From the Institute of Computer Engineering
of the University of Lübeck

Direktor:
Prof. Dr.-Ing. Erik Maehle

Routing and Sensor Search in the Internet of Things

Dissertation for Fulfilment of Requirements
for the Doctoral Degree
of the University of Lübeck

– from the Department of Computer Sciences/Engineering –

Submitted by
M.Sc. Cuong Duc Truong
from Hanoi, Vietnam

Lübeck, January 2014

From the Institute of Computer Engineering
of the University of Lübeck

Direktor:
Prof. Dr.-Ing. Erik Maehle

Routing and Sensor Search in the Internet of Things

Dissertation for Fulfilment of Requirements
for the Doctoral Degree
of the University of Lübeck

– from the Department of Computer Sciences/Engineering –

Submitted by

M.Sc. Cuong Duc Truong
from Hanoi, Vietnam

Lübeck, January 2014

First referee: Prof. Dr. sc. Kay Uwe Römer

Second referee: Prof. Dr. rer. nat. Stefan Fischer

Date of oral examination: 13.01.2014

Approved for printing. Lübeck, den 13.01.2014

Abstract

by M. Sc. Cuong Duc Truong

Institute of Computer Engineering, University of Lübeck

We are witnessing the formation of an Internet of Things (IoT), where real-world entities (e.g., people, plants, cars) augmented with computing devices (e.g., smartphones, tablets, sensor nodes), sensors (e.g., humidity sensors, microphones, cameras), and actuators (e.g., motors, LED) are connected to the Internet, enabling them to publish their generated data on the Web. By mashing up these "Smart Things" with the services and data available on the Web, novel IoT applications can be created.

Two main characteristics of the IoT are its *large scale* interconnecting billions of Smart Things in the next decade, and the *resource limitations* of Smart Things (i.e., of their embedded computing devices). For many IoT applications, *routing* and *sensor search* are essential services. The sensor search service allows for quickly finding Smart Things in the IoT based on the real-world states perceived by their embedded sensors. To facilitate sensor search and also other IoT applications, a routing service is required to enable efficient communication of information among Smart Things and other Internet nodes. Due to the resource limitations of Smart Things, the design of these two services is challenging.

Our thesis is that *despite the large scale of the IoT and the resource limitations of Smart Things, efficient solutions for the routing and sensor search services can actually be provided.* We support this thesis by proposing, implementing, and evaluating two routing algorithms for large-scale wireless sensor networks (which are a building block of the IoT), and two sensor search algorithms for the IoT.

The proposed routing algorithms are *Recursive Multi-region Geocasting* (RMG) and *Stochastic Forwarding-based Routing* (SFR). The RMG algorithm is targeted to large-scale wireless networks where information needs to be delivered from a source to multiple geographical regions that are remotely located from the source, respectively to all Smart Things that are located therein. RMG's approach is to avoid making routing decisions at every intermediate node. Instead, routing decisions and packet duplication are only performed at a few selected nodes on the routing path of a data packet, thus saving processing and energy resources as well as wireless bandwidth. The SFR algorithm aims to balance the energy consumption

caused by the routing task across the network, therefore improving the operational lifetime of the network. Our approach for SFR is to model the route of a data packet as a random walk, such that different packets travel on different routing paths between a given source and destination. The random walk is designed such that the ratio between the average length of the routing paths taken by multiple packets and the length of the shortest routing path (path length overhead) is small, thus saving energy. Evaluation results show the efficiency of our routing algorithms. In particular, (i) RMG minimizes the total number of transmissions needed for the successful delivery of a data packet, and incurs little computation overhead on the network; and (ii) SFR fairly balances the routing load across the network while keeping the path length overhead small. Furthermore, both algorithms are scalable, since routing decisions are made using only local information.

The proposed search algorithms are *sensor similarity search* and *content-based sensor search*. The first algorithm allows for finding sensors whose recent measurements (i.e., perceived states of the real world) are similar to that of an example sensor. The second algorithm allows for finding sensors whose latest measurements fall in a given value range. These two search services are useful because they enable the users to search the real world for objects and places with a given state in real time. Our approach for both algorithms is to encode the properties of the real-world states perceived by sensors using a fuzzy set, and store and index those fuzzy sets in a distributed database system in the Internet, such that they can be used for query resolution. Since the size of a fuzzy set is small, its computation is efficient, and the query resolution using those fuzzy sets is fast, our approach is scalable. Moreover, our approach addresses the inherent imperfections of data generated by sensors as it is based on fuzzy sets. Evaluating our search algorithms using sensor data from real-world deployments, we show the high accuracy of the sensor similarity search algorithm, and the low communication overhead of the content-based sensor search algorithm. We also demonstrate the practical feasibility of our sensor search algorithms by developing an online search engine that allows for finding sensors on the Web based on their published data, using the above sensor search algorithms.

Zusammenfassung

von M. Sc. Cuong Duc Truong
Institut für Technische Informatik, Universität zu Lübeck

Es wird erwartet, dass das Internet der Dinge (IoT) im kommenden Jahrzehnt Milliarden von stark ressourcenbeschränkten smarten Dingen mit dem Internet verbindet. Für viele IoT-Anwendungen sind Routing und Suche grundlegende Dienste. Sensorsuchdienste erlauben das schnelle Finden von smarten Dingen im IoT basierend auf den von ihren Sensoren gemessenen Zuständen der realen Welt. Um die Suche und auch andere IoT-Anwendungen zu ermöglichen, wird ein Routingdienst benötigt der eine effiziente Weiterleitung von Nachrichten zwischen smarten Dingen und anderen Internetknoten erlaubt. Die Ressourcenknappheit der smarten Dinge macht die Realisierung dieser Dienste zu einer wissenschaftlichen Herausforderung.

Unsere These ist, dass es trotz der Größe des IoT und den limitierten Ressourcen der smarten Dinge möglich ist, effiziente Lösungen für das Routing und die Sensorsuche bereitzustellen. Wir unterstützen diese These durch Entwurf, Implementierung und Evaluation zweier Routingalgorithmen für großflächige drahtlose Sensornetze (WSN) sowie zweier Sensorsuchalgorithmen für das IoT.

Die vorgeschlagenen Routingalgorithmen sind Recursive Multi-region Geocasting (RMG) und Stochastic Forwarding-based Routing (SFR). Der RMG-Algorithmus ist auf großflächige WSN zugeschnitten, in denen Nachrichten von einer Quelle zu mehreren entfernten geografischen Regionen bzw. den darin befindlichen smarten Dingen weitergeleitet werden sollen. RMG vermeidet es dabei, auf jedem Zwischenknoten aufwendige Routingberechnungen treffen zu müssen. Stattdessen werden Routingentscheidungen und Paketduplizierungen nur an einigen ausgesuchten Knoten entlang des Routingpfades eines Pakets durchgeführt. Dies spart Rechenaufwand und Energie sowie Bandbreite. Der SFR Algorithmus zielt darauf ab den Energieverbrauch gleichmäßig über die Netzwerkknoten zu verteilen um so die Lebenszeit des Netzwerkes zu optimieren. Dafür wird der Routingpfad eines Pakets als eine Zufallsbewegung modelliert, so dass verschiedene Pakete verschiede Pfade zwischen Quelle und Senke verwenden. Die Zufallsbewegung ist so angelegt, dass das Verhältnis zwischen der durchschnittlichen Länge des Routingpfades von mehreren Paketen und der Länge

des kürzesten Pfades (Pfadlängenoverhead) klein ist. Evaluationen zeigen die Effizienz unserer Routingalgorithmen. Insbeesondere minimiert RMG die Gesamtzahl der benötigten Nachrichtenübertragungen für die erfolgreiche Übermittlung eines Paketes und benötigt nur geringen Rechenoverhead. SFR verteilt die Routinglast gleichmäßig über die Netzwerkknoten während es den Pfadlängenoverhead klein hält. Weiterhin sind beide Algorithmen skalierbar, da Routingentscheidungen nur anhand von lokalen Informationen getroffen werden.

Die vorgeschlagenen Algorithmen für die Sensorsuche sind Sensor Similarity Search (SSS) und Content-based Sensor Search (CSS). Der erste Algorithmus findet Sensoren deren Ausgabezeitreihe ähnlich zu der Ausgabezeitreihe eines vorgegebenen Sensor sind. Der zweite Algorithmus findet Sensoren, deren aktueller Ausgabewert in ein gegebenes Intervall fällt. Diese Algorithmen erlauben es einem Nutzer die reale Welt in Echtzeit nach Objekten und Plätzen mit einem gegebenen Zustand zu durchsuchen. Bei beiden Algorithmen werden die Ausgaben der Sensoren durch Fuzzy-Sets kodiert. Diese Fuzzy-Sets werden in einer verteilten Datenbank im Internet indiziert und verwendet um Suchanfragen effizient zu beantworten. Fuzzy-Sets können effizient berechnet werden, haben einen geringen Speicherbedarf, bilden die Eigenschaften von Sensordaten gut ab und sind damit gut für die Sensorsuche geeignet. Die Evaluation der Sensorsuchalgorithmen auf Basis von realen Sensordatensätzen zeigt die hohe Genauigkeit des SSS-Algorithmusund den geringen Kommunikationsaufwand des CSS-Algorithmus. Wir demonstrieren außerdem die praktische Relevanz der Algorithmen durch die Entwicklung einer Suchmaschine zum Auffinden von im Web publizierten Sensoren auf Basis dieser Algorithmen.

Acknowledgements

Pursuing my PhD degree is perhaps the most challenging task that I have done in my life, which I would have not completed without the support of many people, especially those who have been member of the Institute of Computer Engineering at the University of Lübeck. It has been a great experience to spend three years of my life in there working and sharing many memorable moments of my doctoral journey with them.

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Dr. Kay Uwe Römer, for his support and guidance throughout my research. From him, I have learnt how to do research right in all of its stages, from contemplating the problem to developing the approach to evaluating the solution and to presenting the result. I want to thank Kay for being a supportive adviser to me during my PhD time as well as serving as a role model to me as a junior member of academia. His continued guidance always kept me on the right track and led me to the completion of my PhD degree today.

Special thanks to Dr. Long Le, Prof. Dr. Sahin Albayrak, and Dr. Nuri Kayaoglu of the Distributed Artificial Intelligence Laboratory (DAI Labor) at the Technical University of Berlin, who introduced me to the academic environment and supported me during my first days in Germany. Also of DAI Labor, I would like to thank Dr. Fikret Sivrikaya, Dr. Manzoor Ahmed Khan, and Thomas Geithner for their help during my time at DAI Labor.

I would like to thank Kris Erik Schwerdt for his valuable help with the implementation work. I also want to thank Richard Mietz, Carlo Alberto Boano, Matteo Lasagni, Felix Jonathan Oppermann, and Christian Renner for being my helpful colleagues at work as well as good friends in life. Many valuable scientific discussions that we had during my PhD time really helped me in completing my PhD thesis.

Last but above all, my deepest gratitude goes to my family members who have always been there for me with love and encouragement. This dissertation is dedicated to them. Without their support, I would never be able to pursue my PhD degree till its completion.

Contents

List of Figures

Chapter 1

Introduction

Recent fast advancements in various technological fields including hardware miniaturization, embedded computing, wireless networking, and sensing and actuating allow for augmenting real-world things (e.g., a desk, a car, a plant) with an unique identification (ID) and the capabilities to process information, to sense and respond to the environment, thus making them smart, and for smart things to be able to wirelessly communicate with other smart things. By connecting smart things to the Internet such that they can publish their ID and status (i.e., the real-world states perceived by their embedded sensors) on the Web, an Internet of Things (IoT) is formed. By mashing up smart things with the services and data available on the Web, novel and valuable IoT applications for human users will be created. This thesis is devoted to *routing* and *sensor search* in the IoT, which are two essential services for many IoT applications.

In this inaugural chapter, we offer a brief introduction of the IoT, motivate the need for routing and sensor search in the IoT, and give an overview of the main contributions of our work. We conclude this chapter with the structure of this thesis.

1.1 Motivation

In 1999, Kevin Ashton, co-founder of the MIT Auto-ID Center, initially used the term "Internet of Things" to describe a vision where objects and people in the physical world can be managed and inventorized by computers via RFID (Radio-Frequency IDentification) technology. Since then, this vision was broaden and consolidated over time by the advancements in many different technological fields, among which the major ones are micro-controllers, sensors and actuators, wireless networking, and Web technologies. Although a final definition of the IoT is still subject to debate [1], there is a consensus on the vision of the IoT, which is three-fold:

(i) To give the unanimated things of the physical world (e.g., a cup, a pair of shoes) the ability to gather, process, and act on information, therefore making them "smart".

(ii) To unite the cyber world of computers and information with the physical world we live in by connecting all smart things to the current Internet.

(iii) To enable the intuitive interaction between humans and technology, such that human users are unobstrusively assisted by technology in performing everyday activities.

In the last two decades, embedded computing and hardware miniaturization technologies advanced to a stage where it became possible to pack processing, wireless communication, sensing, and power supply capabilities into a volume size of a cubic centimeter [2] or even a few cubic millimeters [3], creating a miniaturized computing device. Due to their tiny size, these computing devices could be attached to objects (e.g., people, desks, food items), embedded into places (e.g., homes, offices), and dispersed in large quantities into the environment (e.g., forests, farm fields), forming wireless networks of embedded computing devices that can be used as tools for tracking, observing, and influencing the real world. RFID systems and wireless sensor & actuator networks are typical examples of these tools, where objects that are equipped with an RFID tag can be tracked and aspects of the real world can be observed via sensors and controlled via actuators.

In parallel with these technological advancements, developments in the wireless networking field have led to the creation of important wireless communication and identification technologies such as IEEE 802.15.4, BLE (Bluetooth Low Energy), UWB (Ultra-Wide Bandwidth), IPv6, RFID, and NFC (Near-Field Communication). Embedded computing devices, therefore, can be uniquely addressed as well as communicate among each other.

