The Dissertation Committee for Agisilaos Georgios Ziotopoulos certifies that this is the approved version of the following dissertation:

Design of Platforms for Computing Context with Spatio-Temporal Locality

Committee:

Gustavo De Veciana, Supervisor

Vijay Garg

Christine Julien

Al Mok

Nur Touba

Mauricio Breternitz Jr.

Design of Platforms for Computing Context with Spatio-Temporal Locality

by

Agisilaos Georgios Ziotopoulos, B.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 2011

To the great Greeks of the past, the heroic ones of the present, and the blessed ones of the future.

Acknowledgments

I am debtor both to the Greeks, and to the Barbarians; both to the wise, and to the unwise. St. Paul. Romans 1:14.

The length of the acknowledgments section in a doctoral dissertation should be directly proportional to the time it took to complete it and inversely proportional to its contribution. Well, here is mine.

Working with Prof. G. de Veciana was a blessing which came initially in disguise. It would take me long to describe the various ways I benefited from him but Gustavo's reputation precedes him. Therefore, I will not repeat his well known virtues and talents. I will just express my gratitude to him for his trust and will paraphrase the famous mathematician G. H. Hardy [1] in writing that "I still say to myself when I am depressed, and find myself forced to listen to pompous and tiresome people, Well, I have done one the thing you could never have done, and that is to have collaborated with Gustavo on something like equal terms. It is to him that I owe an unusually late maturity".

Writing this section it is my duty to mention the late Prof. Margarida Jacome. Margarida recruited me to UT Austin and introduced me to the field of pervasive computing. With her enormous engineering intuition she set the

[1] G. H. Hardy "A mathematician's apology".

direction for my thesis. I am sorry she is not still with us to see the fruits of the seeds she planted.

I would also like to thank the members of my PhD committee for their patience, if not anything else. Prof. V. Garg introduced me to distributed systems through his excellent course and gave me an idea of what a relentless researcher looks like. Prof. A. Mok was sensitive enough to hire me as his TA for his course on fault-tolerant distributed systems at the time of Margarida's illness. Interacting with him was one of the most motivating experiences in my academic life. Prof. C. Julien, the local expert on pervasive computing, influenced me positively through the academic environment she created with her lab and her course on Formal Methods. Additionally, she provided the most helpful and well articulated 'criticism' to my thesis. Prof. N. Touba and Dr. M. Breternitz Jr. added helpful diversity to my committee.

In my long journey to graduate school, the two most stable companions were P. Aliferis and D. Peroulis. Friends, mentors, and drivers at different times, they never failed to maintain communication in person, through email, Skype, or on the phone. It is not an exaggeration to say that they were the closest to family I ever had, apart from my family.

At UT the names of two colleagues stand out. V. Rajamani and B. Balasubramanian have formed the 'inner sanctum' of my social circle. They stayed close through thick and thin. Above all, they are two interesting individuals and it still gives me pleasure to discuss with them a multitude of topics. Additionally, Vasanth mentored me successfully through my job search.

Good fortune brought me and Y. Kim in similar paths at the same time. It was a pleasure working in the same lab with him, interning with him and going through the steps of writing a dissertation, defending it and searching for a job.

Being a member of different groups at different times, I benefited from the social and academic interaction with a multitude of brilliant colleagues. Frankly guys, interacting with you has been the most rewarding part of my life in graduate school. I will mention just a few of you. E. Mizan was my only Greek companion at UT but he was enough. J. Jo and S. Banerjee volunteered to hear a preliminary version of my defense talk. V. Shah, V. Joseph, Z. Lu, and B. Rengarajan are peers from Gustavo's group. I hope to continue to interact with them in the future, researching topics in networked systems. Having an active interest in software engineering I was happy to meet and be impressed by H. Wright, T. Petz, T. Hartin, D. Stovall, B. Grant, D. De Angelis, and S. Budalakoti. S. Holloway deserves special mention for his organizing skills and his interest in Greece. B. ElKarablieh was kind enough to endorse my job application. S. Han showed a pleasant interest on my progress.

My personal, educational, and professional history goes long back. It is impossible to mention all the people that deserve it. D. Marco, A. Panagopoulos, P. Oikonomakos, S. Georgoulis, M. Foteinos, D. Tsoukalos, and M. Perdikeas have been close friends. B. Hollingsworth was a fine manager and mentor. My primary school teacher Ms I. Varti showed an interest in my education 'far and beyond duty'. All my high-school mathematicians and especially D. Saravanos

cultivated a long life interest in mathematics which gave me the courage to follow the path leading here. At UT Austin Profs. C. Chase and A. Aziz offered excellent courses and advice at different times. Profs. A.Hero, D. Neuhoff, and my undergraduate advisor Prof. J. Roumeliotis supported me with reference letters. I could never have imagined how helpful Prof. Stafylopatis' course on Data Structures would be for my job interviews.

C. Harris, M. Gulick, S. Cardenas, and RA. Goewey helped me with excellent administrative services. Thank you for making my life easier.

It would be impossible not to mention the Very Reverend Father D. Barr of the St. Elias Orhtodox Christian Church in Austin which embraced me with pastoral love. Dr. J. Tai and my uncle Dr. G. Dellaportas helped me with their scientific expertise.

It is not an exaggeration to admit that most of my successes are due to my family; the biological as well as the national. It is sad to admit that my 'finest' hour is also my 'worst' hour and I have to be separated from them. It gives me comfort to think that they do not need me, they just love me. My sisters Mary and Denia did not let the distance that separates us to separate our lives. I acknowledge the increased burden they have undertaken in my absence. My parents Panagiotis and Margarita have blessed my life through the freedom, respect, and love with which they treated me from day one. Of course, this should not come as a surprise to me knowing that these qualities come from all my grand parents, especially Dionysis and Maria, who I still miss after all those years.

Design of Platforms for Computing Context with Spatio-Temporal Locality

Publication No. _____

Agisilaos Georgios Ziotopoulos, Ph.D.
The University of Texas at Austin, 2011

Supervisor: Gustavo De Veciana

This dissertation is in the area of pervasive computing. It focuses on designing platforms for storing, querying, and computing contextual information. More specifically, we are interested in platforms for storing and querying spatio-temporal events where queries exhibit locality.

Recent advances in sensor technologies have made possible gathering a variety of information on the status of users, the environment machines, etc. Combining this information with computation we are able to extract context, i.e., a filtered high-level description of the situation. In many cases, the information gathered exhibits locality both in space and time, i.e., an event is likely to be consumed in a location close to the location where the event was produced, at a time which is close to the time the event was produced. This dissertation builds on this observation to create better platforms for computing context.

We claim three key contributions. We have studied the problem of designing and optimizing spatial organizations for exchanging context. Our thesis has original theoretical work on how to create a platform based on cells of a Voronoi diagram for optimizing the energy and bandwidth required for mobiles to exchange contextual information that is tied to specific locations in the platform. Additionally, we applied our results to the problem of optimizing a system for surveilling the locations of entities within a given region.

We have designed a platform for storing and querying spatio-temporal events exhibiting locality. Our platform is based on a P2P infrastructure of peers organized based on the Voronoi diagram associated with their locations to store events based on their own associated locations. We have developed theoretical results based on spatial point processes for the delay experienced by a typical query in this system. Additionally, we used simulations to study heuristics to improve the performance of our platform. Finally, we came up with protocols for the replicated storage of events in order to increase the fault-tolerance of our platform.

Finally, in this thesis we propose a design for a platform, based on RFID tags, to support context-aware computing for indoor spaces. Our platform exploits the structure found in most indoor spaces to encode contextual information in suitably designed RFID tags. The elements of our platform collaborate based on a set of messages we developed to offer context-aware services to the users of the platform. We validated our research with an example hardware design of the RFID tag and a software emulation of the tag's

functionality.

Table of Contents

Acknowledgments **v**

Abstract **ix**

List of Tables **xvi**

List of Figures **xvii**

Chapter 1. Introduction **1**

1.1 Conclusions 8

Chapter 2. Design and Optimization of Spatial Organizations For Context Exchange and Surveillance **9**

2.1 Introduction 9

2.2 Modeling context regions and scaling 15

2.2.1 Model for context regions 15

2.2.2 Model for context content of a cell 17

2.3.1 Mobility model 21

2.5.1 Cost model 23

2.7 Analysis of aggregative tessellations under additive context content scaling 24

2.7.1 Selective context 27

2.8.1 Dynamic context 30

2.10 Analysis of aggregative tessellations under generalized context scaling 32

2.10.1 Intuition: the $\alpha = \frac{1}{2}$ case 36

2.10.2 Selective context 37

2.12 Hierarchical organization for context exchange 38

2.13 Assessing the cost of surveillance in ubiquitous environments . 44

2.18 Conclusions and future work 48

Chapter 3. P2P Network for Storage and Query of Spatio-Temporal Flow of Events 50

3.1 Introduction 50
3.1.1 Motivation 50
3.1.2 Centralized solution is undesirable 51
3.1.3 Peer-to-peer distributed infrastructures 52
3.1.4 Related work 52
3.2 Event and query models 56
3.4 Architecture 59
3.4.1 Overlay topology management & routing 60
3.7.1 Associating with the P2P network 64
3.7.2 Data management and query processing 65
3.9 Performance and scalability analysis 68
3.9.1 Scalability analysis 77
3.10 Fault-Tolerance 78
3.12.1 Fault-Tolerant Data Management 86
3.12.2 Event Deletion 89
3.12.3 Handling a new peer joining the P2P overlay 89
3.13.1 Handling a peer leaving the P2P overlay 94
3.13.2 Handling a peer's failure 97
3.13.3 Fault-Tolerant query processing 100
3.14 Conclusions 101

Chapter 4. Addressing the Impact of Non Uniformities in Topology and Traffic 104

4.1 Introduction 104
4.2 Storage pooling 105
4.4.1 How far from its closest peer should an event be stored? 107
4.5.1 On the interplay between storage pooling and fault-tolerance 109
4.7 Congestion load-balancing 112
4.7.1 Modifying the location of peers 114
4.7.2 Adapting the edges among peers to non-uniform traffic . 128
4.7.3 Adapting the number of overlay peers 137
4.8 Conclusion 140

Chapter 5. An RFID-based Platform Supporting Context-Aware Computing in Complex Spaces **141**
5.1 Introduction . . . 141
5.2 Related work . . . 143
5.3 Key abstractions supporting spatially contextualized ubiquitous computing . . . 146
5.3.1 The space type and its attributes . . . 149
5.3.2 The serviced entity: SE . . . 150
5.4 Companion protocol . . . 154
5.5 Prototyping efforts . . . 160
5.5.1 Hardware implementation . . . 160
5.5.2 Platform prototype . . . 161
5.5.3 Timing considerations . . . 162
5.6 Conclusions . . . 163

Chapter 6. Conclusions and Future Work **165**

Appendices **169**

Appendix A. Proofs of Theorems in Chapter 2 **170**
A.1 Proof of Theorem 2.7.1 . . . 170
A.2 Proof of Fact 2.8 . . . 172
A.3 Proof of Theorem 2.10.1 . . . 173
A.4 Proof of Fact 2.11 . . . 176
A.5 Proof of Theorem 2.12.1 . . . 178
A.6 Proof of Theorem 2.12.2 . . . 179
A.7 Proof of Fact 2.16 . . . 180

Appendix B. Propositions Related to our P2P Network **182**
B.1 A proof of Prop. 3.9.1 . . . 182
B.2 A proof of Fact 3.6 . . . 184
B.5 A proof of Fact 3.13 . . . 188
B.6 Generating Kleinberg edges . . . 190

Appendix C. RFID design **192**

Bibliography **196**

Vita **209**

List of Tables

2.1 Aggregative vs. hierarchical organizations. 45

3.1 Definition of the class *EventRecord* for an event e. We assume the existence of the primitive class *Peer*. 79

3.2 Data structures for fault-tolerance. We assume the existence of the primitive classes *Event* and *Peer*. The class *EventRecord* is defined in Table 3.1. 79

5.1 Protocol basic messages 155

5.2 Message times 162

List of Figures

2.1 'Finest grain' and 'aggregative' Voronoi tessellations modeling contextual spaces. 17

2.2 Context aggregation using aggregative tessellations with additive context scaling. 26

2.3 Minimum cell interaction probability p vs. overhead, to achieve savings using aggregation. 30

2.4 Minimum percentage for interest in context vs. overhead, to achieve savings using aggregation. 31

2.5 Context aggregation, $\alpha < \frac{1}{2}$. 34

2.6 Context aggregation, $\alpha > \frac{1}{2}$. 35

2.7 Intuitive explanation for $\alpha = \frac{1}{2}$. 37

2.8 Hierarchical vs. aggregative when $\alpha > \frac{1}{2}, c(V_f) = 1$. 42

2.9 Hierarchical vs. aggregative when $\alpha > \frac{1}{2}, c(V_f) = 10$. 43

2.10 Hierarchical vs. aggregative $\alpha < \frac{1}{2}$. 44

3.1 Event model. 57

3.2 Elements of the P2P architecture. 60

3.3 If l lies outside the strip defined by the dashed lines and $r < \min(|l-\pi_1|_2, |l-\pi_2|_2)$ then all the points in the edge defined by π_1, π_2 are farther away than r to l. In the other case it suffices to check the distance of $\hat{l}$ to l. 68

3.4 Resolution of range query $\mathrm{RQ}(l, r)$. 69

3.5 Idealized grid model. 71

3.6 Average query delay vs. cell side δ. 72

3.7 Average query delay vs. locality. 76

3.8 Data-Structures example 85

3.9 Fault-tolerant query resolution. 100

4.1 Mean storage distance for an infinite square grid (in hops). . . 109

4.2 Replica survival probability for a $n \times n$ square grid. 113
4.3 Move Heuristic, d_{old} is the distance between p and q before the heuristic, d_{new} is the distance after and $\frac{d_{new}}{d_{old}} = f_d$. 117
4.4 Figure shows in logarithmic scale the mean delay to process an event as the spatial intensity of events γ_e grows in the overlay network. Two cases are shown, the first where peers are randomly located in space, and the second after the move heuristic is carried out. 118
4.5 Resulting overlay by applying the 'move' heuristic for $\gamma_e = 2$ (up) and original overlay topology (down) (the facets extending to infinity have been omitted). 120
4.6 Figure shows in logarithmic scale the mean delay to process a query as the spatial intensity of queries γ_q grows in the overlay network. Two cases are shown, the first where peers are randomly located in space, and the second after the move heuristic is carried out. 121
4.7 Resulting overlay by applying the 'move' heuristic for $\gamma_q = 0.32$. The facets extending to infinity have been omitted. 122
4.8 Mean traffic intensity per peer vs. the query intensity on the network without using the 'move' heuristic. The two curves correspond to the 'inner' and 'outer' areas shown in Fig. 4.7. . 123
4.9 Mean traffic intensity per peer vs. the query intensity on the network using the 'move' heuristic. The two curves correspond to the 'inner' and 'outer' areas shown in Fig. 4.7. 124
4.10 Figure shows the mean number of hops/query as the query intensity grows in the overlay network. Two cases are shown, the first where peers are randomly located in space, and the second after the move heuristic is carried out. 126
4.11 Mean delay to process an event/query, $\gamma_e = 0.05$. 128
4.12 Resulting overlay by applying the 'move' heuristic for $\gamma_q = 0.28$ and $\gamma_e = 0.05$ (the facets extending to infinity have been omitted). 129
4.13 A 'dipole' model for non-uniform traffic. Non-uniform traffic from location *source*, which falls into C_t is destined to location *destination*, which falls into C_q. We denote this traffic by a solid arrow. The bigger the length of the arrow, the bigger the *separation* between the source and the destination of the dipole and more peers are affected by the traffic it generates. The dashed arrow, represents a non-uniform edge between peers t and q. The effect of this edge is to remove traffic from peers t_1, t_2 which would have to support this traffic in the edge's absence. 130

4.14 Average traffic delay vs. dipole separation for $\gamma_{\text{non-uniform}} = 0.1$. The traffic consists of events, uniform queries and non-uniform queries. 133
4.15 Average traffic delay vs. dipole separation for $\gamma_{\text{non-uniform}} = 0.25$. The traffic consists of events, uniform queries and non-uniform queries. 134
4.16 Standard deviation of traffic delay vs. dipole separation for $\gamma_{\text{non-uniform}} = 0.1$. The traffic consists of events, uniform queries and non-uniform queries. 135
4.17 Standard deviation of traffic delay vs. dipole separation for $\gamma_{\text{non-uniform}} = 0.25$. The traffic consists of events, uniform queries and non-uniform queries. 136
4.18 Heuristic effectiveness vs. separation for low non-uniform intensity, $\gamma_{\text{non-uniform}} = 0.1$. 137
4.19 Heuristic effectiveness vs. separation for high non-uniform intensity, $\gamma_{\text{non-uniform}} = 0.25$. 138

5.1 Main elements of a complex space realizing our platform . . . 142
5.2 Airport example 149
5.3 Key SE's abstractions 150
5.4 A simplified view of the SE's state machine (reset omitted for simplicity). The notation x/y denotes the message triggering a transition and its response. 156
5.5 Software emulation stack 161

A.1 Behavior of selective context transfer for aggregative organization for $\alpha < \frac{1}{2}$ (left) and $\alpha > \frac{1}{2}$ (right). 178

B.1 Peer p_{old} leaves the P2P network. 186
B.2 Peer p_{new} joins the P2P network. 187
B.3 The thick black solid line represents the Voronoi edge between u and v. The thin dashed disks define the locations for which p_{new} is closer to a point in $ve(u, v)$ than u and v. 189

C.1 Implementation 193
C.2 Implementation (cntnd) 193
C.3 Message coding 194
C.4 Message coding (cntnd) 195

Chapter 1

Introduction

Ubiquitous, or pervasive, computing is considered the third major step in the history of computing, following the main-frame and the personal computer. At the very core of this vision, first articulated by Mark Weiser in [83], ubiquitous computing embraces the multitude of devices characterizing modern computing environments, i.e., sensors, netbooks, smart-phones, PDAs, laptops, etc., and strives for innovative approaches to handle the resulting challenges in human-computer interaction and system scalability. *Interaction transparency* and its pre-requisite *context awareness* are the two most promising system properties identified so far to address the above challenges, see [39, 72, 84, 85].

This thesis focuses on novel platforms for managing context. Our work encompasses issues like gathering, storing, querying and computing context. Production and consumption of context is frequently performed by humans. A key observation that drives our work is that humans tend to be interested in context/information that occurred close to them in space and time. In other words, context exhibits spatio-temporal *locality*. For example, in a smart parking management application, information about a vacant parking spot

may be valid for say, 5 minutes and its area of interest is roughly 100 meters. Clearly, the characteristics of this type of information are radically different than those of, e.g., generic data files, which corresponds to the majority of traffic carried by the Internet today. We will capitalize on context's locality to achieve scalability in our design.

Spatial organizations for context. In Chapter 2 we explore a fixed platform to support context storage and query. We consider a model where locations are randomly placed on the plane representing 'information booths', that store and compress the information/context around them - we call the region associated with each location its 'cell'. Suppose a mobile device queries for context after it enters a cell and acquires context associated with that cell *only*. To evaluate scalability we consider a simple model where the amount of context associated with a cell scales as a function of its area, obviously the bigger the area the more the context it contains, but as we argue in Chapter 2 the rate of growth need not be linear. Indeed, pieces of spatial information at two nearby locations may exhibit correlation, allowing compression and thus reducing the rate of functional form of the growth.

We explore scalability by optimizing the density of cells to best mediate context exchange. Specifically, we optimize a cost function which is a first-order proxy for the bandwidth and energy consumption of mobiles trying to exchange context relevant to their locations as they move in space.

In this thesis we explore two optimization strategies, the 'aggregative' and the 'hierarchical'. In the aggregative approach we try to use as big cells as

possible to amortize the fixed cost associated with connecting with a new cell. Additionally, in our aggregative approach we potentially reap the benefits of context redundancy by exchanging compressed information corresponding to a bigger cell. We explore various cases for the scaling of the average amount of content found in a cell. Our analysis shows that when there exists a lot of information redundancy, i.e., specifically when the average amount of context found in a cell grows slower than the square root of the average cell size, the aggregative approach can achieve unbounded gains over an unoptimized case.

As the average amount of context increases with cell size, the maximum gain through aggregation diminishes. Moreover, the optimization occurs only on a limited scale of aggregation, i.e., there exists an upper bound on the size of the cells used for exchanging context. Our approach stops reducing the cost when the size of the cells used for aggregation is too large. In that case we end up exchanging context not truly local to the user, and thus it is irrelevant. For the special case where the average amount of context grows linearly in the size of a cell, we offer an example where the maximum gain is 42.5%. For the case that context grows super-linearly, we offer an example where the maximum gain is 27.5%.

Our 'hierarchical' approach tries to achieve the best of two worlds when there exists redundancy in the context. It exchanges the unique context pertaining to each cell at a fine spatial granularity and in addition it exchanges shared context among cells using larger, aggregative cells. We prove that when context grows relatively fast, i.e., the average amount of context found in a

cell grows faster than the square root of the cell's size, then the hierarchical approach is asymptotically better than the aggregative one. Otherwise, when context grows slowly, the hierarchal approach will asymptotically produce savings but it will not be better than the aggregative one.

We conclude Chapter 2 with an application of our ideas to the problem of surveillance of a group of mobiles. We propose two strategies, a 'direct' one, ideally suitable for passive RFID technology, which detects the movement of mobiles at designated points corresponding to natural barriers in space, e.g., doorways. Our 'indirect' strategy detects the locations of mobiles through a location positioning system. We prove that for services offered on a 'personalized' scale, i.e., when context is exchanged for a space of roughly human dimensions, the direct approach is preferable to the indirect one.

A P2P platform for managing events. In Chapter 3 we turn our attention to a more concrete representation of contextual information. Events are a well established abstraction to represent real-time information. We propose a formal model for events that encodes spatio-temporal locality explicitly. Using our event model as a building block, we define a special type of query, the 'range' query, which returns all 'current' events inside a disk. This allows us to present other types of queries spanning a wide range of use-cases found in ubiquitous computing, e.g., checking proximity of events to each other, coverage of an area by events, etc. This increases the semantic power of our platform compared to the context exchanges we discussed in Chapter 2.

In Chapter 3 we explore a possible organization of infrastructure to

support context-awareness. We propose a P2P platform for storing events and define associated protocols for storage and query. To promote locality in storage, we require events to be stored at the closest peers to their location. This also increases the scalability of our platform by exploiting the role of context locality. We evaluate the performance and scalability of our platform, by studying the average query delay. To develop intuition we first study a grid model based on rigorous queuing theoretic assumptions. Our main conclusion is that, unlike 'traditional' content distribution P2P networks, our context delivery P2P network has an optimal peer intensity, i.e., more is not necessarily better. Its scalability is limited by the routing cost of traversing a large number of peers.

We verify these idealized models by simulating a prototype of our P2P platform and compare its average query delay to that predicted by our model. The results show convergence of the two views in the case of uniform traffic and topology. Additionally, we quantitatively study the benefits of locality; our simulations show up to a 34% improvement on the average query delay as compared to the case of events not exhibiting locality. We conclude the chapter by introducing fault-tolerance into our design via replication.

Addressing non-uniformities. In Chapter 4 we study the effect of limited storage capacity per peer as well as non-uniform traffic and topology. We propose a modified storage scheme that is motivated by our work on fault-tolerant storage; a peer having reached its storage capacity can be considered a special case of a failed peer. Our modified storage scheme re-distributes

events from overloaded peers and increases the overall storage capacity; we call that *storage pooling*. We are able to compute the blocking probability associated with our modified storage scheme for a grid topology using Erlang's loss formula. We provide a specific example where, according to our formula, our modified scheme reduces the blocking probability from 10% to 0. Moreover, using the same grid model we argue that storage pooling does not significantly increase traffic in our platform since events tend to be stored close to their original locations maintaining locality. Finally, we study the 'survival probability' of an event, i.e., the probability that at least one replica of an event gets stored and survives its life-time. We derive an approximate expression for the survival probability using Erlang's loss formula. Contrary to intuition we argue that more replicas do not always lead to bigger survival probabilities when storage space is limited. Examining specific cases we show that in some cases there exists an optimal number of replicas.

In the sequel we turn our attention to addressing non uniformities in topology and traffic. We classify traffic to two categories: 'local' traffic that is mainly affected by the size of a peer's cell and 'end-to-end' traffic, that is mainly affected by the location of a peer as well as the quality of the edges comprising the topology. An example of local traffic is the traffic resulting from storing, events and an example of end-to-end traffic is the traffic resulting from routing queries among peers. To address non uniformities in the peers' locations, we propose a novel heuristic, the 'move' heuristic, which dynamically adjusts the peers' locations to balance the traffic load they receive. We study

the performance of the move heuristic experimentally and are able to report that under heavy traffic, the heuristic can improve the average traffic delay up to $2-3$ orders of magnitude, stabilizing the network.

To address non-uniformities in the traffic experienced by our network we propose a novel heuristic, the 'congestion edge' heuristic. Our heuristic, dynamically identifies for each peer the source peer that contributes the biggest number of queries towards it and creates a 'shortcut' edge between the two peers. The effect of our heuristic is to significantly alleviate the network from traffic and smooth the traffic reducing the traffic variability. We simulate the effect of our heuristic for two configurations: a 'low' non-uniform traffic and a 'high' one. Our heuristics improve the end-to-end query delay roughly by factors 2x and 5x respectively.

A context-aware application platform with locality. In Chapter 5 we present a concrete example of a platform supporting context and exhibiting locality in users' interests. We focus on context that is generated by spaces that support a hierarchical organization, i.e., spaces contain other sub-spaces that might contain other sub-sub-spaces etc. We encode this structure in a stack that contains a frame for each level of the spatial hierarchy the user is located in. Information relevant to each level is contained in the corresponding frame of the stack. We present a companion protocol leveraging the operations of the spatial stack and investigate how the proposed protocol can be implemented. For our implementation we choose to focus on RFID technology. Passive RFID is a light-weight technology ideally suited for exploiting locality

given its limited range. The limited range of an RFID tag naturally implements the 'interaction transparency' that characterizes ubiquitous computing, [83], and the resulting pruning of information ensures the scalability of the approach. Passive RFID tags are powered wirelessly from the entities querying them, thus they do not suffer from power management issues plaguing other candidate technologies for ubiquitous computing.

1.1 Conclusions

The contributions of this thesis are in the area of ubiquitous computing. We address the problem of designing efficient platforms for managing context with locality. In Chapter 2 we describe a fixed infrastructure used by mobiles for exchanging localized context. In Chapter 3 we describe a P2P platform for storing and querying events based on locality. In Chapter 4 we address the limitations of our P2P platform arising from non-uniformities and limited storage. In Chapter 5 we describe an application platform for managing localized context. Finally, in Chapter 6 we conclude this thesis by examining future directions for our research.

Chapter 2

Design and Optimization of Spatial Organizations For Context Exchange and Surveillance

2.1 Introduction

Context-awareness refers to the ability of applications to recognize the 'environment' in which they are executing. Such a capability is a key prerequisite for ubiquitous computing. Its wide applicability has led to many radically different, but equally valid, manifestations, e.g., services available in a space, sensor values describing the local conditions, personal data about people present in a room for a social-networking application, etc., can all be considered context. Spatial context, e.g., the gas stations in the neighborhood or a person's shopping preferences in a mall to be used by a targeted advertisement application, is a special type of contextual information that exhibits strong locality.

Exchanging this type of data has become increasingly prevalent in ubiquitous computing. Thus, we believe that optimizing such contextual exchanges between mobile users and applications is a problem of primary importance. Our focus in this chapter is on contextual information that is encoded, stored and available for use, but *tied* to an area, i.e., is relevant only

to users/applications at certain spatial locations.

Optimization issues. How often should a mobile user interact with its environment to exchange contextual information? Is it preferable to frequently exchange small amounts of context or bulk amounts of context less often? Should location information come from a shared infrastructure mechanism or is it better for mobile users to individually calculate their locations? These are the types of questions that we aim to address in this chapter. The answers will depend primarily on the scaling characteristics of the information exchanged. Intuitively, one might expect that, e.g., temperature measurements, taken from adjacent sensors will be highly correlated. Therefore, temperature contextual information need not be acquired from all sensors but only from a selected subset of them appropriately distanced from each other. Additionally, the mechanism used to perform the exchange can complicate the picture, e.g., the most efficient ways to perform context exchange using a wireless protocol will be affected by its overheads.

Clearly, the way we perform contextual exchanges will have a major impact on the performance and the lifetime of a ubiquitous computing system. For example, if users carry laptops, PDAs, phones etc., to access contextual information, energy consumption will be a primary concern. Shorter and less frequent exchanges of context result in reduced energy consumption, which in turn will affect the required size of batteries, their cost and the time horizon until pausing to re-charge a battery. One cannot overemphasize the importance of these issues for ubiquitous computing especially as new excit-

ing avenues such as wearable computers and smart-objects emerge, see, e.g., [56, 81]. Understanding some of the trade-offs in designing such infrastructure is the goal of this chapter.

System model. In order to quantitatively assess the relevant merits of particular architectures for context exchange, we propose a simple but novel model, capturing the salient features of such systems; see Fig. 2.1. We assume space is partitioned as a tessellation with each cell corresponding to a region whose context will be acquired together. This is a first-order approximation that can be used as a guide when designing/optimizing spatial organizations for context exchange. Similar approaches have been successfully applied to the problem of network design, see, e.g., [4, 5]. In the sequel we will present a simple taxonomy capturing different ways in which the amount of contextual data associated with a cell may be modeled depending on the character of the associated applications.

Our model assumes a mobile traversing cells of the tessellation; a surveillance mechanism that is part of the space's infrastructure notifies the mobile about traversal events and the mobile in response acquires/transmits the context relevant within each cell, e.g., when a mobile enters a building it acquires the associated context. However, each exchange of contextual data incurs a cost in terms of bandwidth or energy plus some overhead. Our goal is to find organizations that minimize such costs. In particular, we wish to determine the granularity that cells should have. For example, inside a mall one could exchange context at a floor, shop or even finer granularity. Thus,

on one hand, if we exchange context more frequently from small cells we exchange only the context that we need, but might incur a higher overhead. On the other hand, if we exchange context less frequently from larger cells, a mobile user will download irrelevant context from the finest-grain cells it will *not* actually visit in addition to the useful context that comes from cells to be visited, but the overhead is amortized. As such, depending on the manner in which context content scales and the nature of exchange overheads, one might expect to find optimal cell granularities.

Related work. Context acquisition is a problem of recognized importance in the ubiquitous computing community as well as in the networking community, see, e.g., [76],[22] and [55]. However, to the best of our knowledge this is the first work to focus on quantitative assessment, albeit based on simplified models, of spatial context exchange. Geometrical modeling of spaces and their use in ubiquitous computing is not a new idea, see, e.g., [17]. The key difference between this work and others focusing on geometrical modeling is that we use a stochastic model for the typical architectural features. We introduce a few parameters, based on which we compute performance metrics that are representative of a large variety of scenarios.

Context aggregation is widely recognized as an efficient policy for coping with scalability issues challenging ubiquitous systems, see, e.g., [20] and [21]. In [20] context aggregation is modeled with the help of a graph connecting context sources, context operators, and context sinks. Context information flows uni-directionally from sources to sinks along the edges of the graph in the

form of context update events. Key insights drawn from the lessons learned from deploying Solar, a concrete realization of the proposed architecture, are provided. In contrast, our work takes a more detailed view on the scalability characteristics of context to assess the limitations of aggregation and allows for modeling bi-directional context exchanges between mobile users and applications. Our work shares with [70] a concern about context aggregation for resource limited mobile devices. They favor a distributed approach to context aggregation as opposed to our approach, which uses different aggregation scales. The work in [91] and ours share a concern with scaling issues of context but we choose to focus on studying context exchange while [91] focuses on the quality of the acquired contextual information. The work presented in [13] describes the use of context for improving network performance and assumes, as in this chapter, that context data is exchanged by the same mechanisms used for communication. Other research in the area of context acquisition is [88],[82], [89],[42]. A preliminary version of this work has appeared in [92].

Contributions and organization. The key contributions and organization of this chapter are summarized as follows. In Section 2.2 we formally propose a simple first-order stochastic geometric model, based on cells from a random Voronoi tessellation, for a spatial organization of context exchanges to/from mobile users/terminals. In order to argue quantitatively about the relevant merits of different architectures, we also propose a taxonomy for how context content scales with the area of a cell for various applications.

In Sections 2.7 and 2.10 we consider a flat organization, which aggre-

gates contextual data via cells and exchanges the associated data as a batch when a mobile traverses a cell. For this scenario we show that when the average context content of a cell scales slower than the square root of its average area then aggregative organizations increasingly reduce overall costs, i.e., bandwidth/energy. However, when the amount of cells' context grows faster than the latter then it is either not beneficial or there is an optimal granularity for aggregation which depends critically on the exchange overheads and the characteristics of context scaling. In other words, the case where context content scales roughly as the square root of the area, is a critical case in considering optimization of context exchanges to mobile users.

In Section 2.12 we consider hierarchical organizations for context exchange. In particular, if the context content of a cell scales sub-linearly with the area, one expects to have contextual redundancy in the space. Thus, it is natural to exchange shared context using a coarse organization of data, while data specific to a given location is exchanged at finer scales. We will show that such hierarchical organizations are indeed beneficial, but once again the benefits relative to aggregative organizations depend critically on the context scaling characteristics. In Section 2.13 we elaborate on different mechanisms to surveil a space's cells and consider their contribution to the energy costs. The chapter concludes with Section 2.18.

2.2 Modeling context regions and scaling

Spatial context is usually formed around designated points of interest in the environment, e.g., information about art exhibits targeted to visitors of a museum. The corresponding regions formed are hardly ever regular, usually the more the context in a region the bigger the size of the region formed around it. For example, art masterpieces are often allocated more space than other exhibits. The resulting partition of the space is very similar to the Voronoi tessellation formed by the points of interest as nuclei. To express the multitude of possible configurations of regions, a stochastic approach is needed.

Stochastic geometry, [79], has recently proven to be a useful tool for modeling the architecture and performance of communication networks as well as the role of mobility, see, e.g., [4, 5]. The general idea is to develop simple first-order models that depend on a few parameters and that capture the salient features of the problem at hand, allowing one to roughly consider optimizing system designs. This is the character of the model we consider below.

2.2.1 Model for context regions

We shall start by considering a non-hierarchical, 'flat', partition of the environment into cells. After a user crosses a cell's boundary, an exchange of context takes place. The exchange might happen at any later point as long as the user is inside the cell. The cell's localized context is transferred to the user, and the user's cell-specific context is transferred to the application(s) serving the cell. We model such a partition based on the cells of a Voronoi

tessellation induced by a homogeneous Poisson Point Process, on the plane which we very briefly describe next, see additionally [79]. The geometry of spaces found in the real world is far too complex to be described by a single model. We think that homogeneous stochastic Voronoi tessellations form a *reasonable* first-order model controlled by a single parameter that is amenable to optimization.

A Poisson Point Process P on the plane with intensity λ is defined as a random set of points $P = \{p_0, p_1, \dots\}$ such that:

- for every set $S \subset \mathbb{R}^2$ the number of points in it follows a Poisson distribution with rate $\lambda|S|$, where $|S|$ is the area of S;
- and for $S_1, S_2 \in \mathbb{R}^2$ such that $S_1 \cap S_2 = \emptyset$ the number of points in S_1 and S_2 are statistically independent.

The Voronoi tessellation $V(P) = \{C_{p_0}, C_{p_1}, \dots\}$ corresponding to the Poisson Point Process P is defined as a collection of closed and convex cells C_{p_i} covering the plane where

$$C_{p_i} = \{p \in \mathbb{R}^2 \ \mid \ |p - p_i| \leq |p - p_j|, \forall p_i, p_j \in P\}.$$

The intensity λ of the Poisson Point Process captures with a single parameter the granularity of the cells of a tessellation – the average area of a cell is given by $\frac{1}{\lambda}$. Higher values of λ lead to finer grain cells, while lower values of λ correspond to a tessellation with coarser cells. Additionally, we

shall assume that a tessellation induced by a Poisson process P_f with intensity λ_f models the natural, *finest grain*, spatial organization of context in the environment. We consider a second *independent* Poisson process, P_a with rate $\lambda_a < \lambda_f$, modeling a coarser *aggregative* view to study the potential benefits of exchanging context from larger cells. For the remainder of the chapter we will refer to these tessellations as the 'finest grain' and 'aggregative' tessellations respectively, these are exhibited in Fig. 2.1.

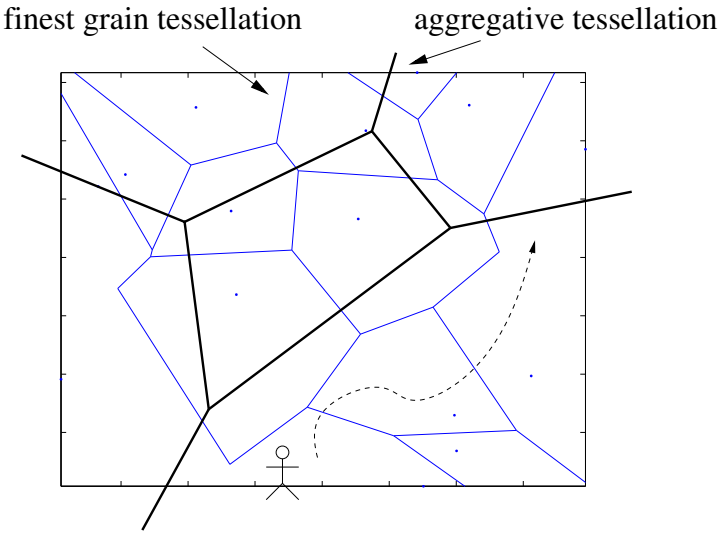

Figure 2.1: 'Finest grain' and 'aggregative' Voronoi tessellations modeling contextual spaces.

2.2.2 Model for context content of a cell

Each cell of a tessellation is associated with a certain amount of context to be exchanged. In general, this amount may depend on the size and shape of the cell. For example, a cell with bigger area is expected to have a higher

number of services in it. Or, in the case of a library or supermarket, contextual content may depend on the perimeter of the shelves storing books, items, etc.

Definition 2.2.1. Context content function. *The context content function* $c : K \to \mathbb{R}^+$ *where* K *denotes the set of bounded convex sets, models the amount of context associated with a region in the plane. We assume this function is translation invariant.*

Depending on the specifics of the application considered, the amount of context content can refer to a cell's context transferred to a mobile and/or a mobile's cell-specific context that is transferred to the application(s) serving the cell.

Definition 2.2.2. Context scaling. *Consider* $A \in K$*, we say a context content function* $c(\cdot)$ *is:*

- additive *iff* $c(A) = \sum c(A_i)$;
- sub-additive *iff* $c(A) < \sum c(A_i)$;
- super-additive *iff* $c(A) > \sum c(A_i)$;

for any partition $A_1, \ldots, A_n \in K$ *of* A*. Note that since* $c()$ *is translation invariant, an additive context content function must be proportional to the area of a set.*

Examples of additive, sub-additive and a super-additive context content functions are: $|A|$, $\sqrt{|A|}$, and $|A|^2$ where $|\cdot|$ denotes the area of a set.

A sub-additive context content function might arise in situations where there is spatial redundancy in contextual information, e.g., as might be the case when context comes from geographically adjacent sensors or cells that are part of a shared hierarchical structure, e.g., the same building. In this case, shared context can be 'compressed' resulting in a context content function that is sub-additive. An example of an additive context scaling would be a situation where a number of services are spatially deployed, say with intensity 1 service per unit area. Consider the context content of a cell with area $|A|$. If we merely want to collect the status of the services in the cell, e.g., operational or not, the amount of context per region would be proportional to its area $|A|$. However, if contextual information should include both the status and location of each service we will need information proportional to $\log_2(|A|) * |A|$, since to describe the location of a service inside a space of area $|A|$, within a precision of a unit area, we need at least $\log_2(|A|)$ bits. This then is an example of a super-additive context content function.

