):8–13, 2007.

[80] N. Kayastha, D. Niyato, P. Wang, and E. Hossain. Applications, architectures, and protocol design issues for mobile social networks: A survey. *IEEE*, 99(12):2130–2158, Dec 2011.

[81] L. Kazemi and C. Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In *Proc. of SIGSPATIAL*, 2012.

[82] G. Kollios, D. Gunopulos, and V. Tsotras. On indexing mobile objects. In *Proc. of PODS*, pages 261–272, 1999.

[83] G. Kortuem, F. Kawsar, D. Fitton, and V. Sundramoorthy. Smart objects as building blocks for the internet of things. *IEEE Internet Computing*, 14(1):44–51, 2010.

[84] J. Krumm and K. Hinckley. The NearMe wireless proximity server. In *Proc. of UbiComp*. 2004.

[85] E. Kuhn, R. Mordinyi, H. Goiss, T. Moser, S. Bessler, and S. Tomic. Integration of shareable containers with distributed hash tables for storage of structured and dynamic data. In *CISIS*, pages 866–871, 2009.

[86] F. Li and Y. Wang. Routing in vehicular ad hoc networks: A survey. *Vehicular Technology Magazine, IEEE*, 2(2):12–22, 2007.

[87] F. Lin, A. Rahmati, and L. Zhong. Dandelion: A framework for transparently programming phone-centered wireless body sensor applications for health. In *Proceedings of Wireless Health 2010*, pages 74–83, 2010.

[88] Y. Liu, Y. He, M. Li, J. Wang, K. Liu, L. Mo, W. Dong, Z. Yang, M. Xi, J. Zhao, and X. yang Li. Does wireless sensor network scale? a measurement study on greenorbs. In *Proceedings of IEEE INFOCOM*, pages 873–881, 2011.

[89] C. Lu, G. Xing, O. Chipara, C.-L. Fok, and S. Bhattacharya. A spatiotemporal query service for mobile users in sensor networks. In *ICDS*, pages 381–390, 2005.

[90] M. Lyell, D. Voyadgis, M. Song, P. Ketha, and P. Dibner. An ontology-based spatio-temporal data model and query language for use in gis-type applications. In *GEO*, pages 15:1–15:9. ACM, 2011.

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

[92] S. Madden, M. Shah, J. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *ACM SIGMOD*, 2002.

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

[94] C. R. Mann, R. O. Baldwin, J. P. Kharoufeh, and B. E. Mullins. A trajectory-based selective broadcast query protocol for large-scale, high-density wireless sensor networks. *Telecomm. Sys.*, 35(1-2):67–86, 2007.

[95] B. Martins, M. Silvia, and L. Andrade. Indexing and ranking in geo-ir systems. In *Proc. of GIR*, pages 31–34, 2005.

[96] R. Mayrhofer, C. Holzmann, and R. Koprivec. Friends radar: Towards a private P2P location sharing platform. In *Proc. of EUROCAST*. 2012.

[97] R. Menezes and A. Wood. The fading concept in tuple-space systems. In *SAC*, pages 440–444, 2006.

[98] J. Miao, O. Hasan, and L. Brunie. Leveraging node centrality and regularity for efficient routing in mobile peer-to-peer networks. In *Data Mgmt. in Grid and P2P Sys.*, pages 83–94. Springer, 2011.

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

[100] J. Michel, C. Julien, and J. Payton. Gander: Mobile, pervasive search of the here and now in the here and now. *IEEE J. IoT*, 1(5):483–496, 2014.

[101] J. Michel, C. Julien, J. Payton, and G.-C. Roman. Gander: Personalizing search of the here and now. In *Proceedings of the 8th International ICST Conference on Mobile and Ubiquitous Systems (MobiQuitous)*, 2011.

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

[103] J. Michel, C. Julien, J. Payton, and G.-C. Roman. mygander: A mobile interface and distributed search engine for pervasive computing. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (Demonstrations Track)*, 2012.

[104] J. Michel, C. Julien, J. Payton, and G.-C. Roman. A spatiotemporal model for ephemeral data in pervasive computing networks. In *Proceedings of the 1st International IEEE Workshop on Hot Topics in Pervasive Computing (PerHot)*, 2012.

[105] M. Minami, Y. Fukuju, K. Hirasawa, S. Yokoyama, M. Mizumachi, H. Morikawa, and T. Aoyama. DOLPHIN: A practical approach for implementing a fully distributed indoor ultrasonic positioning system. In *Proc. of UbiComp*. 2004.

[106] A. Mondal, Y. Lifu, and M. Kitsuregawa. P2PR-Tree: An R-Tree-based spatial index for peer-to-peer environments. In *Proc. of EDBT Workshops*. 2005.

[107] M. Motani, V. Srinivasan, and P. S. Nuggehalli. PeopleNet: engineering a wireless virtual social network. In *MobiCom*, pages 243–257, 2005.