In the Web domain, the Web has undergone two major transformations, from Web 1.0 with the focus on building static Web pages to Web 2.0 with the focus on the online collaboration and sharing among users (e.g., facebook, youtube). The current Web is now moving to the next generation, the Web 3.0 or Semantic Web with key technologies such as RDF (Resource Description Framework), OWL (Web Ontology Language), LOD (Linked Open Data), and SPARQL. The focus of this latest generation is making data and services available on the Web understandable to machines, therefore allowing computer programs to automatically perform certain tasks such as finding, sharing, and combining information on behalf of human users, making information accessible to users more easily and intuitively.

The vision of the IoT emerged from the combination of all these technological advancements. By connecting real-world objects, places, and environments (i.e., their embedded computing devices) to the Internet and allowing them to publish and exchange information on the Web, we will be able to browse as well as mash up the physical world with information and services available on the Web, much like how we browse the current Web, to create novel IoT applications.

The creation of these mashed-up IoT applications often requires the discovery of objects and places with certain properties in the physical world. For example, a car driver wants to find in his proximity a *parking spot* that is currently *empty*, or an ethologist wants to find *individual animals* that have *living habits similar* to a particular one in study. The underlying service in these applications is *search for embedded sensors in the IoT whose sensor measurements match the given properties*. This is one of the two foci of our thesis.

The second focus of our thesis is *routing*, which is an essential service in the emerging IoT, since it enables efficient communication of information among IoT entities, i.e., smart things, Internet nodes, Web services and applications, thus facilitating IoT applications including the sensor search service.

1.2 Contributions

This thesis is devoted to routing and sensor search in the IoT. The main challenges to the provision of these services are the large scale of the IoT and the resource limitations of smart things (i.e., of their embedded computing devices). The scale of the IoT is unprecedentedly large both in terms of the number of computing devices that will be connected to it, which is anticipated to be billions in the next decade, and the geographic area that it will cover, which is expected to be the entire earth. The resource limitations of embedded computing devices are due to their small physical size, which is required in order for them to be embedded into smart things. Routing and sensor search algorithms designed for the IoT, therefore, should be scalable and light-weight.

A main contribution of this thesis is two novel routing algorithms designed for large-scale wireless sensor networks (a building block of the IoT). The novelty of our proposed routing algorithms, namely *resursive multi-region geocasting* and *stochastic forwarding-based routing*, is three-fold: (i) their simple design makes them suitable for resource-constrained embedded computing devices; (ii) they require only local information for operation, thus scale with the size of the IoT; and (iii) they are aimed at IoT-specific routing scenarios. We also implement and evaluate our algorithms using simulation as well as discuss their strengths and weaknesses based on evaluation results.

Another main contribution of this thesis is the proposal of two original sensor search services for the IoT, namely *sensor similarity search* and *content-based sensor search*. State-of-the-art search systems that allow for finding sensors available on the Web are based on either the meta-information of the sensors (i.e., their deployment context, location, and type) or the high-level states derived from raw sensor output (e.g., the state of an occupancy sensor embedded in a room is “free” or “occupied” corresponds to the sensor measurements being lower or higher than a certain threshold, respectively). The limitation of the former is that meta-information of sensors is usually provided by human users who tend to make mistakes when entering information or use different terms when describing the same concept,

thus might lower the reliability of the search results. The limitation of the latter is that it requires proper domain expertise to derive high-level states from raw sensor output as well as confines the search to specific applications. Our proposed sensor search services are aimed at addressing these limitations by allowing for finding sensors in the IoT based on their output. In particular, the sensor similarity search service finds sensors based on a time series of measurements of a given example sensor, and the content-based sensor search service finds sensors whose recent measurements fall in a given value range. For each search service, we propose an algorithm, implement it, and evaluate it using sensor data sets from real deployments.

We also built a sensor search engine for the IoT based on our proposed sensor search algorithms, which allows users to search the current Web for sensors (i.e., smart things). We use this sensor search engine to demonstrate the practical feasibility of the proposed sensor search services.

The contributions of this thesis have also been published in various computer science conferences, most notably in [4], [5], [6], [7]. In particular, Chapter 3 is based on [4] and [5][1], and Chapter 4 is based on [6] and [7], respectively.

1.3 Structure

This thesis is divided into 6 chapters with Chapter 1 being this introductory chapter. Chapter 2 serves as the background of this thesis as it gives a detailed introduction of the IoT by discussing different aspects of the definition of the IoT and a review of its possible applications. In Chapter 2 we also present our view on the IoT as we study it, in which the positions of the two services of routing and sensor search are identified and the relationship between them is demonstrated. Finally, we present the underlying technologies that enable the existence of the IoT, and identify a set of technological challenges that come along with it.

The main contributions of this thesis are presented in Chapter 3 and 4. Chapter 3 is devoted to the routing service in the IoT. We first discuss the general routing problem and identify the main challenges introduced by the IoT to the routing service. We, then, present a design space for designing routing protocols in the IoT. After that, we focus our discussion on the geographic routing approach and its forwarding techniques, since this is the general approach that our proposed routing algorithms follow. The main advantage of this approach is that it requires only local information for making routing decisions thus it can scale with the size of the IoT. Our two routing algorithms are presented thereafter in two separate sections. The structure of these two sections is very similar. We first motivate the need for our routing

[1] Our contributions in [5] are the idea of modeling the routing process on an absorbing Markov chain, the analytical framework, and the heuristic.

algorithms. We then discuss state-of-the-art algorithms and show the new contributions of our algorithms. Finally, we present and evaluate our algorithms.

Chapter 4 is devoted to sensor search in the IoT. We first present a set of assumptions for our new sensor search services and draw from it a set of requirements for the design of our sensor search algorithms. Based on these assumptions and requirements, we lay out a generic and scalable architecture for both the sensor search services, and outline a generic approach for designing sensor search algorithms for them. After that, we present the proposed sensor search algorithms in two separate sections. The general structure of the sections is as follow. We motivate the need for our new sensor search services in the IoT, review the literature, formulate the underlying problem, present and evaluate our solutions, and finally give conclusions.

In Chapter 5, we show the practical feasibility of our sensor search algorithms by describing in details an online sensor search engine that we implemented and that incorporates the proposed sensor search algorithms for searching the current Web for available sensors.

We conclude this thesis in Chapter 6 by summarizing the results, discussing limitations, and providing an outlook on future work.

Chapter 2

The Internet of Things

"Any sufficiently advanced technology is indistinguishable from magic.". It was so formulated in the 1973 book titled "Profile of the future" by Arthur C. Clarke, who is one of the world's most famous science fiction writers along with Isaac Asimov and Robert A. Heinlein. The statement is well-known as Clarke's third law, that expresses Clarke's faith in the power of technology to be able to blur the boundary between science fiction and reality and to create what was in the past only considered as magic. Fourty years on, his vision has proved accurate as we are witnessing today how profoundly technologies have been influencing our everyday life. Handhelds and wearable computers (e.g, mobile phones, body sensor networks), human-to-machine conversation (e.g., voice assistant), GPS-based navigation in our pockets, smart technologies (e.g., smart phone, smart watch, smart home/office), and augmented reality (e.g., Google Glass[1]) are few examples of technologies that were exotic or even science fiction in 1970s, are now viable commercial products.

Having similar faith in the power of technology, contemporary visionaries envisioned a world where technology is aware of human users and their activities, and uses this awareness to assist them to accomplish the activities in an unobstrusive fashion, i.e., without the users noticing its assistance. This vision has been described using different terms, among which the most popular are *Ubiquitous Computing* by Xerox PARC, *Pervasive Computing* by IBM, *Ambient Intelligence* by Philips, *Wireless Sensor Networks* (WSN), and the *Internet of Things* (IoT). While the first three terms and their underlying visions are focused on *what* to bring to the users, the last two are focused on the *means* to realize the "what". In particular, technology should be helpful to the users anywhere (e.g., at home, at work, or on the road), at any time (e.g., day or night), for any purpose (e.g., business or leisure), and not distracting. To realize that, technology should be based on *networking*, should be able to *sense* (e.g., via sensors) and *respond* (e.g., via actuators) to the environment, and should thus make the environment *smart*.

[1] http://www.google.com/glass/start/

The IoT is considered as the ultimate means to realize this vision. Through the incorporation of the Internet, wireless networking (e.g., WSN, cellular), hardware miniaturization (e.g., embedded systems), sensors and actuators, and software technologies, the IoT provides a technological foundation for enabling a smart environment at a global scale.

In this chapter, we study the IoT in detail since a comprehlensive understanding of the IoT serves as the background for our thesis. We discuss the definition of the IoT and review its applications in Sec. 2.1 and Sec. 2.2, respectively. In Sec. 2.3, we propose a layered view on the IoT and point out where routing and searching services are placed in this view as well as how the two services are related. In Sec. 2.4, the driving technologies of the IoT that are relevant to this layered view are presented. In Sec. 2.5, we identify several challenges that need to be addressed for a realization of the IoT. Finally, we conclude this chapter in Sec. 2.6.

2.1 Definition

The Internet of Things, as given syntactically by its name, is composed of two terms: "Internet" and "Things". The first term describes a *networking-oriented* aspect of the IoT where the Internet serves as the central building block interconnecting every possible computing device in the world. This aspect is explicitly reflected in the definition for IoT by DG-CONNECT (formerly INFSO) as "a world-wide network of interconnected objects uniquely addressable, based on standard communication protocols" [8].

In the above definition, DG-CONNECT does not refer to "computing devices" but "objects" as the entities being interconnected. This brings us to the second term of the IoT, the "Things" term that describes literally everything that is addressable and communicable, will be connected. This *thing-oriented* aspect of the IoT is further elaborated in the IoT's definition by IERC - the Cluster of European Research Projects on IoT: "a dynamic global network infrastructure with self-configuring capabilities based on standard and inter-operable communication protocols where physical and virtual 'things' have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network". Thus, an object or a thing, according to this definition, can be understood as an entity that is augmented with computing and communication capabilities, thus is able to possess a certain degree of intelligence and interacts with other things that are also connected to a global information network (i.e., the existing Internet).

Other definitions for the IoT such as by [9], or [10] also revolve around these two original aspects of the IoT. However, to appreciate the true value of the IoT, understanding its separate aspects is not enough. As philosopher Aristole once said: "The whole is more than the sum of its parts", there is a "hidden" aspect of the IoT. With every object in the world being interconnected and exchanging information, there will be an enormous amount of heterogeneous data generated by the IoT. How to efficiently manage and correctly extract

valuable information and knowledge from this data to create valuable applications for human users is challenging. Thus, the third aspect of the IoT is about managing and exploiting the data generated by the IoT.

Although the definition for the IoT varies among organizations and authors, these three aspects of the IoT are widely accepted in literature, e.g., [11], [12], or [9]. In the following, we will discuss them in detail.

2.1.1 Making Things Smart

In order for a smart environment to be aware of the users and adapt to their activities, things that constitute the environment should be "smart" to a certain extent. An example of a smart environment is a "smart store" that can automatically regulate the room condition such as air temperature, humidity level, and light level throughout the day, and that can adaptively assist customers while they are engaging with the shopping, e.g., to show a customer the features of a specific item as he is looking at it, to update an info-screen with information about new discounts that fit the customer's preferences as he walks near it, and to complete the payment for the selected items as the customer walks through the exit door. In order for such regulation and assistance to be possible, the store's air-conditioner should be able to measure the room temperature and humidity level, the window blind and the lighting system should be able to adjust themselves to the current room luminance, an item should be able to detect the presence of the customer, an info-screen should be able to recognize the customer and know his preferences, and the exit door should be able to identify the customer's selected items. In brief, all these "Things", i.e., air-conditioner, window blinds, items, info-screens, and exit door, should be able to make a decision of some kind, therefore possessing a certain degree of smartness.

One aspect of the smartness of a thing is that it can *process* information, which can be realized by augmenting the thing with computation capability. The fast advancement of microcontroller and hardware miniaturization technologies in recent years makes it possible for a computing system to be fitted in a volume size of a match box (e.g., Motes[2]), a cubic centimeter (e.g., Egrains[3]), or even a few cubic millimeter (e.g., SmartDust [3]). Due to their small size, such computing systems can be embedded in things (e.g., air-conditioners, info-screens) to perform dedicated functions (e.g., measuring and regulating room temperature). Typically, the heart of these computing systems is a small computer on a single integrated circuit, also called a microcontroller, that contains a processor core, memory units, and programmable input and output peripherals. Examples of microcontrollers are Atmel AVR[4], MSP430[5], or PIC[6].

[2] http://gyro.xbow.com/Products/Wireless_Sensor_Networks.htm
[3] http://www.sopro-projekt.de/index.php?id=ueberblick
[4] http://www.atmel.com/products/microcontrollers/default.aspx
[5] http://www.ti.com/lsds/ti/microcontroller/16-bit_msp430/overview.page?DCMP=MCU_other&HQS=msp430
[6] http://www.microchip.com/pagehandler/en-us/products/picmicrocontrollers

A thing can also be augmented with "remote" computation capability, i.e., the computation capability is not directly embedded in the thing but is placed somewhere outside of it. This is the case for most RFID tags, where a thing is equipped with an RFID tag that uniquely labels the thing, such that a function dedicated for the thing is executed at a remote computing system whenever the tag is detected. In the above example, an info-screen knows the preferences of a customer because there is an entry about the customer in the database of the store, with his RFID tag's number as primary key. Thus, when the customer walks near the info-screen, its RFID reader detects his RFID tag and triggers a dedicated program for the customer. In fact, the very first vision of the IoT orgininated from such usage of RFID technology, as Keven Ashton, co-founder of the MIT Auto-ID, initially used the term "Internet of Things" to describe a vision where RFID-augmented real-world objects are indentified, tracked, and managed via RFID systems.