In practice, for complex environments context content functions may grow arbitrarily with cell size, i.e., they need not fit neatly into the above taxonomy. Our idealized models for the context content function capture only some basic characteristics of such systems.

For mathematical ease and to capture a range of possible context content functions, we introduce the following assumption.

Assumption 2.3. Context content model. *The context content of a cell is a function of its shape. The average context content in a typical cell (as seen by a*

mobile) in the aggregative Poisson Voronoi tessellation $V(P_a)$ with associated intensity λ_a, is denoted by $c(V_a)$ and given by

$$c(V_a) \triangleq \frac{c(V_f)\lambda_f^\alpha}{\lambda_a^\alpha}, \quad \textit{where } \alpha > 0,$$

and $c(V_f)$ is another constant interpreted as average context content of a typical cell in the finest grain Poisson Voronoi tessellation $V(P_f)$ with intensity $\lambda_f > \lambda_a$.

Note that above we refer to a *typical* cell as a cell seen by a mobile where we will assume in the sequel that mobiles have stationary trajectories independent of the tessellation. Under these circumstances one can show that the area of typical cells is proportional to $\frac{1}{\lambda_a}$, i.e., the area of a randomly sampled cell under the Palm probability. To avoid these considerations we directly model the typical average context content as seen by mobiles.

The polynomial model chosen is continuous at $\lambda = \lambda_f$ and fulfills through a single parameter, α, our stated assumption of expressing various context content scalings that depend on the shape of an average cell, i.e., the area scaling as $\frac{1}{\lambda}$, the perimeter or diameter scaling as $\frac{1}{\sqrt{\lambda}}$, etc. In practice, the exact scaling of the context content function can be quite complex, but we think it is unlikely to be exponential. Polynomial functions are *reasonable* first order bounds for the context content scalings of most of the use cases we have identified. Inspired by use cases from the environmental sensor domain, we observe that there is correlation in spatial context. Use cases from

the monitoring, control or social networking domain exhibit combinatorial context scalings. These observations corroborate our choice of polynomial context content scaling.

The parameter α depends on the context content characteristics of a particular space/application. Intuitively, the higher the amount of spatial redundancy in the relevant context, the lower the value of $\alpha < 1$. For $\alpha = 1$ the average amount of context content in a typical cell is proportional to the average area of a cell of the $V(P_a)$ tessellation. More generally, for $\alpha > 0$ the context content growth is proportional to a power of the average area – with cells of higher average area, containing more content, but scaling in different ways. Our results apply to all context content scalings, but the focus is mainly on $\alpha \leq 1$.

Our Poisson-Voronoi model for cells allows for a stochastic amount of context content in each cell through our assumption that context is a function of the shape of each cell. Assumption 2.3 models the *average* amount of context content in an aggregative cell and will be used for optimizing our chosen cost function.

2.3.1 Mobility model

The time instances at which context exchanges happen depend on the specifics of each application. For example, a ubiquitous computing application that serves a particular area and operates based on proximity will perform context exchanges as soon as the mobile is within a certain range. The intensity

of such events depends on the specific characteristics of users' mobility. We shall assume a generic homogeneous model for mobility.

Assumption 2.4. User mobility. *We assume mobiles are initially distributed as a Poisson process with intensity λ_0, their motions are* stationary *and* independent *with mean velocity v. A user's trajectory is assumed to be sufficiently smooth, i.e., continuous and piecewise differentiable. The context associated with a cell is exchanged once when a user is inside the cell.*

Recent advances in mobility modeling suggest that human mobility might follow something akin to Levy-random walks, see, e.g., [64]. Our assumption on the users' mobility is fairly generic and acceptable. A more critical concern with the model is the assumption that the mobility patterns are independent of the spatial organization of contextual information. Most likely, mobility and context content would be linked to actual physical structures. This is a simplification required to attempt to study some of the fundamental properties of the problem. Under this assumption one can show the following fact, see, e.g., [5].

Fact 2.5. *The intensity of cell boundary crossings of a homogeneous Poisson Voronoi tessellation with rate λ seen by a typical user moving at mean speed v is*

$$\frac{4 * v}{\pi}\sqrt{\lambda} \quad \textit{crossings/unit time.} \tag{2.1}$$

We will use the above result to compute the rate of context exchanges as mobiles acquire the context of a cell once according to our mobility assumption.

2.5.1 Cost model

The nature of the ubiquitous computing paradigm is such that communication will take place via a wireless medium. It is plausible to define the cost associated with an architecture for context exchange based on the bandwidth or energy expended to perform such exchanges. As a *first-order* approximation, both bandwidth and energy might be roughly proportional to the total amount of context exchanged, including for example, protocol and packetization overheads. The following model captures these salient features.

Assumption 2.6. Cost model. *The cost to exchange d units of contextual data between a mobile and a cell is*

$$h + K * d. \tag{2.2}$$

where h, K are constants independent of d. We assume the energy cost for exchanging context is, to a first order, proportional to the amount of data and overhead.

The parameter K can model overheads that are proportional to the amount of data, e.g., packet overheads. In the sequel we will assume without loss of generality that $K = 1$. The effect of $K \neq 1$ can be evaluated by scaling the context content function in Assumption 2.3. The parameter h can model, e.g., fixed protocol overheads, the cost to authenticate with a new cell, the cost to wake-up a terminal from sleep mode, etc.

2.7 Analysis of aggregative tessellations under additive context content scaling

In this section we explore the benefits of using an aggregative organization to perform bulk context exchanges versus doing this at the finest grain. Recall that these two organizations are modeled via a coarse aggregative tessellation $V(P_a)$ with intensity λ_a and finest grain tessellation $V(P_f)$ with intensity λ_f where $\lambda_a < \lambda_f$. Given that context exchanges take place once per cell crossed, under the finest grain organization users/mobiles exchange context more often than in the aggregative case.

In this section we focus on a simple case. We assume that the context content function is additive and so proportional to the *average* area of an aggregate cell $1/\lambda_a$. In particular we will take $\alpha = 1$ in Assumption 2.3 and model the typical context content seen by mobiles at different levels of granularity as

$$c(V_a) = \frac{\lambda_f c(V_f)}{\lambda_a}. \tag{2.3}$$

The key idea for our analysis is simple. Under our assumption for users' mobility, the intensity of cell boundary crossings, and thus of context exchanges, is proportional to the square root of the intensity of the cells. Each context exchange corresponds to an average cost including overheads and data exchanged. Thus, in the case of the aggregative organization, the total cost incurred per unit time is proportional to

$$\sqrt{\lambda_a} * (h + c(V_a))$$

with a similar form for the finest grain case. In order to have cost savings under the aggregative organization versus the finest grain the following inequality must hold

$$\sqrt{\lambda_a} * (h + c(V_a)) < \sqrt{\lambda_f} * (h + c(V_f)) \tag{2.4}$$

where $c(V_a)$ is related to $c(V_f)$ by Eq. 2.3. The following result, which is derived in Section A.1 summarizes when aggregation is indeed beneficial.

Theorem 2.7.1. *Under Assumptions 2.4,2.6 and an additive context content function, i.e., Assumption 2.3 with $\alpha = 1$, the aggregative organization with intensity λ_a can achieve a reduced cost relative to the finest grain organization with intensity λ_f if $c(V_f) < h$ and λ_a satisfies*

$$\lambda_a \in \left(\lambda_f (\frac{c(V_f)}{h})^2, \lambda_f \right). \tag{2.5}$$

The optimal rate for the aggregative tessellation that minimizes the cost is given by

$$\lambda_{a,opt} = \frac{c(V_f)}{h} \lambda_f. \tag{2.6}$$

Let x denote the overhead ratio $x \triangleq \frac{h}{c(V_f)}$. Then the maximum relative cost reduction is given by

$$|1 - \frac{\sqrt{\lambda_{a,opt}}(h + c(V_a))}{\sqrt{\lambda_f}(h + c(V_f))}| = 1 - \frac{2\sqrt{x}}{1 + x}. \tag{2.7}$$

We can interpret this result as follows. The aggregated context is beneficial only when the average context content per cell $c(V_f)$ of the finest grain tessellation is less than the overhead h. This might have been expected since

one can amortize overheads through aggregated context exchanges. Note, however, that using increasingly aggregated contextual regions eventually hurts. Indeed there is a lower bound λ_a if cost reductions are to be achieved. Intuitively this lower bound can be explained as follows. If aggregative cells are too large, one will be exchanging context to users that is not actually relevant to them, i.e., associated with spatial regions they will in fact not visit; in such cases finer grain organizations are best. Observe that the optimal rate for the aggregative tessellation is always inside the allowable region since for $\frac{c(V_f)}{h} \in (0,1)$ we have that $1 > \frac{c(V_f)}{h} > (\frac{c(V_f)}{h})^2$.

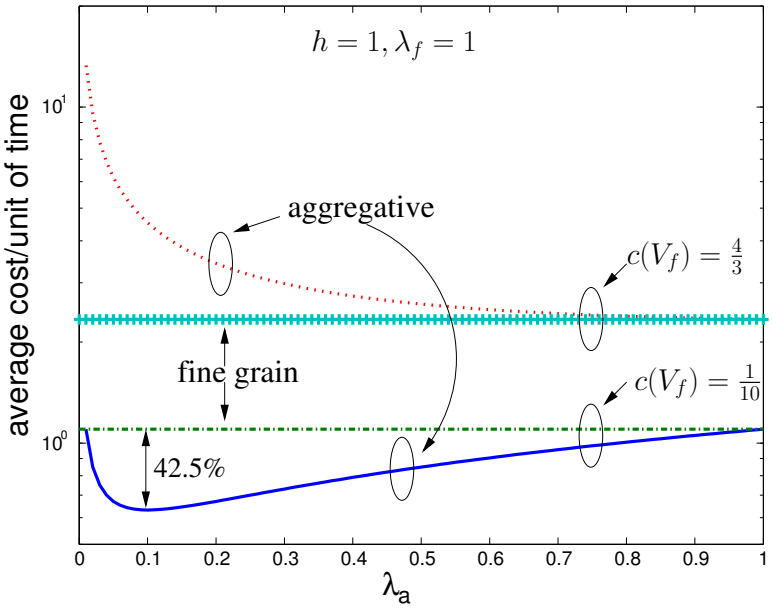

Figure 2.2: Context aggregation using aggregative tessellations with additive context scaling.

Fig. 2.2 exhibits an example of the results. For $c(V_f) > h$, e.g., $c(V_f) =$

$\frac{4}{3}, h = 1$, the average cost per unit of time for a typical user using aggregative cells always exceeds that of the finest grain organization. For the case where $c(V_f) < h$, e.g., $c(V_f) = \frac{1}{10}, h = 1$, one can see the interval for λ_a in which using an aggregative tessellation is beneficial. The left and right boundaries of the interval as well as the location of the optimal rate of the aggregative tessellation are as predicted by Theorem 2.7.1. The relative cost reduction by using the aggregative tessellation with the specific parameter values used in Fig. 2.2 is 42.5%.

The results of Theorem 2.7.1 can be constrained by the limited space resources found in typical mobile devices. Excessively large aggregative organizations may contain too much context content per cell to be downloaded to a mobile device. A designer trying to define the optimal achievable scale of aggregation should choose the biggest aggregative cell whose context fits in the space provided by the mobiles to be used. This policy is guaranteed by the single mode of the cost function in Theorem 2.7.1.

2.7.1 Selective context

Up to now we have assumed that a context exchange occurs for each cell of the underlying tessellation crossed by a mobile. However, this need not hold in practice. Indeed a user/application interacting with a finest grain organization could select exactly in which cells/services it has an interest. As a result, the cost associated with exchanges from the finest grain tessellation may be lower. A simple enhancement to our model capturing this phenomenon

would be that a context exchange with a finest grain cell occurs only with probability p, where p captures the users' selectivity. Additionally, we assume that only a percentage q of the context of each finest grain cell is acquired. We further assume that p and q are independent of each other. The cost of the aggregative tessellation remains unaffected.

In that case the following fact holds, for a proof see Section A.1.

Fact 2.8. *Under the assumptions of Theorem 2.7.1 and assuming that a mobile interacts with the cells of the finest-grained organization with average probability* $p \leq 1$ *and acquires a fraction* $q \leq 1$ *of a cell's context, a necessary condition for an aggregative organization to achieve savings compared to a finest-grained organization is*

$$p * (x + q) > 2 * \sqrt{x}, x > 1 \tag{2.8}$$

where x *is defined as in Theorem 2.7.1.*

Observe that for $p = q = 1$ the condition in Fact 2.8 is consistent with $x > 1$ as required by Theorem 2.7.1 and ensures that the maximum relative cost in Eq. 2.7 is less than 1. The optimal scale of aggregation is still given by Theorem 2.7.1. In the interest of generality, we will present the results for the new range over which aggregation is a win, in Section 2.10 for generic $\alpha \neq 1$.

The interpretation of the previous fact is that if the context associated with finest grain cells is of sufficient interest to users, one can still benefit from exchanging the aggregated context. Fig. 2.3 shows the relationship between the probability of interest p and x, for different values of q. In the $x > 1$

regime where there is more overhead than context associated with a finest-grain cell, the minimum necessary value of p, diminishes as the overhead increases. Indeed, when the overhead is extremely high, using an aggregative tessellation is the correct strategy, and the actual amount of context exchanged does not play a key role. But q, the percentage of context of interest in a finest-grained cell, plays a key role. If q is too low, it might be impossible to achieve savings, see the curves above the $p = 1$ line in the invalid region. For large values of x the role of q is diminished, observe the convergence of the curves in Fig. 2.3. In that case, there is an excessive amount of overhead that dominates the cost, and the amount of context transferred does not matter much. Additionally, observe that for $q = 1$ we can always achieve savings through an aggregative tessellation.

Fig. 2.4 shows the relationship between the minimum value of q necessary to be acquired and x, for different values of p for an aggregative tessellation to be beneficial. Observe that as the overhead increases, the minimum percentage of context necessary to be acquired drops. Indeed, when there is an excessive amount of overhead, it dominates the cost and the amount of context transferred does not play a key role. Note, that for low values of overhead it might not be feasible to achieve savings through an aggregative tessellation if the probability of interaction with the finest-grained tessellation is too low, see the curves above the $q = 1$ line in the invalid region. Additionally, observe that for $p = 1$ we can always achieve savings through an aggregative tessellation.

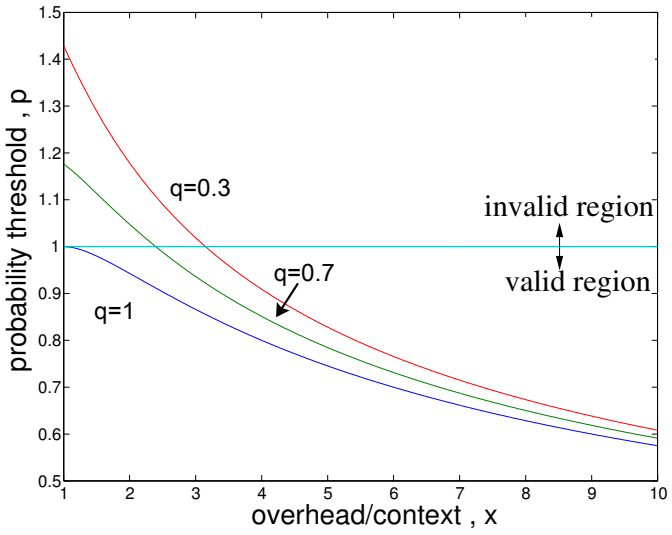

Figure 2.3: Minimum cell interaction probability p vs. overhead, to achieve savings using aggregation.

2.8.1 Dynamic context

Aggregation pays off in terms of bandwidth/energy consumption but this comes at the expense of the accuracy of highly dynamic data. For example, as aggregative cells become larger and larger the readings from highly dynamic sensors provided to a mobile at the time of exchange might be invalid at the time the mobile reaches the sensors. In practice, a wide class of contextual information is static, e.g., a map of the current floor of the mall, or slowly varying e.g., readings from a temperature sensor. A designer trying to decide on the appropriate level of aggregation has to consider the nature of the contextual information as well as the average sojourn time of a mobile through a typical cell. The following fact serves as a rule of thumb for deciding

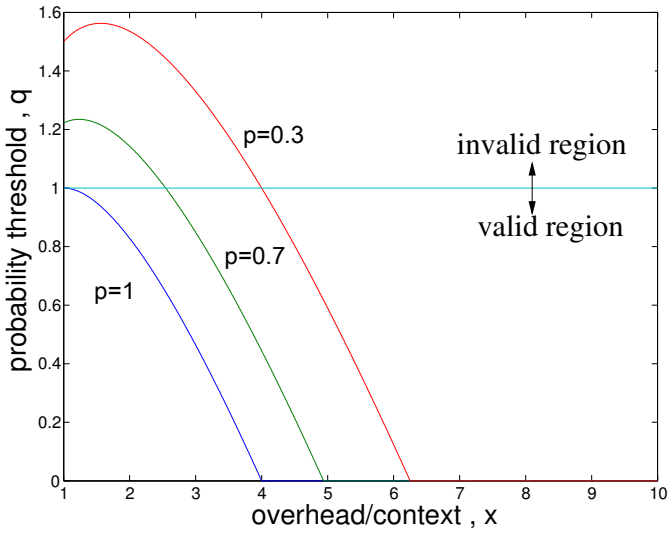

Figure 2.4: Minimum percentage for interest in context vs. overhead, to achieve savings using aggregation.

when aggregation is acceptable for dynamic data.

Fact 2.9. *Aggregation is meaningful for acquiring data from sensors that change with frequency f if*

$$f \approx \sqrt{\lambda_a} v$$

where v is the average speed of the mobiles.

This fact relies on the observation, demonstrated in Section A.7, that the mean sojourn time of a mobile through the cells Voronoi tessellation induced by a homogeneous Poisson Point Process with rate λ is proportional to $\frac{1}{v\sqrt{\lambda}}$.

2.10 Analysis of aggregative tessellations under generalized context scaling

In the previous section we assumed that the context content function was additive. However, as discussed in Section 2.2 one can consider applications with different scaling characteristics. To capture the more general case, we consider context scaling of the form posited in Assumption 2.3. In this case, by analogy with Eq. 2.4, one finds that coarse grained context regions will be beneficial if

$$\sqrt{\lambda_a} * (h + (\frac{\lambda_f}{\lambda_a})^{\alpha} * c(V_f)) < \sqrt{\lambda_f}(h + c(V_f)). \tag{2.9}$$

The results below are shown in Section A.3 and summarize the characteristics of minimum cost organizations for context exchange.

Theorem 2.10.1. *Under Assumptions 2.3, 2.4 and 2.6, an organization for context exchanges based on an aggregative tessellation with intensity λ_a is beneficial if*

- *$\alpha < \frac{1}{2}$ and $\lambda_a \in (0, \lambda_f)$. In this case the cost is strictly increasing in λ_a, thus the intensity should be as small as possible.*
- *$\alpha > \frac{1}{2}$, $\frac{c(V_f)}{h} < \frac{1}{2\alpha - 1}$, and $\lambda_a \in (\hat{\lambda}_a, \lambda_f)$, where $\hat{\lambda}_a$ is the maximum solution to the equation*

$$\sqrt{\lambda_a}(h + (\frac{\lambda_f}{\lambda_a})^{\alpha} c(V_f)) = \sqrt{\lambda_f}(h + c(V_f)) \tag{2.10}$$

such that $\hat{\lambda}_a < \lambda_f$. *In this case the optimal intensity for the aggregative tessellation is*

$$\lambda_{a,opt} = (\frac{2\alpha - 1}{x})^{\frac{1}{\alpha}} * \lambda_f$$

Let x *denote the overhead ratio* $x \triangleq \frac{h}{c(V_f)}$, *then the maximum relative cost reduction of acquiring context from aggregative versus the finest grain organization is given by*

$$|1 - \frac{\sqrt{\lambda_{a,opt}}(h + c(V_a))}{\sqrt{\lambda_f}(h + c(V_f))}| = 1 - \frac{2\alpha}{2\alpha - 1}\frac{x}{1+x}(\frac{2\alpha - 1}{x})^{\frac{1}{2\alpha}}. \tag{2.11}$$

One can interpret this result as follows. Recall that the problem with aggregating and exchanging context from coarse grain cells is that mobile users would obtain contextual data that may not be relevant to them as they move through space. However, when context content scales slowly, i.e., slower than the square root of the area, this corresponds to an application that has quite a bit of spatial redundancy in the contextual information. In this case, coarse aggregative cells effectively compress contextual data, or alternatively the redundancy ensures that most of the context content of an aggregative cell will be relevant. Additionally, the reduced rate of paying for fixed overheads makes the aggregative approach appealing.

Note that when $\alpha < \frac{1}{2}$, the context content function of a cell from the 'finest grain' tessellation scales slow enough that aggregation always helps. As shown in Fig. 2.5 in this case any value for λ_a less than λ_f achieves a cost savings. Note that this is true irrespective of the values of h and $c(V_f)$. Of

course, the higher the values of h and $c(V_f)$, the higher the amount of context exchanged, but asymptotically, the amount of cost per unit time for a typical user goes to 0 as λ_a decreases.

When $\alpha > \frac{1}{2}$, context content grows quickly so more care needs to be taken in using aggregative cells, this was already observed in the special case of Theorem 2.7.1. In particular, the interval for the intensity of the aggregative tessellation to be beneficial is now bounded from below, e.g., compare the lower curves shown in Fig. 2.6. versus the upper curves where aggregation does not pay off.

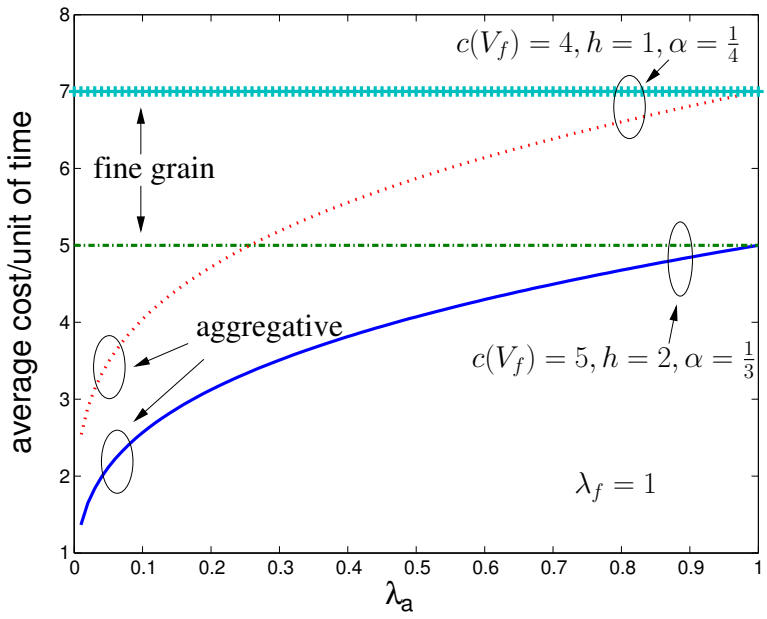

Figure 2.5: Context aggregation, $\alpha < \frac{1}{2}$.

Fig. 2.6 exhibits a case where $\frac{c(V_f)}{h} > \frac{1}{2\alpha - 1}$, e.g., $c(V_f) = \frac{4}{3}, \alpha = \frac{4}{3}, h = 1$.

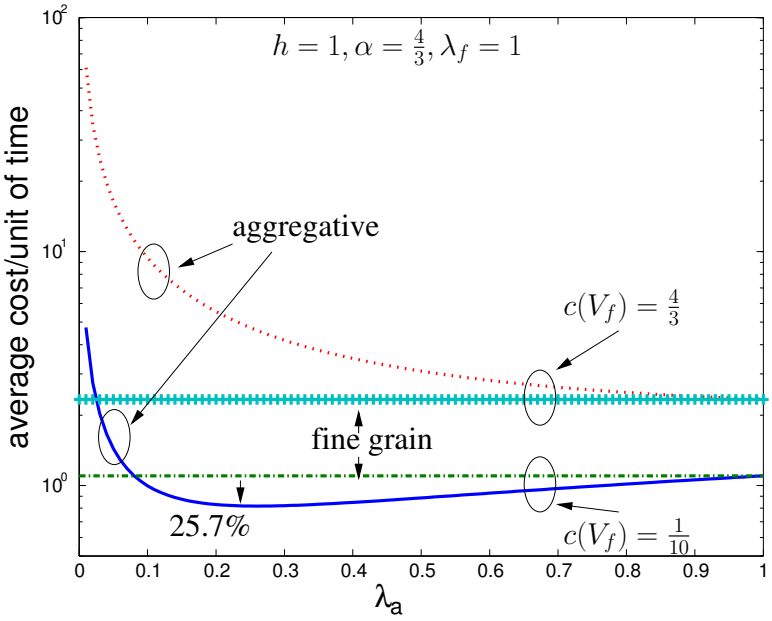

Figure 2.6: Context aggregation, $\alpha > \frac{1}{2}$.

As can be seen, the average cost associated with the aggregative tessellation always exceeds that of the finest grain organization. Thus, aggregation does not pay off. By contrast, when $\frac{c(V_f)}{h} < \frac{1}{2\alpha - 1}$, e.g., $c(V_f) = \frac{1}{10}, \alpha = \frac{4}{3}, h = 1$, there is an interval for λ_a in which aggregation is beneficial. The left and right boundaries of the interval as well as the location of the optimal rate of the aggregate tessellation are those predicted by Theorem 2.10.1. The relative cost reduction achieved by aggregation for the case shown in Fig. 2.6 is 25.7%. Recall that for $\alpha = 1$, Fig. 2.2, we had a higher relative savings of 42.5%. Indeed for $\alpha = \frac{4}{3}$, the benefit of aggregation diminishes because the context content for aggregative cells grows super-linearly.

An important special case of our context model is when h=0, i.e., the cost is simply proportional to the amount of context exchanged. The following result summarizes what happens in this scenario.

Corollary 2.10.2. *Under Assumptions 2.3, 2.4 and 2.6 with $h = 0$, the cost associated with an aggregative tessellation of intensity λ_a results in cost reduction only if $\alpha < \frac{1}{2}$ and $\lambda_a \in (0, \lambda_f)$. The optimal intensity for the aggregative tessellation does not exist since the cost decreases as the intensity λ_a approaches zero.*

The previous result can be obtained by setting the value $h = 0$ to Eq. 2.9.

In this special case the overhead for acquiring contextual information from a cell is proportional to the context content of the cell; there are no fixed overheads. Additionally, the amount of redundancy in a cell's context is reduced for $\alpha > \frac{1}{2}$. Therefore, by using increasing amounts of aggregation, one can never match the cost of acquiring context from the finest scale tessellation.

2.10.1 Intuition: the $\alpha = \frac{1}{2}$ case

By looking at Theorem 2.10.1 and Corollary 2.10.2 the case $\alpha = \frac{1}{2}$ shows special significance. In this section we will offer an intuitive explanation why this is true. For a 2-dimensional homogeneous Poisson Point Process with rate λ, it is known, see [59], that the average chord length is proportional to $\frac{1}{\sqrt{\lambda}}$. Assuming simplistically that a mobile moves in a straight line trajectory inside

a cell of the aggregative tessellation, the average number of finest-grained cells it crosses while inside a coarse-grained cell is $\sqrt{\frac{\lambda_f}{\lambda_a}}$, see Figure 2.7. At the same time, the ratio of the context acquired from an aggregative cell over a finest-grained cell is $(\frac{\lambda_f}{\lambda_a})^\alpha$. If $\alpha < \frac{1}{2}$, the rate at which context is acquired is smaller than the rate at which new finest-grained cells are crossed, amounting to a true economy. If $\alpha > \frac{1}{2}$ this does not hold anymore, and h, the overhead associated with each transfer, plays a dominant role.

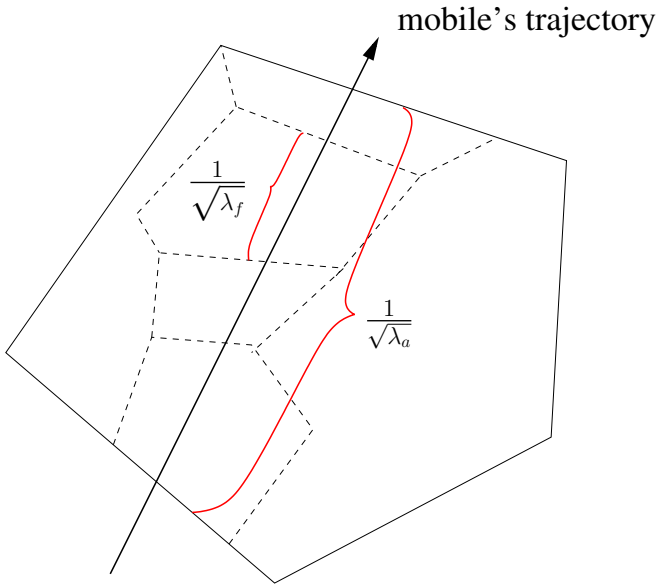

Figure 2.7: Intuitive explanation for $\alpha = \frac{1}{2}$.

2.10.2 Selective context

In Section 2.7 we examined the case of a mobile having selective interest in the context of the cells of the finest-grain organization. We will revisit the same case here for generic α. The following fact holds; for a proof see Section

A.4.

Fact 2.11. *Under the assumptions of Fact 2.8, a necessary condition for an aggregative organization to achieve savings compared to a finest-grained organization is*

- $\alpha < \frac{1}{2}$ *and* $\lambda_a \in (0, \hat{\lambda}_a)$*, where* $\hat{\lambda}_a < \lambda_f$ *is the unique solution to the equation* $r^{2\alpha} - rp(x+q) + x = 0, r \triangleq \sqrt{\frac{\lambda_f}{\lambda_a}}$.
- $\alpha > \frac{1}{2}$ *and* $\frac{p(x+q)}{2\alpha} > \frac{2\alpha}{2\alpha-1}x, x > 2\alpha - 1$*. In this case, there exist* $\lambda'_a < \lambda''_a < \lambda_f$ *s.t.* $\lambda_a \in (\lambda'_a, \lambda''_a)$.

where x *is defined as in Theorem 2.7.1. The optimal scales of aggregation are still given by Theorem 2.10.1.*

The main conclusion of this fact is that the selective context transfer does not alter the structure of the problem, just the range of the beneficial levels of aggregation. We do not discuss the quantitative implications of this fact. We refer the reader to the discussion in Section 2.7 which is essentially analogous.

2.12 Hierarchical organization for context exchange

In this section we focus on applications that have a sub-additive context content function, i.e., $\alpha < 1$ in Assumption 2.3. Recall that sub-additivity likely results from spatial redundancy or shared context across finest grain cells. Intuitively, it makes sense to consider a hierarchical organization, whereby

shared context is delivered via a coarser level of granularity, while context that is specific to a location is delivered via a fine grained organization. For example, for the case of a mall discussed in Section 2.1, the part of the contextual information that is shared among all stores on the same floor, e.g., locations of emergency exit points, could be exchanged *once* at floor level while information specific to each store, e.g., discounts offered by a store, can be acquired once at store level.

In this section a hierarchical organization for context exchanges involves *both* the 'aggregative' and 'finest grain' tessellations introduced earlier, but they are used in different manners. In particular, when a mobile finds itself in a cell of the 'finest grain' tessellation it obtains only the context data that is unique to that cell. The *shared* context is exchanged with mobiles once in each cell of the aggregative tessellation. The idea is to try to minimize overheads while maximizing the relevant context that is exchanged to users.

The effectiveness of our proposed hierarchical organization depends on the average amount of shared context among finest grain cells of the $V(P_f)$ tessellation. We estimate the average shared context as follows. Each cell of the finest-grained tessellation has an average context content $c(V_f)$ and an average *unique* context among their peers denoted by $c(V_f|V_a) < c(V_f)$, since we operate on the $\alpha < 1$ regime and there is spatial redundancy for the context content. The unique part of each finest-grain cell's context, $c(V_f|V_a)$, is stored in the cell. The shared context among all finest-grained cells covered by an aggregate cell, $c(V_f) - c(V_f|V_a)$, is stored in the aggregate cell. A typical cell

from the aggregative tessellation $V(P_a)$ has a total, unique plus shared, average context content $c(V_a)$, area $1/\lambda_a$, and will on average cover λ_f/λ_a finest grain cells. Thus, the total context of the aggregative cell should satisfy

$$c(V_a) = \underbrace{\frac{\lambda_f}{\lambda_a} c(V_f|V_a)}_{\text{sum of unique}} + \underbrace{[c(V_f) - c(V_f|V_a)]}_{\text{shared}},$$

where the first term is the sum of the unique context of its constituent finest grain cells, and the second term is the context shared by the finest grain cells. Denoting the context shared by finest grain cells by $s = c(V_f) - c(V_f|V_a)$ one can solve the above equation to obtain:

$$s = \frac{\lambda_f c(V_f) - \lambda_a c(V_a)}{\lambda_f - \lambda_a}. \tag{2.12}$$

Analogous to the previous sections, the cost per unit of time for a typical user under this hierarchical organization is now given by

$$\sqrt{\lambda_a}(h + s) + \sqrt{\lambda_f}(h + c(V_f) - s),$$

where h is the overhead associated with each context exchange. The first term corresponds to the shared context, which is exchanged from aggregative cells while the second term corresponds to the costs associated with exchanging context that is unique to the 'finest grain' cells. Under this model we can show the following results, where again we have relegated the derivations to Section A.5.

Theorem 2.12.1. *Under Assumptions 2.3, 2.4 and 2.6, the hierarchical organization for context exchanges achieves a cost saving over the finest-grained*

organization if:

$$\alpha < 1 \ , \ \lambda_a \in (0, \lambda_f) \ \textit{and} \ x \triangleq \frac{h}{c(V_f)} < \frac{r^2 - r^{2\alpha}}{r+1} \triangleq f(r)$$

where $r \triangleq \sqrt{\frac{\lambda_f}{\lambda_a}}$.

The interpretation of Theorem 2.12.1 is that when context content scales sub-linearly, for increasing levels of aggregation, the hierarchical organization will always produce savings compared to acquiring context from a finest-grained tessellation. The relationship between a hierarchical organization and an aggregative one is described in the following Theorem proved in Section A.6.

Theorem 2.12.2. *Under Assumptions 2.3, 2.4 and 2.6, the hierarchical organization for context exchanges achieves a cost saving over the aggregative organization if:*

$$\alpha < 1 \ , \ \lambda_a \in (0, \lambda_f) \ \textit{and} \ x \triangleq \frac{h}{c(V_f)} < \frac{r^{2\alpha} - 1}{r+1} \triangleq g(r)$$

where $r \triangleq \sqrt{\frac{\lambda_f}{\lambda_a}}$.

Note that if $\alpha > \frac{1}{2}$ exchanging context from a hierarchical organization will eventually lead to a cost savings. Indeed if $\alpha > \frac{1}{2}$, then $\lim_{r\to\infty} g(r) = \infty$ so the condition in Theorem 2.12.2 is eventually satisfied irrespective of the value of $c(V_f)$ and h. So it suffices to employ a sufficiently coarse granularity (a value of λ_a that is small enough) for the hierarchical approach to result in cost savings. Observe in Fig. 2.8 and in Fig. 2.9 that the hierarchical approach

can reduce the cost for any value of $c(V_f)$, while the aggregative approach has at best a certain range over which it can achieve cost reduction.

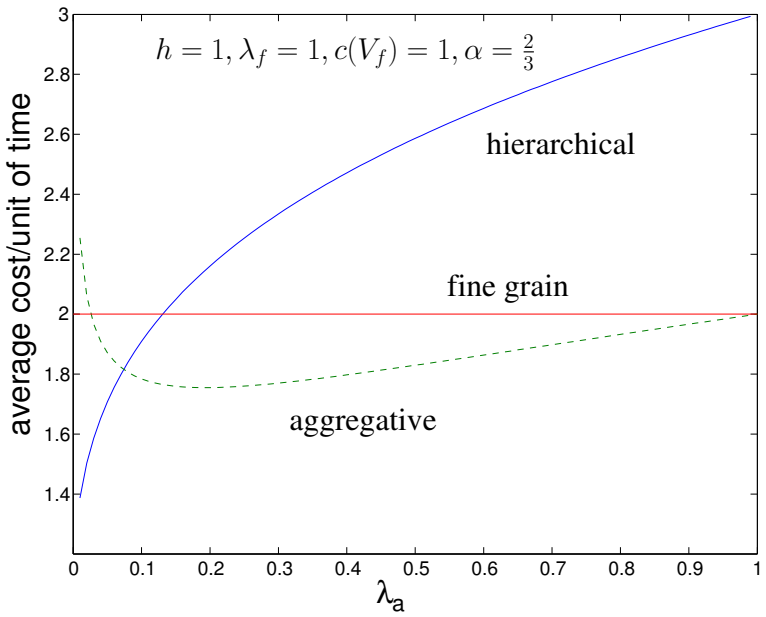

Figure 2.8: Hierarchical vs. aggregative when $\alpha > \frac{1}{2}, c(V_f) = 1$.

By contrast, if $\alpha < \frac{1}{2}$, exchanging context from a hierarchical organization may or may not be preferable to an organization based on aggregation, depending on the coarseness of aggregate cells one can practically achieve. From Theorem 2.10.1 we know that for $\alpha < \frac{1}{2}$ an aggregative approach always results in cost savings. In this case the limit of the upper bound $g(r)$ of Theorem 2.12.2 goes to 0 as $r \to \infty$. Thus in this case, a designer should be careful enough to evaluate both approaches before deciding which one is better. In Fig. 2.10 we observe that the aggregative approach results in cost

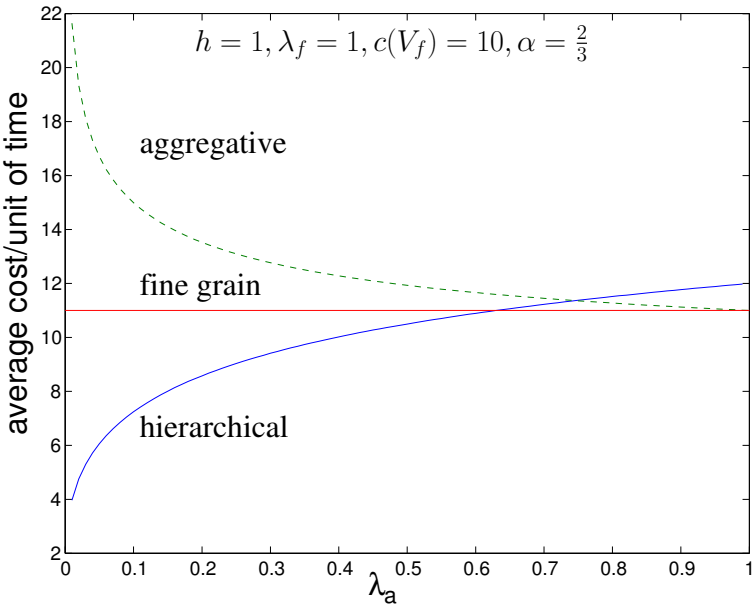

Figure 2.9: Hierarchical vs. aggregative when $\alpha > \frac{1}{2}, c(V_f) = 10$.

savings for all allowable values of λ_a, while the hierarchical approach needs cells to be coarse enough to do so. Once cells are coarse enough, the hierarchical approach produces savings that for the specific values chosen for the graph in Fig. 2.10 outperform the aggregative approach for all practically achievable scales of aggregation.

A comparison summary of the two organizations is shown in Table 2.1.