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

[109] D. Mountain and A. MacFarlane. Geographic information retrieval in a mobile environment: evaluating the needs of mobile individuals. *J. of Info. Science*, 33:515–530, 2007.

[110] 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.

[111] K.-K. Muniswamy-Reddy, P. Macko, and M. Seltzer. Provenance for the cloud. In *Proceedings of the 8^{th} USENIX Conference on File and Storage Technologies*, 2010.

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

[113] M. E. Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

[114] R. Newton, G. Morrisett, and M. Welsh. The regiment macroprogramming system. In *IPSN*, pages 489–498, 2007.

[115] R. Newton and M. Welsh. Region streams: functional macroprogramming for sensor networks. In *Proc. of DMSN*, pages 78–87, 2004.

[116] D. Niculescu and B. Nath. Trajectory based forwarding and its applications. In *MobiCom*, pages 260–272, 2003.

[117] E. Nordström, P. Gunningberg, , and C. Rohner. A search-based network architecture for mobile devices. Technical Report 2009-003, Uppsala University, January 2009.

[118] Open Geospatial Consortium. OGC. `http://www.opengeospatial.org/`.

[119] 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*, pages 1–8, 2010.

[120] J. Ott, E. Hyytiä, P. Lassila, J. Kangasharju, and S. Santra. Floating content for probabilistic information sharing. *Pervasive and Mobile Computing*, 7(6):671–689, 2011.

[121] A. Panisson, A. Barrat, C. Cattuto, W. Van den Broeck, G. Ruffo, and R. Schifanella. On the dynamics of human proximity for data diffusion in ad-hoc networks. *Ad Hoc Networks*, 10(8):1532–1543, 2012.

[122] P. Pantel, M. Gamon, O. Alonso, and K. Haas. Social annotations: utility and prediction modeling. In *Proc. of SIGIR*, pages 285–294, 2012.

[123] C. Parent, S. Spaccapietra, and E. Zimányi. Spatio-temporal conceptual models: data structures + space + time. In *Proc. of the 7th ACM international symposium on Advances in GIS*, pages 26–33, 1999.

[124] C. Peng, G. Shen, Y. Zhang, Y. Li, and K. Tan. BeepBeep: a high accuracy acoustic ranging system using cots mobile devices. In *Proc. of SenSys*, 2007.

[125] 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.

[126] D. Peuquet. Making space for time: Issues in space-time data representation. *GeoInform.*, 5:11–32, 2001.

[127] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches to the indexing of moving object trajectories. In *VLDB*, pages 395–406, 2000.

[128] T. Phe-Neau, M. Dias de Amorim, M. E. M. Campista, and V. Conan. Examining vicinity dynamics in opportunistic networks. In *PM2HW2N*, pages 153–160, 2013.

[129] T. Phe-Neau, M. Dias de Amorim, and V. Conan. The strength of vicinity annexation in opportunistic networking. In *INFOCOM*, pages 3369–3374, 2013.

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

[131] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser. CRAWDAD data set epfl/mobility (v. 2009-02-24). Downloaded from http://crawdad.cs.dartmouth.edu/epfl/m 2009.

[132] M. Pitkanen, T. Karkkainen, J. Greifenberg, and J. Ott. Searching for content in mobile dtns. In *PerCom*, pages 1–10, 2009.

[133] D. J. Pohly, S. McLaughlin, P. McDaniel, and K. Butler. Hi-fi: collecting high-fidelity whole-system provenance. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 259–268, 2012.

[134] A. Pred. The choreography of existence: comments on hagerstrand's time-geography and its usefulness. *Planning-Related Swedish Geographic Research*, 53(2):207–221, 1977.

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

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

[137] V. Rajamani, S. Kabadayi, and C. Julien. An interrelational grouping abstraction for heterogeneous sensors. *ACM Trans. Sen. Netw.*, 5:27:1–27:31, 2009.

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

[139] U. Raza, A. Camerra, A. Murphy, T. Palpanas, and G. Picco. What does model-driven data acquisition really achieve in wireless sensor networks? In

IEEE International Conference on Pervasive Computing and Communications (PerCom), pages 85–94, 2012.

[140] 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.*, 19(3):630–643, jun 2011.

[141] C. Riolo, J. Koopman, and J. Chick. Methods and measures for the description of epidemiological contact networks. *J. Urban Health*, 78(3):446–457, 2001.

[142] N. Ristanovic, J.-Y. Le Boudec, A. Chaintreau, and V. Erramilli. Energy efficient offloading of 3g networks. In *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, pages 202–211. IEEE, 2011.

[143] Y. Rogers, K. Connelly, L. Tedesco, W. Hazlewood, A. Kurtz, R. E. Hall, J. Hursey, and T. Toscos. Why it's worth the hassle: the value of in-situ studies when designing ubicomp. In *Proceedings of the 9th international conference on Ubiquitous computing*, UbiComp '07, pages 336–353, 2007.