However, RFID is not the only identification means for realizing remote computation capability of things. Alternative to the EPCGlobal[7], a major driver of the RFID technology, that promotes the use of the Electronic Product Code (EPC) as the unique identification for RFID-augmented things, the uIDCenter[8] proposes to uniquely identify things with an *uCode* stored in the *uCode tag* that is attached to things. Another alternative is the NFC (Near-Field Communication) technology that allows an NFC-enabled device to perform a dedicated function for a thing when it detects the thing's NFC tag. For example, a customer can select an item in the store by hovering his NFC-enabled smart phone over the item, which is equipped with an NFC tag.

Another aspect of the smartness of a thing is that it can *sense* and *respond* to the real world, which can be realized via sensor and actuator technologies. Usually a sensor maps a certain physical quantity (e.g., humidity level, temperature) to an analog signal such as a variable voltage. This analog signal is, then, mapped by an ADC (Analog-to-Digital Converter) to a digital number that can be stored and processed by a computing system (e.g., a microcontroller). Similarly, an actuator system (e.g., a LED) maps a digital/analog input to a certain action (e.g., turn on/off). In the above example, an item can show a customer its features, because it is augmented with an infrared sensor, that detects an abrupt change in temperature at the customer's standing location using infrared radiation level. This detection triggers an info-screen connected to the item to display necessary information. To regulate the room luminance, the embedded servo of the window blind accepts a steering angle from the embedded microcontroller and steers the blinds accordingly. Thus, sensor and actuator technologies enable things to gather information about a wide range of physical properties (e.g., temperature, light, sound, etc), and to respond to this information accordingly, making things smart.

In summary, a "Thing" in the IoT is defined as any real-world entity that is uniquely addressed and augmented with computation, sensing, and actuating capabilities, such that it

[7] http://www.epcglobalinc.org
[8] http://www.uidcenter.org

can can process and generate information, and can sense and respond to the stimuli from the environment. We refer to the augmented part of a thing, i.e., an RFID/NFC tag or a sensor/actuator device, as an *IoT device*. We call a thing with its augmented part as a whole a *smart thing*, or a *Thing* for short. In this thesis, we use the terms smart thing, Thing, and IoT device interchangeably depending on the specific context to highlight the concept that we want to discuss.

2.1.2 Networking Smart Things

The smartness of Things is just a necessary condition for realizing a smart environment. In order to create valuable applications from the collaboration between Things, they need to be interconnected so that they are able to communicate with each other for exchanging information and behaving in harmony. Moreover, this interconnection between Things needs to be at a global scale for the realization of a global smart environment. The second aspect of the IoT, thus, is about connecting Things together as well as connecting Things to the Internet. This aspect is reflected in the ITU's vision of the IoT, that is "from anytime, anyplace connectivity for anyone, we will now have connectivity for *anything*" [10]. The aspect is also reflected at a more technical view in [8] by DG-CONNECT where the IoT means "a world-wide network of interconnected objects uniquely addressable, based on standard communication protocols", and in [13] by the IPSO Alliance where the specific choice for interconnecting Things is the existing Internet via the extension of the IP stack.

Interconnecting all Things together is challenging due to the huge number of heterogenous Things involved in the process. According to [12], the number of interconnected devices on the planet already overtook the actual number of people. There are currently 9 billion devices and it is expected that 24 billion devices will be interconnected by 2020. A DG-CONNECT report [14] states similarly that the traffic generated and received by humans will be dwarfed by the networking traffic generated by everyday objects. To facilitate this huge number of Things exchanging information, *the underlying networking technology must be scalable.*

The heterogeneity of Things comes from the fact that objects are equipped with different communication technologies (e.g., RFID, Bluetooth, NFC, IEEE 802.15.4, WiFi), hardware platforms (e.g., Motes, Egrain, WISP), and operating systems (e.g., TinyOS, Contiki). The difference between hardware specifications and data formats prevents Things to easily "talk" with each other for collaboration. *The underlying networking technology, thus, must be interoperable and should be standardized.*

A major solution to address the above mentioned issues has been proposed by the IPSO Alliance, that is to wisely re-use the existing IP architecture for IEEE 802.15.4-based devices, since most commercial wireless sensor solutions today are based on the IEEE 802.15.4 standard. According to the IPSO Alliance's white paper [15], IP has proven to be a lightweight, stable, highly scalable communication technology that runs on tiny, battery operated embedded devices, and that already connects billions of devices. "IP therefore has all the qualities

to make the IoT a reality". This proposal is inspired from the work of the IETF's 6LoWPAN working group which has been standardizing IP protocols for sending IPv6 packets over IEEE 802.15.4 based networks, thus enabling seamless integration of sensor nodes or IEEE 802.15.4-based devices into the Internet. A similar effort for seamlessly integrating Things into the Internet is *Internet-0* [16], which promotes routing of IP packets over any media. The authors proposed a general-purpose physical layer that can operate on many media. Their design is based on representing *Internet-0* packets as Manchester codes that can be directly sent over any media, for example as electrical pulses, optical flashs, acoustical clicks, electromagnical waves, or even printed on paper. With this design, things can easily communicate with each other in spite of the different physical media they are operating on, as long as they understand the IP protocol. The common aspect of the IPSO and *Internet-0* approaches is the use of a simplified IP stack for the deployment of the IoT, in which Things are uniquely identified by an IPv6 address, and thus are reachable from anywhere, in an end-to-end fashion.

Besides the fundamental approach of reusing the IP stack, the most common alternative is based on gateways/proxies to connect proprietary networks of Things to the Internet. An end-to-end communication session between an Internet node and a local, proprietary IoT device behind a gateway is possible through the gateway intercepting traffic from the Internet node, converting the data to the proprietary format, sending the formatted data to the particular IoT device, receiving response from the IoT device, encapsulating the information inside the response in IP packets, and sending the packets back to the Internet node. A typical example of this approach is wireless sensor networks (WSN), which are deployed into the environment to perform monitoring tasks via sensors sensing the environment and reporting their sensed data through multihop communication among sensor nodes. This data is accessible to an Internet node via a sensor gateway that acts as a communication bridge between sensor nodes and the Internet node.

Regardless of what approach is used for interconnecting Things, the Internet serves as the central building block of the IoT, which interconnects all Things. Much like how the Internet has revolutionized our society with the interconnection of people, the next revolution will be the interconnection of Things via the Internet, or the Internet of Things.

2.1.3 Managing Data from Smart Things

This aspect of the IoT refers to the management and exploition of the data generated by the IoT, which is very challenging given the massive amounts of information being exchanged in real time by the huge number of Things in the IoT, that are interconnected at a global scale through the Internet. Furthermore, these data are usually heterogenous and imperfect as they come from different sources, are sampled by low-cost hardware, and are transmitted over unreliable communication links. In order to enable a seamless and effective integration of this large-volume, dynamically changing, heterogenous, imperfect, and distributed data

into valuable services and applications, frameworks are required to clean and preprocess the data, to provide data analytics in real time and in a distributed fashion, to represent the data in a standardized way so that it can be easily reused, to index and search the data based on its description or content, and to provide scalable data storage as well as management.

A number of approaches for data cleaning are proposed in [17] and [18] for sensor data, based on sensor recalibration or uncertainty modeling techniques, and in [19] for RFID data which surveys an array of techniques for reducing redundant as well as preventing lost readings by RFID readers. For distributed and real-time data analytic, Google's *MapReduce* [20] framework and its prime implementation, *Hadoop* [21], is a promising solution. Designed for solving parallelizable problems across huge data sets using a large number of computers from heterogenous hardware and distributed across geographical areas and/or administrative organizations, the MapReduce framework is ideal for analyzing data generated by the IoT. Building on top of the Hadoop implementation, systems such as HBase[9] and Pig[10] provide database-like functionalities for scalable data storage and management.

Approaches for searching data in the IoT have been mainly based on the textual description of the data sources (sensors), the similarity of streams of data, and the actual data content. Representatives of the approach based on textual description are Snoogle [22], Microsearch [23], MAX [24], GSN [25], and SenseWeb [26], that allow the user to find data sources by posing a query that contains a list of keywords such as "room" or "book". As the result, the user is presented a rank list of data sources that match the query. The work in [27], [28], [29], and [6] allows finding data sources that produce data streams similar to a given data stream. The work in [30] allows searching for data based on its content.

Data generated by the IoT is large in volume and yet contains too little knowledge about the data sources. Without a clear description of what is available for processing, it is hard for data consumers to use this data effectively. The Sensor Web Enablement[11] standards enable the interoperable usage of sensor resources by allowing "developers to make all types of sensors, transducers and sensor data repositories discoverable, accessible and useable via the Web". Service interfaces such as sensor discovery, access, tasking, eventing, and alerting [31] are standardized so that the heterogeneity of sensor data and sensor networks can be hidden from the perspective of developers. Armed with semantic description of data, developers can easily use and re-use data for building different applications and services. Furthermore, the Semantic Web offers the Resource Description Framework[12] (RDF) to describe semantic information of data using knowledge graphs (e.g., sensor1 is-in room1), where so-called ontologies define the meaning of the vertices (sensor1 and room1) and edges (is-in) of this graph. With RDF, we are able to unambiguously describe a data resource, to specify how data resources are related, and to infer knowledge from data. RDF, therefore, has the potential to effectively represent the huge volumes of data generated by the IoT. The integration

[9] http://hbase.apache.org/
[10] http://pig.apache.org/
[11] http://www.opengeospatial.org/ogc/markets-technologies/swe
[12] http://www.w3.org/RDF/

of RDF-represented data into the linking open data project (LOD)[13], and the use of the SPARQL[14] language, a query language for RDF-based documents, provide a promising set of solutions for efficiently storing, manipulating, and reusing data generated by Things.

2.2 Applications

The technological infrastructure provided by the IoT allows for creating and deploying many novel applications that will improve the quality of our life. In this section, we explore these IoT-enabled applications, which can be grouped into the following application domains:

- Comfort living
- Healthcare
- Automotive
- Security
- Energy saving
- Supply chain

Note that, the IoT vision promises many more IoT-enabled applications across diverse domains, of which only a few are currently available to the public. Thus, we only review here domains and applications that are mostly mentioned in the literature, such that the readers might get an idea about the potential benefits that the IoT may bring about. In the following, we will discuss each application domain in a subsection, present representative applications therein, and outline some possible futuristic applications at the end of the subsection.

2.2.1 Comfort Living

As envisioned by famous visionaries, we will live in smart environments where our living space is filled with sensing, computing, communication, and actuating devices that collaborate with each other in an unobstrusive manner to support our everyday activities. The IoT will be a key to a comfortable life.

2.2.1.1 Home and Office

RFID tags, sensors, and actuators embedded in homes and offices, together with the Internet, can make our life more comfortable in many ways: garage and house doors can automatically open up based on RFID communication; rooms' air condition is automatically regulated to

[13] http://linkeddata.org/
[14] http://www.w3.org/TR/sparql11-federated-query/

our presence, preferences, and to the current weather forecast; home/office can be remotely observed and controlled via smart phone; electronic furnitures (e.g., TV, stereo player) learn our preferences to proactively search the Internet for content; room lighting changes according to the time of the day; and many more. A survey on smart home/office applications can be found in [32], [33], and [34]. There are research projects that already aim at realizing some of these applications. For example, the MavHome [35] and iDorm [36] systems learn and adapt to inhabitant behaviours based on sensor observation and power line control. The Gator Tech Smart House [37] strives to be a so-called "programmable pervasive space" that exposes sensors' and actuators' functionalites as an API for developers to build and deploy powerful applications to users. And in [38], an experiment on real deployment of wireless sensor networks for smart home/office environment is conducted and evaluated.

2.2.1.2 Travelling

The IoT technologies will also deliver a comfortable experience for people on the move. Information about transportation services (e.g., costs, schedules) can be encoded in NFC tags that are attached to markers, posters, and panels. The users can retrieve these information by simply hovering their smartphone over, e.g., a marker, and decide to buy tickets with the smartphone [39]. Touristic maps can also be equipped with NFC tags so that NFC-enabled smartphones can browse them to display touristic information such as hotels, restaurants, points of interest, comming events, etc, for the users [40]. Furthermore, interactive context menu to assist the users in browsing more detailed information can be triggered using NFC tags [41].

2.2.1.3 Shopping

During shopping, RFID and NFC will offer many comfortable applications for the customers. By hovering their NFC-enabled smartphone over NFC-equipped items, customers are informed about allergic information because the smartphone knows what substances its owner is allgeric to. Customers are guided in the shop according to a preselected shopping list (e.g., via interactive touch screens) [42]. Customers can purchase or rent items by simply walking out of the store with them, or can return items without a store receipt as the items' RFID tag and the customers' smartphone contain all necessary information so that they can collaborate to accomplish these tasks [43]. When a customer walks into the store, the store's RFID system recognizes her and will trigger automatic actions such as displaying targeted suggestions, providing offers for products she typically purchases or suprise discounts based on her loyalty level [44]. Such a system has already been realized to some extent by a cooperation between ThingMagic[15] and MIT MediaLab[16] to create a presence-based, touch-sensitive information display that senses the presence of a customer and presents products

[15] www.thingmagic.com
[16] http://www.media.mit.edu/

that are likely appealing to her, and lets her browse further information and buy products via touching the interactive context menu on the screen.

2.2.1.4 Futuristic Applications

Future applications will be based on sensory data, sophisticated actuators (e.g., domestic robots), and advanced artificial intelligence software to further improve the quality of our life. In future homes/offices, smart refrigerators will automatically do inventory of their content, monitor and detect expired or recalled food items, and create shopping lists based on RFID and sensory information [43]. A house or an office will be able to monitor and adapt to its inhabitants' emotions and habits to assist them accordingly, e.g., play emotional musics or cook black coffee every morning at 8:00 AM. Movies, television, and meetings will be more interactive than ever before as we will be meeting with our 3-D virtual friends in our living room or office[17]. Domestic robots will collaborate with wireless smart devices (e.g., smart refrigerator) to effectively perform routine tasks such as cleaning or maintenance in an autonomous fashion (e.g., clean and refill the refrigerator).