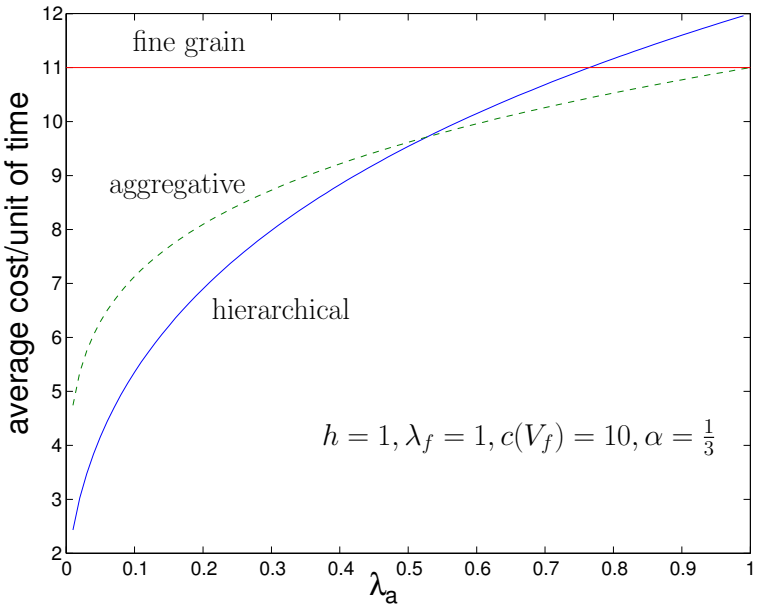

Figure 2.10: Hierarchical vs. aggregative $\alpha < \frac{1}{2}$.

2.13 Assessing the cost of surveillance in ubiquitous environments

Throughout this chapter we have implicitly assumed the existence of a surveillance mechanism that is part of a space's infrastructure and allows mobiles to detect when they cross cell boundaries. We envisage two generic types of surveillance mechanisms.

- A *direct* mechanism that is part of a space's infrastructure and monitors each cell's boundary. An airport or shopping mall with RFID readers installed on the doors exciting RFID tags on the mobiles passing through would be an example of such a mechanism. Such a mechanism is assumed

$\alpha > 1$	Aggregative *may* help depending on the scale of aggregation
	Hierarchical *cannot* help
$1 > \alpha > \frac{1}{2}$	Aggregative *may* help depending on the scale of aggregation
	Hierarchical *eventually* help
	Hierarchical *eventually* outperform aggregative ones
$\alpha < \frac{1}{2}$	Aggregative *always* help
	Hierarchical *eventually* help
	Aggregative *eventually* outperform hierarchical ones

Table 2.1: Aggregative vs. hierarchical organizations.

in [94].

- An *indirect* mechanism that detects boundary crossings by comparing each mobile's location to the location of the cell boundaries. A tracking service that is part of the infrastructure or self-positioning by each mobile device can be used to calculate location. We assume that self-positioning mobile nodes detect boundary crossings using an a-priori downloaded map of the cell boundaries. For a well known location system based on this approach, see e.g., [6].

Let us first consider direct surveillance mechanisms. We abstract the underlying mechanism by assuming that the cost to detect a boundary crossing is E_d units of energy/device, e.g., the energy in an RFID reader's pulse to read a potential tag. Additionally we let f_d denote the frequency with which the mechanism checks for boundary crossings, e.g., an RFID reader on a door sends a pulse every second to detect mobiles, we say that $f_d = 1\ Hz$. Clearly, there will be a trade-off between the surveillance frequency and the timeliness with which contextual data is delivered. Finally, it is known that the average cell

boundary per unit of area for a homogeneous Poisson Voronoi tessellation with rate λ is $2\sqrt{\lambda}$ [79]. Motivated by the use-cases presented in [94] and practical considerations, we note that mobiles moving from cell to cell pass through designated points e.g., doors, and detectors will have a certain coverage range so only a fraction, K_d, of the total cell boundary has to be surveilled directly. The following assumption captures these elements.

Assumption 2.14. *We assume that the average power for a direct surveillance mechanism per unit of area is given by*

$$2 * \sqrt{\lambda} * K_d * f_d * E_d. \tag{2.13}$$

With this additional assumption, the power expended for surveilling cell boundaries and exchanging context using an aggregative tessellation with intensity λ_a is given by

$$2 * \sqrt{\lambda_a} * K_d * f_d * E_d + \lambda_0 * \frac{4 * v}{\pi} \sqrt{\lambda_a}(c(V_a) + h). \tag{2.14}$$

This in turn can be simplified to $\sqrt{\lambda_a} * (\hat{h} + \hat{K}c(V_a))$ where $\hat{h}$ and $\hat{K}$ are appropriate constants. This cost function has the same form as that considered in Section 2.2, which leads to the following corollary.

Corollary 2.14.1. *Theorem 2.10.1 can be applied for optimizing the intensity of an aggregative tessellation for context exchange using direct surveillance. The overhead ratio is given by* $x \triangleq \frac{\hat{h}}{\hat{K}c(V_f)}$.

Note that for the case $\alpha > \frac{1}{2}$, the frequency at which the shared infrastructure surveils boundary crossings, f_d, plays a key role. An increased value

of f_d is beneficial in two ways: it increases the timeliness of mobile detection and increases the parameter overhead ratio x above the $2\alpha - 1$ threshold required for the aggregative tessellation to produce savings, see Theorem 2.10.1. Also note that the added cost of surveillance does not change the underlying structure of the problem, i.e., the existence of optimal values for the scale of aggregation still depends on the scaling (α) of the context content function. Thus, the results derived in the previous sections provide a designer with the tools to roughly evaluate how to optimize the aggregative organizations.

For the indirect surveillance mechanism, we define the frequency with which a mobile acquires location information f_i and the corresponding energy expended E_i in a similar way as in the direct case.

Assumption 2.15. *Under Assumption 2.4 the average power per unit area expended by an indirect surveillance mechanism to track boundary crossings is*

$$\lambda_0 * f_i * E_i. \tag{2.15}$$

Observe that the power expended increases linearly with the intensity of the mobiles. Such an approach would face scalability problems if the number of mobiles increases significantly as expected in ubiquitous computing scenarios.

Note that the frequencies f_d, f_i, must be high enough to ensure that a mobile does not 'miss' acquiring context from a cell in a timely manner. Intuitively, the higher the intensity of the aggregative process, λ_a, the higher the cost for exchanging context. The following fact, formally proved in the Appendix, provides lower bounds on f_d, f_i.

Fact 2.16. *Under Assumption 2.4 and surveillance frequencies f_d, f_i chosen to ensure that a mobile transitioning through an average sized cell of an aggregative organization of rate λ_a is not missed, the frequency f_d must satisfy $f_d > K_d^f$ and the frequency f_i should satisfy $f_i > K_i^f * \sqrt{\lambda_a}$.*

K_d^f and K_i^f are constants depending on the average velocity of the mobiles and the range of the devices used to perform the surveillance. Note that the previous fact is predicated on not missing on *average* sized cell, so in practice the variability in cell sizes would require surveillance frequencies to be higher.

For a given aggregative organization, i.e., fixed λ_a, a designer can consider which surveillance mechanism is more energy efficient.

Fact 2.17. *The direct surveillance is more efficient than the indirect surveillance if*

$$2 * \sqrt{\lambda_a} * K_d * f_d * E_d < \lambda_0 * f_i * E_i \tag{2.16}$$

For services offered on a 'personalized' scale, i.e., $\lambda_f \sim \Theta(\lambda_0)$, $\sqrt{\lambda_a} << \lambda_0$ for an aggregative tessellation and the leverage of the shared infrastructure by a direct surveillance mechanism provides significant gains.

2.18 Conclusions and future work

This chapter is a first attempt at studying the fundamental characteristics of context exchange and surveillance organizations for ubiquitous applications. To allow for quantitative arguments, we propose a simple stochastic

geometric model that naturally represents the main characteristics of such systems. The key results show how the effectiveness of optimal aggregative versus hierarchical organizations depend on the manner in which context content scales with area. We also consider how energy costs for direct and indirect surveillance mechanisms would vary under such organizations. Clearly, our model has several simplifications that it would be of interest to relax, and are part of our future work. Among other issues, it would be of interest to capture how limited caching of contextual data might enable mobiles to reduce their energy expenditures while maintaining updated contextual information.

Chapter 3

P2P Network for Storage and Query of Spatio-Temporal Flow of Events

3.1 Introduction

3.1.1 Motivation

A significant and largely unaddressed challenge in the wide-spread use of networked systems is the harnessing of real-time flows of spatio-temporal information to deliver real-time context-awareness to applications and their users. In this chapter, we study a novel peer-to-peer (P2P) infrastructure enabling wireless and wired users/devices and their associated applications to store and query a flow of events that are short-lived and associated with particular spatial locations. We envisage a system handling a high intensity of events generated by humans, sensing devices, applications or combinations thereof. We focus on systems exhibiting locality in queries, i.e., queries are likely to be related to events located in the vicinity of the device/user making the query, see e.g. [80]. To motivate our design, consider a system for managing available parking spots. Primitive events signifying (possible) vacant parking spots are contributed by a disparate collection of devices/users including sensors, payment stations, cameras surveilling the streets or the human drivers who vacate a spot. A fixed peer overlay is used to store the corresponding events and re-

spond to queries about them from interested parties. Such a system would be a representative application for our proposed system. One can generalize to other classes of applications that can be handled using our proposed system, e.g., resource discovery, predicate computation.

3.1.2 Centralized solution is undesirable

Devising an infrastructure to realize such functionality poses several challenges. A centralized solution requires the flow of events/info to be transported to, and stored at centralized resources, enabling low cost management and sharing of computing resources over the event/information flow. In turn, queries would be routed to the centralized location for processing and back. Such a solution does not exploit the queries' spatial locality and thus can be wasteful when the cost of transporting bits per unit distance is high. Indeed the data associated with events/info would potentially be transported far away, possibly never to be of any use, or at best, to subsequently be retrieved and taken back to devices/users close to where the information was originally generated. In other words when the spatio-temporal intensity of events is high, such a solution would not be particularly scalable. Moreover, a centralized approach may preclude a more ad hoc, or grass roots, distributed infrastructure that might better match application modalities in terms of performance, fault-tolerance and scalability. For example, given we assume there is spatial locality in the queries, if events/information are stored close to where they are generated, network delays and traffic could be substantially reduced.

3.1.3 Peer-to-peer distributed infrastructures

This chapter focuses on exploring the design space of overlay infrastructures of fixed peers that act as gateways for mobiles; this approach has already proved its merits, see [46]. The challenge lies in designing a P2P network with the primary purpose of supporting spatio-temporal context-awareness based on a flow with high spatial intensity of many short lived events over a set of peers possibly exhibiting churn. In this context a simple very low overhead design that achieves low query delays is desirable. We expect to achieve reduced traffic (due to the spatial locality of events and queries) and ultimately reduced query delay (since queries are resolved locally) in addition to inheriting the advantages of P2P architectures (e.g., distribution, scalability, robustness, lack of central administration, use of idling resources, enabling of grass roots deployments). As explained in the related work section below, to date P2P networks have not been designed for storing and sharing spatio-temporal flows of events/info. Thus our approach, although drawing on several key components developed in previous work on geo-located routing, overlays and network protocols, is quite novel.

3.1.4 Related work

Computing context from primitive events is a problem of growing interest in the pervasive computing community. Known approaches based on storing events in P2P overlays and computing on the results of queries do not fully satisfy the requirements imposed by the real-time nature scenario we

target in this chapter. Seminal DHT-based P2P storage systems, e.g., Chord [78] Pastry [68] or GHT [62] are ideally suited for applications performing *location-independent* naming and load balancing across storage capacity, see [7]. They do not preserve locality in storage, i.e., two files contributed by the same peer are not necessarily stored at the same (or close-by) peers nor do they explicitly incorporate notions of the stored information's relevance in space and time; these approaches were not designed with the aim of supporting real-time context-awareness. Attempts to overcome these limitations while still remaining in the realm of DHT-based overlays do not fully succeed. Kuhn et al. in [48], introduce "*containers*" as a new abstraction for storing spatially related events. This way of storing events lacks flexibility when compared to our approach and does not naturally maintain locality, i.e., containers have to be specified ahead of time and two events that are within a query-defined distance to each other are not guaranteed to lie in the same container, this will become obvious subsequently in this chapter when we introduce the 'range' query. PeopleNet, [58], introduces the 'bazaar', a topic specific region to resolve topic-specific queries. Neither of these approaches preserves locality in storage.

Context can be computed and queries resolved more efficiently in a platform providing access to relations between events/info rather than simply to the individual events/info. *Similarity* queries, see [8], are a reasonably powerful family of queries for computing context derived from events exhibiting locality, e.g., retrieving events tied to a given region. Resolving these types

of queries calls for a modified P2P architecture directly factoring *locality* into information storage and retrieval.

Voronoi tessellations have proven to be a useful geometric structure to capture similarity queries and spatial locality in networks. SWAM [8] and VoroNet [11] are designed to enable efficient processing of similarity queries by using Voronoi diagrams. The key idea is to embed data in a d-dimensional attribute space, and, based on an attribute distance metric, determine the induced Voronoi tessellation and Delaunay graph. Edges in the Delaunay graph (in the attribute space) are mapped to edges in the overlay network among the peers that currently hold the data items. By contrast, our focus is on the physical proximity of the locations to which events/information are tied and to their regions of relevance. Links are driven by the spatial proximity of peers and network performance concerns alone, rather than the data's attributes. Challenges for SWAM and VoroNet lie in the overheads associated with managing a d-dimensional Voronoi diagram, particularly when peers join and leave the P2P network frequently. By contrast, in our work peer churn results in moving data among peers to maintain network functionality. Our work, and the above mentioned works, considers the introduction of additional edges between peers in the overlay, such as those proposed in [43], to improve query performance. However, in our work we study their effect on query delay rather than the hop-count, and we show how such edges and topology should be managed to optimize system performance.

A platform for distributing the location updates for players in Massively

Multi-player Online Games (MMOGs) is presented in [33]. Updates about a player's location need only be distributed to a neighborhood around the player. This is a form of locality in a *virtual* space as opposed to our work, which employs locality in the *physical* space. This observation has critical implications on performance as physical locality correlates positively with low delays in the underlay network. The work in [33] uses geometrical routing on the overlay network as we do although the underlying topology is quite different from ours. Our work is based on a *generic* event model that takes explicitly into account spatio-temporal locality. No such model is offered in [33], where the focus is on a special type of information, i.e., location updates. In our work, events have a life-time that is independent of the life-time of the peers forming the overlay. In [33] the information disseminated, i.e., the location updates coincide with the life-time of the players in the MMOG. Moreover, in our platform a peer can store more than one event. In [33] information is not stored, each player knows its own location and publishes updates about it. Finally, in our work we offer procedures for storing, deleting and querying events. No equivalent procedures are offered in [33].

Finally, VoRaQue [2], considers the efficient implementation of *range* queries, a special type of queries that we define in the sequence, on Voronoi based overlays. Their work focuses on algorithmic considerations in resolving such queries, specifically on computing a spanning tree over the specified range. This work has a narrow focus on 2-dimensional data with no protocol, scalability or performance analysis as found in the present work.

In terms of competing platforms, recent technology [51] has been developed that enables devices to efficiently wirelessly advertise resource/events (types) directly to others within proximity. This enables resource discovery but requires periodic advertising throughout the lifetime of the event to assure context awareness. We view this work as complementary, if not an extreme version of the service we target in this chapter. A preliminary version of this work has appeared in [93].

3.2 Event and query models

Our focus is on capturing a spatio-temporal flow of (short-lived) events that can represent a wide range of data/information associated with users, applications, sensors or machines. These are formally defined as follows:

Definition 3.2.1. *An event e is a five-tuple*

$$(e.location, e.range, e.time, e.duration, e.type) \in E$$

where $E = \mathbb{R}^2 \times \mathbb{R}_+ \times \mathbb{R} \times \mathbb{R}_+ \times \mathbb{T}$ and $\mathbb{T}$ denotes a set of possible event types.

As shown in Fig. 3.1, this can be visualized as a cylinder where an event e has associated spatial coordinates, $e.location$ indicating where the event 'occurred' or is centered, a range $e.range$ defining the region where it is relevant (see [36] for a similar concept 'Area Of Interest' (AOI)), as well as a time at which it is generated/starts and duration: $e.time$ and $e.duration$ respectively. We assume that devices/applications that generate events have geo-location

capability and synchronized clocks. This allows them to 'stamp' the events with meaningful spatio-temporal coordinates. Events with larger cylinders can 'interact', e.g., overlap with a larger number of other events possibly producing additional relationships or contextual information. For simplicity we let $B(e)$ denote the disc centered at location $e.location$ with radius $e.range$, i.e., the disc $B(e.location, e.range)$ and refer to events whose duration contains the current time as *active*. The possible event *types*, e.g., 'change-of-status' type, are assumed to be predefined, but for the remainder of this chapter we will not focus on this aspect.

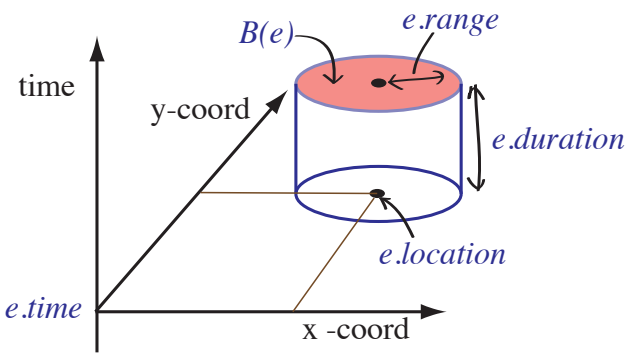

Figure 3.1: Event model.

The goal of the infrastructure is to efficiently support spatio-temporal queries. In this chapter we focus on *range queries* on *active* events. This is done to simplify notation, i.e., remove time, but one can easily generalize this to consider queries on past or even future events. Range queries are defined as follows:

Definition 3.2.2. *The* Range Query $RQ(l, r)$ *returns all active events* e *within*

a range r of a location l, i.e., events currently stored in the system such that $e.location \in B(l, r)$.

For example, return all users/resources currently available within 200m of a given location, e.g., my location. Based on range queries, one can define and compute more complex queries representing relevant spatio-temporal context, some examples are presented below.

k-Nearest Neighbor Query: $\text{NN}(k, l, r) \triangleq \{k$ closest active events $e_1, \ldots, e_k$ within $B(l, r)\}$, e.g., return the 2 closest users to the current location.

Proximity Query: $\text{PQ}(l, r) \triangleq \{$ all pairs of active events $(e_1, e_2) : e_1, e_2 \in RQ(l, r)$ s.t. $e_1.location \in B(e_2)$ and $e_2.location \in B(e_1)\}$, e.g., return pairs of resources that are mutually within communication range of each other and are within a range r of location l.

Intersection Query: $\text{IQ}(l, r) \triangleq \{$ all pairs of active events $(e_1, e_2) : e_1, e_2 \in RQ(l, r)$ s.t. $B(e_1) \cap B(e_2) \neq \emptyset\}$, e.g., return pairs of events with overlapping regions within $B(l, r)$ such as fire overlapping with chemical spill.

Clique Query: $\text{CQ}(l, r, \rho) \triangleq \{$ sets of active events $\{e_1, \ldots, e_k\} : \exists l' : e_i \in B(l', \rho), 1 \leq i \leq k, e_i \in RQ(l, r)\}$, e.g., sets of events in $B(l, r)$ that are within 2ρ of each other.

It should be clear that the family of queries that can be resolved once range queries are enabled is fairly rich, e.g., one can check that they suffice to verify the 9-set intersection relationships [24] among discs. In Section 3.7.2 we will show how to resolve range queries in our infrastructure.

We conclude by formalizing the operational scenario of interest in this chapter, specifically:

Assumption 3.3. Spatial locality of events and queries. *Event and queries submitted by users/devices exhibit locality in that they are associated with close by locations.*

This assumption will drive in part our design choices. It is motivated by the idea that spatio-temporal context awareness for short lived events makes sense and is most likely to be relevant close to where users/devices currently dwell.

3.4 Architecture

Devising a P2P network to store and resolve queries on a flow of spatio-temporal events involves a set of trade-offs that are unique versus P2P systems that were previously devised. In this section we describe the key elements of our proposed architecture.

We consider a platform where the participating entities can play one or more roles: contribute events, make queries, and/or serve as a peer in the P2P overlay network. Further we envisage a setting where there may be (mobile) wireless entities, e.g., sensors, phones etc., that can not serve as peers, and wired fixed devices, e.g., PC and corporate servers, that can. All entities are assumed to know their locations, but exact locations are not necessary. Fig. 3.2 exhibits the elements of the platform that will be discussed below.

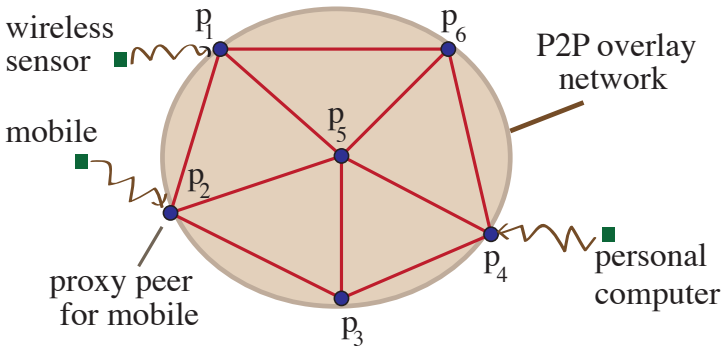

Figure 3.2: Elements of the P2P architecture.

3.4.1 Overlay topology management & routing

We let $P = \{p_1, p_2, \ldots\}$ denote the set of peers identified by their unique locations $p_i \in \mathbb{R}^2$. They are interconnected via links in a structured P2P overlay, represented in Fig. 3.2 by thick straight lines. Overlay links are realized by one or more physical links in the IP layer. The basic overlay connectivity is simple and driven by the goal of exploiting spatial locality of queries – it corresponds to the Delaunay graph induced by the peers' locations. Indeed a set of peer locations P induces a Voronoi tessellation of $\mathbb{R}^2$ and an associated Delaunay graph as follows.

Definition 3.4.1. *The* Voronoi tessellation $V(P)$ *induced by* $P \subset \mathbb{R}^2$ *is a partition of* $\mathbb{R}^2$ *into* cells $\{C(p|P) : p \in P\}$ *such that*

$$C(p|P) = \{x \in \mathbb{R}^2 : ||x - p|| \leq ||x - q||, \forall q \in P\},$$

i.e., the cell $C(p|P)$ *includes all locations that are closest to* p.

Definition 3.4.2. *The* Delaunay Graph *(DG) induced by* $P \subset \mathbb{R}^2$ *is a graph*

$G_D(P, E)$ with vertices P and undirected edges $E = \{(p, q) : (p, q) \in P^2$, where $C(p|P)$ and $C(q|P)$ are neighbors in $V(P)\}$.

In the sequel, when clear we will abuse notation and denote the Voronoi cell $C(p|P)$ corresponding to peer p by C_p. We let $N(p)$ denote the set of peer p's neighbors in the Delaunay graph.

There is already substantial work on distributed protocols to maintain a Delaunay graph among a set of peers, addressing efficiency, correctness, and, in part, peer churn, see e.g., [52, 54] and [77]. As such, in this chapter we will defer to this work, and simply assume that protocols are in place to maintain such overlay structures.

Assumption 3.5. Topology and Routing. *We assume the overlay connectivity of our platform is a* superset *of the Delaunay graph, and peers execute a distributed protocol to maintain the Voronoi tessellation and the Delaunay graph. We further assume peers employ greedy routing to store events and make queries over the overlay.*

Greedy routing here, refers to a policy where each peer forwards a message (event or query) destined to a location, say l, to the neighbor that is closest to l with the intent of eventually reaching the peer closest to the location. These choices for our platform result in the following beneficial properties.

Greedy routing always converges. Greedy routing on a Delaunay Graph always succeeds, see [15]. Moreover such routing on a superset of the

DG also succeeds, see [30], thus we will consider adding additional overlay links to optimize system performance. Castro et al., [19] argue that it is critical for overlay routing to be aware of the underlay network topology. Zahn and Schiller, [90], extend the previous argument for peers with mobility in a DHT overlay. We conjecture that a similar statement holds for non-DHT based P2P systems like ours. This observation drives our decision to perform routing on the Delaunay graph. Note that the length of the shortest path between two peers in the Delaunay graph is a constant times their Euclidean distance, see e.g., [23]. Thus paths chosen on the overlay network might roughly correlate to low cost routes in the IP underlay network.

Limited routing storage overhead per peer. Greedy routing only requires each peer to maintain the location of its neighbors in the DG. Moreover, the routing information exhibits locality, that is inconsistencies in routing information at a given peer will not lead to poor routing outside the peer's local region. We deem this important for networks with peer churn.

Maintaining the DG under churn is scalable. For P2P networks experiencing churn, some members of the network will have to update their neighborhoods. In Section B.2 we prove the following fact:

Fact 3.6. Scalability of Topology Maintenance

1. *If a peer p_{old} leaves the overlay, the only peers that have to update their neighborhood in the $DG(P \setminus \{p_{old}\})$ are the neighbors of p_{old} in $DG(P)$.*

2. *If a peer p_{new} joins the overlay, the only peers that have to update their neighborhood in $DG(P \cup \{p_{new}\})$ will become neighbors of p_{new}.*

From the previous fact, we deduce that the cost of a departure/join, in terms of peers needing to update their neighborhoods depends on the number of neighbors a typical peer will have. In [79] it is proved that for a homogeneous Poisson point process, the average number of neighbors of a point in the DG is 6. We conjecture that on average $O(1)$ peers have to update their neighborhoods when a peer leaves the overlay for a wider class of overlays. In [32] it is proved that when a new point is added to a set of points for which the Voronoi diagram has already been calculated, $O(1)$ structural changes have to be done on average to update the Voronoi diagram. The average is taken over all the possible configurations of points. A structural change denotes an addition of a new Voronoi edge or a deletion of an existing one. Every cell that changes has one or more Voronoi edges change. Thus, *on average* in our platform $O(1)$ peers will be affected. The worst case is given in [3] and is proved to be $O(|P|)$ peers affected. Combining these results we deduce that topology management under churn in our platform is not computationally demanding as relatively few peers are involved in it.

Routing on the overlay graph, assuming a homogeneous distribution of peers, roughly implies that the peer hop count will grow linearly in the distance to be traveled. Adding extra edges among peers, i.e., operating on a superset of the DG, can significantly reduce the number of hops a query has

to traverse. This however requires adding edges with care. In the sequel we will use the well known result of [43] which we summarize as follows:

Fact 3.7. Kleinberg edges. *Let s and t be the source and destination peers of a query respectively and suppose they are drawn uniformly from an $n \times n$ square grid. Suppose each peer u on the grid has edges to its immediate neighbors as well as to one other peer v chosen with probability proportional to $||u - v||^{-2}$, then there exists a decentralized algorithm performing greedy-routing and a constant c, independent of n, so that the expected number of hops between u and v is at most $c(\log(n))^2$, where c is a constant.*

We describe a mechanism to add such edges to our overlay network in Section B.6. A peer need only generate a location at random according to a distribution that is proportional to the inverse of the square of the Euclidean distance from its own location, and then route a message to that location, i.e., the closest peer to the randomly selected location. In the sequel we denote such additional edges, 'Kleinberg' edges, and in Section 3.9 we explore the resulting performance and scalability benefits.

3.7.1 Associating with the P2P network

Wireless devices and wired nodes that are not currently participating in the P2P overlay may still contribute by submitting events and queries. Such nodes access the overlay network by associating with a *proxy peer* currently in the overlay network – such associations are denoted by zig-zag lines in Fig. 3.2. The association process can be divided in two functions.

Bootstrapping. Non-peer entities must be able to bootstrap into the network by identifying a peer with a known IP address with which to associate, e.g., a dedicated peer, a home computer, an enterprise server, or a peer cached from a previous session.

Improving association. Once bootstrapped into the network a node can 'improve' its association. Specifically, given our assumption on spatial locality, it might want to associate with the peer whose physical location is closest to its own, or it might change its association due to poor performance. This can, for example, be achieved by a node sending an association query message to its own location, which eventually will be routed to the desired peer.

Nodes wishing to join the overlay network as peers can use similar mechanisms to do so. Specifically, they can bootstrap into the network through a proxy and then join the overlay network by sending a *join message* to the peer closest to their own location. The topology management protocol can take over from there.

3.7.2 Data management and query processing

Our spatial locality assumption on events and queries motivates the following rule.

Rule 3.7.1. Event storage. *Each event e is stored at the overlay peer $p \in P$ that is closest to e.location, i.e., such that e.location $\in C_p$.*

This rule is easily implemented in our framework by greedily routing and storing the event to the peer p closest to *e.location*. One can consider

various policies for event deletion. Since in this chapter we focus only on *active* events, they are deleted once they expire. However one can envisage policies where events are only deleted when storage space runs out or borrowing from caching policies whereby least recently queried events are deleted when new ones are to be stored.

One of our design goals is to address the possibility of peer churn. Clearly this not only requires managing the overlay topology but also where events are stored. Each time a new peer p_{new} joins the P2P network a subset of the events currently stored in the P2P network may fall within its Voronoi cell, and according to Rule 3.7.1 those events should be moved to p_{new}. Data management in the presence of churn is scalable and has low overheads since only events in p_{new}'s neighborhood may need to be moved:

Fact 3.8. Scalability of data management. *If a peer p_{old} leaves the network, its events will be moved to its current neighbors $N(p_{old})$. If a new peer p_{new} joins the P2P network, only events stored by neighboring peers $N(p_{new})$ in the updated topology may need to be moved.*

The previous fact follows from Fact 3.6, if we take into account that in our model event locations are mapped to points. Since a peer will typically have a low number of neighbors, as proved in Fact 3.6, churn results in low overhead irrespective of the total size of the network, assuring the scalability of our platform.

One can envisage *lazy* data management mechanisms aimed at reducing

the overheads associated with moving events, e.g., if events are very short lived and queries are responded to on a best effort basis, simply placing new events as appropriate at p_{new} may suffice.

The ability to process range queries, see Section 3.2, is key to our proposed solution. A range query $RQ(l, r)$ initially issued by a peer or proxy peer (on behalf of another entity using the overlay) q is greedily routed to the peer p closest to location l. When the query arrives at p, the query resolution process is initiated, see Algo. 1. Specifically p checks if it has events in the range query's disc and forwards the query to its neighbors. Neighboring peers check if their cell overlaps with the range query's disc, if so they check for events and forward the query to their own neighbors, etc.

If a disk $B(l, r)$ overlaps with C_p it will overlap with at least one of its edges. Thus, it suffices to check each edge of C_p for overlap with the disk. We state the following criterion to detect overlaps, for a 'visual' proof see Fig. 3.3:

Proposition 3.8.1. *A disk $B(l, r)$ intersects with an edge e of the Voronoi cell of a point p defined by the vertices π_1, π_2 if*

1. *$r > \min(|l - \pi_1|, |l - \pi_2|)$ and l does not belong to the rectangular strip defined by e and the vertical lines to e passing through π_1 and π_2 respectively, or*

2. *$r > ||l - \hat{l}||$, where $\hat{l} \in e$, is the point where the perpendicular line to e passing through l meets e and l belongs to the rectangular strip defined by e and the vertical lines to e passing through π_1 and π_2 respectively.*

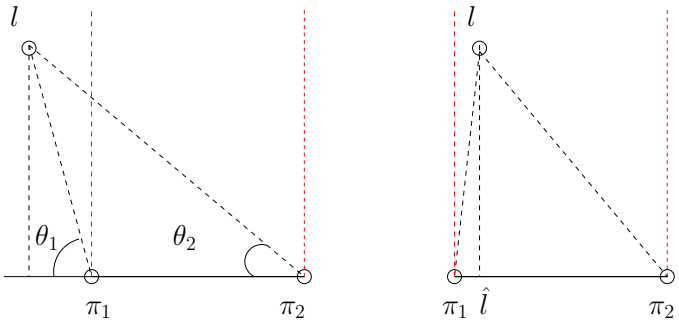

Figure 3.3: If l lies outside the strip defined by the dashed lines and $r < \min(|l - \pi_1|_2, |l - \pi_2|_2)$ then all the points in the edge defined by π_1, π_2 are farther away than r to l. In the other case it suffices to check the distance of $\hat{l}$ to l.

We focus on basic functionality here, but clearly a peer need not forward the query back to the peer that originally forwarded it, and it need not forward it more than once to its neighbors. This recursive query propagation eventually stops if there are only a finite number of peers in the range query's disc. The peers involved in the processing of the query then collect the intermediate results and forward them to the (proxy) peer q that originated the query. The resolution process is depicted in Fig. 3.4. Note that the reply to a query can be sent directly to the IP address of the originating (proxy) peer q, reducing traffic on the overlay network.

3.9 Performance and scalability analysis

Query delays in our overlay network are to a first-order determined by the number of peers a query message traverses and the queuing/processing at

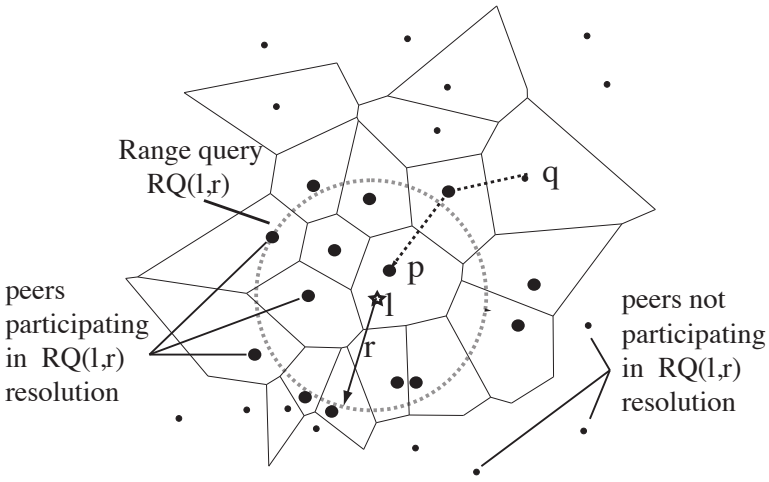

Figure 3.4: Resolution of range query RQ(l, r).

each intervening peer. A high density of peers can make the number of hops a query traverses high – this is a side-effect of our decision to route queries on the Delaunay graph. A low density of peers might result in peers being responsible for larger regions, increasing the storage and traffic loads they see, and thus increasing the per peer queuing delay. This suggests that for our network, "more is not always better." We explore this below.

Performance Analysis. To study these performance trade-offs we consider the following *idealized* model. Assume that peers are spaced δ m apart in a square grid of *physical* dimensions $r \times r$ m^2, see Fig. 3.5, so there are a total of $\lfloor \frac{r}{\delta} \rfloor^2$ peers. To avoid edge effects we assume the grid wraps around, i.e, its geometry is akin to a torus, e.g., peer p_1 in Fig. 3.5 has a distance of 1 hop from the peers p_2, p_3 and p_4. Peers act as sources and destinations for queries. We assume that events are generated according to a

Algorithm 1 Range Query Resolution

1: **Resolve Query**$(q, p, RQ(l, r))$**.** {resolves $RQ(l, r)$ on behalf of peer q at peer p. }
2: **if** $B(l, r) \cap C_p \neq \emptyset$ **then**
3: $response\_set = \emptyset$
4: **for** all events e at p s.t. $e.location \in B(l, r)$ **do**
5: $response\_set = response\_set \cup \{e\}$
6: **end for**
7: **for** $t \in N(p)$ **do**
8: $response\_set = response\_set \cup ResolveQuery(p, t, RQ(l, r))$
9: **end for**
10: send $response\_set$ to q
11: **end if**

spatio-temporal homogeneous Poisson Point Process (PPP) [57] with intensity γ_e events/sec-m^2. Each event is stored at its closest peer, so since each peer is associated with a cell of size δ^2 m^2, each peer sees an intensity of $\gamma_e\delta^2$ events/sec. Similarly, queries are submitted according to an independent spatio-temporal homogeneous PPP with intensity γ_q queries/sec-m^2. They are assumed to be processed by the closest (proxy) peer, so each such peer supports an intensity $\gamma_q\delta^2$ queries/sec. The destination of a query is *uniformly* distributed among the peers in a diamond with 'radius' l m centered around the associated source (proxy) peer. The parameter l is a measure of the spatial locality of the queries. The lower l, the higher the spatial locality. Queries are assumed to be greedily routed on the grid from the source peer to the destination, with ties broken at random. Queries are assumed to be point range queries, i.e., RQ(l, r) where r is very small. To make things tractable, we assume that the service time for processing, storing or relaying a query

at a peer is an exponential with parameter μ. With these assumptions this model corresponds to a network of M/M/1 queues with generalized routing [41]. This grid model is only a first-order caricature of a homogeneous system in terms of both the traffic and peer topology, but as we will see gives some key insights on the characteristics of such systems.

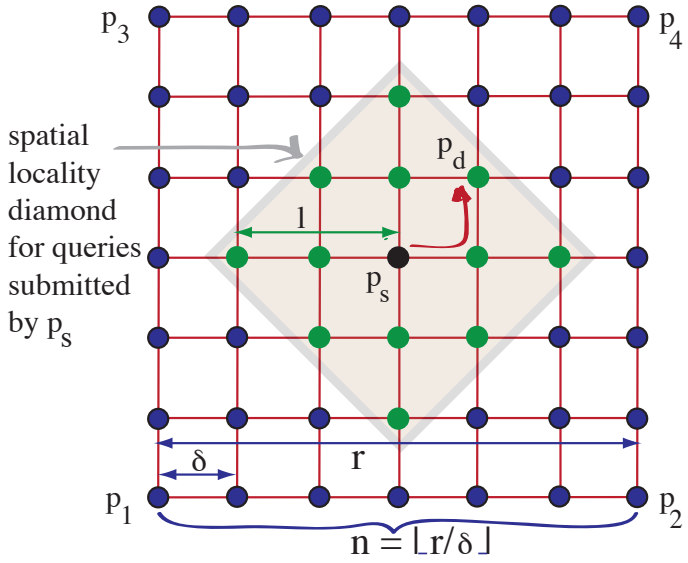

Figure 3.5: Idealized grid model.

Symmetry in this grid model allows us to easily estimate the end-to-end delays on the overlay network. To account for the relayed traffic let $h(\delta)$ denote the average number of hops traversed by a typical query when the peers are spaced δ m apart. Then, the total traffic load that will be serviced in the network consists of two components: (1) storing of events, which is $n^2\gamma_e\delta^2$; and (2) query relaying and processing, which is the number of peers n^2, times the query load per peer, times the average hop count, i.e., $n^2 \times \gamma_q \times \delta^2 \times$

$\max(1, h(\delta))$. By symmetry, the total load is divided equally amongst the n^2 peers giving a load per peer of

$$\gamma(\delta) = \gamma_q \times \delta^2 \times \max(1, h(\delta)) + \gamma_e \delta^2. \tag{3.1}$$

The average end-to-end delay, $\overline{D}$, for a typical query corresponds to traversing $h(\delta)$ M/M/1 queues each supporting a traffic intensity $\gamma(\delta)$. Given the additivity of delays experienced across the network we have that:

$$\overline{D} = \max(1, h(\delta)) \frac{1}{\mu - \gamma(\delta)}. \tag{3.2}$$

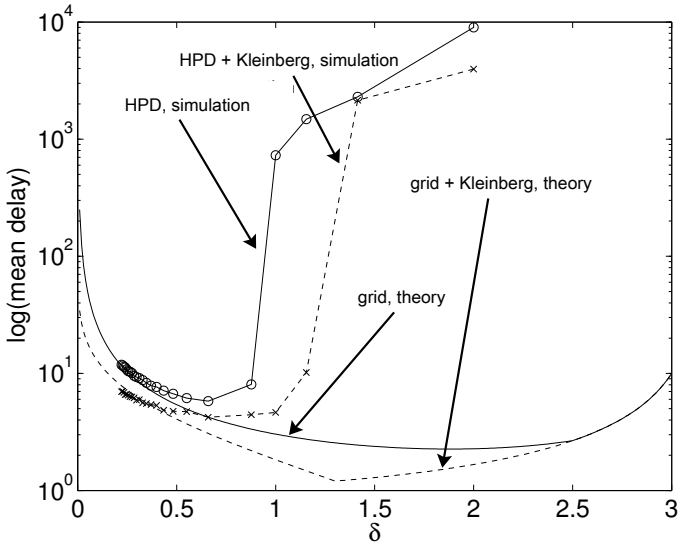

Figure 3.6: Average query delay vs. cell side δ.