[144] K. Romer, B. Ostermaier, F. Mattern, M. Fahrmair, and W. Kellerer. Real-time search for real-world entities: A survey. *Proc. of the IEEE*, 98(11):1887–1902, 2010.

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

[146] R. R. Roy. Mobile ad hoc networks. In *Handbook of Mobile Ad Hoc Networks for Mobility Models*, pages 3–22. Springer, 2011.

[147] G. Russello, E. Scalavino, N. Dulay, and E. Lupu. Coordinating data usage control in loosely-connected networks. In *POLICY*, 2010.

[148] V. Sacramento, M. Endler, H. Rubinsztejn, L. Lima, K. Goncalves, F. Nascimento, and G. Bueno. Moca: A middleware for developing collaborative applications for mobile users. *Dist. Sys. Online, IEEE*, 5(10):1–14, 2004.

[149] F. A. Samimi, P. K. McKinley, and S. M. Sadjadi. Mobile service clouds: A self-managing infrastructure for autonomic mobile computing services. In *Proc. of SelfMan.* 2006.

[150] I. Satoh. Dynamic deployment of pervasive services. In *Proc. of ICPS*, 2005.

[151] M. Satyanarayanan. Mobile computing: the next decade. *SIGMOBILE Mob. Comput. Commun. Rev.*, 15(2):2–10, Aug. 2011.

[152] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE J. of Pervasive Computing*, 8(4):14–23, 2009.

[153] K. Schelfthout and T. Holvoet. A pheromone-based coordination mechanism applied in peer-to-peer. In *Agents and Peer-to-Peer Computing*, pages 109–132. 2005.

[154] J. Schiller and A. Voisard. *Location-Based Services.* Morgan Kaufmann, 2004.

[155] A. Schmieg, M. Stieler, S. Jeckel, P. Kabus, B. Kemme, and A. Buchmann. pSense: Maintaining a dynamic localized peer-to-peer structure for position based multicast in games. In *Proc. of P2P*, 2008.

[156] C. Scholliers, E. Boix, and W. D. Meuter. Totam: Scoped tuples for the ambient. *Electronic Comm. of the EASST*, 19, 2009.

[157] P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, and M. Satyanarayanan. Scalable crowd-sourcing of video from mobile devices. In *MobiSys*, pages 139–152, 2013.

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

[159] J. Sousa and D. Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. In *WICSA*, 2002.

[160] 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.

[161] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *Networking, IEEE/ACM Transactions on*, 11(1):17–32, 2003.

[162] D. Stovall and C. Julien. Rapid prototyping of routing protocols with evolving tuples. In *DAIS*, pages 296–301, June 2008.

[163] C. Tan, B. Sheng, H. Wang, and Q. Li. Microsearch: When search engines meet small devices. In *Per. Comp.*, volume 5013 of *LNCS*, pages 93–110. Springer Berlin / Heidelberg, 2008.

[164] Y. Theodoridis, T. Sellis, A. Papadopoulos, and Y. Manolopoulos. Specifications for efficient indexing in spatiotemporal databases. In *Proc. of 10th International Conference on Scientific and Statistical Database Management*, pages 123–132, 1998.

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

[166] N. Villar, J. Scott, and S. Hodges. Prototyping with Microsoft .NET Gadgeteer. In *Proceedings of the 5th International Conference on Tangible, Embedded, and Embodied Interaction*, pages 377–380, 2011.

[167] H. Wang, C. C. Tan, and Q. Li. Snoogle: A search engine for pervasive environments. *IEEE J. on Parallel and Dist. Sys.*, 21(8):1188–1202, 2010.

[168] J. Whitbeck, M. Dias de Amorim, V. Conan, and J.-L. Guillaume. Temporal reachability graphs. In *MobiCom*, pages 377–388, 2012.

[169] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: A neighborhood abstraction for sensor networks. In *MobiSys*, pages 99–110, 2004.

[170] M. Worboys. A unified model for spatial and temporal information. *The Computer Journal*, 37:26–33, 1994.

[171] X. Wu, S. Tavildar, S. Shakkottai, T. Richardson, J. Li, R. Laroia, and A. Jovicic. Flashlinq: A synchronous distributed scheduler for peer-to-peer ad hoc networks. In *Proc. of Allerton*, pages 514–521, 2010.

[172] T. S. Yeh and B. Cambray. Modeling highly variable spatio-temporal data. In *Proc. of 6th AustraliAsian Database Confererence*, pages 221–230, 1995.

[173] P. Ypodimatopoulos and A. Lippman. Follow me: a web-based, location-sharing architecture for large, indoor environments. In *Proc. of WWW*, 2010.

[174] Q. Yuan, I. Cardei, and J. Wu. Predict and relay: An efficient routing in disruption-tolerant networks. In *MobiHoc*, pages 95–104, 2009.

[175] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.

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

[177] 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.