2.2.2 Healthcare

There will be many IoT applications in the healthcare domain, many of which can be further grouped into the following sub-domains.

2.2.2.1 Smart Monitoring

The combination of sensor- and RFID-based technologies will allow accurate monitoring of vital life functions such as blood pressure, heart rate, liver enzymes, cholesterol and glucose levels in real time while also maintaining best convenience for patients. For example, the US's Food and Drug Administration (FDA) and Federal Communications Commission (FCC) agencies organized a joint meeting on July 26-27, 2010, in Washington DC to discuss how the convergence of the ubiquitous broadband cellular wireless technologies and the implantable wireless sensors could be applied to real-time ambulatory patient monitoring. The Internet and real-time monitored data, i.e., current patient conditions and health records provided by worn sensors and implanted RFID tags, have the potential to remotely monitor patient vital signs and alert physicians about possible emergency situations. A prototype of such monitoring system is presented in [45], that involves dedicated sensors for patient identification (e.g., unique ID number) and medical sensors such as electrocardiogram, oxygen blood saturation, and non-invasive blood pressure. Another prototype is the "magic carpet" developed by researchers at GE and Intel, which uses sensors to monitor and detect erratic movements of ederly people at home thus can predict and detect falls. The mHealth

[17] http://www.futuretechnology500.com/index.php/future-homes/

Alliance[18] promotes using mobile devices such as cell phones, PDAs, and other wireless technologies to collect and transmit health-related data between medical personnels to improve health throughout the world (e.g., via better monitoring of patients). A brief introduction of several already-deployed monitoring systems based on the "The Internet of Medical Things" is given in [46].

2.2.2.2 Smart Assistance

Through reliable broadband and high-speed communication, abundant sensory data, and robotics, telemedecine activities such as telesurgery and telepresence will be significantly improved. Discussions on how IoT technologies improve telemendecine can be found in [47] and [48]. Telesurgery is a technique that allows a surgeon to remotely operate on a patient via robotic and visual means while being at a considerable distance from the operating table. In [49], a telesurgery system is enhanced with accelerometer sensors for measuring acceleration in 3 axes of a moving space (e.g., a van, an elevator). The measurements are used to compensate for any unintended master and slave manipulator motion resulting from accelerations of the moving space within which the surgery is performed, thus enabling mobile telesurgery. The *Telelap Alf-x* [50] prototype initiated by EU Comission and SOFAR S.p.A focuses on providing haptic sensation feedback, adaptive 3D image processing and tracking, and intelligent system setup and docking to realize next generation telesurgery systems.

Telepresence refers to providing physicians with the ability to easily navigate around in a remote hospital as if they were physically at the hospital via a kind of surrogate, thus allowing them to visit patients at bedside to provide medical care remotely. The RP-VITA medical robot is such a surrogate, which has been recently approved by the US's FDA for use in hospitals. An RP-VITA medical robot is a combination of the iRobot's AVA telepresence[19], and the InTouch Health's remote presence[20] technologies.

For further smart assistance applications, a common architecture for providing elderly people with smart assistance at home is proposed in [51]. The "activity-aware computing" mechanism proposed in [52] uses smart environment infrastructure (i.e., sensors, RFID, actuators, wireless and mobile networks) to automatically recognize and inventorize hostpital activities (e.g., patient care, clinical case assessment), thus effectively helping hospital staff to associate resources and services with activities via their smartphone. Another application is the wearable, headband-like device called Muse[21] developed by Interaxon, that can measure human brainwaves in real-time and send the collected data to a smartphone or tablet to be displayed so that users can monitor how well their brain is performing.

[18] http://www.mhealthalliance.org/
[19] http://www.irobot.com/ava/
[20] http://www.intouchhealth.com/products-and-services/products/
[21] http://interaxon.ca/muse/

2.2.2.3 Futuristic Applications

According to [53] and [54], sensor- and actuator-based assistance systems will monitor elderly people's condition at their own home, raise alarms if necessary (e.g., likely to fall down, heart rate exceeds certain threshold), and provide medical care if required (e.g., via robot doctor), thus saving them from being moved into old people's homes. Microchips will be developed to be injected into human body, and use body fluid (e.g., blood) to continuously perform many diagnosis (e.g., inflamation, early tumor cells).

2.2.3 Automotive

The automotive industry has long been using embedded systems to improve consumer experience during the move. The fast advancements of IoT technologies such as sensors and actuators, micro-controllers, and wireless communications will further deepen this usage. Wireless sensor nodes will be placed along roads and rails, and inside moving vehicles (e.g., cars, trains, buses). These embedded systems will exchange data via ad hoc (e.g., MANET, VANET), overlaying (e.g., GSM, WiMAX) networks, and the Internet, to provide real-time traffic information to drivers and passengers for better navigation and safety. Examples of such infrastructures can be found in [55] and several research projects on the newly emerging *Car-to-X* paradigm such as the PATH[22] and simTD[23]. Furthermore, Apple is working with General Motors, Honda, Toyota, and other car-makers to integrate its *Eyes Free* technology onboard[24]. With this integration, a driver can talk to his car to e.g., send an SMS, enter an address for navigation, as well as the car can talk back to the driver, e.g., reads out loud a received SMS or a reminder. The Volvo's V40 car[25] can drive itself in busy traffic without human intervention by automatically maintaining a safe distance to the surrounding vehicles, keeping in lane, and braking when it senses an imminent collision. The *teleservice*[26] feature developed by BMW allows a car to automatically monitor certain functions that require maintenance such as oil and brakes, and contacts the dealer to schedule an appointment when required, so that the dealer will know the full health status of the car before the consumer arrives. All these examples give an indication that vehicles will become a part of the IoT, having their own intelligence, functioning automatically and even autonomously. In [56], a prototype of a completely autonomous vehicle system is given, that, in addition to sensory information provided by embedded sensors, is based on real-time data taken from Internet clouds to plan efficient paths and to avoid obstacles.

[22] http://www.path.berkeley.edu/
[23] http://www.psychologie.uni-wuerzburg.de/izvw/forschung/projekte/fahrerinformation/simtd.php.en
[24] http://www.wired.com/autopia/2012/07/siri-eyes-free-competing/
[25] http://www.economist.com/node/21548992
[26] http://www.bmw.com/com/en/owners/service/bmw_teleservices.html

2.2.4 Security

In the security domain, the term "smart surveillance" refers to the high level of autonomy of surveillance applications in accomplishing their tasks. The use of image processing mechanisms in combination with dedicated video capturing and computation hardware have been shifting the security paradigm from investigation of incidents to prevention of potentially (catastrophic) incidents via automatic decision making of the surveillance systems [57]. The IoT technologies will further contribute to this shift via smart cameras (i.e., cameras that can partly process video streams), sensor networks, and RFID. A detailed review of smart surveillance systems based on smart cameras and sensor networks is given in [58]. With the integration of RFID into surveillance systems, RFID-enabled assets can be monitored by an RFID reader despite that there is no line-of-sight between them and the reader, which is not possible with a camera-based mechanism. Example systems based on this are [59], [60], and SimplyRFID[27].

2.2.4.1 Futuristic Applications

Future systems for the security purpose will be inspired from more sophisticated hardware and intelligent software and their combination in a fast-evolving IoT. A research project funded by DARPA and Carnegie Mellon University aims at developing an artificial intelligence system that is close to human visual intelligence and can watch and predict what a person will "likely" do in the future [61]. Scientists at the University of Illinois introduced a prototype of a tattoo-like patch to be laminated onto human skin, that is able to monitor human nerves and muscles via embedded EEG and EMG sensors [62]. Dispite of its thinness, the patch contains a wide range of hardware components including LEDs, transistors, wireless antennas, sensors, and conductive coils and solar cells for power. A number of DARPA-funded projects such as HIMEM are developing bug-like machines that are equipped with sophisticated sensors (e.g., camera, acoustic) for "hidden" surveillance systems. In the long term, surveillance systems will even be able to read and analyze human brain waves to proactively react to harmful deeds [63]. There may also be parabolic microphones that can pick up conversations a mile away, cameras that learn what and who to photograph, radars that "see through walls"[28], and nano air vehicles (NAVs) resembling drones to fly in swarms by 2030[29].

2.2.5 Energy Saving

Energy saving enabled by IoT technologies refers to systems that use sensory data provided by the sensors embedded into power consuming devices to optimize energy usage.

[27] http://www.simplyrfid.com/nox-core-ata-server/
[28] http://topdocumentaryfilms.com/surveillance-technology/
[29] http://www.activistpost.com/2011/02/how-close-are-we-to-nano-based.html

2.2.5.1 Home, Office, and City

In the home and office sub-domain, the smart metering technology is becoming more popular for measuring energy consumption and transmitting this information to the energy provider via the Internet. This allows energy providers to conveniently provide better services, such as real-time pricing to promote better energy efficiency, instant reporting of detected faults, and more accurate data for profiling usage within their network [64]. Within a house or office, electrical equipment can be automatically switched off when not needed to save energy. This is achieved by an Intelligent Building Automation System (IBAS) that processes sensory data provided by sensors embedded inside the building (e.g., light sensors at windows, motion sensors at doors, pressure sensors at chairs, temperature sensors in rooms), and other cloud data (e.g., weather forecast) to, e.g., switch off lights if room luminance exceeds a certain threshold, or turn off the heater when no person is detected in the room. The SPITFIRE project[30] develops several IBAS use-cases to illustrate such energy saving scenarios. Several studies by KNX[31] show appealing statistics on how much the IBAS technology can save energy [65]. A further survey on research works on IBAS can be found in [66]. Commercial IBASs are also available such as the Siemens's Desigo system [67], and the solutions provided by AVI-SPL[32], and Intelligentibas[33]. Energy usage can also be optimized at city level via sensor networks monitoring and controling light in public street lighting scenarios [68], or via combining smart metering technology with future smart power grids to provide a real-time balance between energy supply and enery demand [69].

2.2.5.2 Futuristic Applications

In an IoT-enhanced future, the status and performance of each urban structure (e.g., sidewalk, bridge, building, railway, bus corridor) of the entire city will be continously monitored and selectively accessible via a series of APIs, thus enabling a live modeling of the entire city, which changes in real time according the live sensory information streams, and is known as City Information Model (CIM) [11]. Based on CIM, urban facility managers and service providers can communicate and collaborate with each other to trade surplus energy, and to calculate prices that match energy supply and demand in real time [70] [71]. Furthermore, with the realization of future smart grids for efficient energy distribution, energy usage optimization across countries will be possible [72].

2.2.6 Supply Chain

Globally identified RFID or NFC tags that contain both current and historical information about the item they are attached to such as its production and expiration date, ownership,

[30] http://spitfire-project.eu/
[31] www.knx-gebaeudesysteme.de
[32] http://www.avispl.com/solutions/intelligent-buildings/
[33] http://www.intelligentbas.com/

states and locations, or destination, make (semi-)automatic, near real-time tracking and management of every link of the supply chain possible, including conceptual design, raw material purchasing, production, warehousing, transportation, distribution, retail, and customer care. Through space and time, arising problems can be taken care of in a timely and economical manner, therefore creating benefit as well as saving money for companies.

Sensor technology can also be combined with RFID to further improve tracking and management of products in the supply chain. Using sensory data we can ensure that a product never was exposed to damaging environmental conditions (e.g., a milk carton has never been stored at a temperature that exceeds 30°C), or in an area that is not along the planned transportation route (e.g., based on a GPS sensor), thus the integrity of the product is guaranteed for end customers. Example systems are [73] and [74] which use sensory information to preserve the quality of perishable goods such as fruits, meat, and dairy products during transportation.

2.2.6.1 Futuristic Applications

In an IoT-enhanced future, smart containers/pallets will automatically and continuously monitor the condition of the goods (e.g., to ensure right temperature, to avoid toxic chemical substances) during transportation, track the transportation route and their actual position to ensure that they are arriving at their destination [42]. Supply chains will be capable of learning and making decisions by themselves, without human involvement. For example, they could reconfigure supply chain networks when disruptions occur, or could acquire rights to use physical assets like production capacity, distribution facilities and transportation fleets on demand through virtual exchanges [75].

2.3 A Layered View on the IoT

Based on the definition of the IoT and its applications, the vision of the IoT can be summarized by the illustration given in Fig. 2.1, which shows four functional layers: the *interaction* layer, the *representation* layer, the *service* layer, and the *application* layer.

The interaction layer consists of Things, i.e., IoT devices such as RFID tags and sensor nodes that are attached to physical entities (e.g., cars, homes, people, animals) or deployed into the environment (e.g., forests, farms, coastlines). Thanks to the wireless communication, computing, sensing, and actuation capabilities of the IoT devices, this layer can directly "interact" with the real world, i.e., monitor (via sensors) and manipulate (via actuators) the state of the real world.

The representation layer resides on top of the interaction layer, and is responsible for enabling the access to the basic functionalities offerred and data generated by the vast numbers of heterogenous IoT devices through a set of standardized methods (e.g., REST interfaces).