Note that topology and locality of the traffic on the overlay network impact the mean hop count $h(\delta)$ of a typical query. If each peer maintains

edges to only its grid neighbors, the number of hops a query traverses is roughly proportional to the mean distance and inversely proportional to δ, i.e.,

$$h(\delta) \approx c_1 \frac{l}{\delta} \tag{3.3}$$

where c_1 is a constant. Fig. 3.6 exhibits a plot of the query's mean delay, i.e., Eq. 3.2, on a log scale versus the grid spacing δ when there is no locality, i.e., $l = r/2$, and $\mu = 1$, $r = 10$, $\gamma_q = 0.05$, $\gamma_e = 0.05$ and $c_1 = 0.5$. As can be seen, a higher density of peers, i.e., lower δ, leads to a high mean query delay, due to the increased hop count of paths, while a lower density, i.e., high δ, leads to high mean query delays due to increased congestion in the traversed peers. Indeed, if δ is too high, the traffic load on a peer may exceed its capacity. For a given set of system parameters, there is an optimal density for peers, see Fig. 3.6.

As mentioned in Section 3.4.1 one can improve the performance in this system by including additional edges in the graph. In particular adding a single additional edge per peer as proposed by Kleinberg (see Fact 3.7) reduces the mean hop count to

$$h(\delta) \approx c_2 (\log(\frac{l}{\sqrt{2}\delta}))^2 \tag{3.4}$$

where c_2 is a constant. Note in Eq. 3.4 we have accounted for the fact that our queries are uniform in a locality diamond of radius l. The diamond centered at each peer includes roughly $2\frac{l^2}{\delta^2}$ peers, giving an 'equivalent' square grid to that considered by Kleinberg with $n = \sqrt{2}\frac{l}{\delta}$ and divided by a factor of 2 to account for distances in wrap-around geometry. As expected and shown in

Fig. 3.6 these new edges substantially reduce the mean end-to-end delay on the overlay – here we set $c_2 = 1$. Indeed the best performance is achieved with a much larger δ, i.e., one can get away with a much lower density of peers.

Note, that the quantity in Eq. 3.4 is the mean number of hops. Each peer, in this case, will be the 'target' of a different number of Kleinberg edges. That means that the traffic intensity received by each peer is *not* the same anymore. Thus, the quantity

$$\frac{1}{\mu - \gamma(\delta)}$$

in Eq. 3.2, where $\gamma(\delta)$ is given in Eq. 3.1, representing the mean per hop delay, is a lower bound for the case of Kleinberg edges, since the mean per hop delay is a convex function of the traffic intensity. In Fig. 3.7 we show that the lower bound is tight.

Admittedly, the grid model is idealized. To capture more realistic topologies, we simulated a Homogeneous Poisson Delaunay (HPD) topology. In an HPD topology the peers' locations are generated by a homogeneous Poisson spatial point process with intensity λ, and peer connectivity follows the edges of the corresponding DG. The peers were placed inside a $r \times r$ square region, and we used the same event/query model as the one described in the previous paragraph for the grid topology. Additionally, we consider a HPD topology augmented by Kleinberg edges. In Fig. 3.6, we include our simulated results for HPD topologies. We scale the horizontal axis for grid and HPD networks via the correspondence $\delta \sim \frac{\sqrt{r^2}}{\sqrt{n^2}} \leftrightarrow \frac{1}{\sqrt{\lambda}}$. The theoretical results based on the grid capture the qualitative behavior of the HPD topology, but

there are significant quantitative discrepancies for high δ/low λ. The latter is due to the arbitrary choice of the constants involved in Eqs. 3.3,3.4 and the effect of the statistical variations of the cells' sizes. Indeed, in contrast to the grid model, in the HPD topology the cell sizes for all the peers are not the same. This means that a peer with a bigger cell than the rest will likely see more traffic - this can lead to instability affecting all the queries routed through that peer. This explains the effect seen for high δ/low λ. When λ is high, the average cell size goes down, and the effect of varying cell sizes is mitigated; this is a fundamental property of HPDs. In this case the traffic becomes more uniform essentially like a grid; observe the convergence of the theoretical and simulated curves. This motivates the need to devise ways to adapt the topology to non-uniformities either by adapting the peers' cell sizes, e.g., by associating a "virtual" location with a peer, allowing modification of the associated Voronoi cells or the edges connecting the peers. We discuss this in Chapter 4.

Fig. 3.7 shows some simulated results for the mean query delay on various overlay topologies as query locality is varied. We compare three topologies: HPD, HPD augmented with Kleinberg edges, and HPD augmented with Kleinberg edges restricted to the locality region of each peer. The values of the specific parameters used are $\mu = 1$, $\gamma_q = 0.1$, $\gamma_e = 0.05$, $r = 10$, $n^2 = 5000$. Each simulation lasted 100000 time units. From Fig. 3.7 it is clear that when the locality of the queries is high, e.g., 0.5, the delay of the queries is mostly due to the processing rather than queuing and the grid overlay performs well.

However, as locality is reduced, i.e., l increases, the benefit of Kleinberg edges appears, with an improvement in performance of up to 34% for this scenario. Further if the scale of locality were known up front, then limiting the edges within the locality region leads to a small yet consistent performance improvement, although simply adding edges throughout the whole network is robust and performs quite well. The graph also shows the analytical performance curves associated with Eqs. 3.2, 3.3 and 3.4, where c_1 and c_2 were estimated by matching the value of the equations with the values reported by our simulations for $l = \frac{r}{2} = 5$. As can be seen the analytical and simulation results match very well. This validates the ability of our model to qualitatively capture the performance of the actual system.

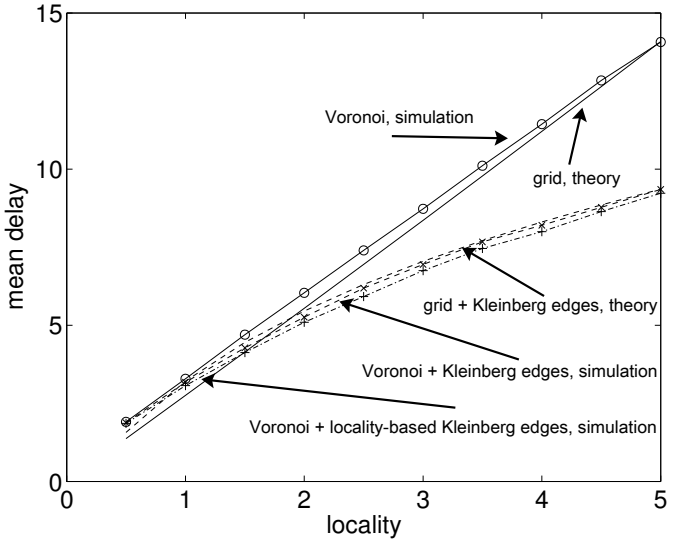

Figure 3.7: Average query delay vs. locality.

3.9.1 Scalability analysis

The above results might be contrasted with P2P networks for sharing (large) files, where increasing the number of peers wanting the same file typically improves the performance. Indeed, in such systems new peers place new demands but also serve as potential sources for the parts of a disaggregated file they are retrieving, i.e., they also increase system capacity. Our system, however, does not have such a characteristic since events are assumed to correspond to relatively small amounts of information and are stored in only one peer. The system is optimized to reduce bottlenecks associated with storing and resolving queries when there is a high intensity of events and queries, i.e., the overheads lie mainly in managing and finding data, rather than transmitting the data. To explore the performance scalability of our platform, let us consider how performance in our grid model scales as the spatio-temporal query intensity $\gamma_q(f)$ grows linearly, i.e.,

$$\gamma_q(f) = \gamma_q^0 f.$$

where f is the scaling factor and assume $\gamma_e(f) \propto \gamma_q(f)$, i.e., both grow linearly. Let us also consider various possible scalings for the service rate of each peer $\mu(f)$, the total number of peers in the network $n^2(f)$, and the locality of queries $l(f)$ as a function of f as follows:

$$\mu(f) = \mu_0 f^p, \quad n^2(f) = n_0^2 f^q, \quad \text{and} \quad l(f) = \frac{l_0}{f^s}, \quad p \geq 0, q \geq 0, s \geq 0$$

Note that such a scaling for the number of peers implies a scaling in the distance among the peers $\delta(f) \approx \Theta(f^{-\frac{q}{2}})$.

The central question we would like to tackle is what the relationships amongst p, q, s should be to ensure bounded mean query delays as f grows. The following proposition shows when this is the case, a proof is presented in Appendix B.

Proposition 3.9.1. Scalability. *For a linear growth in event/query traffic, a necessary condition for the grid network to have bounded delays is the polynomial scaling exponents for the number of peers, service capacity and locality p, q and s respectively to satisfy*

$$p \geq 1 - \frac{q}{2} - \min[\frac{q}{2}, s].$$

If the network is augmented with Kleinberg edges then $p > 1 - q$ may suffice.

This provides some nice insight on the scaling matter. Suppose peers have a fixed service capacity ($p = 0$) then if locality is also fixed ($s = 0$) then peers must scale at least quadratically ($q \geq 2$). Otherwise, with increased locality $s > 0$ peers can scale linearly ($q = 1$) but only if locality scaling is fast enough ($s > 1/2$). By contrast, if Kleinberg edges are added, locality plays a small role, i.e., asymptotically the additional edges have limited impact.

3.10 Fault-Tolerance

In Section 3.4 we discussed the elements of our architecture and how they might be managed in the presence of peer churn but assumed that churn happens gracefully, i.e., peers follow our protocols. In practice, these assump-

Type	Variable
$set < Peer >$	$holders$
$set < Peer >$	$owners$
$dictionary(Peer, set < Peer >)$	$neighbors$

Table 3.1: Definition of the class $EventRecord$ for an event e. We assume the existence of the primitive class $Peer$.

Type	Variable
$set < Peer >$	$neighbors$
$dictionary(Event, EventRecord)$	$events$
$set < Peer >$	$holdsEventsForMe$
$dictionary(Peer, set < Event >)$	$holdEventsForPeer$

Table 3.2: Data structures for fault-tolerance. We assume the existence of the primitive classes $Event$ and $Peer$. The class $EventRecord$ is defined in Table 3.1.

tions might be violated, e.g., peers may fail. This motivates considering mechanisms to achieve fault-tolerance.

The simplest approach to achieving fault-tolerance is replication. Replication is not optimal with respect to minimizing the total amount of storage required, e.g., as opposed to coding, still, it is used by some of the seminal P2P storage systems [68, 78]. Moreover, for our platform this is a good choice due to the high intensity of short-lived real-time events and the need for an approach with low computational complexity. To set the limits of our approach, we shall consider a well defined failure model for the peers under which the proposed protocols should maintain functionality. We start by introducing the updated rules for event storage with replication and the associated failure

model. Then we present fault-tolerant algorithms for data management and query processing. Topology management and routing as well as peer association with the P2P network will not be discussed as they are the same as those presented in Section 3.4.

We introduce a revised rule for event storage that extends Rule 3.7.1: each event is stored in the $k+1$ closest available peers to the event's location - this allows us to tolerate up to k peers' failures. Observe that $k = 0$ amounts to no replication. We refer to these $k+1$ peers storing the event as its "holders". In case some of the $k+1$ closest peers to an event are unavailable to store it because, e.g., they have ran out of space, they are substituted by peers that are further away from the event. Collectively, we refer to all the peers involved in the storage of the event as its "owners", i.e., the set of owners of an event includes its holders as well as any peers that should be holders by virtue of their proximity to the event but were not available to store events. We present formal definitions for these terms:

Rule 3.10.1. Fault-Tolerant Storage. *Each event e is stored at the $k+1$ peers $\{h_i(e), i = 0, \ldots, k\} \subset P$ that are closest to e.location and are available to store it. We will denote this set of peers by holders(e). The* owners *of an event e, owners$(e) \triangleq \{p_0(e), \ldots, p_{n-1}(e)\}$, are all the peers that are closer to e.location than $h_k(e)$, including $h_k(e)$.* [1]

Obviously, $p_{n-1}(e) = h_k(e)$.

[1] In the sequel, when it is clear from the context, we will refer to a peer as an *owner* when it is an owner but not a *holder*.

Storing all the replicas for an event in the closest peers to its location, limits the fault-tolerance of our approach as neighboring peers are more likely to fail together. A similar issue is recognized in the context of the 'Topology-based nodeId assignment' topology-aware routing policy in structured P2P overlay networks, see [19]. We consider this part of the design trade-offs we adopt in our platform.

As will become apparent in Section 4.1 when we address issues of limited storage at the peers, our solutions for fault-tolerance and limited storage share similarities. Therefore, in the interest of generality, in Rule 3.10.1 we require peers to be available to store events. In the unlikely event that there does not exist $k+1$ available peers to store an event, the procedure stops at the last holder. Additionally, the algorithms in the current section will be presented in their most general form, taking storage limitations into account, although the topics of fault-tolerance and storage limitation are in general distinct and can be examined separately.

As explained previously, maintaining $k+1$ replicas per event allows us to tolerate up to k peer failures. Our failure model is consistent with that observation:

Assumption 3.11. Failure Model.

1. *Peers fail according to the* fail-stop *model, i.e., once a peer fails it remains silent and unresponsive.*

2. *For each event stored in our platform, up to k of its holders may fail until one of the remaining holders executes our protocol.*

3. *The underlying distributed protocol for topology maintenance can maintain the overlay topology correctly in the presence of failures. Peers execute our protocol having complete and accurate knowledge of their neighbors.*

4. *No overlay topology change occurs while our data management/query processing algorithms execute.*

5. *All messages sent are eventually acknowledged within a bounded period unless the intended recipient has failed.*

Our failure model is an extension to the one used in [52] with $k > 1$. Our assumption about the number of holders that may fail before one of the remaining holders is notified about their failures, will serve to restore the correct number of replicas. The protocol presented in [52] is an example of an underlying protocol for maintaining the DG. The assumption that no overlay topology changes occur while our data management/query processing algorithms execute will serve to ensure that our algorithm is correct as well as the results of a query are accurate. Finally, our assumption that all successfully received messages are acknowledged within a bounded period will help detect failed peers.

Before we proceed with presenting our protocols for data management, we shall state our assumptions about the data-structures to be maintained

by each peer. These data-structures will be leveraged in the operation of our protocols. Each peer needs to maintain auxiliary data structures to implement Rule 3.10.1. Table 1 exhibits the needed data structures that we explain below.

Assumption 3.12. Data Structures for Fault-Tolerance. *Each peer maintains the data structures shown in Table 3.2.*

Our failure model ensures that the information stored in the data structures maintained by each peer is complete and accurate when it is executing our protocols.

Each peer p maintains its neighbors in the DG $N(p)$ in the variable $p.neighbors$. According to Assumption 3.12:

$$N(p) == p.neighbors.$$

To every event e stored by p we associate, through a $1-1$ mapping, an instance, er, of the class $EventRecord$ and store it in the dictionary $p.events$ under the key e. The class $EventRecord$ works as a place-holder for storing the information about the holders and the owners of events. Additionally, the class $EventRecord$ contains information about the neighbors of e's owners in the DG: the dictionary $p.events[e].neighbors$ has an entry for each owner of e - the value stored is the neighbors of that owner. According to Assumption 3.12:

$$(er.owners == er.neighbors.keys) \wedge$$
$$(\forall o \in er.owners : N(o) == er.neighbors[o]).$$

If p is a holder of the event e, then it maintains a copy of er, i.e.,

$$p \in holders(e) \Leftrightarrow e \in events.keys.$$

Finally, to facilitate processing, we maintain at each holder the following bookkeeping structures: the set $holdsEventsForMe$ and the dictionary $holdEventsForPeer$. The former contains all peers that hold events for which the current peer is an owner. According to Assumption 3.12:

$$(q \in p.holdsEventsForMe) == (\exists e \in q.events.keys :$$
$$(p \in q.events[e].owners) \wedge (p \notin q.events[e].holders)).$$

The latter maps to each owner the set of events for which the current peer is a holder. According to Assumption 3.12:

$$(q \in p.holdEventsForPeer.keys) == \exists e \in p.events.keys :$$
$$(q \in p.events[e].owners) \wedge (q \notin p.events[e].holders)).$$

To better illustrate the concepts of holders and owners and how these are encoded in our data structures, we present a small example. In Fig. 3.8 for the event e and $k = 1$ assuming peer p_1 is unable to store e, the following

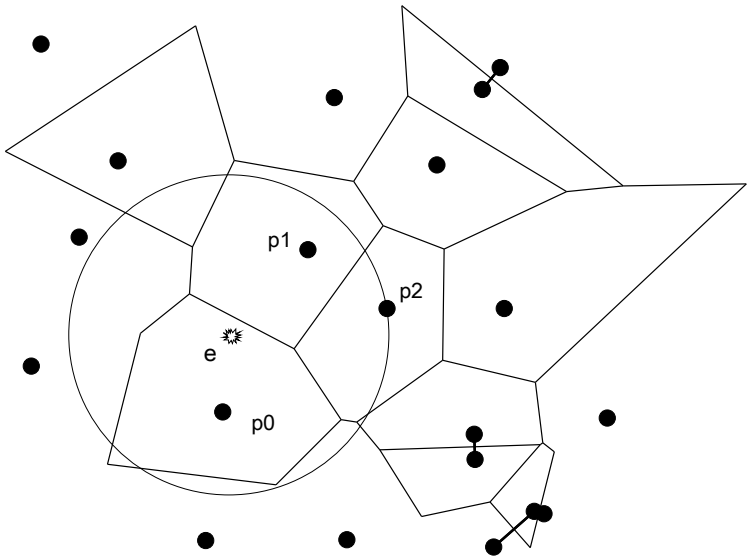

Figure 3.8: Data-Structures example

.

relationships hold:

$$
\begin{aligned}
owners &= \{p_0, p_1, p_2\},\\
holders &= \{p_0, p_2\},\\
p_0.holdsEventsForMe &= \emptyset,\\
p_0.holdEventsForPeer &= \{p_1\},\\
p_1.holdsEventsForMe &= \{p_0, p_2\},\\
p_1.holdEventsForPeer &= \emptyset,\\
p2.holdsEventsForMe &= \emptyset,\\
p2.holdEventsForPeer &= \{p_1\}.
\end{aligned}
$$

3.12.1 Fault-Tolerant Data Management

In this section we will specify how events are stored and deleted from our platform according to Rule 3.10.1. This entails updating the records maintained by the holders of each event stored/deleted. Additionally, we will demonstrate how to update the data structures maintained by the peers when the overlay topology changes due to a peer joining or leaving/failing.

Event Storage and Deletion. We start by describing a distributed algorithm for identifying the holders and the owners of an event. Consider a new event e to be stored in our platform. Our storage algorithm identifies *recursively* the holders/owners of e starting from $p_0(e)$ and terminating with $h_k(e)$. Once all $k+1$ holders of e have been identified, $h_k(e)$ communicates the information about them to the rest of the holders/owners, which in turn update their records about e.

Storing an event e, as in the case without peer failures, starts by greedily routing the event towards the closest peer to its location, $p_0(e)$. We do not present the messages used for that purpose. Upon reception of e, $p_0(e)$ is responsible for 'bootstrapping' the procedure for identifying $holders(e)$ and $owners(e)$, according to Rule 3.10.1. Alg. 2 exhibits pseudo-code describing the actions of peers upon receiving the various messages exchanged.

Alg. 2 requires every potential holder to receive the special message, "recursiveStore(e, er)". The message specifies the event to be stored as well as the corresponding record that will be used for recording the owners/holders

and their neighbors. First, $p_0(e)$ bootstraps the procedure by 'sending' the message "recursiveStore(e, er)" to itself. Assuming it has space to store e, p_0 adds itself to the holders/owners of e, by updating $er.holders$ and $er.owners$ respectively, and proceeds to identify p_1. Otherwise, p_0 records itself as merely an owner and proceeds to identify p_1 as well. The following proposition serves as the basis for identifying the next owner:

Proposition 3.12.1. Evolution of a peer's set of neighbors

1. *The jth closest peer to an event $e, p_j, j = 1, \ldots, n$, is such that $p_j \in N(p_{j-1}|P \setminus \{p_0, \ldots, p_{j-2}\})$, i.e., p_j is a neighbor of p_{j-1} in the Voronoi tessellation induced by the locations of the peers in $P \setminus \{p_0, \ldots, p_{j-2}\}$.*

2. *Let $p, q \in P$ be two neighbor peers in the DG and $N(p|P), N(q|P)$ be the sets of their neighbors respectively. If p leaves/fails, the new set of neighbors for q will be $N(q|P \setminus \{p\}) = (N(q|P) \cup S_{N(p|P)}) \setminus \{p\} \subset N(q|P) \cup N(p|P)$, where $S_{N(p|P)} \subset N(p|P)$ is the set of neighbors of p that will also be neighbors of q after p fails/leaves.*

The first part of Prop. 3.12.1 follows by definition. To see why the second part of Prop. 3.12.1 holds we recognize that the set of q's neighbors can only grow, by the statement of Prop. 3.12.1 no new peer joins to destroy any edge, so all of q's neighbors, apart from p, will remain remain as neighbors in $P - \{p\}$. The new neighbors of q in $P - \{p\}$ can only be p's neighbors in P, since Fact 3.6 ensures that *only* p's neighbors will have to update their

neighborhood. Therefore, the set of q's neighbors $N(q|P \setminus \{p\})$ will evolve to $N(q|P) \cup S_{N(p|P)}$, where $S_{N(p|P)} \subset N(p|P)$.

Based on the previous facts we deduce that

$$\begin{aligned}
p_1 &\in N(p_0|P), \\
p_2 &\in N(p_1|P \setminus \{p_0\}) \subseteq (N(p_1|P) \cup N(p_0|P)) \setminus \{p_0\}, \\
p_3 &\in N(p_2|P \setminus \{p_0, p_1\}) \subseteq (N(p_2|P) \cup N(p_0|P) \cup N(p_1|P)) \setminus \{p_0, p_1\}, \\
&\vdots \\
p_{j+1} &\in N(p_j|P \setminus \{p_0, \ldots, p_{j-1}\}) \\
&\subseteq (N(p_j|P) \cup (\cup_{i=0}^{j-1} N(p_i|P))) \setminus \{p_0, \ldots, p_{j-1}\}.
\end{aligned}$$

The previous equations suggest that p_1 is the closest neighbor of p_0 in the DG to *e.location*. Once p_1 has been determined, another iteration begins by p_0 sending to p_1 the message "recursiveStore(e, er)". Upon reception of the "recursiveStore" message p_1 adds itself to the owners, or the holders of e or both and proceeds to identify p_2. To achieve that, the previous equations suggest that p_2 will be the closest peer to *e.location* among the peers in $(N(p_0|P) \cup N(p_1|P)) \setminus \{p_0, p_1\}$. The information about p_0, p_1 and their neighbors is encoded in the structure er. Using the above strategy recursively one can identify all the holders/owners of an event by identifying the j^{th} closest peer to e at the j^{th} iteration.

When the last holder is identified, it notifies the rest of the owners about the event's storage by sending an "update(e, er)" message. Upon reception of

the "update" message, each holder will update its record corresponding to e as follows:

$$h_0[e] = h_1[e] = \cdots = h_k[e] = er.$$

In addition, each owner will update its $holdsEventsForMe$ set to include the holders of e. Each holder will update the value stored in the dictionary $holdEventsForPeer$ for each owner, to include the event e. Finally, each holder will make sure that past owners for which it does not store any event anymore are removed from its dictionary $holdEventsForPeer$. These owners are notified to remove the holder from their $holdsEventsForMe$ sets.

3.12.2 Event Deletion

We delete events as soon as they expire, as in the case without failures. A peer p holding a recently expired event e should remove it from the value corresponding to each owner of e in the dictionary $p.holdEventsForPeer$. If e is the last event p is holding for a particular peer q, p will remove the entry for q from the dictionary $p.holdEventsForPeer$ and will notify q to remove p from the set $q.holdsEventsForMe$. Finally, the record for e will be deleted from $p.events$. The necessary bookkeeping is described in Alg. 3.

3.12.3 Handling a new peer joining the P2P overlay

Consider a new peer q joining the overlay. For all events e s.t. q is closer to $e.location$ than $h_k(e)$, q should be an owner of e in $P \cup \{q\}$. Additionally, if q has space to store e, it should also become a holder of e, removing $h_k(e)$

Algorithm 2 Fault-Tolerant Event Storage

1: $Event : e; EventRecord : er; Peer : p, q$
2: {peer p receives ***" recursiveStore(*** e, er ***)"*** from peer q}
3: insert p to $er.owners$
4: $er.neighbors[p] \leftarrow N(p)$
5: **if** p has space to store e **then**
6: insert p to $er.holders$
7: **end if**
8: **if** $|er.holders| < k + 1$ **then**
9: $N \leftarrow \cap_{t \in er.owners} er.neighbors[t]$
10: **if** $\emptyset \neq N \setminus er.owners$ **then**
11: find $r \in N \setminus er.owners$ that is the closest to e
12: send message "recursiveStore(e, er)" to r
13: **end if**
14: **else**
15: {we have identified all the holders for e.}
16: **for** $r \in er.owners$ **do**
17: send "update(e, er)" to r
18: **end for**
19: **end if**
20: $Event : e; EventRecord : er; Peer : r, p$
21: {peer r receives ***"update(*** e, er ***)"*** from peer p}
22: **if** $r \notin er.holders$ **then**
23: **for** $t \in er.holders$ **do**
24: insert t to $p.holdsEventsForMe$
25: **end for**
26: **else**
27: **for** $(t \in er.owners) \wedge (t \notin er.holders)$ **do**
28: insert e to $r.holdEventsForPeer[t]$
29: **end for**
30: **for** $t \in r.holdEventsForPeer.keys : (e \in r.holdEventsForPeer[t]) \wedge (t \notin eh.owners)$ **do**
31: remove e from $r.holdEventsForPeer[t]$
32: **if** $\emptyset == r.holdEventsForPeer[t]$ **then**
33: remove the entry for t from $r.holdEventsForPeer$
34: notify t to remove r from $t.holdsEventsForMe$
35: **end if**
36: **end for**
37: $r.events[e] \leftarrow er$
38: **end if**

Algorithm 3 Fault-Tolerant Event Deletion

```
1: Event : e; Peer : p
2: {event e to be deleted from peer p}
3: for (q ∈ p.events[e].owners) ∧ (q ∉ p.events[e].holders) do
4:    remove e from p.holdEventsForPeer[q]
5:    if ∅ == p.holdEventsForPeer[q] then
6:       remove the entry for q from p.holdEventsForPeer
7:       notify q to remove p from q.holdsEventsForMe.
8:    end if
9: end for
10: remove the entry for e from p.events
```

from the set of holders. In this section, we describe a recursive distributed algorithm for updating e's storage. Algo. 4 exhibits the relevant pseudo-code.

'Bootstrapping' the update of e's storage. The underlying topology management protocol ensures that upon q joining the overlay, its neighbors will be notified about it. The following fact guarantees that at least one of the neighbors of q is an owner of e too.

Fact 3.13. New Owner Detection. *Consider any event e stored in our platform and a new owner of e, q, that joins the platform. Then, there exists at least one owner of e among the neighbors of q:*

$$owners(e) \cap N(q) \neq \emptyset.$$

We offer a proof in the Appendix B. Consider, p, the closest peer to $e.location$ among the neighbors of q. We require p to 'bootstrap' the procedure for updating e's storage.

If p is a holder of e, it notifies h_k to release event e by sending it a "release(e, q)" message. Upon reception of the "release" message, h_k removes itself from the set of holders of e and removes the owners of e from e's entry in its $holdEventsForPeer$ directory. If it does not hold events for some peers anymore, they are removed from the dictionary $holdEventsForPeer$ and they are notified to remove h_k from their set $holdsEventsForMe$. Additionally, h_k removes all the other owners between h_{k-1} and itself to ensure that $owners(e)$ does not contain any peer that is farther from $e.location$ than the farthest holder, h_{k-1}. Finally, h_k notifies q to hold event e by sending the message 'recursiveStore$(e, h_k.events[e])$' to it. At that point, there exist k replicas of e stored in the platform, and the procedure to identify the last replica continues as in the case of storing the event in the first place, described in Algo. 2.

If p is just an owner of e, but not a holder, it cannot identify $h_k(e)$. Therefore, p notifies the peers that hold events on its behalf, to do that, by sending them a "join(q)" message.. The closest holder to p that holds a replica for e, say peer r, will notify h_k to release its replica by sending it a "release(e, q) message. From that point, the procedure continues as described for the case that p is a holder.

In a more efficient implementation, messages for different events can be aggregated in a single message reducing the communication cost substantially.

Algorithm 4 Handling a peer joining

1: $Peer : p, q$
2: {peer p detects peer q as a **new neighbor** in the DG}
3: **for** all events $e \in p.events.keys$ **do**
4: **if** $(||q - e.location||) < ||h_k(e) - e.location||) \wedge (p$ is the closest holder of e to $q)$ **then**
5: send msg "release(e, q)" to $h_k(e)$
6: **end if**
7: **end for**
8: **for** $r \in p.holdsEventsForMe.keys$ **do**
9: send "join(q)" to r
10: **end for**

11: $Peer : h_k, q, p; Event : e$
12: {peer h_k receives ***"release(e, q)"*** message from peer p}
13: remove h_k from $h_k.events[e].holders$
14: **for** $t \in h_k.events[e].owners : t \notin h_k.events[e].holders$ **do**
15: remove e from $h_k.holdEventsForPeer[t]$
16: **if** $\emptyset == h_k.holdEventsForPeer[t]$ **then**
17: remove the entry for t from $h_k.holdEventsForPeer$
18: notify t to remove h_k from $t.holdsEventsForMe$.
19: **end if**
20: **end for**
21: **for** $t \in (h_{k-1}, h_k] : t \notin h_k.events[e].holders$ **do**
22: remove t from $h_k.events[e].owners$
23: **end for**
24: send msg "recursiveStore$(e, h_k.events[e])$" to q
25: remove the entry for e from $h_k.events$

26: $Peer : r, q, p$
27: {peer r receives ***"join(q)"*** from peer p}
28: **for** all events $e \in r.holdEventsForPeer[p]$ **do**
29: **if** $(||q - e.location|| < ||h_k(e) - e.location||) \wedge (p$ is the closest owner of e to $q) \wedge (r$ is the closest holder to $p)$ **then**
30: send msg "release(e, q)" to $h_k(e)$
31: **end if**
32: **end for**

3.13.1 Handling a peer leaving the P2P overlay

A peer leaving the P2P network gracefully will have to ensure that replicas for all the events it holds are replaced in the platform. Additionally, the storage of the events for which it is merely an owner has to be updated. In this section, we present a distributed algorithm to achieve the above tasks. The relevant pseudo-code is shown in Algo. 5.

Consider peer p leaving the P2P overlay. For each event, e, that peer p holds, p contacts the closest holder of e to it, q, and 'delegates' to q, the task of looking for a new holder for e. This happens by sending a "handle(e, p)" message to q. Upon reception of the "handle" message, q removes p from the set of owners/holders of e. If p is just an owner, q sends *update* messages to the rest of the holders so as to update their $holdEventsForPeer$ dictionaries. Otherwise, if p is the farthest holder from $e.location$, it notifies the owners of e to remove p from their sets $holdsEventsForMe$ and removes all the owners between h_{k-1} and p to ensure that $owners(e)$ does not contain any peer that is farther from $e.location$ than the farthest holder, h_{k-1}. At that point there exist k replicas of e stored in the platform, and the procedure to identify the last replica continues as in the case of storing the event in the first place, described in Algo. 2.

Finally, p has to notify all the holders of the events for which it is merely an owner, but not a holder, about its departure. For each such event e, the closest holder to p, q, will remove p from the set of owners and notify all the other holders by sending them an "update$(e, q.events[e])$" message.

In a more efficient implementation, messages for different events can be aggregated in a single message, reducing the communication cost substantially.

Algorithm 5 Handling a peer's departure

1: $Peer : p$
2: {peer p **leaves** the overlay}
3: **for** all events $e \in p.events.keys$ **do**
4: $q \leftarrow$ closest holder of e to p
5: send message "handle(e, p)" to q
6: **end for**
7: **for** $r \in p.holdsEventsForMe.keys$ **do**
8: send message "depart()" to r
9: **end for**

10: $Peer : q, p; Event : e$
11: {peer q receives a ***"handle(e, p)"*** message to update the storage of event e after peer p has left/failed}
12: remove p from $q.events[e].owners$
13: **if** $p \notin q.events[e].holders$ **then**
14: **for** $t \in q.events[e].holders$ **do**
15: send "update$(e, q.events[e])$" to t
16: **end for**
17: **else**
18: remove p from $q.events[e].holders$
19: **for** $(t \in q.events[e].owners) \wedge (t \notin q.events[e].holders)$ **do**
20: notify t to remove p from $t.holdsEventsForMe$
21: **end for**
22: **if** $p == h_k$ **then**
23: **for** $t \in (h_{k-1}, h_k) : t \notin q.events[e].holders$ **do**
24: remove t from $q.events[e].owners$
25: **end for**
26: **end if**
27: send msg "recursiveStore$(e, q.events[e])$" to itself
28: **end if**

29: $Peer : r, p$
30: {peer r receives a ***"depart"*** message from peer p}
31: **for** all events $e \in r.holdEventsForPeer[p]$ **do**
32: **if** (r is the closest holder of e to p) $\wedge$ $(p \in r.events[e].owners)$ **then**
33: send message "handle(e, p)" to itself
34: **end if**
35: **end for**

3.13.2 Handling a peer's failure

The case of a peer q detecting that one of its neighbors, peer p, has failed presents similarities with the case of a peer leaving the overlay gracefully. After peer p fails, the underlying topology management protocol will notify its neighbors about its failure, see Assumption 3.11. In this section we will present a distributed algorithm to update the storage for all events for which p was a holder/owner. Algo. 6 has the relevant pseudo-code. Fact 3.13 ensures that for each event, e, for which p is an owner/holder, there exists a neighbor, assume it is q, that is also an owner. If q is the closest holder of e to p, we require that it 'bootstrap' the update of e's storage by sending a "handle(e, p)" message to itself. From that point on, the procedure is similar to the case of peer p leaving the overlay gracefully.

If q is only an owner of e, but not a holder, it will notify all the peers that hold events on its behalf to update e's storage by sending them a "fail(p)" message. Among them, the closest holder of e to p will update e's storage by sending a "handle(e, p)" message to itself. At that point there are k replicas of e in the platform and the procedure to identify the last platform will proceed as in the case where the event was first stored.

In the case that more than one peer fails, the closest holder of e to p might have failed too. In that case, one of the remaining holders of e will update its storage. Such a holder always exists since according to Assumption 3.11 up to k holders of e may fail before one of the remaining holders executes our protocol. Let $u = p_j(e)$ denote one of the remaining holders. When

u executes our protocol, either because one of his neighbor holders failed or because it received a "fail" message from one of the owners of e, it will start a timer waiting for the update message for each event for which p was an owner. The j^{th} time the timer goes off, assuming all the holders from p_0 to p_{j-1} have failed, u will update the storage for the corresponding event.

In a more efficient implementation, messages for different events can be aggregated in a single message reducing the communication cost substantially.

Algorithm 6 Handling a peer's failure

1: $Peer : p, q$
2: {peer q finds out that neighbor peer p has **failed**}
3: **for** all events $e \in q.events.keys$ **do**
4: **if** (q is the closest holder of e to p) $\wedge$ ($p \in q.events[e].owners$) **then**
5: send message "handle(e, p)" to itself
6: **else**
7: start a timer waiting for the "update" message for event e.
8: **end if**
9: **end for**
10: **for** $r \in p.holdsEventsForMe.keys$ **do**
11: send message "fail(p)" to r
12: **end for**

13: $Peer : p, q$
14: {peer r receives ***"fail(p)"*** message from peer q}

15: **for** all events $e \in r.holdEventsForPeer[q]$ **do**
16: **if** (r is the closest holder of e to p) $\wedge$ ($p \in r.events[e].owners$) **then**
17: send message "handle(e, p)" to itself
18: **else**
19: start a timer waiting for the "update" message for event e.
20: **end if**
21: **end for**

22: $Peer : p, q$
23: {timer for the "update" message for event e goes off for the i^{th} time}
24: remove the i^{th} closest holder of e from $p.events[e].owners$,$p.events[e].holders$.
25: **if** q is the $(i+1)^{th}$ closest holder of e to p **then**
26: **for** $t \in \{1^{st}, \ldots, i^{th}$ closest holders of e to $p\} \cup \{p\}$ **do**
27: send message "handle(e, t)" to itself
28: **end for**
29: **else**
30: start a timer waiting for the "update" message for event e
31: **end if**

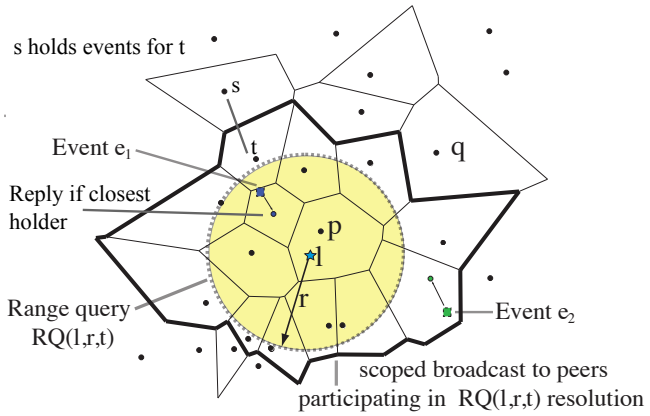

Figure 3.9: Fault-tolerant query resolution.

3.13.3 Fault-Tolerant query processing

In §3.4 we identified the *range* query as the fundamental building block supported by our platform for realizing more complex queries. Alg. 1 described its resolution when peers do not fail. In this section we present Alg. 7, a modified version of Alg. 1, which takes fault-tolerance into account.

Just as in the case with no peer failures, a range query $RQ(l, r)$ is initially issued by a peer or a proxy peer operating on behalf of another entity using the overlay. The decision to store the $k + 1$ replicas for each event at the $k + 1$ closest peers to the event's location allows us to retain the resolving strategy essentially unaltered. The query is initially routed greedily towards location l as depicted in Fig. 3.9. The closest peer to that location bootstraps its resolution by checking for events stored locally that are inside the disk specified by the query. In addition to the events stored locally, a peer has to check for events stored in other peers on its behalf as well. A peer replies

only for the events for which it is the closest holder. After these events have been acquired the resolution proceeds as in the case with no faults with the peer flooding the query to its neighbors in the DG. [2] The performance of a flood-based algorithm is acceptable in a restricted disk under Assumption 3.3, still, the same heuristics as for the case with no failures can be used to reduce the resulting traffic, e.g. never forward the same query twice, never forward the query back to the peer from which the query was received. The recursive resolution stops at the peers whose cells do not overlap with the disk specified at the query. At that point each peer involved in the resolution returns the events it has gathered to the peer from which it received the query. Finally, the closest peer to the location mentioned in the query returns the results to the peer that issued the query. Again, the response can be sent directly to the IP address of the source instead of being propagated on the overlay.

3.14 Conclusions

We have introduced a novel P2P architecture for storing and querying events that exhibit locality in space and time. The core idea is to exploit spatial locality by aligning the overlay topology with the Delaunay Graph induced by peer locations, and augmenting it by Kleinberg edges. We have extended our data management algorithms to address fault-tolerance. Indeed, storing extra replicas for each event in the 2nd,3rd,... k, closest peers to the event's

[2] In practice, the flooding and acquiring the events stored on other peers on a peer's behalf can proceed in parallel.

Algorithm 7 Range Query Resolution

1: **Resolve Query**$(q, p, RQ(l, r))$ {resolves $RQ(l, r)$ on behalf of peer q at peer p. }
2: $response\_set = \emptyset$
3: **if** $B(l, r) \cap C_p \neq \emptyset$ **then**
4: **for** all events $e \in p.events.keys$ s.t. $(e.location \in B(l, r)) \wedge (p$ is the closest peer to e among the peers in $p.events[e].holders$ **do**
5: $response\_set = response\_set \cup \{e\}$
6: **end for**
7: **for** $t \in p.holdsEventsForMe$ **do**
8: $response\_set = response\_set \cup HolderReport(p, t, RQ(l, r))$
9: **end for**
10: **for** $t \in N(p)$ **do**
11: $response\_set = response\_set \cup ResolveQuery(p, t, RQ(l, r))$
12: **end for**
13: **end if**
14: send $response\_set$ to q
15: **Holder Report**$(q, p, RQ(l, r))$ {returns to peer q the events peer p stores on its behalf}
16: $s = \emptyset$
17: **for** $e \in p.holdEventsForPeer[q]$ **do**
18: **if** $(e.location \in B(l, r)) \wedge (q$ is the closest peer to e among the peers in $p.events[e].holders)$ **then**
19: $s.insert(e)$
20: **end if**
21: **end for**
22: send s to q

associated location or utilizing these peers in case the closest peer runs out of storage space are natural extensions to our data management algorithms. In Chapter 4 we will further address the impact of non-homogeneity on the performance of our network.

Chapter 4

Addressing the Impact of Non Uniformities in Topology and Traffic

4.1 Introduction

In Chapter 3 we focused on homogeneous network topologies and traffic. More specifically, we assumed that peers join the P2P network with equal probability at any location. As far as the traffic is concerned, we assumed that events occur at all locations/times with equal probability and peers make queries from every location to every location (possibly within some range) with equal probability. Additionally, we assumed that peers have unlimited event storage capacity.

In practice, these assumptions need not hold. Variations will exist that may impact the performance dramatically. There may be higher event / query loads to some regions that have higher interest at some point in time, e.g., think of the area around a famous exhibit in a museum during the time the museum is open. There may be fewer peers in some locations as compared to others, e.g., think of the area of the downtown of a city during a weekend. Such variations can result in a subset of the overlay peers being overloaded. Moreover, overlay peers may have limited storage capabilities or may choose to

limit the amount of storage they contribute resulting in local resource scarcity.

In this section, we study the following techniques to offset the effect of such non-uniformities:

1. storage pooling, where peers with available storage can store events on behalf of other peers;

2. and, congestion load-balancing, where the overlay dynamically adjusts the number of peers, their locations and the edges between them to mitigate local overloads.

4.2 Storage pooling

Consider an event that according to Rule 3.7.1 in Chapter 3 should be stored at a peer that has run out of storage capacity. That event would have to be blocked or require the deletion of some other event. However, if viewed in the aggregate, the platform has a total storage capacity equal to the sum of the storage capacities of the overlay peers. Still, our rule for storage prevents us from exploiting the total capacity available since each event is to be stored at one peer only. To demonstrate the effect of this decision on performance, we will briefly return to our simple grid model for a back-of-the-envelope calculation:

Fact 4.3. *Consider a square grid overlay network with a unique peer at each of its nodes, such as the one considered in Section 3.9. Events arrive uniformly at each peer with rate γ_e events per unit time, space, i.e., each peer receives*

$\gamma_e\delta^2$ events per unit time since each peer has a distance δ from its neighboring peers in the grid. Suppose each peer has storage capacity for m events and a typical event's duration is an arbitrary independent random variable with mean β. For such a system the storage blocking probability for a typical event, i.e., at a typical peer, is given by $p_{block} \triangleq E(\gamma_e\delta^2\beta, m)$, where $E(,)$ corresponds to the standard Erlang loss function.

For a definition of the standard Erlang loss function, see [44]. Clearly, as the number of peers increases and the dimensions of the grid remain constant, the distance between two neighboring peers decreases, i.e., $\delta \to 0$. As a result, according to the previous formula, the blocking probability vanishes since the Erlang function is monotonic in its first argument but this happens at the expense of an increased mean query delay, as discussed in Section 3.9. Using a similar approach, the corresponding storage blocking probability if peers could share the total storage capacity of the platform is $E(n^2\gamma_e\delta^2\beta, n^2m)$, where n^2 is the total number of peers in the grid. For reasonably high n, the previous expression will be much smaller than the one derived above. For example, for the parameters $\gamma_e = 0.05$, $n^2 = 10000$, $\beta = 600$, i.e., 10 minutes, $\delta = 0.5$ and $m = 10$ we have an improvement from 0.1 to roughly 0.0.

To enable pooling of storage resources we modify our storage policy to allow peers that have sufficient storage space to store events on behalf of other peers. In the interest of self-containment of the current chapter we re-state Rule 3.10.1, the modified rule for storage, and the concepts of *holders* and *owners* for an event.

Rule 4.3.1. Fault-Tolerant Storage. *Each event e is stored at the $k+1$ peers $\{h_i(e), i = 0, \ldots, k\} \subset P$ that are closest to e.location and are available to store it. We will denote this set of peers by $holders(e)$. The* owners *of an event e, $owners(e) \triangleq \{p_0(e), \ldots, p_{n-1}(e)\}$, are all the peers that are closer to e.location than $h_k(e)$, including $h_k(e)$.*

Furthermore, to disentangle issues of load-balancing and fault-tolerance, we will initially focus on the case where only 1 replica per event is stored in our platform, i.e., $k = 0$.

Observation 4.4. Storage Capacity. *Using Rule 4.3.1 for storing events, an event can always be stored in our platform as long as there is available space in at least one peer, i.e., we achieve the optimal storage performance.*

Indeed, an event might be stored arbitrarily far away from its location. In the sequel, we tackle this issue.

4.4.1 How far from its closest peer should an event be stored?

In our previous observation we allowed for an event, e, to be stored an unbounded distance from its closest peer if all the owners that are closer to it have run out of space. In this section we will argue that this is an unlikely event. Let us once again consider our grid model for a back-of-the-envelope calculation. Suppose that an event is blocked at each peer independently with probability p_{block}. In the sequel, we will address the case of correlated blockings.

In the grid, each peer has 4 neighbors that are 1 hop away, 8 peers that are 2 hops away, etc. The total number of peers at a distance up to j hops away from a peer, including itself, is $j^2 + (j+1)^2$.

Fact 4.5. Storage Distance. *Consider an infinite square grid with a unique peer at each one of its nodes. Suppose an event, e, can be stored at a given peer with probability $1 - p_{block}$ independently of the other peers. Then, the probability e is stored at a peer that is $j \geq 1$ hops away from $p_0(e)$ is*

$$(1 - p_{block}^{4j}) \times p_{block}^{(j-1)^2+j^2}. \tag{4.1}$$

The mean distance, in hops, between the holder of event e and $p_0(e)$ (mean storage distance) is

$$\sum_{j=1}^{\infty} j * (1 - p_{block}^{4j}) p_{block}^{(j-1)^2+j^2}. \tag{4.2}$$

The mean storage distance as a function of p_{block} is plotted in Fig. 4.1. Fig. 4.1 suggests that for reasonable values of p_{block}, an event e is unlikely to be stored far away from its closest peer, $p_0(e)$. In practice, events are not blocked independently by peers. A event that is blocked by a peer is likely to be blocked by neighboring peers as well since we try to store to neighboring peers. Let X_1, X_2 be indicator functions for the events that 'e is blocked at peer p_1 and its neighbor p_2'. Since there is positive correlation between them, we expect $\mathbb{E}[X_1 X_2] \geq \mathbb{E}[X_1]\mathbb{E}[X_2]$. So our calculation above yielded a lower bound.

Storage Pooling Scalability. The previous result implicitly suggests that the performance of our platform is unlikely to suffer when we perform

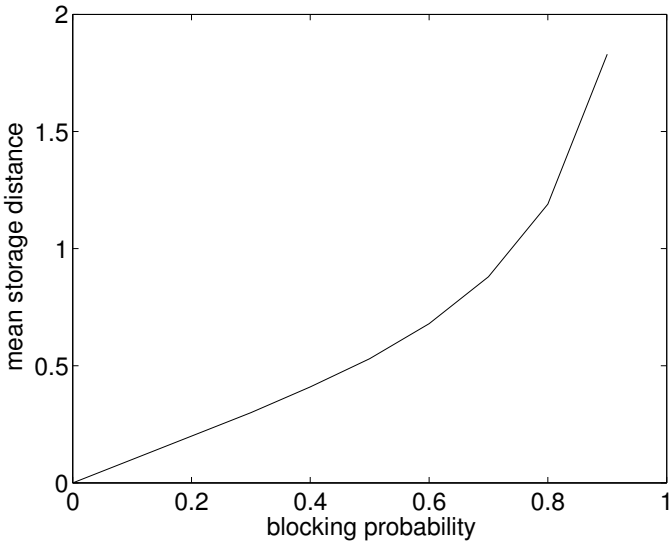

Figure 4.1: Mean storage distance for an infinite square grid (in hops).

storage pooling. Rule 4.3.1 requires events to be moved as a result of changes in the overlay topology. The penalty for the increased performance in storage will come in the form of an increased overhead traffic to adjust an event's storage when overlay topology changes occur. For example, assume that event e ends up being stored at owner $p_j(e), j > 0$. Any peer q joining inside the disk $B(e.location, ||e.location - p_j(e)||)$ will cause e to be moved from peer $p_j(e)$ to peer q. The smaller the distance $||e.location - p_j(e)||$, the less likely the above event.

4.5.1 On the interplay between storage pooling and fault-tolerance

Using $k + 1 > 1$ replicas/event for fault-tolerance is guaranteed to increase the storage load on all peers. Peers with big cells and/or in areas

where the intensity of events is high are likely to see high loads. Keeping more replicas to ensure a copy of an event is always available prevents other events from being stored due to lack of space and vice versa. This reveals a trade-off between storage pooling and fault-tolerance: high degrees of fault-tolerance through replication will lead to higher storage loads for the peers, and less storage loads for the peers have to come at the expense of reduced fault-tolerance through replication.

We attempt to capture the above trade-off through the following quantity: the likelihood that 'a typical event e has at least one of its k replicas remain alive for its entire lifetime'. A replica stops being alive if the corresponding peer holding it fails. Again, we assume that an event is deleted as soon as its lifetime expires. Alternative policies, e.g., a new event overwrites an old event, will be considered in our future work.

Fact 4.6. Storage vs. Fault-Tolerance Tradeoff. *Consider n peers arranged in a square grid where each peer has a distance δ from its neighbors. Suppose that events arrive according to a homogeneous spatio-temporal Poisson Process with intensity γ_e events per unit time,space, and events have an average duration of β units of time. Assume that for each event, e, we maintain $k+1$ replicas at the closest peers to its location available to store it. Suppose, that each peer can store up to m events otherwise the peer is unavailable to store events. Suppose a peer hosting a replica of e fails before e expires with probability p_{fail} independently of the other peers and is replaced immediately by another peer. Then, the probability at least one replica remains alive in our platform*

throughout the entire lifetime of the event is approximately

$$\sum_{i=1}^{k+1}(1-\hat{p}_{block}(i))\hat{p}_{block}(1)^{1(i\neq k+1)}(1-p_{fail}^{i}) \tag{4.3}$$

where $\hat{p}_{block}(i) = E(n^2\gamma_e\delta^2\beta, n^2\frac{m}{i})$*, is a lower bound on the probability a 'batch' of* i *replicas gets blocked from storage and* $E(,)$ *corresponds to the Erlang standard loss function.*

The previous formula takes into account the individual events that only $i = 1, \ldots, k+1$ out of the $k+1$ replicas can be stored in the grid by the peers. A replica is available at the end of its lifetime if it is stored at a peer and the corresponding peer does not fail during its lifetime. The 'batch' arrival process is Poisson, since event arrivals are assumed to be Poisson, but we require every replica to be stored in a different peer. For example, if peer p has space for two events, still according to Rule 4.3.1 only one replica will be stored in it. Thus, our expression for $\hat{p}_{block}$ yields a lower bound on the true blocking probability of a batch.

Based on Eq. 4.3 we cannot conclude that adding more replicas per event ensures that the probability of at least one replica surviving its lifetime increases. An optimal value for the number of replicas/event might exist, depending on the lifetime of a typical event and the storage capacity of each peer. In Fig. 4.2 we demonstrate this effect by plotting Eq. 4.3 for the following parameter values: $\gamma_e = 0.05$, $n^2 = 10000$, $\delta = 0.5$ and $p_{fail} = 0.1$. Curve 1 corresponds to $\beta = 600$, $m = 10$, Curve 2 corresponds to $\beta = 180$, $m = 10$, and Curve 3 corresponds to $\beta = 600$, $m = 100$. Because of the relatively

big lifetime of each event in Curve 1, $\beta = 600 = 10$ minutes, and the low storage capacity of each peer, adding more replicas per event *decreases* the probability an event survives its entire lifetime. This happens because most events will be blocked from storage in the first place. Decreasing the lifetime of a typical event to $\beta = 180 = 3$ minutes, as in Curve 2, reveals that an optimal number of replicas, $k+1 = 4$, exists. Indeed, the decreased lifetime of a typical event reduces the number of events blocked. The increased number of replicas stored per event increases the fault-tolerance of the platform. Once we exceed the optimal number of replicas per event, there are 'too' many replicas in the system, so adding more replicas decreases the fault-tolerance of the platform. Finally, if we increase the available storage space per peer, as in the case corresponding to Curve 3, the optimal number of replicas/event for the platform, not shown in Fig. 4.2, increases too. This gives the platform an excellent fault-tolerance, in comparison to Curve 1, for the same number of replicas.

It is part of our future work to research further the conditions under which an optimal number of replicas for our platform exists.

4.7 Congestion load-balancing

In this section, we explore ways to adapt the overlay topology, i.e., the number of peers, their locations and the edges connecting them, so as to balance the traffic load on the peers. As discussed in the introduction, different peers may receive widely different amounts of traffic due to non-uniformities

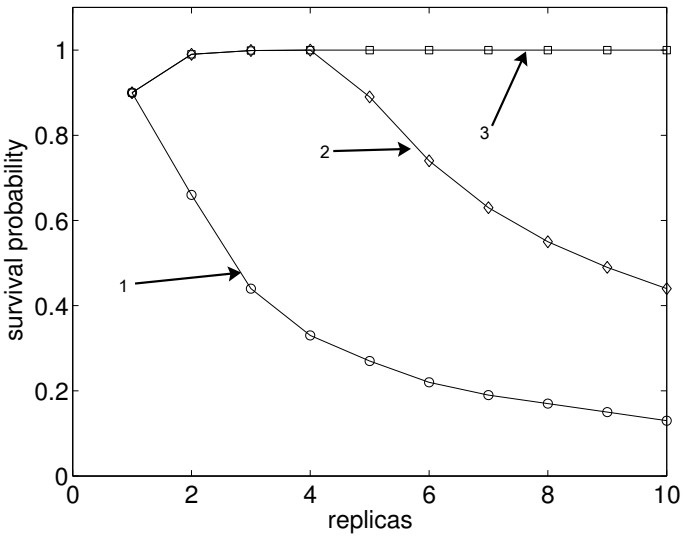

Figure 4.2: Replica survival probability for a $n \times n$ square grid.

in the peers' and/or traffic's spatial distribution.

'Local' vs. 'End-to-End' Congestion. Congestion at the peers can be conceptually divided into 'local' and 'end-to-end' congestion. We will define the former as caused by exceedingly high event traffic generated in a cell or query traffic whose source and destination lie in the same cell, while the latter is caused by routing of queries across the overlay network. Local congestion depends on the size of a peer's cell; the bigger the cell the more local traffic it might be expected to receive. End-to-end congestion depends mainly on the location of a peer, the more 'central' a peer is, the more routes cross its cell.

End-to-end congestion also depends on the quality of the edges in the topology long edges connecting peers that share a lot of traffic are better.

In this section we consider the following approaches to mitigating congestion: modifying the locations of the peers, adapting the connections among peers to the underlying traffic, as well as adapting the number of peers in the overlay so as to approximate the optimal peer intensity suggested in the work discussed in Section 3.4.

The first approach addresses mainly the local congestion, the second approach addresses the end-to-end congestion, and the third addresses both.

4.7.1 Modifying the location of peers

The locations of the peers uniquely determine the size of the cell of each peer. Consider the set P containing all the peer locations $\{p_1, \ldots, p_n\}$ such that for each $p_i \in P, p_i \in R$, for some region R. We denote by $Q_i(P)$ the queue size of the i^{th} peer under the peer placement suggested by P. Ideally one would like to find an algorithm that provides a solution to the following optimization problem.

$$\min_P \{\mathbb{E}[\sum_{i=1}^{|P|} Q_i(P)] | p_i \in R \text{ for all } p_i \in P\} \tag{4.4}$$

under the constraint of greedy routing and a fixed average intensity of events γ_e and queries γ_q. By Little's law, this is a proxy for the average traffic delay in the system.

For local traffic *only*, any arrangement that divides the topology in

shapes of equal area, e.g., square grid, grid of equilateral triangles, etc would be a solution for Eq. 4.4. The addition of 'end-to-end' traffic complicates things.

In Section 3.9 we studied the performance of our platform using a uniform square grid as the overlay topology. Eq. 3.1 asserts that the bigger the area of a peer's cell, the more traffic it receives. If peers could advertise a different location than their current one, provided it is still close to the original (so as not to lose the benefits of locality), a more 'balanced' overlay topology could be achieved.

This observation motivates the following simple heuristic:

Rule 4.7.1. Move Heuristic. *An overloaded peer, p, may invite a non-overloaded peer, q, to drop its current location and join the overlay with a different 'virtual' location that is closer to p and thus take some of its load.*

For purposes of implementing this heuristic, a peer will be considered overloaded with respect to another peer if its queue size exceeds a fixed multiple, $f_{qs} > 1$, of the other peer's queue size. Below we discuss how a peer estimates its queue size. To account for the effect of changing topology, each peer maintains an exponentially weighted estimate of its queue size. In other words, if the current estimate at time t for a peer's queue size is $Q(t)$ and a new sample Q' of the queue size is obtained at time $t' > t$, the estimate of the queue size is updated as follows

$$Q(t') = (1 - e^{-\frac{t'-t}{\tau}})Q(t) + e^{-\frac{t'-t}{\tau}}Q'. \tag{4.5}$$

The time constant, τ, of the exponential weighting function is chosen such that the decay for an event that arrives an 'average inter-arrival' interval after the previous event is $\frac{1}{e}$.

For simplicity, we will require the peers q and p involved in such move heuristics to be neighbors in the DG. The peer q will be restricted to move to a new location that lies along the line connecting its original location to the location of p. The resulting distance between the two peers will be a fraction, $f_d < 1$, of the original distance between them, see Fig. 4.3.

The above mentioned heuristic has the effect of reducing the area of the overloaded peer's cell, which in turn will lower the local traffic the peer sees. Moreover, a peer with a big cell is likely to lie in the path of an increased number of queries and suffer increased end-to-end traffic as well. Thus, reducing a cell's size is likely to have a positive impact on end-to-end traffic as well.

Evaluation. To evaluate the effectiveness of the 'move' heuristic on the mean delay to process an event, we performed the following experiment: we varied the average intensity of events arriving at a peer per m^2-sec γ_e, and measured the mean delay to process an event with and without the 'move' heuristic, see Fig. 4.4. We assumed that the event arrival process is Poisson with mean rate γ_e, ranging from 0.2 to 2 events per m^2-sec. New event arrivals are assumed to be homogeneous in space, and arriving events enter the queue of the closest peer. To decouple the study of the local congestion from the end-to-end congestion we set $\gamma_q = 0$ for this experiment. The time to process an individual event is assumed to be an independent exponential random variable

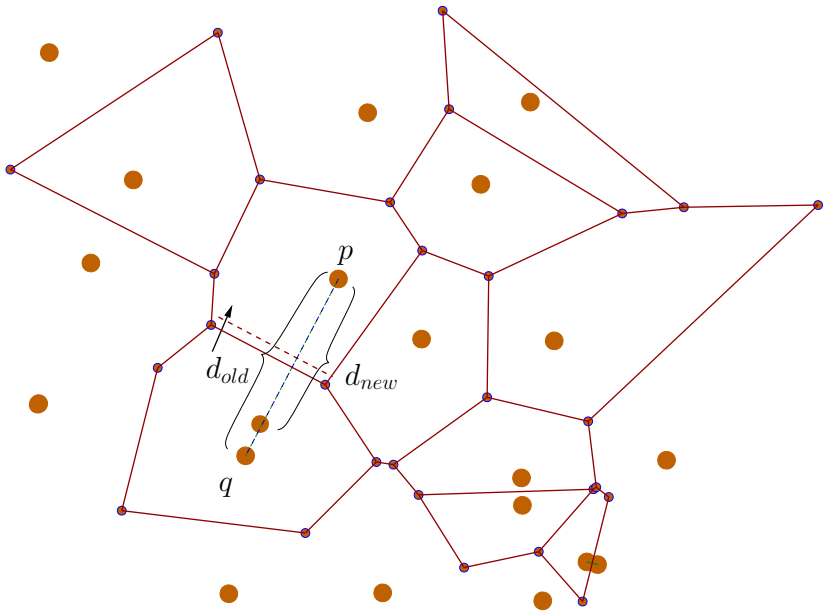

Figure 4.3: Move Heuristic, d_{old} is the distance between p and q before the heuristic, d_{new} is the distance after and $\frac{d_{new}}{d_{old}} = f_d$.

with mean $\mu = 1$. The overlay topology is generated by placing 60 peers independently in a 5×5 region according to a homogeneous Poisson process with rate $\lambda = \frac{60}{5^2} = 2.4$ peers per m^2. Two peers are connected if and only if they are neighbors in the Delaunay graph. For this experiment, since $\gamma_q = 0$, the Kleinberg edges would not play any role. For our preliminary evaluation of the 'move' heuristic, we used the following parameters $f_{qs} = 1.3$ and $f_d = 0.95$.

Peers were selected to perform the heuristic at random, i.e, uniformly, at each tick of an exponential clock with rate $\mu_{move} = 0.05$. The rate of the clock has been selected such that on average each peer will have at least 100 event arrivals before being selected to implement the heuristic. This allows for our heuristic to operate on meaningful queue size statistics, i.e., the queue size estimates obtained via Eq. 4.5.

The simulation lasted for 120000 units of time, ensuring that on average 100 'cycles' where performed, each cycle corresponding to a period in which every peer performs the heuristic at least once.

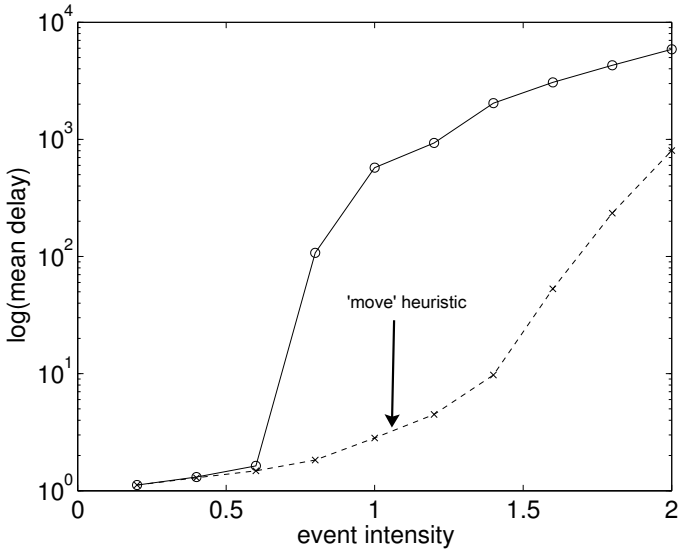

Figure 4.4: Figure shows in logarithmic scale the mean delay to process an event as the spatial intensity of events γ_e grows in the overlay network. Two cases are shown, the first where peers are randomly located in space, and the second after the move heuristic is carried out.

The 'move' heuristic has an obvious beneficial effect on the mean delay to process an event for the entire range of event traffic. Although the maximum mean event arrival intensity at a cell is $\overline{\gamma} = \frac{\gamma_e}{\lambda} = \frac{2}{2.4} = 0.84 < \mu = 1$, due to statistical variations of the cell sizes, some peers will receive more traffic than they can handle and will become unstable. In Fig. 4.5 we show that the 'move' heuristic manages to create a homogeneous topology where most cells are of roughly equal size, thus minimizing the probability a peer overflows. This is consistent with our observation about Eq. 4.4.

Beneficial effect on 'end-to-end' congestion. Previously, we claimed that although the 'move' heuristic aims to address 'local' congestion, it has a beneficial impact on 'end-to-end' congestion as well. To evaluate the effect of the 'move' heuristic on the mean delay to process a query, we performed the following experiment: we varied the intensity of query traffic generated per m^2-sec, γ_q, and measured the mean delay for a query starting from its source to reach its destination, with and without the 'move' heuristic. The query traffic is assumed to arrive according to an independent homogeneous Poisson process with intensity ranging from $\gamma_q = 0.04$ to $\gamma_q = 0.32$; we kept $\gamma_e = 0$. Each query is immediately placed by its source in the queue of the first hop. For this experiment the clock rate μ_{move} has been set equal to 0.04. The duration of the experiment has been set equal to 150000 sec. The rest of the parameters of the experiment have been chosen following the same rationale as in our previous experiment. Throughout the simulation, each peer has a Kleinberg edge to a fixed point, the 'target' of that edge is the peer

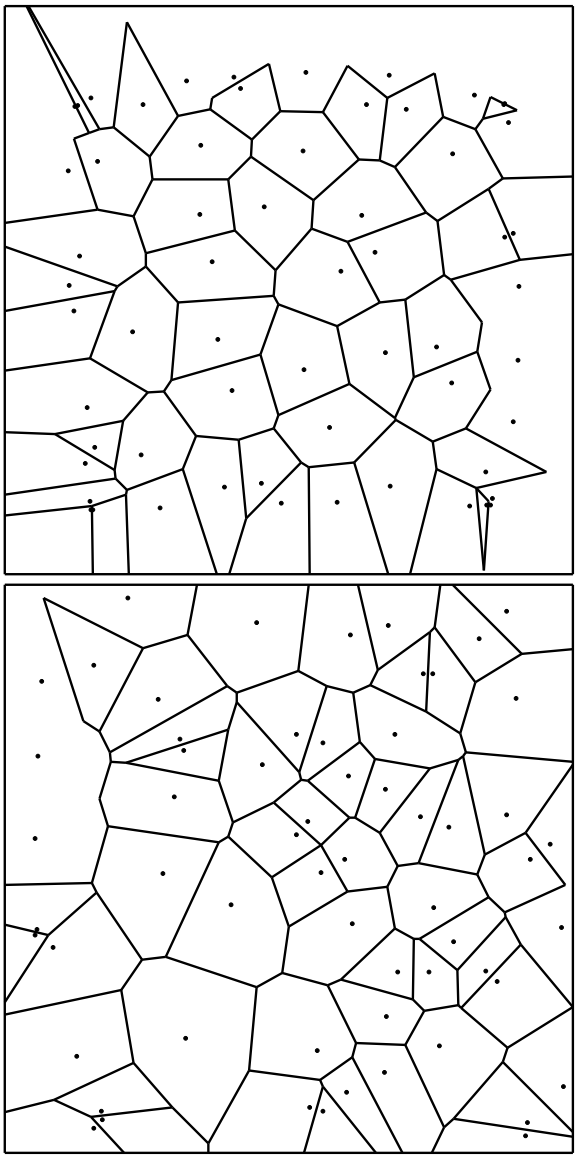

Figure 4.5: Resulting overlay by applying the 'move' heuristic for $\gamma_e = 2$ (up) and original overlay topology (down) (the facets extending to infinity have been omitted).

that happens to be closest to that point at any time. The initial placement of the peers, depicted in Fig. 4.5, is the same as the one used in our previous experiment. The results are shown in Fig. 4.6. The 'move' heuristic appears to be beneficial for high query intensities; we further discuss that point in the sequel.

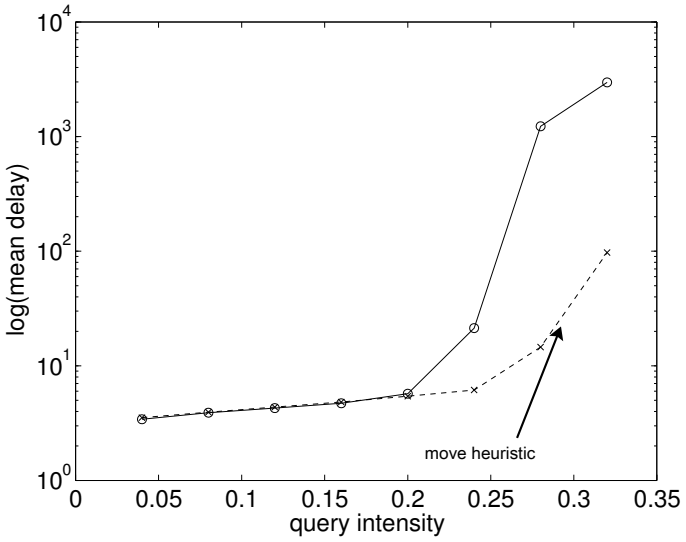

Figure 4.6: Figure shows in logarithmic scale the mean delay to process a query as the spatial intensity of queries γ_q grows in the overlay network. Two cases are shown, the first where peers are randomly located in space, and the second after the move heuristic is carried out.

A rough, yet enlightening, view of the effect of our heuristic is that it divides the overlay topology in two areas: the inner area, which consists of uniformly arranged peers with small cells that suffer primarily from end-to-end cross congestion, and the outer area, which consists of uniformly arranged

peers with big cells that suffer primarily from local congestion, i.e., with high probability queries originate and end at their cell. Fig. 4.7 exhibits the resulting topology achieved by our heuristic when $\gamma_q = 0.32$. The inner area is defined by the dashed disk. The resulting arrangement ensures that *all* peers receive on average the same amount of total traffic, local plus end-to-end.

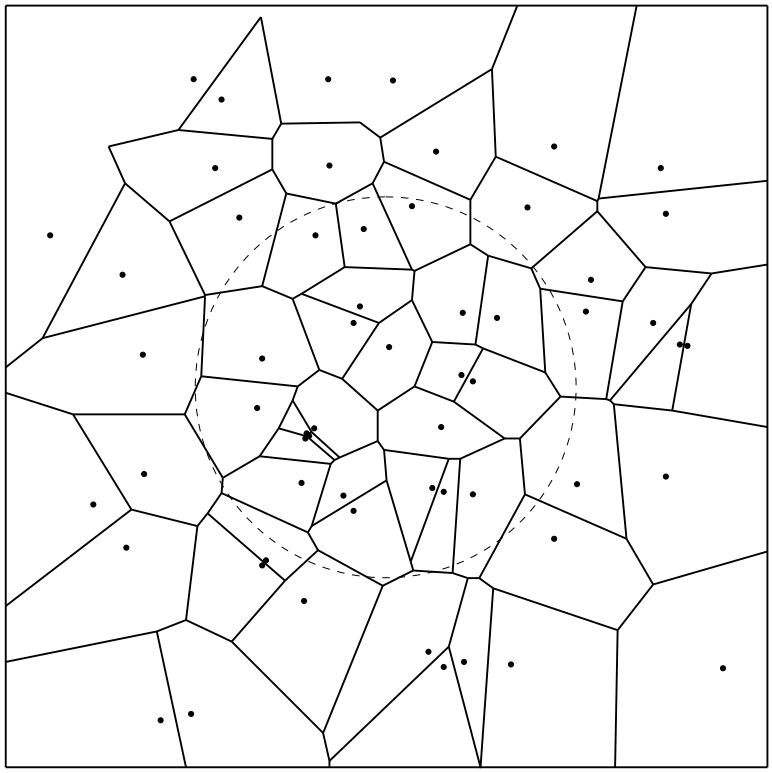

Figure 4.7: Resulting overlay by applying the 'move' heuristic for $\gamma_q = 0.32$. The facets extending to infinity have been omitted.

To further illustrate our point, in Figs. 4.8 and 4.9 we plot the mean traffic intensity per peer vs. the query intensity on the network. We provide two sets of two curves: one set without our heuristic and another set with our

heuristic. In each set we provide two curves: one for the peers ending up in the inner area mentioned above and another for the peers ending up in the outer area (remember that peers' locations change according to our heuristic). The quantity averaged across peers is the exponentially weighted estimate of the traffic intensity observed by each peer. Looking at Figs. 4.8 and 4.9, the balancing effect of our heuristic becomes apparent.

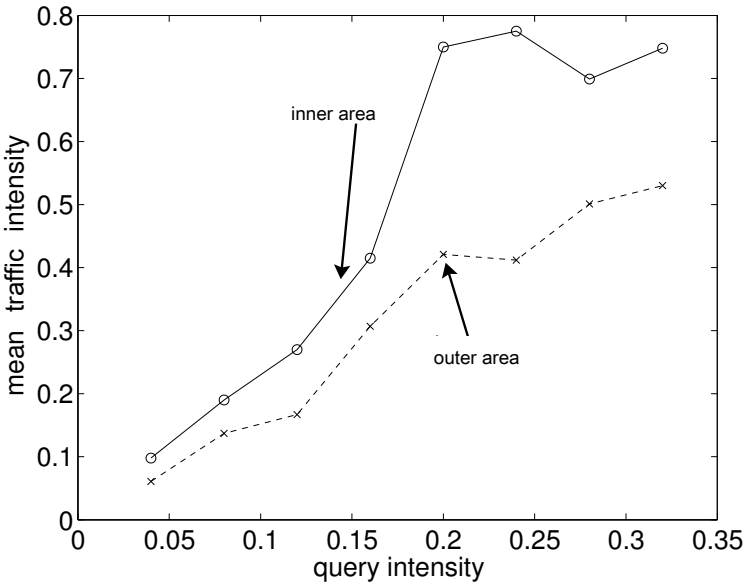

Figure 4.8: Mean traffic intensity per peer vs. the query intensity on the network without using the 'move' heuristic. The two curves correspond to the 'inner' and 'outer' areas shown in Fig. 4.7.

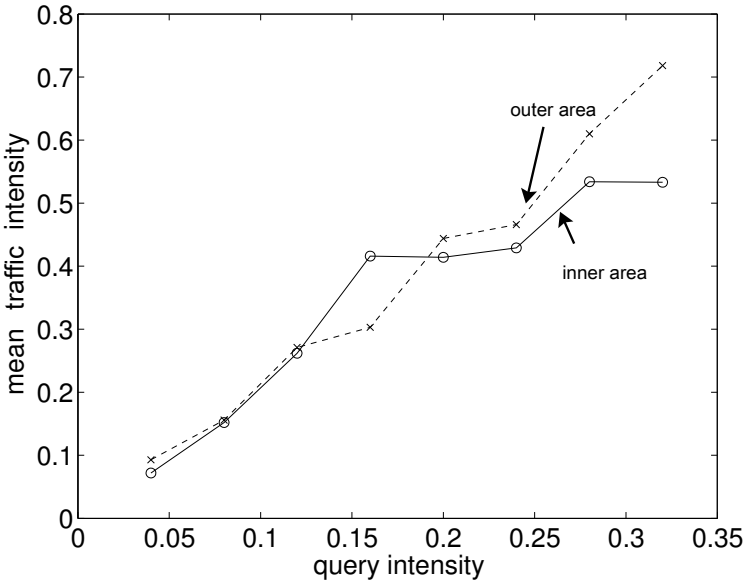

Figure 4.9: Mean traffic intensity per peer vs. the query intensity on the network using the 'move' heuristic. The two curves correspond to the 'inner' and 'outer' areas shown in Fig. 4.7.

Going back to Fig. 4.6, we deduce that the performance of our heuristic for low query intensities is slightly worse than when omitting it but as the traffic increases, the heuristic pays off. The initial performance gap is due to the following reasons:

1. **Our heuristic increases the average hop count for queries.** Our heuristic balances the traffic among peers, thus peers tend to move from

the periphery of the topology towards the center, see Fig. 4.7. Due to the increased peer intensity at the center of the overlay, routes driven by greedy routing, crossing the center have to go through more hops. In Fig. 4.10 we plot the time average number of hops/query averaged over all peers. Fig. 4.10 supports our claims. As expected the average number of hops/query is independent of the query intensity.

The average end-to-end query delay equals the average hop delay times the average number of hops/query. For low query intensities, the average hop delay is small, so the number of hops/query is the deciding factor for the end-to-end performance. As the query intensity increases, the hop delay dominates the end-to-end delay, and the effect of the number of hops is less important. Note that the average number of hops/query in Fig. 4.10 is attained with the help of Kleinberg edges. In the absence of Kleinberg edges, the difference would be even more dramatic.

Combining the results of Fig. 4.10, Fig. 4.8, Fig. 4.9, and Fig. 4.6, we observe that due to the improved traffic balancing achieved by the move heuristic, the effect of the increased number of hops/query is mitigated to some extent and does not affect the average end-to-end delay significantly.

2. **The movement of peers affects the delay of the queries they hold in their queues.** Every time a peer moves to a different location according to our heuristic, the progress of a subset of the queries it has in

its queue is 'impeded' if the movement is in a direction opposite to their current destination. This results in an increased delay for these queries. For low amounts of query traffic this delay is observable. As the query intensity increases, the effect of congestion at the peers dominates this phenomenon, and the benefit from the resulting balanced arrangement of peers pays off for our heuristic.

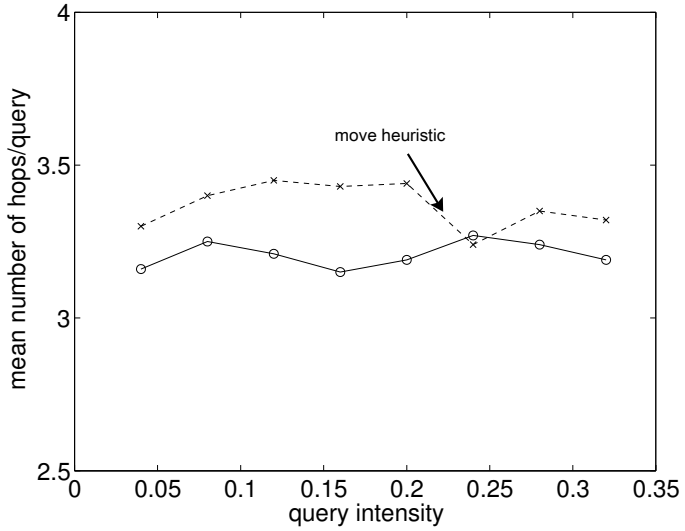

Figure 4.10: Figure shows the mean number of hops/query as the query intensity grows in the overlay network. Two cases are shown, the first where peers are randomly located in space, and the second after the move heuristic is carried out.