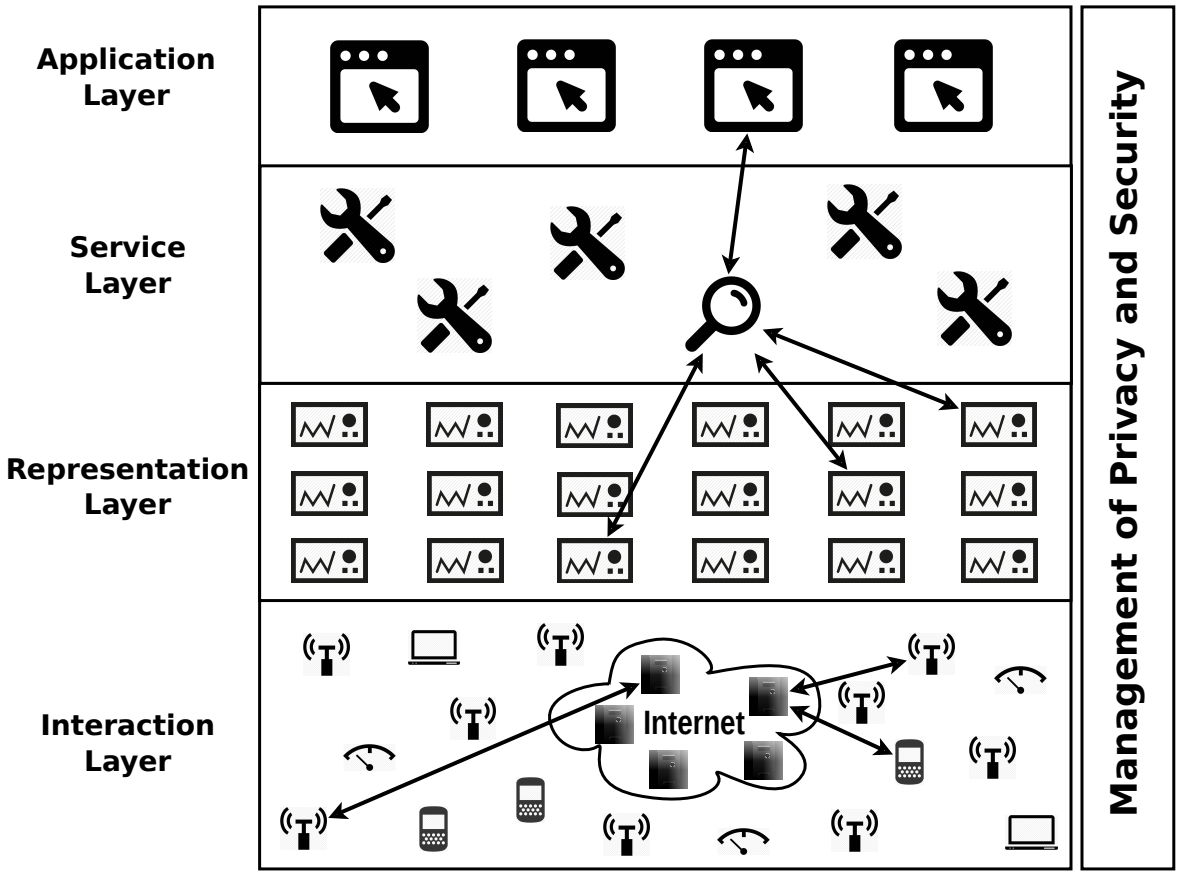


FIGURE 2.1: A layered view on the Internet of Things

Each Thing is given a logical representation that is addressed using a unique identification (e.g., a URI/URL) and is accessible using the standardized methods. This way, generated data and offered functionalities of Things can be directly accessed by human users, applications, and other Things (via their representations), without being hampered by possible heterogeneity issues (e.g., heterogenous sensor data, diverse hardware architecture of IoT devices, and different operating systems that run on them).

Using these data and functionalities, the service layer on top of the representation layer is responsible for building and maintaining composite services that are used by applications developed by programmers and end-users at the application layer.

In this layered view, the *sensor search* service is placed in the service layer and can be considered as a composite service. This placement is because the search service finds Things based on the real-world states perceived by their embedded sensors, thus it requires access to the IoT devices via their representation. The *routing* service is placed in the interaction layer, because communication between entities in the IoT (i.e., Things, services, and applications) is eventually translated to communication between IoT devices, and routing enables this communication. Thus, *routing facilitates sensor search, and also other IoT applications.* More specifically, the sensor search service requires communication with IoT devices in order to verify if they match a certain search query or to request their current status. This

communication is eventually translated to the communication between the computer on which the search service is running and the IoT devices. At this point, the routing service is needed to reliably and efficiently deliver the search query to the destination IoT devices, as well as the response from them to the computer hosting the search service. An illustration of this relationship is given in Fig. 2.1, where the three communication links between a search engine and three representations of three IoT devices are translated into three corresponding routing sessions between the computers that host the search engine in the Internet and three IoT devices. In the following section, we will discuss the most important and relevant IoT technologies in this layered view.

2.4 Driving Technologies

In Sec. 2.2, we have reviewed a wide range of possible applications of the IoT. Among these applications, some have already been put in practice, some are still existing as prototypes, and some others are just speculations but are realizable given the assumption that the technologies supporting them will mature (very quickly) over time. The realization and evolution of the IoT-based applications will, thus, depend on what are the technologies that drive the IoT and how fast they are maturing.

In the following, we will present a survey on the driving technologies that are either most relevant to our thesis or the major building blocks of the IoT. In particular, technologies that enable the operation of the interaction layer in Fig. 2.1 are presented in Sec. 2.4.1, Sec. 2.4.2, Sec. 2.4.3, and Sec. 2.4.4. Among these sections, Sec. 2.4.4 provides a brief discussion of the routing service in the IoT. In Sec. 2.4.5, we present the technologies that enable the operation of the representation and service layers in which the sensor search service for the IoT takes place. To gain an in-depth understanding of a wider range of IoT technologies, we refer the reader to documents such as [76] and the references therein.

2.4.1 Radio-Frequency Identification

Radio Frequency IDentification (RFID) systems consist of 3 parts: an RFID tag/transponder that carries the data, an RFID reader/interrogator that sends a radio signal to an RFID tag and reads the tag's response (the tag's carried data), and an RFID software/middleware that receives and processes information sent from the RFID reader. The data carried by an RFID tag is typically its unique identification (ID) such as EPC, and/or some product-related information such as production date and batch number. In general, RFID tags are composed of two basic components: (i) an integrated circuit for storing and processing information, modulating and demodulating a radio frequency (RF) signal, and optionally for harvesting power from a reader's RF signal; and (ii) an antenna for receiving and transmitting RF signals.

RFID systems operate in the unlicensed radio frequency bands known as ISM (Industrial, Scientific, and Medical), with the common frequency of 125 to 135 KHz in low frequency (LF) band, 13.56 MHz in high frequency (HF) band, 868 to 928 MHz in ultra-high frequency (UHF) band, and 2.45/5.8 GHz in microwave band.

The working principle of an RFID system is as follow. The RFID reader initiates the data transfers by broadcasting a query signal. More than one RFID tags will respond to the reader by sending their stored data (e.g., EPC code). To avoid collisions, two main approaches are used. In a slotted ALOHA [77] based approach, a parameter is included in the query signal that is used by RFID tags to randomly delay their response. In an adaptive binary tree [78] based approach, the RFID reader sequentially broadcasts a string of bits 0 and 1, each time appending one bit to the string. Only RFID tags whose ID match the bit sequence will respond. This way, a reader can effectively discover all RFID tags within its transmission range.

RFID systems are usually classified according to the type of the RFID tags in use, which are either passive or active. A passive tag only transmits signals when triggered by a reader. The transmission is powered by the RF signal generated by the reader. In addition to the basic components, an active tag has an on-board battery to power the transmission of its signals (either periodically or when triggered by a reader), and optionally embedded sensors and actuators for data collection and react to the outside world. When compared with passive tags, active tags are bigger in size, have higher production cost, can be read from greater distance (up to 100 meters), and thus are reserved for costly products e.g., cars, containers. Passive tags are read from much closer distance (up to 3 meters), and mostly reserved for cheaper items such as clothes or wine bottles.

Besides being classified as active or passive, RFID tags can also be classified according to how they store data. There are three storage types: read-write, read-only, and write once read many (WORM). While new data can be added or overwritten on read-write tags, the same is not possible with read-only tags and they only contain the data that is stored in them when they were made. For WORM tags, new data can only be added once and cannot be overwritten thereafter.

There is currently no universally accepted standard for use in RFID systems. However, most RFID technical specifications conform to either the International Standard Organization's (ISO) or the EPCGlobal's standards. In the following we will briefly mention the most commonly used standards. For a detailed description of all standards, we refer the reader to [79] and the ISO14443/15693/18000 sets of standards for RFIDs.

2.4.1.1 EPCGlobal UHF Gen 2

This is the current and mostly used air interface RFID standard of the EPCGlobal. The UHF Gen 2 tag is of type passive WORM and contains an EPC of 96 bits long. An UHF

Gen 2 system operates in the UHF band using a carrier frequency of 860 MHz to 960 MHz depending on local regulations. For querying tags, the reader uses an amplitude-shift keying (ASK) modulation technique (e.g., DSB-ASK, SSB-ASK) and pulse interval encoding (PIE) architecture, and transmits data at rates of 26.7-128 kbit/s. For responding to queries, a tag uses ASK or phase-shift keying (PSK) for modulation and transmits data at rates of 5-640 kbit/s. A slotted ALOHA based protocol is used for collision avoidance.

2.4.1.2 ISO14443/NFC

This standard is also called ISO standard for *proximity cards*, which the near-field communication (NFC) technology is based on. An ISO14443/NFC tag offers a maximum reading range of only up to 10 cm, thus is promising for applications that require high level of security such as electronic payment, building access, or banking activities. The system operates in the HF band using a carrier frequency of 13.56 MHz. The tag uses ASK technique for modulation. Its data transfer rate can be from 106 kbit/s to 847 kbit/s. For querying tags, the reader uses ASK for modulation and the Miller sub-carrier coding algorithm for encoding, and transmits data also at a rate of 106-847 kbit/s. For collision avoidance, an adaptive binary tree protocol is used.

2.4.1.3 ISO15693

This standard is also known as the ISO standard for *vicinity cards* or *smart tags*, which also operate in the HF band using 13.56 KHz carrier frequency as do ISO14443/NFC cards, but offers a much better maximum reading distance (1.5 m VS. 10 cm). For communication to cards, the reader uses ASK modulation and pulse-pause modulation (PPM) coding. The data transmission rate can be either 1.65 kbit/s or 26.48 kbit/s. A card has two ways to communicate with the reader: (i) based on ASK modulation on a 423.75 KHz subcarrier with a data rate of 6.62 kbit/s or 26.48 kbit/s, or (ii) based on switching between a 423.75 KHz subcarrier and a 484.25 KHz subcarrier with a data rate of 6.67 kbit/s or 26.69 kbit/s. Collision avoidance is based on a slotted ALOHA-like algorithm called the slotted terminal adaptive collection (STAC) [80].

2.4.2 Wireless Personal Area Networks

Wireless Personal Area Networks (WPAN) are characterized by low cost, low bit rate, short to medium communication range, and low power consumption [81], which in most cases are networks of IoT devices such as sensor nodes and RFID/NFC tags. As IoT devices are made of diverse hardware, standardized physical signal manipulation (e.g., modulation, coding) and wireless medium access (e.g., scheduling) mechanisms are needed for effective and efficient communication between Things. We summarize in the following a few widely used physical and medium access standards/technologies.

2.4.2.1 IEEE 802.15.4

The IEEE 802.15.4 [82] is a communication standard maintained by the IEEE 802.15 Task Group 4. The standard specifies physical layer and medium access control protocols for use in WPANs. Many industrialized standards such as ZigBee, WirelessHART, and MiWi take IEEE 802.15.4 as the basis and extend it by introducing higher layers that are not specified in IEEE 802.15.4.

The current specification of the 802.15.4 standard features 3 different data rates of 20 kbps, 40 kbps, and 250 kbps operating at the frequencies of 868 MHz, 915 MHz, and 2.4 GHz, respectively. The standard supports power management at physical layer to put the radio on duty cycle (can be inactive up to 99% of the time), thus ensuring low power consumption. Furthermore, spread spectrum techniques are used prior to transmission to alleviate environment noise, interference, and jamming. For medium access control, CSMA/CA based protocols are used.

2.4.2.2 Bluetooth

The Bluetooth physical layer standard [83] was originally created by Ericsson in 1994 and intended to replace RS-232 data cable. It is mainly used in WPANs and Body Area Networks (BANs) which are short range communication, require low power and low cost. The Bluetooth physical layer operates in the unlicense ISM band at 2.4 GHz, using spread spectrum, adaptive frequency hopping, on a set of 79 channels each of which is 1 MHz wide, at a nominal hopping rate of 1600 hops/seconds. Transmission range can be up to 10 meters (e.g., smartphones) or 100 meters (e.g., industrial use cases) with a peek power consumption smaller than 30 mW . For medium access control and data transmission, a TDMA-based polling channel access scheme is used. A Bluetooth master device manages a number of other Bluetooth slave devices to form a single-hop star topology network (also called Piconet). During each time slot, only either the master or one of the slaves can transmit at a particular frequency that is selected according to the adaptive frequency hopping (AFH) technique. AFH reduces inteference between concurrent transmissions of different Piconets that are in the same geographical area.

2.4.2.3 Bluetooth Low Energy

The Bluetooth Low Energy (BLE) [83] was originally named Wibree by Nokia in 2006, but was merged with and renamed under the Bluetooth standard version 4.0 in 2010, thus it can be seen as a simplified version of the classic Bluetooth standard. Compared to classic Bluetooth, BLE has a similar transmission range (approx. 50 meters), a slower data transfer rate (approx. 200 kbit/s), but provides further reduced power consumption and much faster setup time (time for Bluetooth device discovery and connection). Due to these features,

BLE has become an alternative to the NFC standard. Some manufacturers (e.g., Texas Instruments, Nordic Semiconductor) are already developing BLE-complient devices.