Mixed traffic: Events and Queries. To evaluate the effect of the 'move' heuristic on the mean delay to process an event/query for traffic consisting of a mixture of events and queries, we performed the following experiment: we varied the intensity of query traffic generated at a peer per m^2-sec, γ_q, and measured the mean delay to process traffic consisting of events/queries with and without the 'move' heuristic, see Fig. 4.11. Individual events and queries contributed equally to the average delay reported. In Fig. 4.11, we present the curve for the mean delay for $\gamma_e = 0.05$. The query traffic is assumed to arrive according to an independent homogeneous Poisson process with intensity ranging from $\gamma_q = 0.04$ to $\gamma_q = 0.28$. The source and destination of each query are chosen independently by point sampling the underlying space homogeneously and 'truncating' to the closest peer. The dimensions of the underlying space and the initial placement of the peers are the same as in our previous experiment. We used an independent exponential clock to select the peers to perform the 'move' heuristic. The intensity of the clock, μ_{move}, has been set equal to 0.05, following the same rationale as in our previous experiment. The duration of the simulation has been set equal to 120000 time units following the same rationale as before. The parameters for the 'move' heuristic are the same as in our previous experiment as well. Throughout the simulation, each peer, in addition to the edges to its neighbors in the Delaunay graph, has a Kleinberg edge to a fixed point, the 'recipient' of that edge is the peer that happens to be closest to that point at any time. The resulting overlay topologies are depicted in Fig. 4.12. The results show the same qualitative

characteristics as for the previous experiment, thus validating our conclusions.

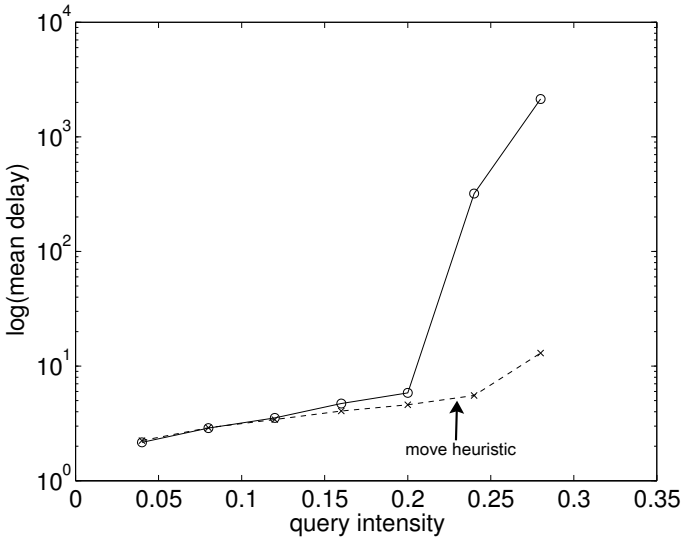

Figure 4.11: Mean delay to process an event/query, $\gamma_e = 0.05$.

How does a peer estimate the queue size of its neighbors? As discussed previously, our heuristic is triggered depending on the relative queue sizes of the current peer and its neighbors peers. A peer can piggyback the current estimate of its queue size on the queries originating from its cell. This way, neighboring peers can estimate its relative queue size to theirs and choose whether to invite it to move closer to their cell.

4.7.2 Adapting the edges among peers to non-uniform traffic

In Section 3.4 we proposed augmenting our overlay topology with Kleinberg edges. The effect of these edges is to create short, on average, routes be-

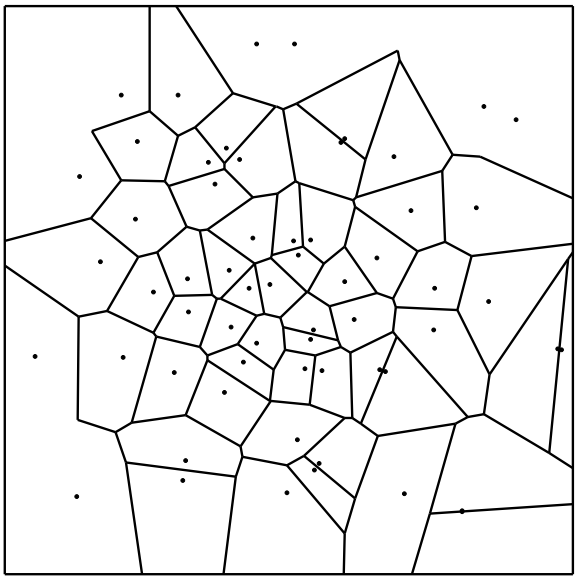

Figure 4.12: Resulting overlay by applying the 'move' heuristic for $\gamma_q = 0.28$ and $\gamma_e = 0.05$ (the facets extending to infinity have been omitted).

tween any two peers in the overlay. A limitation of Kleinberg edges is that they are oblivious to the congestion in the network, and they are not sufficiently flexible to handle non-uniformities in the traffic. For example, consider the case of a group of people attending a conference at a convention center. Some of their queries could be destined to locations corresponding to the restaurant where the conference banquet will take place. Obviously, in this case traffic is not homogeneous in space. The peers that happen to be in the path between them and the destination peer answering their queries will be disproportionally stressed by traffic.

To model the non-uniformity in traffic, we introduce the concept of a traffic 'dipole', see Fig. 4.13. Each dipole creates traffic that originates

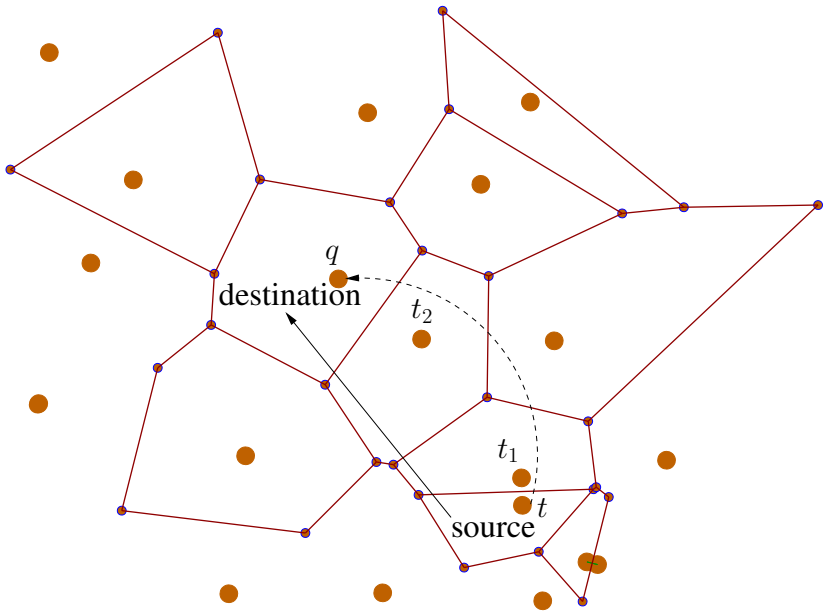

Figure 4.13: A 'dipole' model for non-uniform traffic. Non-uniform traffic from location *source*, which falls into C_t is destined to location *destination*, which falls into C_q. We denote this traffic by a solid arrow. The bigger the length of the arrow, the bigger the *separation* between the source and the destination of the dipole and more peers are affected by the traffic it generates. The dashed arrow, represents a non-uniform edge between peers t and q. The effect of this edge is to remove traffic from peers t_1, t_2 which would have to support this traffic in the edge's absence.

from one location, i.e., its *source*, to another location, i.e., its *destination*. The distance between the source and the destination of a dipole is termed its *separation*.

To realize a traffic-aware P2P topology, one can further augment the

topology by a traffic-dependent edge per peer. A simple heuristic would be:

Rule 4.7.2. Congestion Edge Heuristic. *Each peer, q, estimates the average traffic intensity, $\gamma_{t,q}$ and the average traffic delay, $d_{t,q}$, for the queries originating from every peer $t \in P$ in the network destined to q. The peer with the maximum $\gamma_{t,q} \times d_{t,q}$ is selected to create an edge to q. We call the associated edge, the* congestion *edge of peer t.*

The quantity $\gamma_{t,q} \times d_{t,q}$, by Little's law, is a proxy for the average number of queries 'in-flight' from peer t to peer q. By creating an edge between peers t and q, we offload the corresponding traffic from the overlay network and restrict it to the source/destination. In our implementation, each peer, t, maintains the cost, $\gamma_{t,q} \times d_{t,q}$ associated with the peer q to which it directs its congestion edge. In case another peer, r, invites t to establish its congestion edge toward it, t accepts only if $\gamma_{t,r} \times d_{t,r}$ exceeds $\gamma_{t,q} \times d_{t,q}$. A peer can estimate the load and the delay from another peer through, e.g., exponentially weighted estimates as in Eq. 4.5. We estimate the associated decay parameter as the average inter-arrival time of queries from the other peer.

How often do peers implement the congestion edge heuristic? A peer can be triggered to switch its congestion edge at random times generated by, e.g., an exponential clock with rate $\mu_{\text{congestion edge}}$. We set the value of $\mu_{\text{congestion edge}}$ such that, on average, a peer receives at least 100 queries before selecting the peer to invite to direct its congestion edge.

Evaluation. We studied the performance of the congestion edge heuris-

tic via the following experiment. We used the peer placement depicted in Fig. 4.5, the same as the one used in the experiments for the move heuristic. Additionally, we augmented the topology with edges derived from the Delaunay Graph as well as with Kleinberg edges. We simulated a combination of homogeneous and non-homogeneous traffic. The homogeneous traffic consisted of events arriving with intensity $\gamma_e = 0.05$ events/m^2-sec and queries arriving with intensity $\gamma_q = 0.04$ queries/m^2-sec. This is the same traffic intensity as in the experiment for the move heuristic. For the non-homogeneous traffic, we created a fixed number, 10, of traffic 'dipoles'. The source and destination of each dipole were placed at random on the topology. For simplicity, in our implementation all dipoles had a traffic intensity equal to $\gamma_{\text{non-uniform}}$. For the non-uniform traffic we simulated two different intensities, a 'low' one with $\gamma_{\text{non-uniform}} = 0.1$ queries/sec and a 'high' one with $\gamma_{\text{non-uniform}} = 0.25$ queries/sec. The service time for each event/query was an independent exponential random variable with rate $\mu = 1$. The rate of the exponential clock triggering the heuristic at the peers was set to $\mu_{\text{congestion edge}} = 0.2$ and the simulation lasted for $T_s = 50000$ time units. The simulation duration was chosen so as to ensure that on average at least 100 cycles of the heuristic are performed, where a cycle is defined as the period where all the peers perform the heuristic.

In Figs. 4.14 and 4.15 we vary the dipoles' separation and plot the average delay for traffic. The traffic consists of events, uniform queries, and non-uniform queries. The intuition is that as the separation of the dipoles

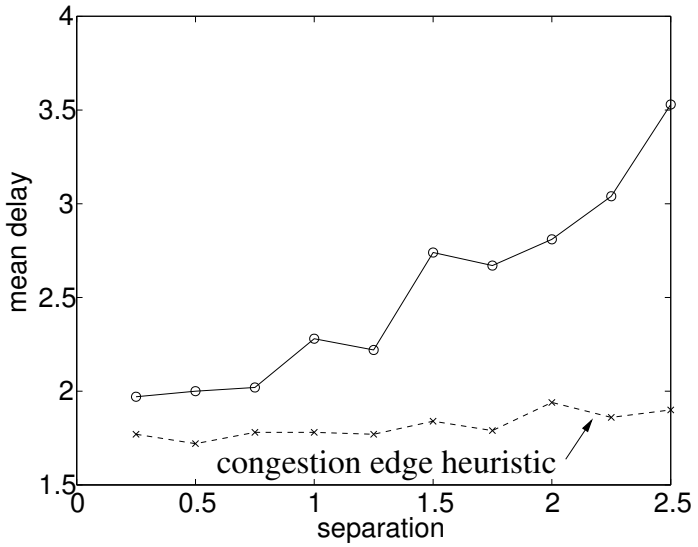

Figure 4.14: Average traffic delay vs. dipole separation for $\gamma_{\text{non-uniform}} = 0.1$. The traffic consists of events, uniform queries and non-uniform queries.

increases more peers will suffer from the resulting transit traffic. In Fig. 4.14 the overall traffic is relatively low, so the contribution of the non-uniform traffic is noticeable but small. In Fig. 4.15 we increase the intensity of the non-uniform traffic, driving the increases in overall average delay.

The relative performance of the two schemes exhibited in both figures matches our intuition. Indeed, in both figures the mean delay achieved by our heuristic remains essentially the same regardless of the dipole separation, as opposed to the delay of the base case without our heuristic. The apparent non-monotonicity of the delay for the case without our heuristic in Fig. 4.15 can be attributed to the high variance of the measurements, we explain this in the sequel.

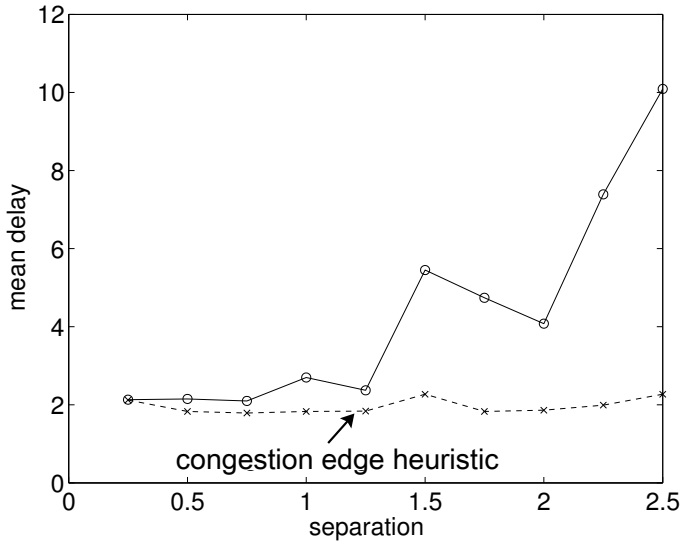

Figure 4.15: Average traffic delay vs. dipole separation for $\gamma_{\text{non-uniform}} = 0.25$. The traffic consists of events, uniform queries and non-uniform queries.

The traffic delay in the network exhibits high variability as some peers are particularly stressed by the non-uniform traffic. This effect becomes apparent from Figs. 4.16 and 4.17. We ensured that the simulation duration is not the cause of this phenomenon by running the simulation for exceedingly long intervals. Still, our heuristic manages to reduce the variability of the delay in the network. Our heuristic, by creating congestion edges between the sources and the destinations of non-uniform traffic, 'flattens' the traffic, regardless of the separation of the dipoles.

In Figs. 4.18 and 4.19 we show the 'effectiveness' of our heuristic, i.e., the percentage of the edges that are established from a source of a dipole to its destination. We observe that the effectiveness of our heuristic is always greater

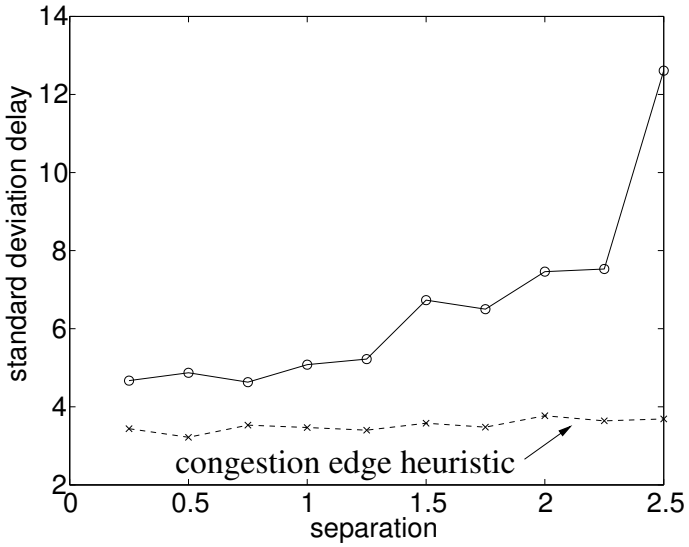

Figure 4.16: Standard deviation of traffic delay vs. dipole separation for $\gamma_{\text{non-uniform}} = 0.1$. The traffic consists of events, uniform queries and non-uniform queries.

than 80% and in many cases close to 100%, regardless of the separation. One would expect that as the non-uniform traffic increases in intensity, our heuristic would make each source establish its congestion edge to the destination of the dipole. Fig. 4.19 confirms that as the non-uniform traffic increases the effectiveness of the heuristic increases as well. By studying the cases where the heuristic failed to connect the source with the destination of a dipole, we were able to identify the underlying reason is that the source and the destination of a dipole, being generated randomly, were in the same cell so the corresponding peer could not establish its congestion edge to itself.

One could argue that the comparison we presented in our previous

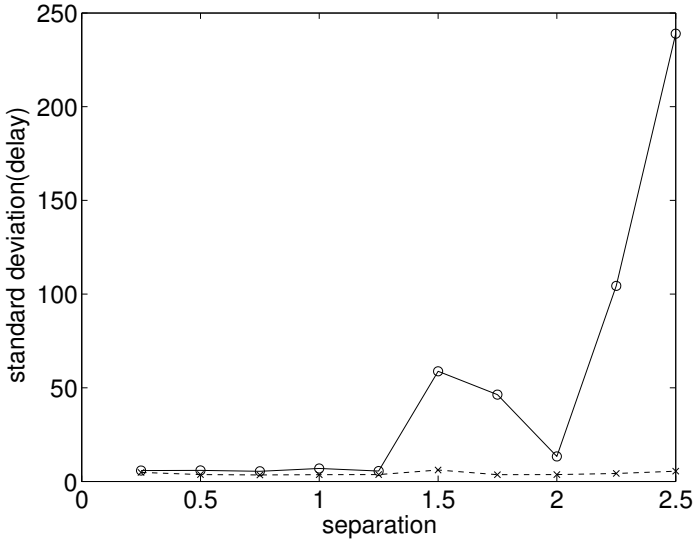

Figure 4.17: Standard deviation of traffic delay vs. dipole separation for $\gamma_{\text{non-uniform}} = 0.25$. The traffic consists of events, uniform queries and non-uniform queries.

experiments is not fair since peers under our heuristic have one more edge to route queries, the congestion edge. Note that even if additional Kleinberg edges were added to the scheme without our heuristic, the performance would still be poor. It is the edges that are congestion-aware that provide the real additional benefit when traffic is non-homogeneous.

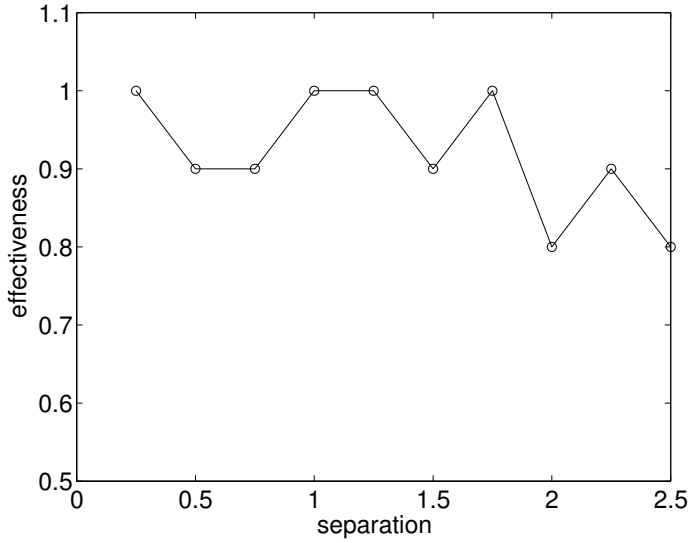

Figure 4.18: Heuristic effectiveness vs. separation for low non-uniform intensity, $\gamma_{\text{non-uniform}} = 0.1$.

4.7.3 Adapting the number of overlay peers

In Section 3.9 the analysis of our platform for the idealized square grid overlay topology revealed that the average query performance in our system is a trade-off between having too few or too many peers in the overlay, see Fig. 3.6. In the former case, performance suffers due to traffic congestion of the peers, in the latter due to excessively long routing. The addition of Kleinberg edges can help but, in the limit, the average delay of queries increases without bound. This motivates adopting policies that operate the overlay near the optimal intensity of peers.

Rule 4.7.3. Leave Heuristic. *A non-overloaded peer may leave the overlay if*

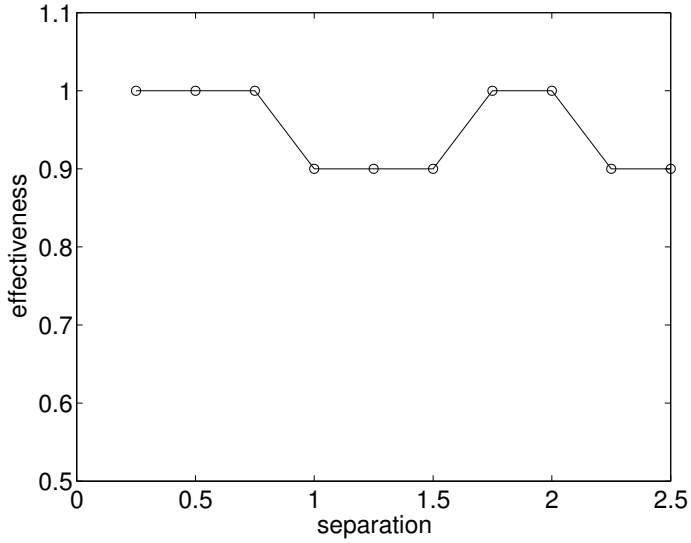

Figure 4.19: Heuristic effectiveness vs. separation for high non-uniform intensity, $\gamma_{\text{non-uniform}} = 0.25$.

it is the least loaded among its neighbors and they are not overloaded as well.

A peer is considered overloaded if the total traffic intensity it receives, a combination of the event traffic arriving in its cell and the query traffic that it has to route or process, is close to its service capacity. Empirically, values of the total traffic intensity received, that are close to, e.g., 90% of the peer's capacity, render the peer overloaded.

The intuition behind the previous heuristic is that when all peers in a neighborhood are not overloaded, it is likely that there are too many peers in the overlay. In this case, removing peers from the overlay makes the platform operate closer to the optimal intensity of peers. As the leaving peer is

not overloaded by definition, the overhead due to the messages exchanged for updating the storage of the events it owns, is not expected to be significant. Again, information about the load on each peer can be piggybacked on the queries that originate from that peer.

To address the 'reverse' situation, i.e., when there are too few peers to keep up with the traffic, we propose the following heuristic:

Rule 4.7.4. Join Heuristic. *An overloaded peer whose neighbors are all overloaded, and which is the most loaded among them may ask an entity that is not part of the overlay to join.*

The intuition is to add more peers to the overlay so as to operate the platform closer to its optimal intensity of peers. As stated in Section 3.4, in our platform there are entities that do not participate in the overlay and access its services through a proxy. Some of these entities could very well play the role of the peer. For example, a peer p receiving queries from an entity en through proxy q can invite en to join the overlay. Under Assumption 3.3, en is likely to be close to p so by joining the overlay it will 'relieve' the peers in p's neighborhood by taking up some of their load. The incentive for en is that its own queries will see a speedup as they are routed through those peers.

We do not explore the effect of the heuristics experimentally as they are straight-forward consequences of our theoretical model.

4.8 Conclusion

In this chapter we address the storage and performance issues resulting from non-uniformities in the topology and the traffic. To address storage limitations, we modify our storage scheme to re-distribute events to peers who have not reached their storage capacity. To address non-uniformities in the topology, we propose and experimentally evaluate a heuristic that dynamically adapts the locations of the peers to balance the load. To address non-uniformities in traffic, we propose and experimentally evaluate a heuristic that creates 'shortcut' edges in the topology where they are needed most.

Chapter 5

An RFID-based Platform Supporting Context-Aware Computing in Complex Spaces

5.1 Introduction

The vision of ubiquitous computing was originally conceived by Mark Weiser at Xerox Parc, see [83]. The realization of this vision requires overcoming significant obstacles, many of which have at their root the use of abstractions more appropriate for the PC world than the new field of ubiquitous computing. To name just a few, the weight, energy consumption, monetary cost and interface of future mobile computers will significantly influence the adoption of the ubiquitous computing paradigm. Additionally, issues like poor wireless connectivity, effective sharing of limited bandwidth and privacy threats call for more 'localized' user-application interactions as opposed to 'centralized' interactions where service is provided from a few master points only.

In this thesis, we advocate for a ubiquitous computing platform that exploits locality in context in order to tackle the above issues. We assume that the spaces where entities reside are *complex*, i.e., spaces where one space is 'nested' inside another, e.g., in Fig. 5.1, we see that the bookstand (defined

to be at depth 2) lies inside the airport (defined to be at depth 1). We chose to focus on these types of spaces because they can easily model the vast majority of spaces humans spend most of their time, e.g., homes, business buildings, shopping malls, etc.

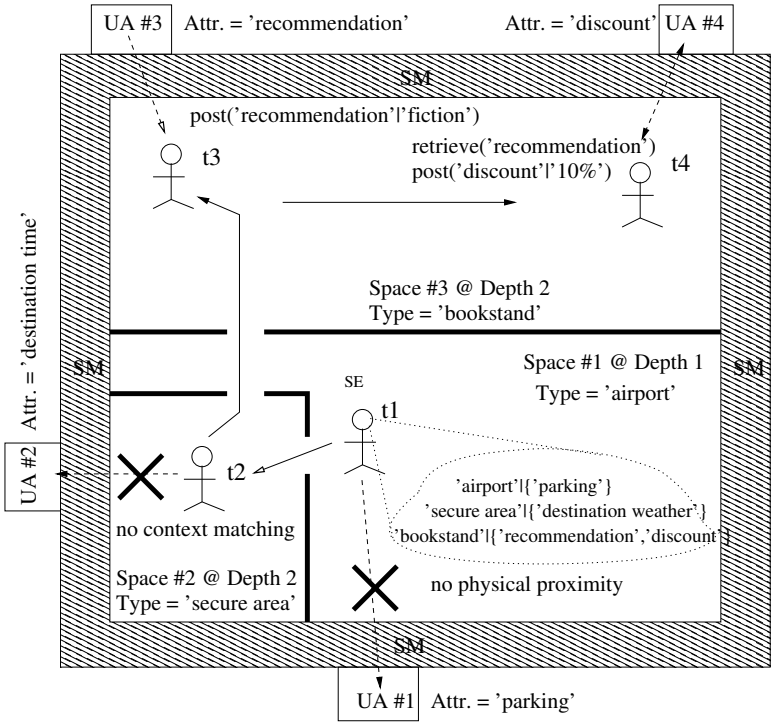

Figure 5.1: Main elements of a complex space realizing our platform

Our work is motivated by passive RFID technology being a candidate fabric for our platform. Passive RFID technology offers distinctive advantages in terms of cost, weight and energy consumption, giving our platform some of the desired properties alluded to before almost 'by construction'. Extensive use of RFID technology as a means to store context and data is assumed. Specifically, we propose a platform in which mobile entities, e.g., humans,

robots, pets carry with them a minimalistic artifact to hold information describing the current context and application data only. All the rest, i.e., energy, computational power and logic are provided by applications embedded in the environment.

The rest of this chapter is organized as follows. In Section 5.2 we review related work. In Section 5.3, we discuss the key abstractions underlying the proposed platform. Section 5.4 presents the actual protocol used by our abstractions. In Section 5.5, we discuss a low cost implementation platform on top of RFID technology – this 'low end' implementation demonstrates the immediate benefits of our approach's inherent simplicity. Finally, Section 5.6 concludes the chapter.

5.2 Related work

There has been a substantial body of work focusing on providing users with transparent and efficient access to their data, irrespective of where it is stored, see [72]. While such efforts address an important need, their focus is still akin to a PC-like, centralized view of the world, where a person's relevant data/information is, for the most part, explicitly owned by that person and accessed via some personal all encompassing device (PC, PDA, etc.), i.e., they still adhere to a more 'user-centric' view of the world, quite distinct from that advocated in this thesis.

Since then, highly successful prototypes of key building blocks and enabling technologies for pervasive computing include [25, 28, 35, 38, 40]. More

recently, the interest in ubiquitous computing has culminated especially in the context of 'smart' spaces, e.g., see the work in [9, 16].

A major approach to the problem of making a space smart relies on a 'meta-operating system' that administers the entire space and coordinates devices and applications running on top of traditional operating systems. For example, in [66, 67], GAIA, a distributed software middleware is demonstrated that manages resources and provides location information and event services within a managed space. GAIA supports the notion of context through first order predicates, yet it does not directly address space dependent characterization and discrimination of arbitrary mobile entities, as well as entity-targeted communication, which are key contributions of our work. The work in [31] focuses on information to enable service migration and discovery.

Location-aware computing is an orthogonal approach to context-aware computing. In [74] the authors try to address the issues currently hampering the global adoption of location-aware ubiquitous computing, in the sense of using GPS coordinates. A platform is proposed, 'Place Lab', which attempts to exploit the already large installed base of Wi-Fi networks. Our approach shares the idea that location forms a natural hierarchy but is oblivious to the exact location of a mobile entity within such spaces.

The use of context is paramount in our approach. An excellent general discussion about context, its role, and the advantages of using a tuple-like formalism is given in [87]. One of the first efforts to exploit context and build applications that react to an individual's changing context is described in [73].

It relies on a system akin to a rule-based expert system. Upon detection of the presence of a user, an action is triggered depending on the identity of the user and the rules pertaining to the event. The definition of context used is more narrow than ours, and context information is stored centrally while our approach encapsulates context in the SE abstraction that goes with the mobile entity. In [61], context information is used to select the best possible device and method of communication. A contextual model of first order predicates is used in one of their implementation platforms. The tuple formalism we use shares similarities with Linda, see [18], yet the latter addresses issues relative to parallel computing.

Our approach assumes the existence of a suitable ontology to model spaces and attributes. In this work we have assumed the simplest version of such an ontology. For most comprehensive ontologies, see [34, 45].

Little work has been published in the area of methodologies for designing ubiquitous devices. The closest match to this type of work is [69, 75]. In [75] a high-level design methodology for building context-aware devices is presented. A set of rules for detecting whether an application can benefit from the knowledge of context, e.g., a temperature sensor, and ways to capture and represent context are provided. In [69] a software framework for interacting with mobile devices supporting different communication technologies is presented. In [29, 47] the challenges for creating embedded hardware prototypes for ubiquitous computing research are presented. In [63] a case study for the use of constructing a virtual prototype of an RFID reader embedded in a cell

phone is presented.

Security and privacy are a concern in our platform as well. Our mechanism for assigning IDs, and the profile concept (to be defined in the sequel), which is essentially an anonymizing mechanism, contribute towards this goal. In [65] an overview of RFID security challenges and solutions is presented. In [71, 86], a thorough examination of problems relevant to RFID security is made. A list of security protocols are evaluated, with an emphasis on their suitability for the low-cost limited hardware capability of RFID tags. In [14] the authors deal explicitly with the problem of location privacy. They advocate in favor of a mechanism that will allow users to frequently change ID's (pseudonyms in their terminology) and introduce the concept of a mixed zone. A mixed zone for a group of users is defined as a maximum size connected space in which the users are not visible to any application. Users change aliases each time they enter the mixed zone, which enhances the system's privacy. The work in [50] discusses a vast array of privacy issues in the field of ubiquitous computing. In [49], an in depth discussion on principles/policies towards privacy-awareness in ubiquitous computing is provided.

5.3 Key abstractions supporting spatially contextualized ubiquitous computing

In this section, see also Fig. 5.1, we will present the main abstractions identified for supporting context-aware ubiquitous computing, namely:

- *Space Manager* (SM): an OS-like layer that administers the actual com-

puting fabric embedded in the physical environment.

- *Ubiquitous Applications* (UAs): the set of ubiquitous computing applications executing on the particular fabric; and
- *Serviced Entities* (SEs): interface to the actual mobile entities being currently served by UAs, in the particular physical space.

The space manager, depicted as a layer that *logically* surrounds all spaces in Fig. 5.1, provides an OS-like (distributed) layer, i.e., offers a set of basic resource management and operational services/primitives to be used by both local UAs and SEs. Inside the physical space administered by an SM, there can be an arbitrary number of UAs and SEs. We sometimes refer to an actual physical space provided with an SM as a '*managed space*'. Throughout the chapter, we illustrate our approach using a running example of an airport structured as a managed space. To make our scenario more realistic, we assume that the SE models a human passenger and all the SE abstractions are stored in an RFID chip in the passenger's ticket.

The logic underlying the operation of a managed space is as follows. Local ubiquitous applications, running on the myriad of (smart) objects populating the particular physical space, provide the actual computing services. For example, in Fig. 5.1, UA #1 represents a ubiquitous application offering parking services, e.g., directions to parking space and prices, inside the airport. The 'serviced entities' represent mobile entities, e.g., people, robots, pets, etc., that temporarily enter the managed space and potentially benefit

from its services. In Fig. 5.1, the little human figure that moves around is modeled as a SE. Each SE carries with it information describing its current context as well as application data. Only when one such SE comes to physical proximity with a ubiquitous application, will it transparently react depending on whether there is a context match (as specified by the SE and the application). The reaction can be the provision of the object's service or total lack of it. *No* involvement from the SE is required. For example, at times $t1$ and $t2$ the SE in Fig. 5.1, does not receive any service since it is not close to UA #1 and does not share any interest in UA #2's services (UA #2 lets SEs find out about the time at their destination, while the SE only cares about the weather at its destination inside this space).

An application can write its data to the SE, data that will be available for other applications to inspect and update when the SE comes to physical proximity to them. This mechanism realizes a form of application communication. For example, at time $t3$ UA #3 that runs inside the bookstand makes a recommendation about a fiction book to the SE by posting an appropriate message *on* it. At $t4$ this information is exploited by UA #4 to give a 10% discount to the SE. (This would not have happened unless the SE had specified that inside bookstands it cares about the attributes 'recommendation' and 'discount').

In this sequel, we expand more on some elements of the key abstractions: the *space type* and the *SE*.

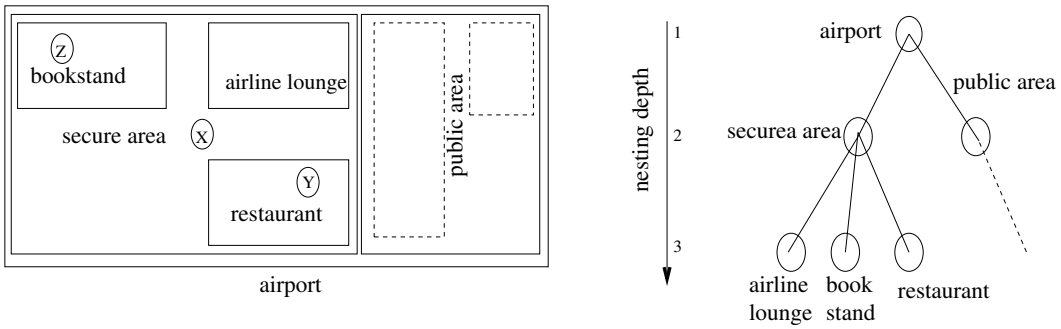

Figure 5.2: Airport example

5.3.1 The space type and its attributes

As alluded to in the introduction, a key objective of our approach is to enhance entities immersed in a given physical space with information to be used by ubiquitous computing applications to seamlessly establish the potential relevance of their services to such entities. Towards that, our approach consists of abstracting the very services meaningful inside a particular space in terms of attributes. A *space type* can thus be defined as a collection of attributes.

Fig. 5.2a, presents a simplified view of an airport. This figure shows several space types, e.g., a generic airport space and other more specialized ones, e.g., a secure area. Attributes abstracting the relevant services in a, e.g., restaurant include vegetarian food, kosher restrictions, low fat diet, etc., while attributes relevant to a, e.g., secure area used for departures include gate information (denoted from now on as 'gate'). While above we give suggestive names to space types and attributes, truly those are just identifiers. Thus, the combination, e.g., $< secure\ area, gate >$ will unambiguously abstract gate

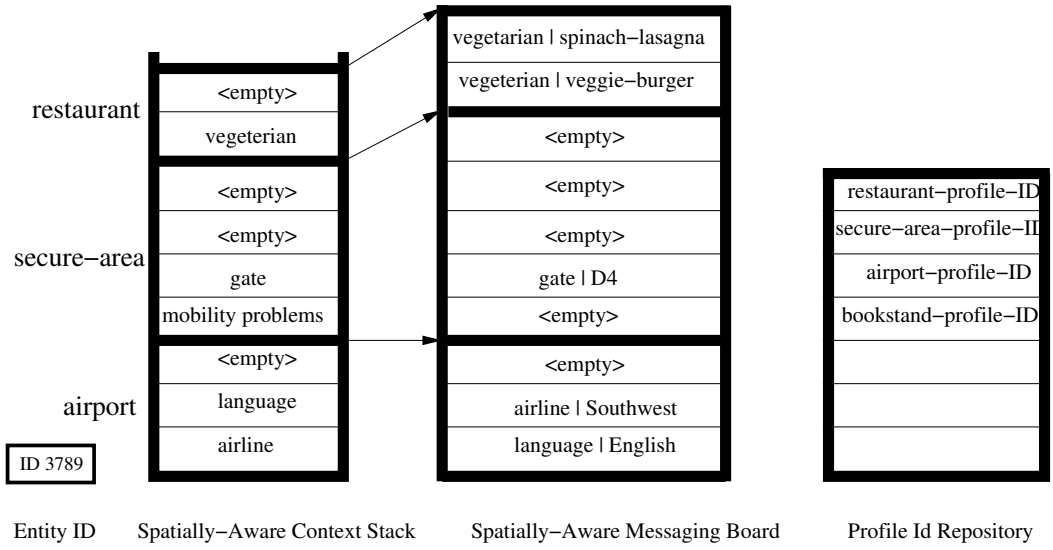

Figure 5.3: Key SE's abstractions

related services inside the secure area of the airport. Naturally, the definition of such service taxonomies will ultimately require a flexible standardization effort, enabling new space types and attributes to be added to the standard, as their need becomes apparent.

5.3.2 The serviced entity: SE

A serviced entity or SE is defined by the following set of abstractions which are physically stored in it:

Temporary ID. Requiring entities to provide some form of 'permanent identification' upon entering a managed space could potentially infringe on one's privacy and is actually unnecessary – rather, in our approach each SE entering a managed space is simply assigned a temporary ID – see Section 5.4

for details on the protocol used for the purpose. (Note also that privacy considerations may further dictate that such temporary IDs are to be periodically dropped and reassigned.)

Spatially-Aware Context Stack. The spatial organization assumed in this paper supports complex spaces, nested inside each other. We use a tree to encode this structure, see Fig. 5.2b. The actual location of any entity can thus be defined as a path from the root of the tree to the node corresponding to the 'innermost' space where the entity is currently located. For example, entities X and Y in Fig. 5.2a, are currently located in 'airport→secure area', and 'airport→secure area→restaurant', respectively.

Services in our platform are related to the spaces in which they are offered. A stack is used for representing that since it naturally captures the containment relationships alluded above. The *Spatially-Aware Context Stack* is responsible for holding the attributes characterizing the classes of services the SE is interested in, in the context of its current physical location. Stack frame (or entry) i+1 represents a physical space enclosed within the physical space associated to stack frame i. Thus, for example, an entity positioned at Y in Fig. 5.2a will have its stack filled as symbolically depicted in Fig. 5.3, i.e., with frames 1, 2 and 3 of the stack representing the airport, the secure area, and the restaurant, respectively. Fig. 5.3 shows a simple context stack for an airline passenger – attributes characterizing services for this passenger in the broad context of the airport may include his/her airline, language and so forth.

Naturally while at a given position the entity is characterized not only by its attributes for the innermost space (stored in the frame currently at the the top of the stack), but also by its attributes associated to all of the enclosing spaces – that is, it continues to be sensitive and respond to applications and messages addressing it in the context established by all such enclosing spaces (established by the remaining stack frames). As the entity moves around across different spaces, frames that are no longer relevant are popped from the stack and new relevant ones are pushed into it – Section 5.4 gives details on the adopted stack update protocol.

Location models have been used in the literature, e.g., [10, 12] to provide applications with a way to describe information about the current location of an entity. In our approach, the underlying location model for a space is a tree, but location information is *encoded* in the spatially-aware context stack of each SE. Moreover, the stack responds dynamically to the movement of the SE in contrast to statically mirroring the structure of a space.