2.4.2.4 Ultra Wide Band

The Ultra Wide Band (UWB) [84] wireless communication technology is finding its use in applications requiring high data rate over a short transmission range (e.g., multimedia traffic or wireless USB to replace the USB cable), and high precision ranging (e.g., WSN location and tracking applications). According to the US FCC, UWB signals must occupy a bandwidth greater than 500MHz or a bandwidth at least 20% of the carrier frequency, and the physical layer operates in the frequency range of 3.1 GHz to 10.6 GHz with a limited transmit power of -41dBm/MHz. UWB differs from other conventional narrow band and spread spectrum systems (e.g., ZigBee, Bluetooth, 802.11a/b/g) in that it uses the so-called Impulse Radio technique [85] to transmit baseband pulses of very short duration (less than 1 ns) in a discrete manner, thus enabling very low duty cycle transmission. The advantages are that multipath fading and interference can be significantly reduced due to the very short pulse duration, low power consumption due to low duty cycle, inexpensive production of the radio device due to the pulses being baseband or "carrierless" (no additional carrier modulation is required), and the very wide bandwidth allows very high data rate (e.g., Gigabit/s which is well beyond that of 802.11a/b/g or WiMax) as well as high precision (few centimeters) of ranging measurements (which is ideal for geolocation-related applications).

The Impulse Radio UWB technique is adopted as physical layer in the 802.15.4a standard [86] to enable robust data communication and precision ranging of devices. In the market, there are several UWB devices that have been certified by the WiMedia Alliance (the group behind the UWB technology). They are UWB chipsets from 12 manufacturers including Alereon, Intel, and Staccato.

2.4.3 Wireless Sensor Networks

Together with RFID/NFC, Wireless Sensor Networks (WSN) (which generally use WPAN technologies) is the main building block of the IoT. A WSN typically consists of two components: (i) a (large or very large) number of low-cost, small-size sensor nodes, that can sense and react to the environment, process information, and communicate with each other wirelessly; and (ii) one or more information sinks that receive and process sensory information reported by sensor nodes, as well as control the operation of the WSN. Each sensor node typically consists of a hardware architecture (i.e., microcontroller), a software framework (i.e., operating system), a wireless interface (i.e., radio chip), a sensor board that contains several sensors (e.g., light, humidity), and possibly a set of actuators to react (e.g., to move, to make sounds) to the outside stimuli. Equipped with these capabilities, a WSN can be deployed into the environment to "interact with the physical world", i.e., to provide us with

the actual states of the physical world in (near) real-time via rich sensory information as well as the ability to control the physical world via actuators. Low cost and small size allow sensor nodes to be deployed in large quantity and allow WSNs to operate in an unobtrusive fashion, which contributes greatly to the realization of the IoT. A comprehensive discussion on various WSN technologies, deployments, and applications can be found in [87].

There are many sensor node platforms currently existing in the market[34], which are composed of different kinds of hardware including microcontrollers, wireless radio chips, sensors, actuators, and batteries from various popular manufacturers such as Atmel[35], Microchip[36], and Texsas Instruments[37]. Examples of the representative sensor node platforms are TelosB, Imote2, MicaZ from Crossbow[38], BTnode from Ethz[39], Sun Spot[40] from Oracle, and Smart-Dust [3]. To help the readers get an idea of what enables the operation of these platforms, we briefly present below the components that are commonly found on a typical sensor node platform.

2.4.3.1 Computing Subsystem

The main function of this component is performing any required computations on the sensor node such as controlling the operation of other components (e.g., sensing, actuating, wireless communication) and processing information. The component usually consists of one or more microcontrollers whose clock rate ranges from a few tens of KHz to hundreds of MHz, a flash memory unit for storing program code, and a RAM memory unit for storing sensory as well as run-time data. The size of the memory units ranges from a few KB to a few MB. Popular microcontrollers that are used for sensor nodes are Atmel's AVR, Texas Instruments' MSP430, and Microchip's PIC families.

2.4.3.2 Wireless Communication Subsystem

This component is responsible for the wireless communication among sensor nodes, and between them and a sink. Recently, most wireless radio chips for sensor nodes are compliant to the IEEE 802.15.4 standard and operate in license-free ISM bands (e.g., 868 MHz, 915 MHz, 2.4 GHz). However, other technologies could be used as well such as Bluetooth (e.g., BTnode), optical-based (laser) (e.g., SmartDust), and infrared. Examples of popular radio chips include the Texas Instruments' CC2xxxxx and Atmel's AT86RF2xx families.

[34] http://en.wikipedia.org/wiki/List_of_wireless_sensor_nodes
[35] http://www.atmel.com/products/microcontrollers/default.aspx
[36] http://www.microchip.com/pagehandler/en-us/products/picmicrocontrollers
[37] http://www.ti.com/lsds/ti/microcontroller/16-bit_msp430/overview.page
[38] http://bullseye.xbow.com:81/General_info/companyoverview.aspx
[39] http://www.snm.ethz.ch/snmwiki/Projects/BTnodeRev3
[40] http://www.sunspotworld.com/

2.4.3.3 Input/Output (IO) Interfaces

This component facilitates the communication among different components within a sensor node as well as between the sensor node and external devices. Examples of these interfaces are UART (Asynchronous Receiver/Transmitters), I^2C (Inter-Integrated Circuit), SPI (Serial Peripheral Interface), ADC (Analog-to-Digital Converter), which are directly supported by many digital sensors, and USB (Universal Serial Bus) for connecting external devices to the sensor node.

2.4.3.4 Sensors and Actuators

In order to sense and control the physical world, sensor nodes must be equipped with sensors and actuators. Usually a sensor maps a certain physical quantity (e.g., humidity, temperature) to an analog signal such as voltage or current, and an ADC, which is typically integrated into the microcontroller, converts this analog signal to a digital number. A sensor with a built-in ADC is called digital sensor. Similarly, an actuator (e.g., a LED) can accept a digital or an analog input depending on whether it is supported by an ADC. Sensors/actuators can either be directly integrated on-board or on so-called sensor boards. The latter is more common as sensor boards provide modularity and flexibility (can be removed, replaced, and upgraded), and dedicated functionalites for supporting sensors such as ADC, multiplexers, or even an extra microcontroller. Some examples for sensors are light, humidity, gas, pressure sensors; examples for actuators are LED, servo, speaker, and vibrator.

2.4.3.5 Power Source

Sensor nodes may be powered by batteries or may harvest energy from the environment, e.g., solar radiation, heat, motion, or vibration. The batteries are commonly classified according to the electrochemical material used for the electrodes, including NiCd (nickel-cadmium), NiZn (nickel-zinc), NiMH (nickel-metal hydride), and lithium-ion.

2.4.4 Routing

Routing refers to the process of selecting a path for information to travel on the network from its source to its destination. From an application layer's point of view, the main routing goals for a piece of information are its successful arrival at the destination, its integrity, and its travel time. From the lower layers' point of view, there are many influencing factors to be taken into consideration to achieve these routing goals, especially when information is travelling on networks of IoT devices (e.g., IEEE 802.15.4-based), which are typically characterized as low-cost, low-power, short-range, and lossy-link wireless networks. Energy consumption, next hop selection, link quality, medium access control, physical layer noise and interference are just a few of these factors to name.

At an IoT scale, routing of information generated by Things takes place in two major domains: the Internet and the networks of IoT devices. Routing inside the Internet is mainly based on the IP stack which is mature, well-documented, and has been working well in practice. In the networks of IoT devices (i.e., devices that mostly are IEEE 802.15.4-compliant), standardized protocols are based on the 6LoWPAN standard and can be divided into 2 schemes: mesh-under routing and route-over routing.

In the mesh-under scheme, the 6LoWPAN's adaptation layer is responsible for forwarding packets from a source to a destination over multiple radio hops (a radio interface is considered a next hop of the current radio interface if it is within the transmission range of the current radio interface). Nodes (i.e., radio interfaces) are uniquely identified using either IEEE 16-bit short or IEEE 64-bit address. This way, a mesh network topology is created underneath and unbeknownst to the IPv6 layer (layer 3 or network layer). The IPv6 sees the entire 6LoWPAN network as a single IP link. Thus, multiple link layer (layer 2) hops are used to complete a single IP hop. Once the adaptation layer receives all fragments of an IPv6 packet, it reassembles the packet and sends it to the IPv6 layer. Examples of this routing scheme include the LOAD [88] and DYMO-low [89] protocols that are variants of the well-known AODV [90] routing protocol, with appropriate modifications to operate on networks of IoT devices. For example, they eliminate the use of destination sequence numbers and disable the maintenance of a precursor list (a list of IP addresses of neighbor nodes that are the candidate for the next hop forwarding) at intermediate nodes. Another example of mesh-under routing is the HiLow [91] protocol, that adopts a hierarchical approach to reduce the size of the routing table that each node has to maintain.

In the route-over scheme, the IPv6 layer takes care of the routing decisions. Each node acts as an IP router and each link layer hop is an IP hop. Forwarding of packets, therefore, is performed based on IP routing tables. In particular, the adaptation layer of a forwarding node divides the IPv6 packet into fragments and forwards them to the next IP hop based on the node's IP routing table. Once the adaptation layer of the next IP hop has successfully received these fragments, it reassembles the IPv6 packet from them and sends the packet to the IPv6 layer. If the packet is destined for the node, the IPv6 layer sends the packet to the upper layer, otherwise forwards the packet to the next IP hop in the same forwarding fashion. An example of this routing scheme is the RPL [92] protocol, which is optimized for the *multipoint-to-point* traffic pattern, i.e., data travels from multiple IoT devices to a gateway (e.g., WSN).

Alternative to the standardized routing protocols, there is a vast number of non-standardized routing protocols. Depending on the application scenarios, their design is focused on different (but possibly overlapping) sets of goals. For example, traffic pattern (e.g., multipoint-to-point, point-to-multipoint, point-to-point), packet delivery time, control overhead, location-based, hierarchy-based, etc. Although varying, their design still shares two essential goals of minimizing energy consumption and scaling to the large number of network nodes. A comprehensive survey of these protocols can be found in [93].

2.4.5 The Web of Things

The realization of the IoT is driven by the integration of huge numbers of Things (i.e., real-world objects that are augmented with sensing, computing, communication, and actuating capabilities) into the Internet. As such, sensor, actuator, and RFID devices will play a key role in this integration. We have reviewed in the previous subsections the technologies that enable the operation of and the communication among these types of devices. Effectively exploiting this infrastructure to build novel and valuable IoT applications, however, is challenging.

This is due to the heterogeneity of hardware platforms (i.e., made of different hardware architectures and components although they can provide similar functionalities), software platforms (i.e., different operating systems and programming languages), and ownership (i.e., manufactured and/or owned by different organizations) of IoT devices, which requires application developers to be an expert on heterogenous technological fields. Another reason is that the open and dynamic nature of the IoT makes it difficult for application developers to manually discover IoT devices and the functionalities that they offer, to mash them up to create necessary services for use in developing applications, and to keep track of all IoT devices, functionalities, and services. Obviously, application developers do not want to deal with any of these problems. In fact, they should only care about problems that are directly pertinent to their interest, that is the development of the specific application using IoT infrastructures.

In today's Internet, the Web has become the de facto platform for creating interoperable and platform-independent applications. These applications can be further customized and adapted to build new applications according to user needs, based on the easily reusable and combinable Web services. The Web, therefore, has hidden the heterogeneity of the underlying Internet technologies from the application developers, such that they can focus more on the functionalities of the Web-based applications that they are developing. Given the success of the current Web, it is logical to reuse and extend the existing Web technologies for the IoT, resulting in a Web of Things (WoT). Much like with the current Web, where Internet resources and entities are given a Web representation, the WoT aims at giving Things in the IoT a Web representation that is accessible using Web technologies. For example, a chair augmented with a pressure sensor can be accessed from a Web browser via a URL to query if it is currently being sat on or not. The Web representations of Things, therefore, exist in the representation layer (see Fig. 2.1).

The biggest advantage of the WoT is interoperability. Regardless of the hardware, software, and communication technologies that IoT devices are made from, they can publish their data and expose their functionalities on the Web, as well as consume the data and and use functionalities that are pulished and offered by other Things on the Web, through their Web representation. The data and functionalities published by Things (i.e., originated from the physical world), then, can easily be mashed up with the existing Web data and services (i.e.,

originated from the world of information and computers), to create novel IoT services and applications which exist in the service and application layers (see Fig. 2.1). For example, the sensor search service could be built based on the published output of sensors which is accessible via their Web representation.

Given the IoT, there are two approaches to realize the WoT, i.e., integrating Things into the Web. In the *direct* approach, a tiny Web server is embedded directly in the IoT devices, which supports light-weight Web-level communication interfaces such as the POST and GET methods of the RESTful framework. Functionalities offered by the IoT devices can be wrapped in the payload of these methods so that they can be exposed on the Web. Recent work shows that this approach is promising as tiny Web servers can be built within a size of only a few kilobytes [94] [95] [96], thus can be fit in resource-constrained IoT devices (e.g., sensor nodes). In the *indirect* approach, a gateway (or proxy) is located between the Web and the IoT devices. The gateway is responsible for wrapping the heterogeneous functionalities offered by the IoT devices into standard Web services, such that any Web-based entity can communicate with the IoT devices behind the gateway as if there was a direct Web-level communication link between them. This approach is adopted when either the IoT devices are too resource-constrained (e.g., passive RFID tags) or are not allowed to be directly connected to the Web (e.g., due to privacy, security, or authority reasons). An example of this approach is the implementation of the Smart Service Proxy in the SPITFIRE project[41]. Some other examples are [97] and [98].

Once Things have been integrated into the Web and start publishing data and services, mashed-up IoT applications can be built based on them. However, due to the scale and the dynamicity of the IoT, these data and services will be huge in amount and dynamic in content. Furthermore, the data are heterogenous, unstructured, and unannotated as they come from diverse sources (or Things) and are usually just streams of bytes. Thus, it will be difficult for human users to manually discover and compile them to create applications. The Semantic Web[42] is an effort by the World Wide Web Consortium (W3C) to address this difficulty. The core concept is that information in the Web should be represented as *resources* each of which is identified by an unique URI, and by descriptions that precisely define these resources based on their contents and properties (which also are identified by URIs), values assigned to them (which can be URIs or literal values), and the relationships between them (which are also URIs). This way, Web information has exact meaning and the relationships between Web information are precisely defined, which enables computers to "understand" the Web and to automatically (i.e., without direct support from humans) gather, process, and integrate Web information, thus easing the otherwise difficult manual process of service and data discovery and management in the WoT.

The core technologies that enable the Semantic Web concept are the REST (REpresentational State Transfer) [99] architecture, the RDF (Resource Description Framework), SPARQL

[41] http://spitfire-project.eu/ssp
[42] http://www.w3.org/standards/semanticweb/

(SPARQL Query Language for RDF)[43], and LOD (Linking Open Data)[44]. Among these, RDF is used to define the above interconnected network of resources as a knowledge graph of *triples* of the form $(subject, predicate, object)$ whose vertices (i.e., "subject" and "object") define the contents (which are also resources) of a resource and edges (i.e., "predicate") define the relationships between them. By navigating the knowledge graph, computers can explore and infer knowledge. The LOD project is an attempt to connect distributed data across the Web and present it as a knowledge graph of triples. To build services and applications on top of the Semantic Web framework, SPARQL and REST are used. SPARQL is a query language for expressing queries that are executed over RDF-described data sources (e.g., LOD). The results of a SPARQL query are usually RDF graphs. Typically, SPARQL endpoints, an URI to which SPARQL queries can be sent and RDF graphs are returned, are RESTful applications (i.e., they support REST interfaces). As an overview, REST is a client-server based architecture for designing networked applications, with the core idea of using lightweight HTTP methods (e.g., PUT, POST, GET) as a replacement of complex mechanisms such as RPC (Remote Procedure Call) and CORBA (Common Object Request Broker Architecture) to facilitate invocation of functions on remote computers. As REST operations are independent of platforms (e.g., Linux, Windows), independent of programming languages (e.g., C, Java, Delphi), self-contained (i.e., each REST-based request contains within it all necessary information for the server to complete it), and standardized (using HTTP which the current Web is based on), the architecture has been widely accepted as the "mainstream" mechanism for designing Semantic Web applications.

2.5 Challenges

The IoT promises to reshape the scociety. Althougth the driving technologies discussed in the previous section make the realization of the IoT feasible, more research efforts are still required to overcome challenges at all layers as well as across layers, in order to make the IoT vision a reality. In the following, we present the major challenges to a realization of the IoT.

2.5.1 Small Physical Size

The IoT vision predicts a future where we will be surrounded by smart environments that constantly take care of and unobstrusively assist us in every aspect of our life. We will hardly notice this assistance as it will be seamlessly wowen into our daily activities. In order to realize this future, it is desired that the augmenting devices, i.e., devices that are to be attached to real-world objects to form Things, are as small in physical size as possible. For example, in healthcare applications, body-worn sensors should be as small in physical size as

[43] http://www.w3.org/TR/rdf-sparql-query/
[44] http://linkeddata.org/

possible in order to not interfere with patients' daily activities. Current technologies already help to reduce the size of a sensor node to fit in a cubic centimeter [2] via micro-integration techniques and system-on-chip solutions, or even in a few cubic millimeters [3] via laser-based communication.

2.5.2 Limited Resources

A direct consequence of physical size reduction are limited resources, namely energy, computation, memory, and communication, that can be provided by a single IoT device such as a sensor node or an RFID tag. A passive RFID tag, which is the most popular type of RFID tag, has from 64 bits to 1 KB of non-volatile memory, and an antenna for transceiving RF signals within the typical range of several meters. An RFID active tag has, in addition, an on-board battery, longer tranmission-range antenna (could be up to 100 meters), more memory (rarely but could be up to 128 KB), and possibly interfaces to external sensors (for gathering information about the surrounding environment), and an external processing unit (e.g., microcontroller). In a typical sensor node, the capabilities of the microcontroller are a few MIPS processing power, few kilobytes of program memory, and few hundred kilobytes of general purpose memory [87]. Such limited resources require a high level of optimization and simplification of programs that run on IoT devices. Furthermore, energy saving is of paramount important for IoT devices, since they are usually cheaply manufactured and deployed in large quantities into the environment, and are expected to operate over long periods of time without manual intervention, making replacement of energy supply (i.e., battery) for each IoT device extremely difficult. Thus, processing tasks must be optimized as well. Last but not least, the IoT implies that every IoT devices should be connected to the IoT network infrastructure and wirelessly communicable. Since wireless communication usually dominates the energy consumption of an IoT device, it should be kept to the absolute minimum.

2.5.3 Interoperability

As we discussed in Sec. 2.4.5, the wide range of heterogeneity issues introduced by the IoT hampers the seamless interoperation among different IoT devices. Standardization therefore is a must, but is not enough as no single standard can cover everthing, as well as some organizations (manufacturers, software companies) would like to follow different standards or even proprietary protocols. A solution is to extend IoT devices with multiple adapters, each of which conforms to a specific standard. However, the complexity of this extension would grow quadratically with the number of standards involved, which is inefficient at the level of low-end IoT devices. To mitigate this problem, bridges between standards are introduced. Implementation of bridges is usually in the form of a border gateway or proxy that understands the "languages" of a number of different sets of IoT devices, thus acting as a translator among them. Standard bridges, unfortunately, still do not scale with the number

of standards, and especially, the number of the IoT devices. Therefore, middleware solutions will play an important role of wrapping the functionalities of the underlying heterogenous technological layers into well-defined and well-organized services that can be used for communication among IoT devices, or used by upper layers (e.g., application layer). For example, some works, such as in [100] and those referenced therein, propose the embedding of a Web server directly on the IoT device, making the device accessible to the outside world via Web interfaces. The Semantic Web also has been developing data formats that can be readable by IoT devices and understandable among them (e.g., RDF, SensorML), making their interoperation seamless. Future research efforts, hence, should investigate and elaborate these directions.

2.5.4 Dynamic Topology

One of the main characteristics of the IoT is the high dynamicity of its network topology. There are several reasons for this dynamicity. Firstly, as IoT devices are made of low-power, resource-constrained hardware platforms, systems that run on or functionally based on these platforms should employ at least one mechanism for energy saving, making the IoT devices switching between on and off states. Secondly, IoT devices are connected to and disconnected from the Internet at unknown rate, as doing so is driven by specific user needs, application scenarios, and authority regulations. Thirdly, Things change their locations from time to time (e.g., a car, a person), which affects the underlying network topology as the wireless links between the mobile IoT devices and other IoT devices in their proximity are reconfigured accordingly. Finally, the low-power wireless link is unreliable, asymmetric, and transitional, making the network topology highly dynamic.

2.5.5 Scalability

The size of the IoT is unprecedentedly large both in terms of the number of IoT devices connected to it (which is anticipated to be multi-billions), and the geographic area that it covers (which is expected to be the entire earth). Hence, scalability is of paramount importance when building IoT-enabled applications. To illustrate this, we take the simple example of an application that requires to know the number of empty parking spots on a street of the city of Berlin at this moment. Assuming that each parking spot is equipped with an occupancy sensor, answering this query would mean to sequentially communicate with every occupancy sensors to check if they are reading “empty” or “occupied”, and count the empty ones. This straightforward approach would work for a small system with only a few occupancy sensors that cover the street. However, if we consider the entire city of Berlin with millions of occupancy sensors in Berlin (which is a reasonable assumption), the same approach would not scale due to the unacceptable delay incurred by and energy cost required for the communication between the application and all occupancy sensors of Berlin. Research efforts will be required to develop scalable systems, e.g., based on caching

mechanisms, parallel computations, hierarchical architectures, on top of which IoT-enabled applications are built.

2.5.6 Imperfect and Heterogenous Data

IoT devices are typically low-cost, low-power, and small in size, so that they can be deployed in large quantity (e.g., large-scale RFID systems or WSNs), operate on their battery for long time (e.g., environmental monitoring applications using WSNs), and be unobtrusive to human users (e.g., convenience for patients in healthcare applications). Due to these factors, the data collected by IoT devices will be subject to redundancy (e.g., due to RFID tags are read by multiple RFID readers at multiple times), heterogeneity (e.g., data come from different kinds of sensors of different organizations, with different sampling rates), and noise, jitter, outages, and outliers (e.g., due to environmental influences or hardware malfunction). As IoT-enabled applications and services are built based on the data generated by the IoT, appropriate mechanisms will be needed to compensate for possible impacts of these imperfections on IoT applications.

2.5.7 Security and Privacy

The attraction of the IoT comes from the pervasiveness of vast numbers of IoT devices that are embedded into and constantly report information about the real world, so that we could interact with the real world much like we can now with the "virtual" world of the Internet and the Web. Unfortunately, this pervasiveness also poses serious security and privacy problems that need to be addressed in order for the IoT to be widely accepted by the public. The reasons for this are due to the nature of how the IoT works. Firstly, IoT devices spend most of their time unattended, thus can be easily physically attacked. Secondly, the wireless communication between Things and between Things and the Internet is vulnurable to eavesdropping. Thirdly, complex and resource-demanding security mechanisms are not suitable to be implemented on resource-constrained IoT devices (e.g., passive RFID tags or low-end sensor nodes). Fourthly, information about the environment is autonomously and constantly collected by IoT devices without human awareness (e.g., smart home applications recording inhabitants' living habits). Finally, how the extremely massive amount of heterogenous data generated by the IoT is exploited, i.e., who has the right to access what kind of data and when, is not clear. Example problems are replacement of IoT devices with harmful ones (due to the devices' unattendedness), man-in-middle attacks (due to wireless communication), and mis-use of sensitive data. For more information, readers can find detailed discussions about these and other problems as well as potential solutions in [101] [102] [103] [104].

2.6 Summary

This chapter discussed important concepts of the IoT in detail. In particular, we discussed the definition of the IoT, highlighted its three aspects, and presented its applications. We introduced a layered view on the IoT in which we pointed out where the routing and sensor search services can be placed, as well as how they are related to each other. We reviewed the technologies that enable the existence of the IoT as well as drive its evolution. Finally, we identified several technological challenges that need to be addressed for a realization of the IoT.

Chapter 3

Routing in the IoT

Routing is an essential service in the IoT, since it enables the exchange of information between Things, by efficiently directing and reliably delivering data on the network from their sources to their destinations. However, routing in the IoT is also challenging, due to the global scale of the IoT, the massive number of Things in the IoT, the dynamic topology of the IoT, and the resource constraints of the IoT devices.

This chapter is devoted to the routing service in the IoT that addresses the above challenges[1]. Specifically, we propose, implement, and evaluate two routing algorithms for wireless sensor networks (a building block of the IoT), namely *recursive multi-region geocasting* and *stochastic forwarding-based routing*. The novelty of these routing algorithms is three-fold: (i) their simple design makes them suitable for resource-constrained sensor nodes; (ii) they require only local information for operation, thus scale with the size of the IoT; and (iii) they are aimed at IoT-specific routing scenarios. The structure of this chapter is as follow. In Sec. 3.1, we discuss the general routing problem in the context of the IoT. In Sec. 3.2, we focus our discussion on the geographic routing approach, which is the base for our two proposed routing algorithms in this thesis. In Sec. 3.5 we conclude the chapter.

3.1 The Routing Problem

Routing, in general, answers the question of "how an entity is brought from an origin to a destination". In the context of the IoT, the entity is a *data packet*, and the origin and destination of the data packet are two computing devices and are called *the source* and *the destination*, respectively. A computing device can either be an IoT device (such as an RFID tag, a sensor node, or a smartphone) or an Internet device (such as a PC or a server computer). We call a computing device a *routing node*. Due to the fact that there is not always a direct physical connection between the source and the destination of a data packet,

[1] This chapter is based on our work in [4] and [5].

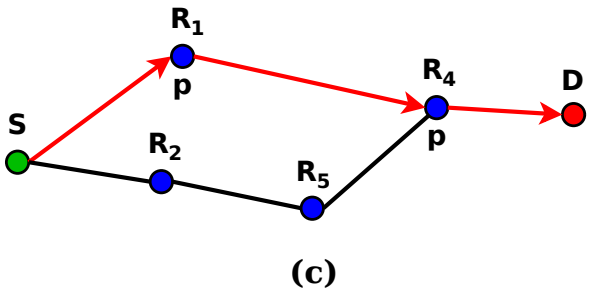


FIGURE 3.1: The general routing problem

the packet must be relayed from one intermediate routing node to another before arriving at the destination. This approach is known as *multi-hop routing*. The series of hops, i.e., intermediate routing nodes that are involved in relaying the data packet, is called a *routing path* or a *routing route*. Suppose that it costs $c(RP)$ units of network resources (e.g., energy, wireless bandwidth) to send the data packet from a source over a routing path RP to a destination, where $c(\cdot)$ is a cost function, the multi-hop routing problem is *finding a routing path RP such as $c(RP)$ is minimized*.

The multi-hop routing problem can be illustrated as in Fig. 3.1, which shows a generic and simple routing scenario where the source S wants to send the data packet p to the destination D. For the sake of explanation, we abstract the network as a graph whose vertices (e.g., S, D, R_1) represent routing nodes and edges (e.g., $\overline{SR_1}$) represent the *routing links* between them. If we define the cost function as the number of hops, the routing path RP_{R_1,R_4} will be selected instead of RP_{R_2,R_5,R_4} for sending the packet p, because the former requires only 2 hops as opposed to 3 hops required by the latter. Specifically, p will be relayed from S to R_1, from R_1 to R_4, and finally from R_4 to D.