The Profile Repository of an Entity. A *profile* is a subset of a space's attributes, defining a compatible/coherent set of services within that space. In practice, we use standardized sets of attributes for each space. Moreover we expect that, for many space types, it will be possible to define a number of typical profiles that can be then directly adopted by entities when they enter a space of that particular type for the very first time. Each such profile will be encoded using an ID, the *profile ID*. Entities should choose the profile that best describes the services they are interested in for a given space. The fact

that the profile used will typically be a proper superset of the actual services of interest to the user adds to the privacy of the platform, since a potential attacker will be provided with information, only a subset of which is true. The profile is a novel anonymizing mechanism that nicely fits in a ubiquitous environment. Each SE holds a repository of its profile IDs for relevant space types – see Fig. 5.3.

The Spatially-Aware Messaging Board. Communication and cooperation between ubiquitous applications in our platform are scoped by the SEs it is intended to service. SEs contain messaging boards for that purpose. For each space held in the context stack of an SE, a corresponding messaging board is defined. As depicted in Fig. 5.3, messages posted by ubiquitous applications in these boards are indexed by attributes. In the simple example shown in Fig. 5.3, there is a message posted on the 'secure area' messaging board of the particular entity referring to the passenger's gate. (In the figure, for clarity, the attribute used to index each message is separated from the body of the message by a vertical bar.) Only messages guarded by attributes contained in the SE's profile can be posted to a messaging board. Thus, attributes in our platform have a *filtering* functionality. This highly focused form of communication adds to the scalability of our design in contrast to other centralized solutions found in the literature using blackboards, e.g., [38].

Supporting Persistent State. In addition to the abstractions presented previously, a persistent version of the profile repository and the messaging board exists, independent of spatial context. For example, a message

describing the allergies the patient suffers from should be stored in a persistent way independent of the space the passenger is in. We do not discuss this feature any further.

5.4 Companion protocol

In this section, we describe the protocol used with SE devices implementing our abstractions. Table 5.1 summarizes the messages defined in our protocol – the first two columns describe the function of the message and provide an abbreviation name for it, the third column specifies the message's sender and receiver, and the fourth column describes the information exchanged through the message. In what follows, we discuss how the protocol supports the different types of actions listed above, using such messages, and we also present a FSM representing the intended behavior of an SE during such actions – see Fig. 5.4.

ID Assignment. The SM periodically broadcasts the ⟨man_space⟩ message to inform entities entering the particular physical space that this is a managed space. Different spaces are distinguished by the space identifier used. SEs reply with a message requesting a unique ID – ⟨req_id⟩ – using some bootstrapping identifier for the purpose. Upon receiving it, the SM sends a unique ID to the requesting entity using the ⟨assign_id⟩ message. This action and corresponding interactions are captured in states "initial", "waiting_id" and "named" of the SE's FSM shown in Fig. 5.4.

	Message Description	Message Name	Direction	Information Exchanged
SM Messages	announce managed space	man_space	SM → SE	⟨space identifier⟩
	announce space identification	space_id	SM → SE	⟨space depth⟩⟨space type⟩
	request managed space_id	req_id	SE → SM	⟨bootstrapping ID⟩
	assign managed space_id	assign_id	SM → SE	⟨new entity ID⟩
	reset SE	reset	SM → SE	⟨entity ID⟩
	request spatially contextualized profile	req_profile	SE → SM	⟨space depth⟩⟨space type⟩⟨entity ID⟩⟨profile ID⟩
	send spatially contextualized profile	send_profile	SM → SE	⟨space depth⟩⟨space type⟩⟨entity ID⟩⟨profile ID⟩⟨profile⟩
UA Messages	qualified ping	qping	UA → SE	⟨space depth⟩⟨attribute type⟩⟨attribute⟩
	qualified ping response true	qping_rsp_t	SE → UA	⟨space depth⟩⟨attribute type⟩⟨attribute⟩⟨entity ID⟩
	qualified ping response false	qping_rsp_f	SE → UA	⟨space depth⟩⟨attribute type⟩⟨attribute⟩⟨entity ID⟩
	post message	post	UA → SE	⟨space depth⟩⟨attribute type⟩⟨attribute⟩⟨entity ID⟩ ⟨data⟩
	retrieve messages	retrieve	UA → SE	⟨space depth⟩⟨attribute type⟩⟨attribute⟩⟨entity ID⟩
	retrieve messages response	retrieve_rsp	SE → UA	⟨message number⟩⟨attribute type⟩⟨attribute ⟩⟨entity ID⟩ ⟨data⟩
	retrieve messages response end	retrieve_end	SE → UA	⟨message number⟩⟨attribute type⟩⟨attribute ⟩⟨entity ID⟩ ⟨data⟩

Table 5.1: Protocol basic messages

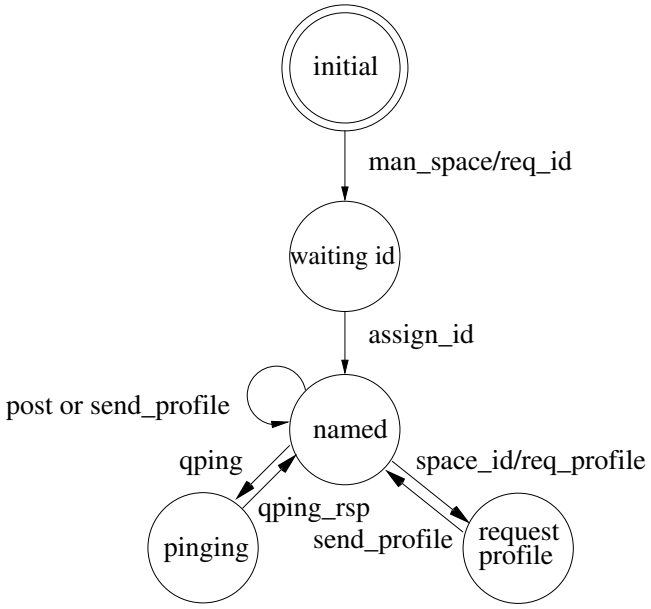

Figure 5.4: A simplified view of the SE's state machine (reset omitted for simplicity). The notation x/y denotes the message triggering a transition and its response.

Spatial Context Update and Profile Acquisition. Upon entering a new physical subspace, the entity eventually receives a ⟨space_id⟩ message from the SM, identifying the space type and level of nesting (within the overall managed space). As a response, the entity's SE is updated, i.e., the corresponding stack frame is initialized and the messaging board that will be holding messages for this space type is initialized as well. Once this is done, the entity requests its profile for the new space type, using the ⟨req_profile⟩ message. Upon receiving the latter, the SM sends the requested profile, using the message ⟨send_profile⟩. Subsequent ⟨space_id⟩ messages, announcing the same space, are ignored by the SE's FSM. This action and corresponding in-

teractions are captured in the SE's FSM states "named" and "request_profile", see Fig. 5.4. The persistent profile does not change with the space, in fact we expect that it rarely changes that's why its' modification is done in a docking platform.

Qualified Query. Applications (UAs), and the SM, can broadcast messages inquiring for entities matching a certain attribute, using the ⟨qping⟩ message. Entities matching the attribute reply with their unique ID, using the message ⟨qping_rsp⟩. This way, applications can subsequently send direct messages to an intended subset of the present entities, appropriately filtered. A wild-card value can be used for the "attribute" field. The SE's states corresponding to this action and corresponding messages in Fig. 5.4 are "named" and "pinging". The "qping" message is used for querying both the context-sensitive stack and the persistent profile repository. The semantics of the message are interpreted differently for the two cases. In the context-sensitive stack case the attribute type is interpreted as the space type of the SM querying, in the persistent profile repository case it is interpreted as the attribute type for which the SM is querying. In both cases a single "qping_rsp" message with a true or false bit set in the message code indicates whether the attribute queried is part of one of the SE's profiles.

Posting/Retrieving Qualified Messages. UAs can post and retrieve messages to/from the SE's messaging boards. Upon receiving a ⟨post⟩ message, the addressed entity (indicated by the entity ID), checks its profiles for a match with the attribute type — attribute pair contained in the message.

A wild-card entity ID can be used that matches all entities, to implement a broadcast delivery. Each messaging board performs the check independently. For the context-sensitive messaging board if the check is true, it stores the received data in the messaging board allocated for this attribute type, indexed by the provided attribute. For the persistent messaging board the rules for interpreting the attribute type are the same as the ones described in the "query" subsection. In case a messaging board, regardless of its type, is full it behaves like a cyclic buffer overwriting old contents.

Other UA's can access the data stored in the messaging boards by sending a ⟨retrieve⟩ message. In this case, the addressed entity will reply by sending all messages (stored in the specified attribute type's messaging board) that match the specified attribute. The message used is ⟨retrieve_rsp⟩ for the initial messages and ⟨retrieve_end⟩ for the last message. A similar version of retrieve messages exists for messages from the persistent board. A wild-card can be used for the "attribute" field. All messages matching the fields of the retrieve message are lazily deleted. An application that wants a particular message to remain in the messaging board has to explicitly repost it. The main rationale behind such a decision is to ensure that only the absolutely up-to-date messages will remain in the messaging board. In such a dynamic environment as the one encountered in a pervasive computing scenario message tend to "age" very quickly and loose their significance.

Finally, most likely it will be useful to give the SM the ability to reset an SE. Once reset, the entity will be able to receive a new ID, and its context

stack can be incrementally (re)built.

Limitations. Time in general and time ordering in particular, is an important design feature that is absent from our protocol. The messages in our protocol do not carry any timestamps so it is not possible to compare two messages with respect to who came first. Instead we rely completely on the context of each message and our replacement policies to replace older messages with newer ones. Up to now we have not encountered the need for comparing two messages based on their delivery or generation time, but in case such need arises we believe that a structure akin to a logical clock will be appropriate. In fact, due to the passive operation of some of the devices, having a synchronized clock in all devices is infeasible.

Additionally in our protocol there is no provision for any Quality-of-Service(QoS) metric. Message delivery is best-effort and is totally dependent on the underlying medium access layer. We believe that QoS is less of an issue in low-end ubiquitous computing than it is in e.g. wireless communications. In the latter case it is not infrequent for multiple terminals to compete for medium access. In the former case due to the relatively short range of communication, which is comparable to human dimensions, and the social conventions that govern our behavior it is rather improbable that two devices will be competing for service. For example, in an automatic bus ticket collector equipped with RFID technology, people tend to form a queue that implicitly serializes requests for service, thus eliminating competition for service. Nevertheless, a more rigorous study of the usage patterns of this new style of communication

is needed before reaching definite conclusions about the level of QoS needed.

5.5 Prototyping efforts

The SE embodies most of the innovative aspects of our approach, and, most importantly, its cost/complexity defines how popular can this type of technology eventually become. Towards assessing such potential, we:

- developed a 'low cost' implementation of an SE, potentially targeted at RFID technology.
- emulated an SE on top of existing RFID tags.

Details on our design are given below.

5.5.1 Hardware implementation

We implemented the state diagram in Fig. 5.4 in Verilog – our design assumes that the interface to the analog part of the device implementing the SE is comprised of two FIFO queues, one for input and another for output messages. We further assumed that each SE has a byte-addressable memory of 16KBytes allowing unaligned accesses. We note that this amount of memory is within the order of magnitude offered by current RFID technology, see e.g., [1]. In our implementation, the context stack, the messaging boards and the profile ID repository reside in different segments of the SE's memory. The data bus and the address bus connecting the protocol controller to the memory

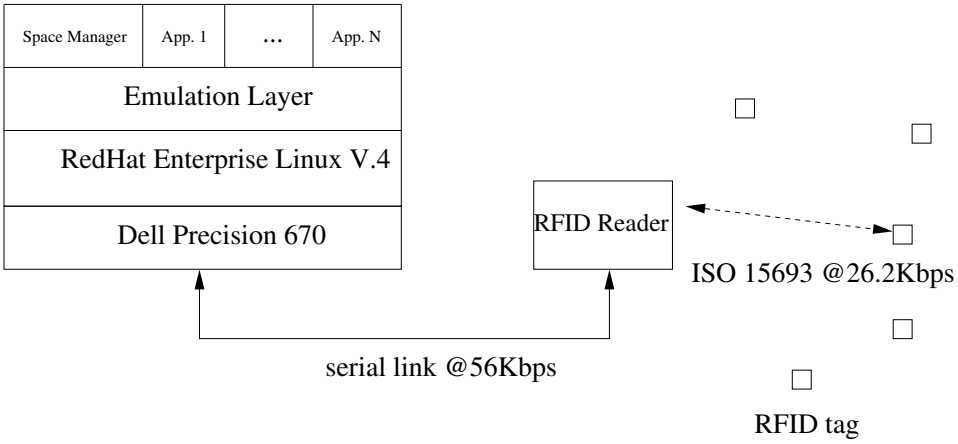

Figure 5.5: Software emulation stack

have widths of 32 and 16 bits, respectively. Our current implementation supports fixed length messages of 64 bits each. We synthesized our design using Synopsis's Design Vision V-2004.06-SP2 - the FSM implementing the protocol and the FIFO queues require 9800 gate equivalents, a number comparable to the gate count found in current passive RFID tags [86]. See Appendix C for more details.

5.5.2 Platform prototype

Our platform for developing pervasive applications relies on a software emulation layer, see Fig. 5.5. The purpose of the emulation is to

- export a suitable API for implementing the space manager and applications alluded to in Section 5.3.
- emulate the operation of a SE on top of real RFID tags.

Message Sequence	Time (sec)
post	0.67
qping, qping_rsp	0.23
retrieve, retrieve_end	0.36

Table 5.2: Message times

Each time the space manager or any of the applications wants to communicate with an SE, the communication goes through our emulation layer. The role of the emulation layer is to translate all messages of our protocol to standard RFID messages, which are then relayed to the RFID tags, and vice versa.

Our implementation uses passive RFID tags from Texas Instruments. The tags are compatible with the ISO15693 standard specification and were included in the TI S4100 evaluation kit along with the readers. In each one of the RFID tags, capable of holding 256 bytes of non-volatile memory, reside data structures implementing all the abstractions presented in Section 5.3.2. The choice of passive tags was made to assert that our protocol does not have any power constraints. See Appendix C for more details.

5.5.3 Timing considerations

Our platform's time and space performance is suitable for adoption by current entities. The following back-of-the-envelope calculations demonstrate the validity of the previous claim. Assuming an entity moving with a speed of 1.5 m/sec = 5 km/hr=3.1 miles/hr and an average data bit-rate of 10 Kbps (see [26] for actual "fast mode" and "long distance mode"

bit-rates), a typical message in our protocol (64 bits under the used encoding) would require 6.4 msec and would take approximately 1 cm worth of distance for the moving entity. Our emulation approach has a significantly higher overhead since our protocol's messages are encapsulated in a series of RFID messages. For example, a POST message in our implementation is implemented with 4 READ_MULTIPLE_BLOCKS messages and a WRITE_SINGLE_BLOCK message. The encoding used by TI results in an exchange of $((4 * 86) + 26) * 8 = 2960$ bits through a serial link operating at 56Kbps. This amounts to 0.052 sec, an order of magnitude increase (ignoring the time communicating through the RF medium, reading/writing the tag, emulation layer execution overhead and the cost of context-switches to the OS for writing to the serial port). The actual times per message are shown in Table 5.2.

The performance achieved, allows adoption of our technology even with today's limited support for our protocol/abstractions.

5.6 Conclusions

We proposed a low complexity platform for supporting spatially contextualized ubiquitous computing. The platform provides applications with the ability to assess the relevance of their services to mobile entities within a given space, and enables such applications to communicate via a post/retrieve interface, physically implemented on the memories of participating entities. The exchanged messages are filtered through a simple yet powerful mechanism

based on aliases, that takes into account each entity's profile in a given context space. The filtering mechanism ensures that each entity receives only the subset of messages that are truly destined and/or relevant to it – key towards enabling context awareness and iteration transparency. Finally, the platform offers privacy gains since global/permanent unique identifiers are never used revealing the true identity of any of the entities.

As part of our future work we intend to focus our attention on crafting more mature application managers, thus assessing the expressiveness of our protocol.

Moreover, our proposed abstractions offer a rudimentary level of privacy by using temporary ID's. Nevertheless, more sophisticated security solutions can (and should) be used as our abstractions are orthogonal rather than incompatible with such solutions. We intend to show that such approaches are feasible.

Chapter 6

Conclusions and Future Work

Ubiquitous computing applications are increasingly permeating our lives. The role of context in these applications is absolutely critical. In this chapter we briefly summarize key conclusions from our work on storing, querying and computing context. Additionally, we touch upon future directions for our research.

In Chapter 2 we examined three different classes of context scaling and identified necessary conditions for them to hold. The opportunities for optimization, depending on the way the volume of contextual information "scales", highlight the importance of context scaling for ubiquitous applications. We argue that developers and system architects of ubiquitous computing applications should take into account how context scales and choose an appropriate aggregation policy.

As part of future work, it would be interesting to conduct empirical studies to validate how context scales in various types of systems. Additionally, in our work we assumed a fairly generic model for the movement of mobiles. Recent developments in modeling human mobility, e.g., [53, 64] offer insight on the way we move. An experimental comparison between our theoretical

results and experimental results based on these models would strengthen the applicability of our work.

In Chapters 3 and 4, we examined a distributed platform for storing, querying and computing context. Our work is the first, to the best of our knowledge, to argue that a P2P platform, already established as a viable alternative for storing content, has great potential in this application as well. However, the nature of a P2P-based network to support context awareness is quite different. In particular, locality in the queries users will perform dictates particular design choices to allow a reduction in traffic and allow the platform to scale. Developers of similar platforms should recognize the key role of locality for context and take advantage of it. At the same time, we demonstrated the effect that non-homogeneities in the overlay topology and the traffic pattern might play. The heuristics we proposed for addressing the effect of non-homogeneities are, to the best of our knowledge, the first attempts to adapt the *topology* of an overlay architecture to the underlying traffic in real-time.

Experimenting with alternative data management policies for P2P networks storing context is one of our future goals. For example, 'sectoring' the storage, i.e., allocating half of the storage of a peer to local events, a quarter to events from neighboring peers, i.e., 1 hop away, etc., is a promising alternative policy akin to trunk reservation used in telecom networks. Additionally, a more formal justification of the heuristics presented in Chapter 4 would be desirable.

Our formal model for spatio-temporal context events takes locality into account and can be used to model other types of systems, e.g., [60] where similar assumptions arise. In fact, studying the propagation properties of spatio-temporally local information is the subject of recent research, e.g., [37]. It would be interesting to define a quantitative model of context awareness for such types of information and study how to maximize context awareness for a configuration of information sources under a given network's capacity constraints.

In this direction, we have defined in our preliminary work, 'unary' and 'binary' context awareness, as the fraction of true predicates consisting of a single or a pair of events respectively, that are known to a randomly placed observer. We are currently studying distributed event scheduling algorithms to maximize such context awareness measures. As the number of events and predicates increases, it makes sense to adopt a macroscopic view of context awareness. In that respect, one can study the 'velocity' of information propagation, in the spirit of [27], and compute context awareness as a function of propagation velocity.

Finally, in Chapter 5 we demonstrated the power of localized context by creating a lightweight platform targeted at RFID technology. We expect that the abstractions demonstrated in our work, i.e., the spatially-aware context stack and spatially-aware messaging boards, are general purpose solutions that could be used by future developers of ubiquitous applications. Additionally, we believe that the profile abstraction is a novel anonymizing mechanism with

potential to address some of the privacy-related issues plaguing the large-scale adoption of ubiquitous technologies.

Appendices

Appendix A

Proofs of Theorems in Chapter 2

A.1 Proof of Theorem 2.7.1

Thm. 2.7.1 establishes the ranges of scales producing savings in context acquisition compared to the fine grain tessellation. Starting with Eq. 2.4 and using the additive scaling of context we get

$$\begin{aligned} \sqrt{\lambda_a} * (h + \frac{\lambda_f}{\lambda_a} c(V_f)) &< \sqrt{\lambda_f} * (h + c(V_f)) \Rightarrow \\ h + \frac{\lambda_f}{\lambda_a} c(V_f) &< \sqrt{\frac{\lambda_f}{\lambda_a}} (h + c(V_f)). \end{aligned}$$

By substituting $r \triangleq \sqrt{\frac{\lambda_f}{\lambda_a}}$ and $x \triangleq \frac{h}{c(V_f)}$ we get

$$x + r^2 < r(x+1) \Leftrightarrow r^2 - r(x+1) + x < 0.$$

Observe that this inequality has a solution since

$$\Delta = (x+1)^2 - 4x = (x-1)^2 \geq 0.$$

The inequality can be equivalently rewritten

$$\begin{aligned} (r - \frac{(1+x)+(x-1)}{2})(r - \frac{(1+x)-(x-1)}{2}) &< 0 \Rightarrow \\ (r-x)(r-1) &< 0. \end{aligned}$$

The solution to the previous inequality is either

$$(r < x) \wedge (r > 1) \Leftrightarrow 1 < r < x \Leftrightarrow 1 < \sqrt{\frac{\lambda_f}{\lambda_a}} < \frac{h}{c(V_f)} \Leftrightarrow \lambda_a \in (\lambda_f(\frac{c(V_f)}{h})^2, \lambda_f),$$

or

$$(r > x) \wedge (r < 1) \Leftrightarrow x < r < 1 \Leftrightarrow \frac{h}{c(V_f)} < \sqrt{\frac{\lambda_f}{\lambda_a}} < 1 \Leftrightarrow \lambda_a \in (\lambda_f, \lambda_f(\frac{c(V_f)}{h})^2).$$

The first solution corresponds to aggregation, i.e., $\lambda_a < \lambda_f$ and is feasible if $c(V_f) < h$. The second solution does not correspond to aggregation, i.e., $\lambda_a > \lambda_f$ and is feasible if $c(V_f) > h$. We reject the second solution because it does not correspond to aggregation.

To calculate the optimal rate for the aggregative tessellation we will minimize the amount of context exchanged in that case. Taking the derivative with respect to λ_a for the left hand-side of Eq. 2.4 and setting it equal to zero we get

$$\frac{h}{2\sqrt{\lambda_a}} - \frac{\lambda_f c(V_f)}{2\lambda_a^{\frac{3}{2}}} = 0 \Leftrightarrow \lambda_{a,opt} = \frac{c(V_f)}{h}\lambda_f.$$

This solution is guaranteed to lie in the interval $(\lambda_f(\frac{c(V_f)}{h})^2, \lambda_f)$ since $(\frac{c(V_f)}{h})^2 < \frac{c(V_f)}{h} < 1$ for $\frac{c(V_f)}{h} < 1$, therefore, it is a true minimum.

The maximum relative cost reduction compared to acquiring context from the finest grain organization $V(\Pi_f)$ is

$$1 - \frac{\sqrt{\lambda_{a,opt}}(h + c(V_a))}{\sqrt{\lambda_f}(h + c(V_f))}.$$

By plugging in the value for $\lambda_{a,opt}$ we get

$$1 - \frac{\sqrt{hc(V_f)}}{\frac{h+c(V_f)}{2}},$$

which can be simplified to

$$1 - \frac{2\sqrt{x}}{1+x}.$$

A.2 Proof of Fact 2.8

We want to determine the effect of selective context in the interval of possible scales achieving gains compared to the finest grain tessellation. The cost of the aggregative tessellation remains unaffected. The cost of the finest grain tessellation will be reduced due to the selective exchange of context. The necessary condition to achieve savings is

$$\sqrt{\lambda_a}(h + \frac{\lambda_f}{\lambda_a}c(V_f)) < \sqrt{\lambda_f}(h + qc(V_f))p.$$

Performing the substitutions $r \triangleq \sqrt{\frac{\lambda_f}{\lambda_a}}$, $x \triangleq \frac{h}{c(V_f)}$, the previous inequality becomes

$$r^2 - rp(x+q) + x < 0.$$

We define the function $f(r) \triangleq r^2 - rp(x+q) + x$ and study its behavior in the interval $r > 0$. In order to achieve aggregation we need the minimum of f to be achieved in the $r > 1$ interval. So,

$$f'(r) = 0 \Leftrightarrow r_{opt} = \frac{p(x+q)}{2} > 1 \Leftrightarrow p(x+q) > 2.$$

The minimum value achieved should be negative

$$\begin{aligned} f(r_{opt}) = \frac{p^2(x+q)^2}{4} - \frac{p^2(x+q)^2}{2} + x = x - \frac{p^2(x+q)^2}{4} &< 0 \Leftrightarrow \\ p(x+q) &> 2\sqrt{x}. \end{aligned}$$

Combining the previous two equations we get

$$p(x+q) > 2\max(1, \sqrt{x}).$$

If $x < 1$, the maximum value of the left hand-side of the previous equation is $x+1$ and corresponds to the case $p = q = 1$. Also $\max(1, \sqrt{x}) = 1$ and as a result $x+1 > 2 \Leftrightarrow x > 1$ which contradicts our assumption. Therefore, we have $x > 1$ and the necessary condition is

$$p(x+q) > 2\sqrt{x}.$$

A.3 Proof of Theorem 2.10.1

Starting from Eq. 2.9 the necessary condition to achieve savings is

$$\sqrt{\lambda_a} * (h + (\frac{\lambda_f}{\lambda_a})^{\alpha} * c(V_f)) < \sqrt{\lambda_f}(h + c(V_f)).$$

Substituting $x \triangleq \frac{h}{c(V_f)}$ and $r \triangleq \sqrt{\frac{\lambda_f}{\lambda_a}}$ we get

$$x + r^{2\alpha} < r(x+1) \Leftrightarrow r^{2\alpha} - r < x(r-1). \tag{A.1}$$

We care about solutions that correspond to aggregation, i.e., $r > 1$. We will do a case analysis:

- $\alpha < \frac{1}{2}$. In this case $x(r-1) > 0$ and $r^{2\alpha} - r < 0$, so by default we can see that the inequality is satisfied for every $r > 1$. That means that the allowable range for λ_a is $(0, \lambda_f)$.

- $\alpha > \frac{1}{2}$. Observe that in this regime the term $r^{2\alpha}$ will grow asymptotically bigger than the term $r(x+1)$. This means that the inequality has at best an interval over which it can be satisfied. We will identify the conditions under which this interval does exist. Manipulating Eq. A.1 we get

$$f(r) \triangleq r^{2\alpha} - r(x+1) + x < 0, r > 1. \tag{A.2}$$

Observe that $f(1) = 1 - (x+1) + x = 0$. To ensure the interval in question exists we impose a decreasing condition on $f()$ at $r = 1$,

$$f'(1) = 2\alpha r^{2\alpha-1} - (x+1) = 2\alpha - (x+1) < 0 \Rightarrow \frac{x+1}{2\alpha} > 1, \alpha > \frac{1}{2}.$$

The minimum for $f()$ is attained at r_0 s.t.

$$r_0^{2a-1} = \frac{x+1}{2\alpha}.$$

The condition for decreasing behavior imposed on $f()$ guarantees that $r_0 > 1$, i.e., we are in the aggregation regime. The value at r_0 is indeed a minimum for $f()$ since

$$f''(r_0) = 2\alpha(2\alpha-1)r_0^{2\alpha-2} > 0.$$

Additionally, we also need to prove that the minimum value for $f()$ is negative, thus satisfies Eq. A.2. As noted previously $f(1) = 0$, therefore the minimum $f(r_0) < 0$.

All the previous properties proven for $f()$ taken collectively ensure that the interval $(\hat{\lambda}_a, \lambda_f)$ where $\hat{\lambda}_a$ is the maximum solution to Eq. 2.10 is the

interval where aggregation is a win. The existence of $\hat{\lambda}_a$ is guaranteed by the fact that f is continuous, has a negative minimum for $r_0 > 1$ and asymptotically goes to infinity for $r \to \infty$.

To find the value where the amount of context transfered by an aggregative organization is minimized we define the following auxiliary function.

$$g(\lambda_a) \triangleq \sqrt{\lambda_a}(h + \frac{\lambda_f^\alpha}{\lambda_a^\alpha} c(V_f)) - \sqrt{\lambda_f}(h + c(V_f)).$$

We take the derivative of $g()$ with respect to λ_a and set it equal to 0

$$\frac{h}{2\sqrt{\lambda_a}} + (\frac{1}{2} - \alpha)\lambda_f^\alpha c(V_f) \frac{1}{\sqrt{\lambda_a}\lambda_a^\alpha} = 0,$$

which gives us the value for $\lambda_{a,opt}$

$$\lambda_{a,opt} = (\frac{(2\alpha - 1)c(V_f)}{h})^{\frac{1}{\alpha}} \lambda_f.$$

This expression is valid since $\alpha > \frac{1}{2}$. The optimality is verified by checking the sign of the second derivative of $f()$ at $\lambda_{a,opt}$

$$f''(\lambda_\alpha) = -\frac{h}{4\lambda_\alpha^{\frac{3}{2}}} - \frac{1 - 2\alpha}{2} c(V_f)\lambda_f^\alpha (\alpha + \frac{1}{2}) \frac{1}{\lambda_a^{\alpha + \frac{3}{2}}}.$$

Manipulating the previous expression we get

$$f''(\lambda_a) = \frac{1}{4\lambda_\alpha^{\frac{3}{2}}} (\frac{(2\alpha + 1)(2\alpha - 1)c(V_f)\lambda_f^\alpha}{\lambda_a^\alpha} - h).$$

Substituting the expression for $\lambda_{a,opt}$ we get

$$f''(\lambda_{a,opt}) = \frac{\alpha h}{2\lambda_{a,opt}^{\frac{3}{2}}} > 0.$$

therefore, we have achieved a minimum.

The optimal rate for the aggregative tessellation corresponds to aggregation if

$$\frac{(2\alpha-1)c(V_f)}{h} < 1 \Leftrightarrow \frac{c(V_f)}{h} < \frac{1}{2\alpha-1}.$$

which is the condition we have already identified above.

We will not provide a proof for the expression of the maximum relative cost reduction. The steps are essentially the same as with the case $\alpha = 1$.

A.4 Proof of Fact 2.11

Fact 2.11 establishes the benefitial range for aggregative tessellations for a given α when we exchange context selectively. We will follow the steps of the proof of Fact 2.8. Let

$$f(r) \triangleq r^{2\alpha} - rp(x+q) + x.$$

We are interested in r s.t. $f(r) < 0$.

For $\alpha < \frac{1}{2}$, observe that $f(1) = 1-p(x+q)+x = (1-pq)+(1-p)x > 0$, but r grows faster than $r^{2\alpha}$ as $r \to \infty$. Therefore, there exists a point $\hat{r}_a$ s.t. $f(\hat{r}_a) = 0$ and for $r > \hat{r}_a$ the aggregative organization is a win, see Fig.A.1 (left).

For $\alpha > \frac{1}{2}$, observe that $f(1) > 0$ and $r^{2\alpha}$ grows faster than r as $r \to \infty$. The aggregative organization can be a win only if there exists an interval where

$f()$ is decreasing and its minimum is negative. Therefore,

$$f'(r) = 2\alpha r^{2\alpha-1} - p(x+q) \Rightarrow f'(1) = 2\alpha - p(x+q) < 0 \Rightarrow 2\alpha < p(x+q).$$

The minimum is achieved at r_0 s.t.

$$f'(r_0) = 0 \Leftrightarrow 2\alpha r_0^{2\alpha-1} - p(x+q) = 0 \Leftrightarrow r_0^{2\alpha-1} = \frac{p(x+q)}{2\alpha} > 1.$$

The previous result combined with the observation that we are operating in the $\alpha > \frac{1}{2}$ regime leads to the conclusion that $r_0 > 1$.

We will identify the necessary conditions so that the minimum $f(r_0)$ is negative. Evaluating $f()$ at r_0 we get

$$r_0 \frac{p(x+q)}{2\alpha} - r_0 p(x+q) + x.$$

Since $r_0 > 1$ as proved earlier an upper bound for the minimum is

$$r_0 \frac{p(x+q)}{2\alpha} - r_0 p(x+q) + r_0 x = r_0 (\frac{p(x+q)}{2\alpha} - p(x+q) + x).$$

A necessary condition for the miminum to be negative is

$$\frac{p(x+q)}{2\alpha} - p(x+q) + x < 0 \Leftrightarrow p(x+q) > \frac{2\alpha}{2\alpha - 1} x.$$

Combining this result with the result for the decreasing behavior of $f()$ we get

$$p(x+q) > 2\alpha \max\{1, \frac{x}{2\alpha - 1}\}.$$

Assume $x < 2\alpha - 1$, then $p(x+q) > 2\alpha$. But, at the same time $p(x+q) < x + 1 < 2\alpha - 1 + 1 = 2\alpha$, which is a contradiction. Therefore,

$$p(x+q) > \frac{2\alpha}{2\alpha - 1} x, x > 2\alpha - 1.$$

Since $f(1) > 0, f(r) \to \infty$ as $r \to \infty$ and $f()$ has a negative minimum, there exist r'_a, r''_a s.t. for $r \in (r'_a, r''_a)$ an aggregative organization is a win, see Fig. A.1 (right).

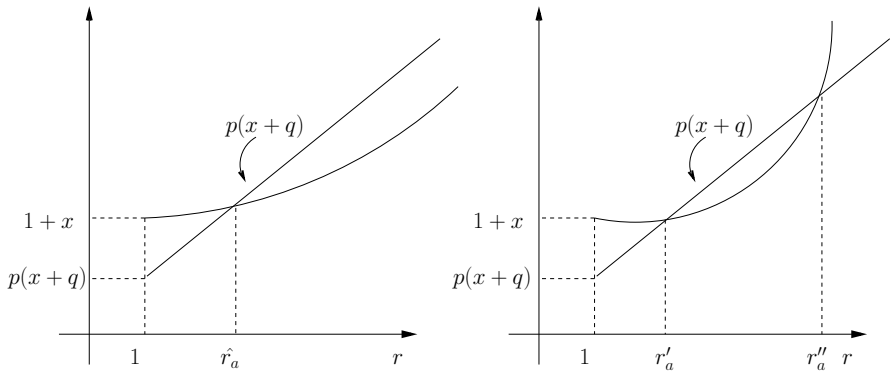

Figure A.1: Behavior of selective context transfer for aggregative organization for $\alpha < \frac{1}{2}$ (left) and $\alpha > \frac{1}{2}$ (right).

A.5 Proof of Theorem 2.12.1

The condition for a hierarchical organization to obtain savings compared to acquiring context from the finest-grained tessellation is

$$\sqrt{\lambda_a}(h+s) + \sqrt{\lambda_f}(h + c(V_f) - s) < \sqrt{\lambda_f}(h + c(V_f)) \tag{A.3}$$

where $r \triangleq \sqrt{\frac{\lambda_f}{\lambda_a}} > 1, s \triangleq \frac{r^2 - r^{2\alpha}}{r^2 - 1} c(V_f)$ is the amount of shared context content between the cells of a finest-grained tessellation that are covered by a cell of the aggregative tessellation. Introducing r and x to the previous equation we get

$$(h+s) + r(h + c(V_f) - s) < r(h + c(V_f))$$

which in turn gives

$$(x + \frac{s}{c(V_f)}) + r(x + 1 - \frac{s}{c(V_f)}) < r(x + 1), x \triangleq \frac{h}{c(V_f)}.$$

Deleting common terms, re-arranging and plugging-in the expression for s we get

$$x < r\frac{s}{c(V_f)} - \frac{s}{c(V_f)} = \frac{s}{c(V_f)}(r - 1)$$

equivalently,

$$x < \frac{r^2 - r^{2\alpha}}{r^2 - 1}(r - 1) = \frac{r^2 - r^{2\alpha}}{r + 1}.$$

If $a \geq 1 \Rightarrow r^2 < r^{2\alpha}$, for $r \in [1, \infty)$. So the previous inequality cannot be satisfied. If $a < 1 \Rightarrow r^2 > r^{2\alpha}$, for $r \in [1, \infty)$. So there exists a level of aggregation, i.e., r_0 s.t. for $r > r_0$ the previous inequality is satisfied and the hierarchical organization wins.

A.6 Proof of Theorem 2.12.2

We are trying to compute the conditions under which the hierarchical approach is more beneficial than the aggregative one when context grows in a sub-additive way. This regime is equivalently described as:

$$0 < \alpha < 1 \text{ and } \lambda_a \in (0, \lambda_f).$$

In order for the hierarchical approach to have reduced cost compared to the aggregative approach we would like the following inequality to hold:

$$\sqrt{\lambda_a}(h + s) + \sqrt{\lambda_f}(h + c(V_f) - s) < \sqrt{\lambda_a}(h + c(V_a))$$

where $s \triangleq \frac{r^2 - r^{2\alpha}}{r^2 - 1} c(V_f)$ is the amount of shared context content between the cells of a finest-grained tessellation that are covered by a cell of the aggregative tessellation.

Substituting the expression for $c(V_a)$ from Section 2.12 and setting $r \triangleq \sqrt{\frac{\lambda_f}{\lambda_a}} > 1$ we get

$$(h + s) + r(h + c(V_f) - s) < h + r^2(c(V_f) - s) + s)$$

or equivalently,

$$x + \frac{s}{c(V_f)} + r(x + 1 - \frac{s}{c(V_f)}) < x + r^2(1 - \frac{s}{c(V_f)}) + \frac{s}{c(V_f)}, x \triangleq \frac{h}{c(V_f)}.$$

Substituting the expression for s we get

$$x + \frac{r^2 - r^{2\alpha}}{r^2 - 1} + r(x + 1 - \frac{r^2 - r^{2\alpha}}{r^2 - 1}) < x + r^2(1 - \frac{r^2 - r^{2\alpha}}{r^2 - 1}) + \frac{r^2 - r^{2\alpha}}{r^2 - 1}.$$

Cancelling common terms we get the expression of Theorem 2.12.2.

$$rx < r^2(1 - \frac{r^2 - r^{2\alpha}}{r^2 - 1}) - r(1 - \frac{r^2 - r^{2\alpha}}{r^2 - 1}) \Leftrightarrow$$
$$rx < r(r - 1)\frac{r^2 - 1 - r^2 + r^{2\alpha}}{(r - 1)(r + 1)} \Leftrightarrow$$
$$x < \frac{r^{2\alpha} - 1}{r + 1}.$$

A.7 Proof of Fact 2.16

For the direct case, note that the detection of a boundary crossing event must happen in an area of radius R around the device performing the detection, provided that R is smaller than the mean cell size. This assumption

currently holds in practice, e.g., passive state-of-the-art RFID readers have a range $R \sim 50\ cm$ while the mean diameter of, e.g., a store in a mall is at least $5\ m$. Thus, the frequency f_d has to be greater than $\frac{\bar{v}}{R}$ where $\bar{v}$ is the average speed of mobiles.

For the indirect case, since cells are almost surely bounded, a mobile entering a typical cell will eventually leave the cell. Consider a typical cell, by Little's theorem, the average sojourn time in the cell T_{soj} must be the ratio of the mean number of mobiles in the cell to the average rate of users into the cell. Assuming that the initial distribution of mobiles is Poisson with intensity λ_0 and under our the mobility model in Assumption 2.4 one can show that the mean number of mobiles in a typical cell is λ_0/λ_a while the mean rate of mobiles entering the cell is proportional to their intensity, mean velocity and typical cells perimeter, i.e., $\frac{\lambda_0 v}{\sqrt{\lambda_a}}$. Thus, the mean sojourn time is proportional to $\frac{1}{v\sqrt{\lambda_a}}$ giving the desired result.

Appendix B

Propositions Related to our P2P Network

B.1 A proof of Prop. 3.9.1

We start with the plain grid case. Substituting the values

$$\gamma_q(f) = \gamma_q^0 f, \quad \gamma_e(f) = \gamma_e^0 f, \quad n^2(f) = n_0^2 f^q, \quad \mu(f) = \mu_0 f^p, \quad \text{and} \quad l(f) = \frac{l_0}{f^s}.$$

to Eqs. 3.1, 3.2, and 3.3 and ignoring constants:

$$h(f) = O(\frac{1}{f^s})O(\frac{1}{f^{-\frac{q}{2}}}) = O(f^{\frac{q}{2}-s})$$

and

$$\begin{aligned} \gamma(f) &= O(f)O((f^{-\frac{q}{2}})^2)\max(1, h(f)) + O(f)O((f^{-\frac{q}{2}})^2) \\ &= O(f^{1-q})\max(1, O(f^{\frac{q}{2}-s})) + O(f^{1-q}). \end{aligned}$$

So, the average delay from source to destination for a query is roughly

$$D(f) = \frac{\max(1, O(f^{\frac{q}{2}-s}))}{O(f^p) - O(f^{1-q})\max(1, O(f^{\frac{q}{2}-s})) - O(f^{1-q})}.$$

We want to ensure that $D(f)$ remains bounded as f grows without bound. We will examine two cases:

1. $\frac{q}{2} - s \leq 0$

In this case, $\max(1, O(f^{\frac{q}{2}-s})) = O(1)$ for large values of f. The delay, becomes:

$$D(f) = \frac{1}{O(f^p) - O(f^{1-q}) - O(f^{1-q})}.$$

So $p \geq 1 - q$ is a necessary and sufficient condition for $D(f)$ to remain bounded. Since $\min(\frac{q}{2}, s) = \frac{q}{2}$, the previous condition can be expressed as

$$p \geq 1 - \frac{q}{2} - \min(\frac{q}{2}, s). \tag{B.1}$$

2. $\frac{q}{2} - s \geq 0$

In this case $\max(1, O(f^{\frac{q}{2}-s})) = O(f^{\frac{q}{2}-2})$ for large values of f. The delay becomes:

$$\begin{aligned} D(f) &= \frac{O(f^{\frac{q}{2}-s})}{O(f^p) - O(f^{1-q})O(f^{\frac{q}{2}-s}) - O(f^{1-q})} \\ &= \frac{O(f^{\frac{q}{2}-s})}{O(f^p) - O(f^{1-\frac{q}{2}-s}) - O(f^{1-q})}. \end{aligned}$$

The necessary and sufficient condition for $D(f)$ to remain bounded as f grows without bound is that $p \geq \max(1 - \frac{q}{2} - s, 1 - q) = (1 - \frac{q}{2}) + \max(-\frac{q}{2}, -s)$ and that $p \geq \frac{q}{2} - s$. But, $\max(-s, -\frac{q}{2}) = \min(\frac{q}{2}, s)$. So, the condition becomes:

$$p \geq 1 - \frac{q}{2} - \min(\frac{q}{2}, s), p \geq \frac{q}{2} - s. \tag{B.2}$$

Comparing Eqs. B.1 and B.2, the necessary condition for $D(f)$ to remain bounded is

$$p \geq 1 - \frac{q}{2} - \min(\frac{q}{2}, s).$$

We now examine the case where we augment the topology with Kleinberg edges. As discussed in Section 3.9 the expression in Eq. 3.2 is only a lower bound of the delay when using Kleinberg edges. Performing the same scalability substitutions as previously but using Eq. 3.4 and ignoring constants we get:

$$h(f) \approx (\log(\frac{1}{O(f^s)} \frac{1}{O(f^{-\frac{q}{2}})}))^2 = O(\log(f^{\frac{q}{2}-s})^2)$$

and

$$\begin{aligned} \gamma(f) &= O(f)O((f^{-\frac{q}{2}})^2) \max(1, h(f)) + O(f)O((f^{-\frac{q}{2}})^2) \\ &= O(f^{1-q}) \max(1, O(\log(f^{\frac{q}{2}-s})^2)). \end{aligned}$$

Substituting, to the expression for the delay we get:

$$D(f) = \frac{\max(1, O(\log(f^{\frac{q}{2}-s})^2))}{O(f^p) - O(f^{1-q}) \max(1, O(\log(f^{\frac{q}{2}-s})^2))}.$$

For large f, $f^p > \log(f^{\frac{q}{2}-s})^2$, the interpretation of this fact is that in the presence of Kleinberg edges, locality does not make a big difference. Thus, the necessary and sufficient condition for $D(f)$ to remain bounded as f grows without bound is:

$$p > 1 - q.$$

Note the strict inequality in the previous expression. This condition ensures that the lower bound of the delay stays bounded.

B.2 A proof of Fact 3.6

We start with a basic observation that will prove helpful in the sequel:

Fact B.3. *Consider any peer $p \in P$.*

1. *If a new peer p_{new} joins the overlay P2P network $C(p|P \cup \{p_{new}\}) \subseteq C(p|P)$.*

2. *If a peer $p_{old} \in P$ leaves/fails the overlay P2P network $C(p|P) \subseteq C(p|P \setminus \{p_{old}\})$.*

Fact B.3 expresses the idea that a new peer will 'acquire' some space at the expense of other peers to form its Voronoi cell while a peer that is leaving will 'release' the space of its Voronoi cell to some of the peers.

We will start with the case of a peer, p_{old}, leaves the overlay, and show that only the neighbors of p_{old} have to update their neighborhoods. Fact B.3 asserts that in case that a peer leaves the overlay, the remaining peers can change their cells only by growing. Consider a point $\sigma \in C_{p_{old}}$, see also Fig. B.1. When p_{old} leaves the P2P network σ will have to belong to the Voronoi cell of peer p_1 that is the next closest peer to σ. We will prove that p_1 has to be a neighbor of p_{old} in the DG. Assume p_1 is not a neighbor of p_{old}. Consider the line connecting σ to p_1. Without loss of generality assume that there exists a peer p_2 s.t. p_2 is a neighbor of p_{old} and the line (σ, p_1) crosses the shared edge between p_1, p_2. By our assumption p_1 is the next closest peer to the event σ so $|\sigma - p_1|$ has to be minimum among all the peers in $P \setminus \{p_{old}\}$. But

$$|\sigma - p_1| = |\sigma - \pi| + |\pi - p_1|$$

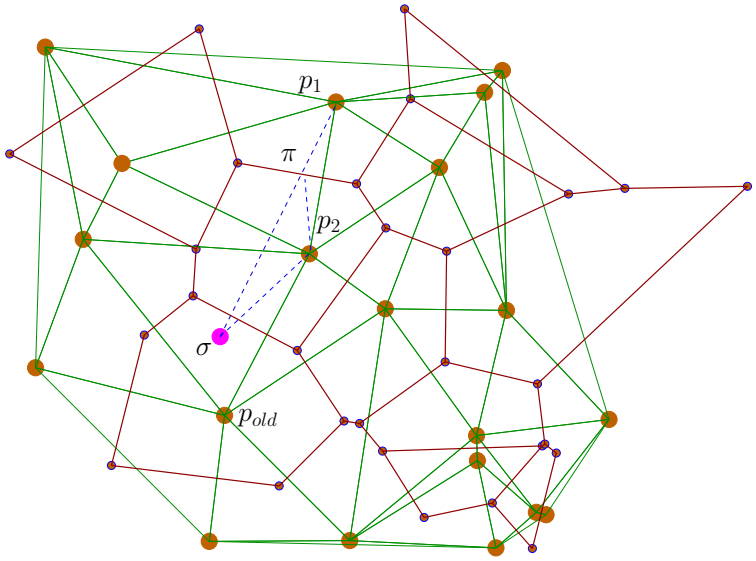

Figure B.1: Peer p_{old} leaves the P2P network.

where π is the unique point where the line (σ, p_1) intersects the shared edge between p_1 and p_2. By definition, since π belongs to the shared edge $|\pi - p_1| = |\pi - p_2|$, we have that

$$|\sigma - p_1| = |\sigma - \pi| + |\pi - p_2|.$$

Using the triangle inequality we have that

$$|\sigma - p_2| < |\sigma - \pi| + |\pi - p_2|.$$

Combining the two previous equations we get that

$$|\sigma - p_2| < |\sigma - p_1|.$$

which is a contradiction. So, p_1 has to be a neighbor of p_{old}.

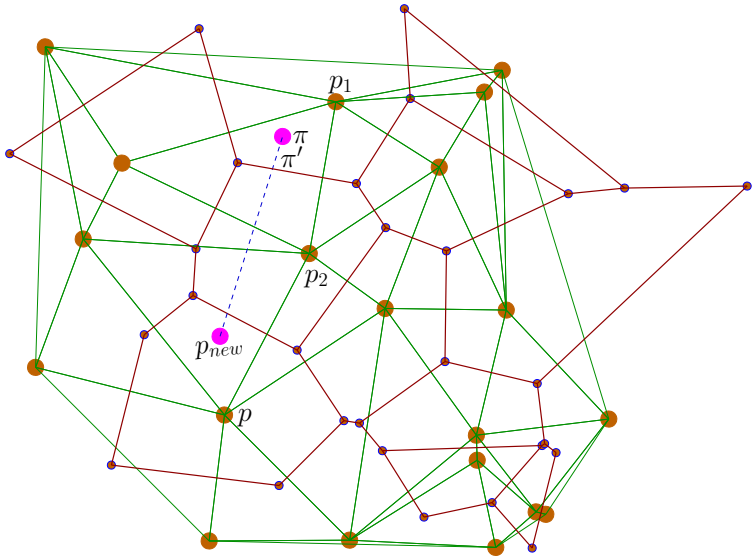

Figure B.2: Peer p_{new} joins the P2P network.

Next we examine the case where a peer p_{new} joins the overlay. Assume that $p_{new} \in C_p$, see Fig. B.2. Let p_1 be a peer whose cell will change as a result of p_{new} joining the P2P network. According to Fact B.3, the cell of p_1, C_{p_1} must shrink. Let $\pi \in C(p_1|P)$ and assume that after p_{new} joins, $\pi \in C(p_{new}|P \cup \{p_{new}\})$. Every point in the line between π and p_1 will belong

either to C_{p_1} or to $C_{p_{new}}$. Thus, p_1 and p_{new} have to be neighbors. Moreover, consider the line connecting p_{new} and π. The point π is internal to $C(p_1|P)$ and p_{new} is external, so there has to exist a neighbor of p_1 in $DG(P)$, call it p_2, and a point π' in the Voronoi edge between p_1 and p_2 s.t $\pi' = C(p_1|P) \cap line(p_1, p_{new})$. $C_{p_{new}}$ is convex, therefore π' has to be inside $C_{p_{new}}$ in the resulting Voronoi tessellation. So we proved that for any peer whose cell changes, it has to become a neighbor of p_{new} and at least one of the edges of C_{p_1} will change.

Locating the new neighbors. Are the peers in $N(p_{new})$ necessarily neighbors of p, i.e., $N(p_{new}) \subseteq N(p|P) \cup \{p\}$? Once p finds out about p_{new} joining the overlay, it could notify the peers in $N(p|P)$ and the process would conclude there. We offer without proof the following fact:

Fact B.4. $N(p_{new}) \not\subseteq N(p|P) \cup \{p\}$.

How can the peers in $N(p_{new})$ be identified at the time p_{new} joins? Consider a peer $v \in N(p_{new})$. If v is a neighbor of p in DG it can be contacted immediately. Otherwise, p can broadcast to its neighbors an advertisement of p_{new}'s join event. All peers recursively forward this request to their neighbors using the following criterion: a peer u forwards the advertisement about p_{new} to its neighbor v iff $p_{new} \in \cup_{\alpha \in ve(u,v)} B(\alpha, ||\alpha - u||)$, see Fig. B.3.

B.5 A proof of Fact 3.13

We want to prove that for any event e stored in our platform and a new owner of e, q, that joins the platform, there exists at least one owner of e among

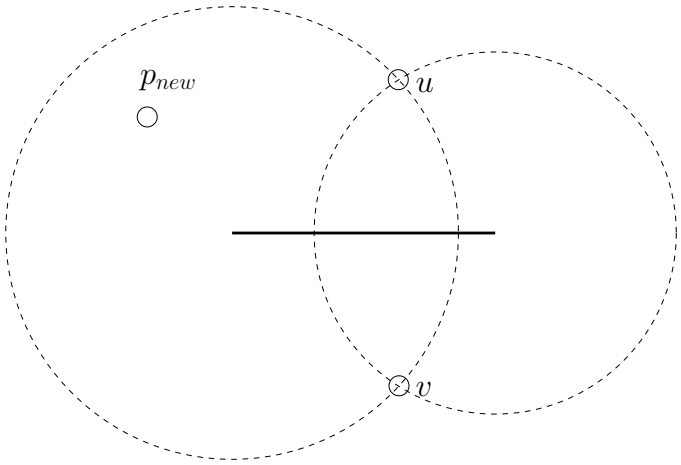

Figure B.3: The thick black solid line represents the Voronoi edge between u and v. The thin dashed disks define the locations for which p_{new} is closer to a point in $ve(u, v)$ than u and v.

the neighbors of q. Let $\{p_0, \ldots, p_{n-1} = h_k(e)\}$ be the set of owners of e *after* q joins the overlay. If $q = p_0$, then by the first statement of Prop. 3.12.1 (applied for $p_!$), it has to be a neighbor of p_1, which proves our claim. Otherwise, we assume that $q = p_j, j \in [1, n-1]$. By the first statement of Prop. 3.12.1, q and p_{j-1} are neighbors in the $G_D(P \setminus \{p_0, \ldots, p_{j-2}\})$. The addition of the peers from $\{p_0, \ldots, p_{j-2}\}$ will either remove the edge between q and p_{j-1} or not. If it doesn't then q and p_{j-1} will still be neighbors in $DG(P \cup \{q\})$, otherwise at least one of the peers in $\{p_0, \ldots, p_{j-2}\}$ will become a neighbor of q (replace p_{j-1}). In any case, q will have at least one neighbor from the set $\{p_0, \ldots, p_{j-1}\}$ in $DG(P \cup \{q\})$ which proves our claim.

B.6 Generating Kleinberg edges

The work in [43] describes an algorithm to generate 'long' edges between the points of a $n \times n$ square grid. The effect of those edges is that the average distance between two uniformly randomly chosen points in the grid is $O(\log(n)^2)$. We call those edges, 'Kleinberg' edges. In this section we describe an algorithm to generate Kleinberg edges for any topology. Additionally, we discuss a locality constraint on them, i.e, the source and the destination of an edge can be separated by a maximum distance.

Let R denote the length of such an edge. We modify the distribution of R suggested in [43] as follows to encode our locality constraint:

$$f_R(\rho) = \begin{cases} K\rho^{-2}, & \epsilon \leq \rho \leq \rho_{max} \\ 0, & \text{else} \end{cases} \tag{B.3}$$

where ρ_{max} is the maximum distance between the source and the destination of the edge, and ϵ is a parameter to avoid the singularity of the distribution at $\rho = 0$.

Using the above distribution we get:

$$Pr[R \leq \rho] = \begin{cases} 2\pi \int_{\epsilon}^{\rho} Kt^{-2}tdt = 2\pi K \log(\frac{\rho}{\epsilon}), & \epsilon \leq \rho \leq \rho_{max} \\ 0, & \text{else} \end{cases} \tag{B.4}$$

where log() is the natural logarithm.

Normalizing the probability we get:

$$2\pi K \log(\frac{\rho_{max}}{\epsilon}) = 1 \Leftrightarrow K = \frac{1}{2\pi \log(\frac{\rho_{max}}{\epsilon})}.$$

To generate a Kleinberg edge with the above distribution we generate a random number x with a uniform distribution in the interval $[0, 1]$. That value corresponds to an edge length ρ:

$$\begin{aligned} x &= 2\pi \frac{1}{2\pi \log(\frac{\rho_{max}}{\epsilon})} \log(\frac{\rho}{\epsilon}) \Leftrightarrow \\ \log(\frac{\rho}{\epsilon}) &= x \log(\frac{\rho_{max}}{\epsilon}) \Leftrightarrow \\ \frac{\rho}{\epsilon} &= (\frac{\rho_{max}}{\epsilon})^x \Leftrightarrow \\ \rho &= \epsilon(\frac{\rho_{max}}{\epsilon})^x. \end{aligned}$$

Appendix C

RFID design

For both implementations (hardware and software) we chose to implement a SE with the same characteristics so that direct comparisons could be drawn. We used passive RFID tags from Texas Instruments. The tags were compatible with the ISO15693 standard specification and were included in the TI S4100 evaluation kit along with the reader. Each one of the RFID tags is capable of holding 256 bytes of non-volatile memory. Given the memory constraints of the tags used we selected the parameters shown in Figs. C.1 and C.2 for the structures described in Section 5.3.

The parameters have to obey the following constraint imposed by the total tag memory:

$$4 * (6 + x) + 4 * 5 * z + 2 * y + 2 * w + 6 * k \leq 256.$$

The solution we chose is

$$x = 2, z = 8, y = 4, w = 4, k = 8.$$

The parameters chosen allow for 8 levels of nesting, 2^8 different attribute types and 2^8 attributes per attribute type, 2 attributes per profile, 2^8 profile types per attribute type. Each messaging board can hold up to 8

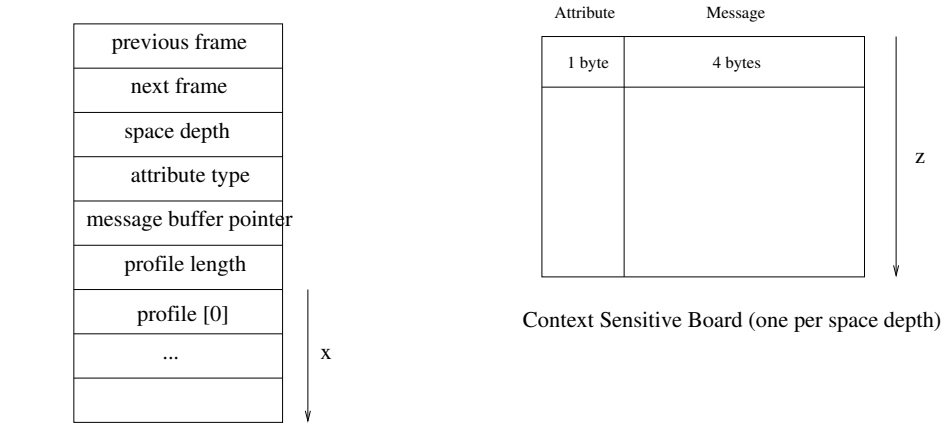

Figure C.1: Implementation

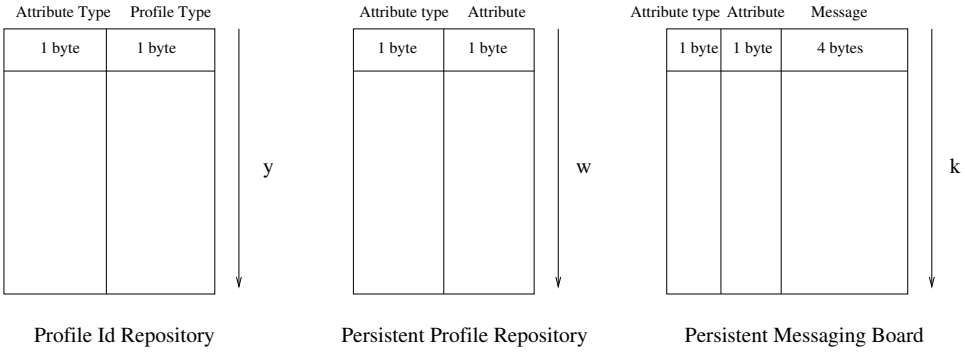

Figure C.2: Implementation (cntnd)

messages and a total of up to 4 profiles can be in an entity's profile repository. The actual message coding used for the messages of our protocol is depicted in Figs. C.3 and C.4.

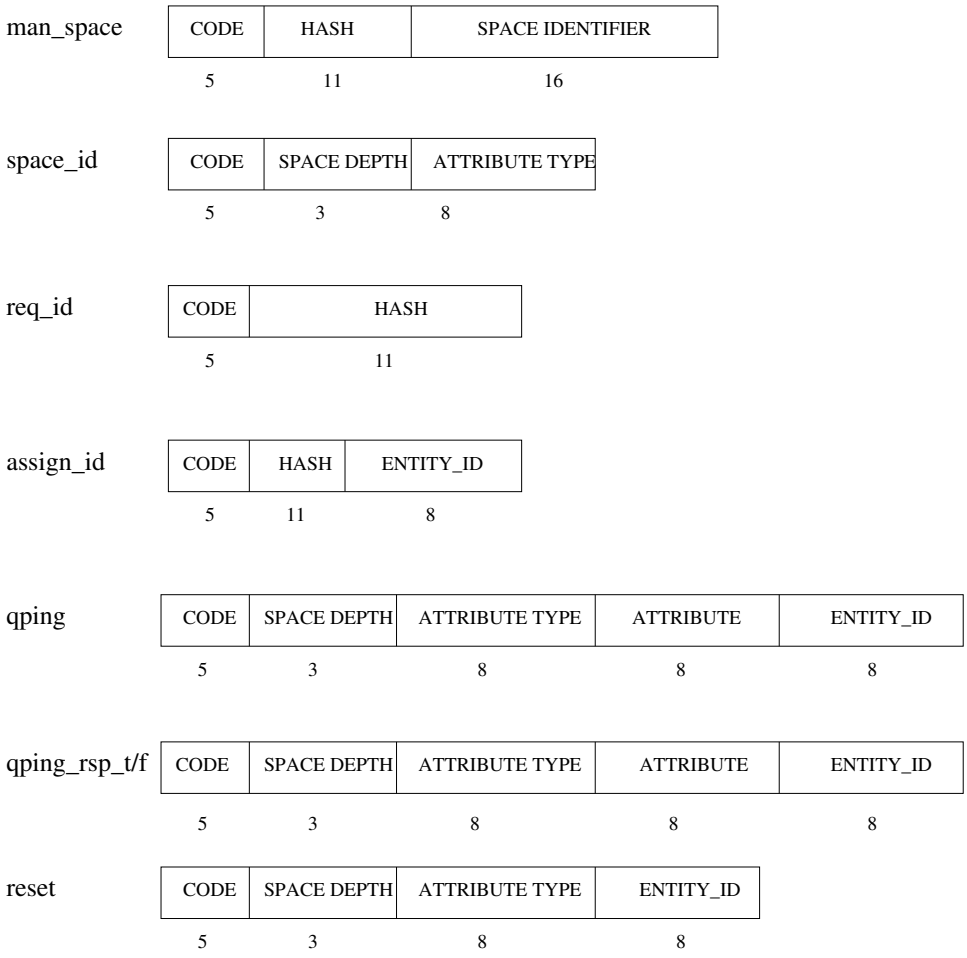

Figure C.3: Message coding

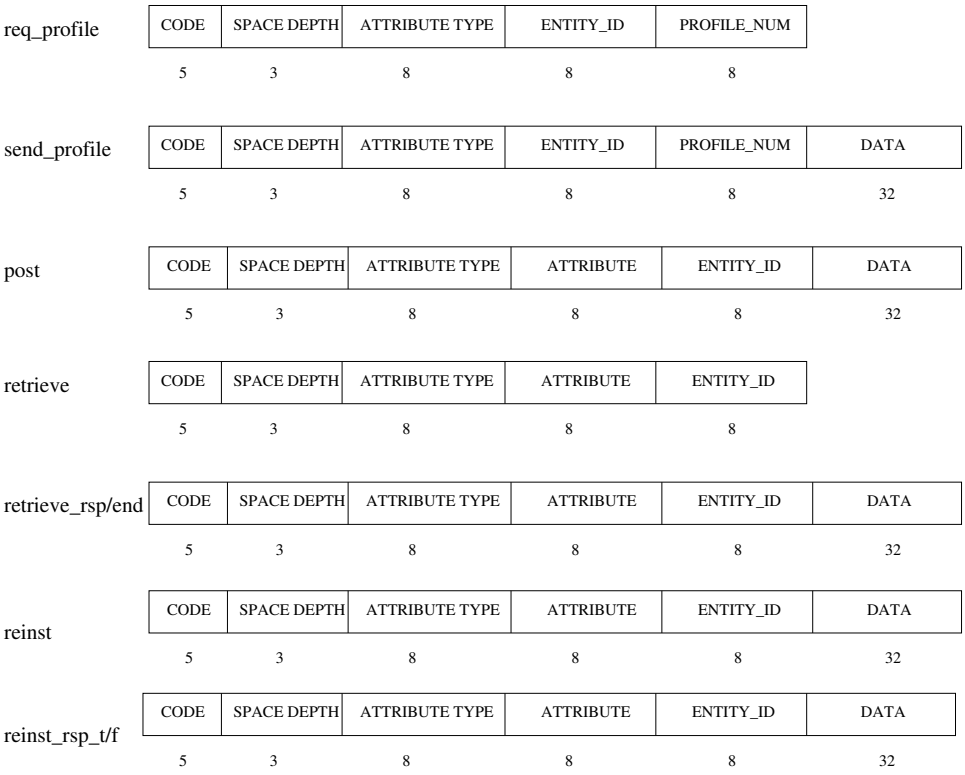

Figure C.4: Message coding (cntnd)

Bibliography

[1] Here comes the wallet phone. `http://www.spectrum.ieee.org/nov05/2150`.

[2] M. Albano et al. VoRaQue: Range queries on Voronoi overlays. In *ISCC*, pages 495–500. IEEE, 2008.

[3] F. Aurenhammer. Voronoi diagrams a survey of a fundamental geometric data structure. *ACM COMPUTING SURVEYS*, 23(3):345–405, 1991.

[4] F. Baccelli et al. Stochastic geometry and architecture of communication networks. *Communications journal*, 1996. Select Proc. of the third INFORMS telecommunications conference.

[5] F. Baccelli and S. Zuyev. Poisson-Voronoi Spanning Trees with Applications to the Optimization of Communication Networks. *Oper. Res.*, 47(4):619–631, 1999.

[6] P. Bahl and V. N. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *INFOCOM (2)*, pages 775–784, 2000.

[7] H. Balakrishnan et al. Looking up data in P2P systems. *Commun. ACM*, 46(2):43–48, 2003.

[8] F. Banaei-Kashani and C. Shahabi. SWAM: A Family of Access Methods for Similarity-Search in Peer-to-Ppeer Data Networks. In *CIKM '04: Proc. of the 13th ACM international conference on information and knowledge management*, pages 304–313, New York, NY, USA, 2004. ACM.

[9] T. Basten et al., editors. *Ambient Intelligence: Impact on Embedded System Design*. Springer Verlag, 2004.

[10] M. Bauer et al. Location Models from the Perspective of Context-Aware Applications and Mobile Ad Hoc Networks. *Personal Ubiquitous Comput.*, 6(5-6):322–328, 2002.

[11] O. Beaumont et al. VoroNet: A scalable object network based on Voronoi tessellations. In *Proc. of the 21st International Parallel and Distributed Processing Symposium (IPDPS 2007)*. Society Press, 2007.

[12] C. Becker and F. Durr. On location models for ubiquitous computing. *Personal Ubiquitous Comput.*, 9(1):20–31, 2005.

[13] M. Beigl et al. Awarecon: Situation aware context communication. In *Proc. of Ubicomp*, pages 12–15, Seattle, October 2003.

[14] A. R. Beresford and F. Stajano. Location privacy in pervasive computing. *IEEE Pervasive Computing*, pages 46–55, January 2003.

[15] P. Bose and P. Morin. Online Routing in Triangulations. In *ISAAC '99: Proc. of the 10th International Symposium on Algorithms and Computation*, pages 113–122, London, UK, 1999. Springer-Verlag.

[16] B. Brumitt et al. Easyliving: Technologies for intelligent environments. In *HUC*, pages 12–29, 2000.

[17] B. Brumitt et al. Ubiquitous computing and the role of geometry. In *IEEE Personal Communications*, pages 41–43, 2000.

[18] N. Carriero and D. Gelernter. Linda in context. *Commun. ACM*, 32(4):444–458, 1989.

[19] M. Castro et al. Future directions in distributed computing, 2003.

[20] G. Chen and D. Kotz. Context Aggregation and Dissemination in Ubiquitous Computing Systems. In *WMCSA '02: Proc. of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, page 105, 2002.

[21] N. Cohen et al. Challenges in flexible aggregation of pervasive data. Technical report, IBM Research Division, Thomas J. Watson Research Center., 2001.

[22] A. Dey. *Providing architectural support for building context-aware applications*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2000. Director-Gregory D. Abowd.

[23] D. Dobkin et al. Delaunay graphs are almost as good as complete graphs. *Discrete Comput. Geom.*, 5(4):399–407, 1990.

[24] M. Erwig and M. Schneider. Spatio-Temporal Predicates. *IEEE Transactions on Knowledge and Data Engineering*, 14:881–901, 1999.

[25] M. Esler et al. Next century challenges: Data-centric networking for invisible computing. In *Mobile Computing and Networking*, pages 256–262, 1999.

[26] Klaus Finkenzeller. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification.* John Wiley & Sons, Inc., New York, NY, USA, 2003.

[27] R. Ganti and M. Haenggi. Bounds on information propagation delay in interference-limited aloha networks. In *Workshop on Spatial Stochastic Models for Wireless Networks (SpaSWiN'09)*, June 2009.

[28] D. Garlan et al. Project aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, April-June 2002.

[29] H. Gellersen et al. Physical prototyping with smart-its. *IEEE Pervasive*, pages 10–18, July 2004.

[30] M. Ghaffari et al. On the necessity of using Delaunay triangulation substrate in greedy routing based networks. *IEEE Communication Letters*, 14:266–268, 2010.

[31] R. Grimm. One.world:experiences with a pervasive computing architecture. *IEEE Pervasive*, pages 22–30, July 2004.

[32] L. Guibas et al. Randomized incremental construction of Delaunay and Voronoi diagrams. In *Proc. of the 17th international colloquium on Automata, languages and programming*, pages 414–431, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[33] B. Hariri et al. Using geometrical routing for overlay networking in MMOGs. *Multimedia Tools and Applications*, 45(1):61–81, 2009.

[34] S. Helal. Programming pervasive spaces. *Pervasive Computing*, January 2005.

[35] S. Holloway et al. Opening Pervasive Computing to the Masses Using the SEAP Middleware. In *Proc. of the Middleware Support for Pervasive Computing Workshop*, Galveston, Texas, 9–13 March 2009.

[36] Shun-Yun Hu, Jui-Fa Chen, and Tsu-Han Chen. VON: a scalable peer-to-peer network for virtual environments. *IEEE Network*, 20(4):22–31, July 2006.

[37] E. Hyyti et al. When does content float? characterizing availability of anchored information in opportunistic content sharing. In *Infocom*, April 2011.

[38] B. Johanson et al. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. *IEEE Pervasive Computing*, 1(2):67–74, 2002.

[39] G. Judd and P. Steenkiste. Providing contextual information to pervasive computing applications. In *IEEE International Conference on Pervasive Computing*, pages 23–25, March 2003.

[40] C. Julien and G.-C. Roman. EgoSpaces: Facilitating Rapid Development of Context-Aware Mobile Applications. *IEEE Transactions on Software Engineering*, 32(5):281–298, May 2006.

[41] F. Kelly. *Reversibility and Stochastic Networks*. John Wiley and Sons, 1979.

[42] M. Khedr and A. Karmouch. Negotiating Context Information in Context-Aware Systems. *IEEE Intelligent Systems*, 19(6):21–29, 2004.

[43] J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proc. of the 32nd ACM Symposium on Theory of Computing*, pages 163–170, 2000.

[44] L. Kleinrock. *Theory, Volume 1, Queueing Systems*. Wiley-Interscience, 1975.

[45] P. Korpipaa et al. Managing context information in mobile devices. *IEEE Pervasive*, pages 42–51, July 2003.

[46] N. Kotilainen et al. Mobile Chedar - A Peer-to-Peer Middleware for Mobile Devices. In *PerCom Workshops*, pages 86–90, 2005.

[47] M. Kranz and A. Schmidt. Prototyping smart objects for ubiquitous computing. In *Proc. of the International Workshop on Smart Object Systems in Conjunction with the Seventh International Conference on Ubiquitous Computing*, September 2005.

[48] E. Kühn et al. Integration of shareable containers with distributed hash tables for storage of structured and dynamic data. In *CISIS*, pages 866–871, 2009.

[49] M. Langheinrich. Privacy by design: Principles of privacy-aware ubiquitous systems. In Gregory D. Abowd, Barry Brumitt, and Steven A. Shafer, editors, *Third International Conference on Ubiquitous Computing (UbiComp 2001)*, pages 273–291, Atlanta, USA, 2001. Springer-Verlag. Lecture Notes in Computer Science 2201.

[50] M. Langheinrich. A privacy awareness system for ubiquitous computing environments. In Gaetano Borriello and Lars Erik Holmquist, editors, *4th International Conference on Ubiquitous Computing (Ubicomp 2002)*, pages 237–245. Springer-Verlag, September 2002. Lecture Notes in Computer Science 2498.

[51] R. Laroia. Future of wireless - the proximate internet? Workshop on Frontiers of Controls, Games, and Network Science with Civilian and Military Applications, February 2010.

[52] D-Y. Lee and S. Lam. Efficient and Accurate Protocols for Distributed Delaunay Triangulation under Churn. In *ICNP '08: Proc. of the International Conference on Network Protocols*, 2008.

[53] K. Lee et al. SLAW: A new mobility model for human walks. In *Infocom*, 2009.

[54] J. Liebeherr et al. Application-layer multicasting with delaunay triangulation overlays. *IEEE Journal on Selected Areas in Communications*, 20:1472–1488, 2001.

[55] J. Loughney et al. Context Transfer Protocol, RFC 4067, July 2005.

[56] D. Marculescu et al. Electronic textiles: A platform for pervasive computing. Proc. of the IEEE, No. 12., Vol. 91, 2003.

[57] J. Moller. *Lectures on Random Voronoi Tessellations*, volume 87 of *Lecture Notes in Statistics*. Springer-Verlag, 1992.

[58] M. Motani et al. PeopleNet: engineering a wireless virtual social network. In *Proc. of the 11th annual international conference on Mobile computing and networking*, MobiCom '05, pages 243–257, 2005.

[59] L. Muche et al. Contact and chord length distributions of the Poisson Voronoi Tessellation. *Journal of Applied Probability*, 29(2):467–471, 1992.

[60] J. Ott et al. Floating content: Information sharing in urban areas. In *Percom*, March 2011.

[61] A. Ranganathan and H Lei. Context-aware communication. *Computer*, April 2003.

[62] S. Ratnasamy et al. Data-Centric Storage in Sensornets with GHT, a Geographic Hash Table. *Mobile Networks and Applications*, 8:2003, 2003.

[63] P. Repo et al. Virtual Product Design Case Study: The Nokia RFID Tag Reader. *IEEE Pervasive*, pages 95–99, October 2005.

[64] I. Rhee et al. On the levy-walk nature of human mobility. In *Infocom*, 2008.

[65] M. Rieback et al. The Evolution of RFID Security. *IEEE Pervasive*, pages 62–69, January 2006.

[66] M. Roman and R. Campbell. Gaia: Enabling active spaces. In *9th ACM SIGOPS European Workshop*, 2000.

[67] M. Roman et al. A middleware infrastructure for active spaces. *IEEE Pervasive*, pages 74–83, October 2002.

[68] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *in MIDDLEWARE*, pages 329–350, 2001.

[69] E. Rukzio et al. A framework for mobile interactions with the physical world. In *Proc. of the Wireless Personal Multimedia Communication (WPMC'05) conference*, September 2005.

[70] H. Ryu et al. A task decomposition scheme for context aggregation in personal smart space. In *SEUS*, pages 20–29, 2007.

[71] S. Sarma et al. Radio-frequency identification: Security risks and challenges. *Cryptobytes*, 6(1), 2003.

[72] M. Satyanarayanan. Augmenting cognition. *IEEE Pervasive Computing*, 3(2):62–70, 2004.

[73] B. Schilit et al. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, 1994.

[74] B. Schilit et al. Challenge: Ubiquitous location-aware computing and the place lab initiative. In *First ACM International Workshop on Wireless Mobile Applications and Services on WLAN (WMASH 2003)*, San Diego, CA, September 2003.

[75] A. Schmidt and K. Van Laerhoven. How to build smart appliances? *IEEE Personal Communications*, August 2001.

[76] Albrecht Schmidt. *Ubiquitous Computing - Computing in Context*. PhD thesis, University of Lancaster, 2002.

[77] G. Simon et al. Distributed dynamic Delaunay triangulation in d-dimensional spaces. Technical report, Institut Eurecom, 2005.

[78] I. Stoica et al. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proc. of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM.

[79] D. Stoyan et al. *Stochastic Geometry and its Applications*. John Wiley and Sons, Chichester, 1987.

[80] A. Villalba Castro et al. Hovering Information - Self-Organising Information that Finds Its Own Storage. In *SUTC*, pages 193–200, 2008.

[81] R. Want. An Introduction to RFID Technology. *Pervasive Computing*, January 2006.

[82] E. Wei and A. Chan. Towards Context-Awareness in Ubiquitous Computing. In *EUC*, pages 706–717, 2007.

[83] M. Weiser. The computer for the twenty-first century. *Scientific American*, pages 94–10, September 1991.

[84] M. Weiser. The world is not a desktop. *Interactions*, pages 7–8, January 1994.

[85] M. Weiser. Turning pervasive computing into mediated spaces. *IBM SYSTEMS JOURNAL - Pervasive Computing Issue*, 38(4), 1999.

[86] S. Weiss. Security and privacy in radio-frequency identification devices. Master's thesis, EECS Dept MIT, May 2003.

[87] T. Winograd. Architectures for context. *Human-Computer Interaction*, pages 16:2–3, 2001.

[88] T. Yamabe et al. Citron: A Context Information Acquisition Framework for Personal Devices. In *RTCSA '05: Proc. of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 489–495, Washington, DC, USA, 2005.

[89] S. Yang et al. Context Model and Context Acquisition for Ubiquitous Content Access in ULearning Environments. In *SUTC '06: Proc. of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing - Vol 2 - Workshops*, pages 78–83, Washington , DC , USA, 2006.

[90] T. Zahn and J. Schiller. Designing structured peer-to-peer overlays as a platform for distributed network applications in mobile ad hoc networks. *Computer Communications*, 31(3):643–654, 2008.

[91] T. Zimmer. QoC: Quality of context - improving the performance of context-aware applications. In Tom Pfeifer et al., editors, *Advances in Pervasive Computing*, volume 207, May 2006.

[92] A. Ziotopoulos and G. de Veciana. Design and optimization of spatial organizations for context exchange and surveillance. In *Proc. of the In-*

ternational Workshop on Context Modeling and Reasoning (CoMoRea) in conjunction with the International Conference on Pervasive Computing, 2009.

[93] A. Ziotopoulos and G. de Veciana. P2P network for storage and query of a spatio-temporal flow of events. In *Proc. of the International Workshop on Mobile Peer-to-Peer (MP2P) in conjunction with the International Conference on Pervasive Computing*, 2011.

[94] A. Ziotopoulos, M. Jacome, and G. de Veciana. An RFID-based platform supporting context-aware computing in complex spaces. In *Second International Workshop on Managing Context Information and Semantics in Mobile Environments (MCISME), in conjunction with the 8th International Conference on Mobile Data Management (MDM'07)*, May 2007.

Vita

Agisilaos Georgios (Ayis) Ziotopoulos was born in 1974. He received his diploma in ECE from the National Technical University of Athens in 1997. In 2000 he received a Master's degree in EE:Systems from the University of Michigan, Ann Arbor. From 2001 to 2004 he worked as a software engineer/consultant for Siemens and Vodafone respectively. In 2004 he joined the University of Texas at Austin as a PhD student in Computer Engineering. During the course of his doctoral studies he had extended internships with AMD and Qualcomm.

Permanent address: 62 Solomou St. 18345 Moschato Greece

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.