As we mentioned in Sec. 2.4.4, routing in the IoT takes place in two domains: the Internet and the networks of IoT devices such as RFID networks or WSNs. In this chapter, we focus on routing in the latter domain, since IoT devices motivate the existence and evolution of the IoT. Therefore, we define "routing in the IoT" as "routing in networks of IoT devices" hereafter.

In the rest of this section, we discuss the major challenges in designing a routing protocol for the IoT. After that, we present a set of routing requirements that usually are specified by applications and derive a set of properties that a routing protocol should possess to meet the set of routing requirements. Finally, we outline at the end of the section the design space of IoT routing protocols.

3.1.1 Challenges

There are many challenges that can affect routing in the IoT. The challenges can come from the routing layer itself, and/or from the layers underneath it such as physical and medium access control (MAC) layers. Although a cross-layer approach can be employed to take

advantage from the properties of the lower layers in the design of a routing protocol, such approach would tighten the routing protocol to a (set of) specific MAC and physical designs. This would limit the use of the routing protocol to only a few types of IoT devices, whereas IoT devices are known to be extremely heterogeneous. In this thesis, our focus is the routing layer. In the following, we present the major challenges that directly affect routing in the IoT.

3.1.1.1 Limited Resources

One of the main challenges to the IoT, as we mentioned in section 2.5.2, is the limitation of resources, including energy supply, processing power, memory capacities, wireless communication range, and wireless communication bandwidth. This limitation affects routing in many ways. The short wireless communication range dictates that routing must be done in a multihop fashion, i.e., the data packets must be forwarded by multiple relay nodes in order to reach to their destination. The low processing power and program memory require that the routing process running on the IoT devices must be highly optimized and light-weight. The small storage memory and scarce communication bandwidth may limit the size of the packets to be forwarded. The scarce energy source (either battery-supplied or harvested) makes it difficult to decide which nodes should forward the data packets, since wireless communication dominates the energy consumption of the IoT devices.

3.1.1.2 Dynamic Routing Topology

The cause of the dynamicity of the routing topology is manyfold. Firstly, due to energy constraint, IoT devices are usually scheduled to be idle or working (e.g., by turning the wireless radio on/off) to minimize energy consumption, making the routing topology dynamic. Secondly, since users deploy or remove their IoT devices at will, routing nodes will be connected to and disconnected from the IoT at unknown rate, which adds the unpredictability to the dynamicity of the routing topology. Thirdly, node failures are common in the IoT. The causes of a failure include hardware malfunctioning (e.g., antenna damage), exhausted energy supply (e.g., depleted batteries), and environmental impact (e.g., the air humidity level is unexpectedly high causing shortcuts). Fourthly, node mobility causes the wireless links between the mobile nodes and other nodes in their proximity to be reconfigured. Finally, the low-power wireless links in networks of IoT devices (e.g., WPAN, WSN) are unreliable and transitional, which also contributes to the dynamicity of the topology. The routing protocols, hence, must be flexible enough to deal with such dynamicity of the IoT's topology.

3.1.1.3 Scalability

The IoT will be large in scale, both in terms of number of nodes and geographically. As routing means to decide over which routing path the data packet should be sent, the more

candidate relay nodes to be evaluated for inclusion in a routing path, the more complex routing is. This complexity is manyfold, including what cost function to be used, how to decide which of the neighbours of a node is the relay node, what is the cost to setup and maintain a routing path, how to setup new a routing path when another one is broken, etc. Such complexity will quickly grow unmanagable if the routing protocol was not carefully designed with scalability challenge being taken into account.

3.1.1.4 Partitions and Voids

Another major challenge to routing in the IoT is the presence of network partitions and voids in the network. A partition is a disconnected part of the network, such that nodes inside a partition cannot communicate with nodes in the other parts of the network, because there is no routing path to exchange data packets. A void is an area that is not covered by the network. Since there is no node located inside the void that is connected to node(s) outside it, data packets can only be forwarded around the void to reach to their destination. For example, a WSN has been deployed by randomly scattering a large number of sensor nodes over a geographical area. Due to the structure of the area, there may be lakes that cause voids, or rivers that cause partitions in the WSN.

3.1.2 Requirements and Properties

We review here the application requirements to routing, and present a set of properties that a routing protocol designed for the IoT should possess in order to meet these requirements. The requirements are given below:

- *Delivery time (R1)*: Delivery time is the traveling time of a data packet from its source to its destination, and is usually specified by the application. For example, a WSN-based forest fire detection application may require that the base station of the WSN receives warnings within a hard time constraint of 3 seconds after the sensed temperature exceeds a certain threshold.
- *Delivery rate (R2)*: Delivery rate refers to the ratio between the number of the data packets that successfully arrive at their destination and the total number of the data packets that have been sent by their source. Obviously, the higher the delivery rate, the better. The reason that causes unsuccessful delivery are *routing loops*, which is usually due to a poor routing design (note that, reasons that come from MAC and physical layers are beyond the scope of this thesis as we focus only on the routing layer).
- *Energy-awareness (R3)*: As IoT devices usually operate on battery for long time periods (e.g., WSNs to monitor the environment), it is desired that the routing protocol is always aware of the energy status of the network, and acts on that accordingly.

Due to the challenges presented in the previous subsection, any routing protocol designed for the IoT should possess the following properties in order to meet the above requirements:

- *Highly adaptive ($P1$)*: The routing protocol should be highly adaptive to be able to quickly react to the dynamicity of the network topology. For example, if a routing path is broken due to node failure or the energy source of the nodes on the path being exhausted, the routing protocol should be able to quickly find alternate paths.
- *Load balancing ($P2$)*: The routing protocol should distribute its operation load, including energy consuming, computation, and communication activities evenly across the network, such that no part of the network would run out of resources faster than the others. The operation load depends on how the cost function $c(\cdot)$ is defined for each routing node or link in the network. For example, the cost function can be defined in terms of residual energy of the routing node and/or the level of reliability of the routing link.
- *Loop-freedom ($P3$)*: As we mentioned in $R2$, the routing protocol should result in no routing loop in order to archieve successful data packet delivery.
- *Low overhead ($P4$)*: Due to resource constraints, the routing protocol should incur low routing overhead, i.e., the cost required to meet some or all of the routing requirements. The cost can be in terms of any type of resources, namely energy, computation, memory, and communication, or a combination of them. Again, the cost function $c(\cdot)$ needs to be defined in order to evaluate the incurred overhead.

3.1.3 Design Space

In this subsection we discuss the space of possible approaches a routing protocol for the IoT could follow. We do not firmly conclude which approach a routing protocol should follow, but rather discuss the advantages and the disadvantages of them while putting them in relation to each other. We also refer the readers to some representative routing protocols that follow certain approaches during discussion if needed.

Note that the approaches are not mutually exclusive but can be combined, and the choice for an approach is driven by what kind of specific applications and/or what routing challenges are more pronounced to a designer.

3.1.3.1 Distributed vs. Centralized

These two approaches refer to where the routing decision is made, i.e., deciding which path to send the data packet along. There are two choices: centralized and distributed.

In a centralized approach, there is a super node that is assumed to have abundant resources and knowledge about the state of the entire network. This super node has control over all

other nodes, computes the optimal routing path for every data packet, identifies bottlenecks and underutilized nodes, and adapts the routing paths accordingly. The advantage of centralized routing is a complete control over all aspects of the network, therefore optimal routing paths could be computed. The disadvantages are, however, costly maintainance of the super node, the super node could potentially be the central point of failure, high control overhead as instructions need to be communicated between the super node and other nodes, and last but not least, the computed optimal routing paths may become obsolete quickly due to the dynamicity of the network, especially in the context of the IoT.

In a distributed approach, an individual node or a set of nodes that are in proximity of each other make the routing decision. These nodes do not have knowledge about the state of the entire network, but only about their local state (and possibly the state of their neighbors). The routing decision, therefore, is made only according to this limited knowledge. The advantages of distributed routing are flexibility as decision making is distributed and performed by each node, and responsiveness because nodes in proximity can quickly react to any dynamicity-related issue that occurs locally. The disadvantages are possibly suboptimal routing paths and potentially unbalanced load distribution, since only local information are used. Almost every routing protocol designed for the IoT is distributed to ensure scalability.

3.1.3.2 Flat vs. Hierarchical

Once one decides to follow the distributed approach, one could further decide to follow either the flat or the hierachical approaches. These two approaches refer to where the routing algorithm is placed and run in the network. In a flat approach, a relatively simple routing algorithm is implemented on every individual routing node of the network. A node makes routing decisions based solely on its own state and the state of a number of other nodes in its proximity. As the design is relatively simple, typical IoT devices such as sensor nodes or active RFID tags can afford to run the routing algorithm. There is no super node that controls the routing of the network in this case. The routing paths computed are the emergent results of many nodes executing the same routing algorithm. This approach is also known as *flat routing* (see [93] for some representative protocols), because every routing node plays the same role in the network.

In a hierarchical approach, the routing nodes are divided into several hierarchical levels. Nodes that belong to the same level are assumed to have similar resource budgets, while nodes belonging to different levels have significantly different resource budgets. The routing algorithm is also divided into components with different degrees of complexity. More complex components are implemented on nodes that belong to the higher hierarchy level. Usually, a node manages inferior-level nodes, reports to superior-level nodes, and only collaborates with nodes at the same level. With such a distribution of roles and complexity, network resources could be efficiently utilized for calculating routing paths. This approach is also known as *hierarchical routing*. A set of routing protocols following this approach can be found in [93].

3.1.3.3 Location-based vs. State-based

These two approaches refer to the type of information used by the routing protocol to forward data packets. In a location-based routing protocol, information about the location of the routing nodes are used for addressing nodes and forwarding data packets. The nodes' locations can be obtained via dedicated hardware (e.g., GPS sensors) or software (e.g., location discovery algorithms such as in [105]). The forwarding decision is usually made based on a distance metric (e.g., Euclidean distance). Sometimes, information about network resources are also combined with the distance metric if one or more routing properties are to be integrated into the design (e.g., $P2$). The advantages of location-based routing are low control overhead, scalability, and robustness against network dynamicity, since the processes of route discovery and maintenance (i.e., finding the destination and maintaining an established path to it) are spared, and information about network topology is not required. The disadvantage is the dependence on means for location discovery, which can be costly in terms of money (e.g., buying GPS receiver hardware) or network resources (e.g., distributed algorithm for location discovery). Several representative protocols following this approach can be found in [93].

In a state-based routing protocol, a data packet is forwarded based on the information about the current state of the network. The network state can be (i) stored at nodes and/or (ii) included in the data packet. In the case (i), each node has a view on the current topology of the network in terms of which nodes are connected to which other nodes, or the distances from the node to all other nodes (distance is a measure of the cost to reach a certain node, that usually is a cost function of hop count and/or a set of resources such as the node's residual energy). These two approaches are known as *link state* and *distance vector* routing, respectively. Representative routing protocols that follow these approaches are [106] for link state routing, and [107] for distance vector routing, and their variants. In the case (ii), a routing path that the data packet should traverse to its destination is stored in its header and is specified by the source. A relay node uses this information to make routing decisions. This approach is known as *source-based* routing, and [108] is the representative protocol. The main disadvantage of state-based routing is poor scalability, since the storage required to store the network state at each node and the amount of information required to be exchanged across the network to update topology changes do not scale with the number of nodes. Additionally, network state may get obsolete quickly if topology changes are not updated fast enough, leading to inefficiently computed routing paths.

3.1.3.4 Data-centric vs. Address-centric

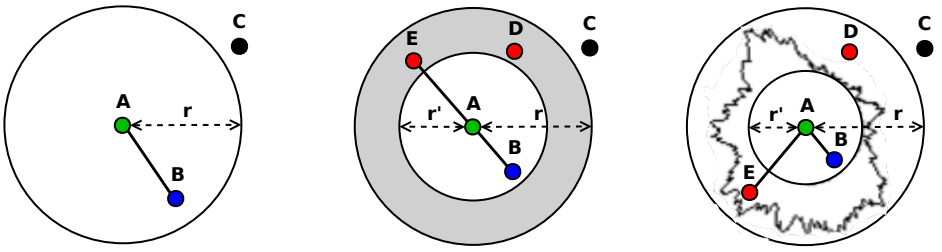
The design choice to follow one of these two approaches depends on the type of the application running on the network. In traditional networks such as the Internet or networks of wireless computer devices (e.g., laptops, smart phones), data packets usually are routed based on the addresses of their destination nodes. For example, in a video conferencing application,

multimedia data packets are destined only to the addresses of the participants in the video conference (i.e., their laptops or smart phones). An address is unique to a network node, which could be the node's MAC address, IP address, or any other type of unique identification (e.g., RFID, uCode). This approach is known as *address-centric* routing.

Many IoT-enabled applications require that data generated by all or a large percentage of nodes are reported to a sink node for further processing. For example, an RFID reader scans all RFID tags within its communication range, or sensor nodes in a WSN periodically send their sensed data about a certain event to the WSN's base station. In such applications, it is important that nodes with certain types of data (e.g., temperature readings, free parking lots) rather than with specific addresses (or identifications) send data packets to the sink. Due to multiple nodes sampling the same type of data or observing the same event, there are data redundancies which can be eliminated by performing data fusion at relay nodes as the data packets travel to the sink. Data fusion at a relay node means to integrate similar information contained in multiple data packets into a consistent, accurate, and useful piece of information, that is to be forwarded by the relay node towards the sink. This type of forwarding is known as *data-centric* or *query-based* routing. The data-centric routing is usually a consequence of the sink dispersing a query (e.g., all sensor nodes that are reading above 50°C) into the network with the help of a routing protocol (e.g., a location-based routing protocol to deliver the query to multiple geographical regions). Two representative data-centric routing protocols are [109] and [110].

3.2 The Geographic Routing Approach

In this section, we focus our discussion on the geographic routing approach, which is the foundation of our proposed routing algorithms in this thesis. The geographic routing approach follows two design approaches, namely distributed and location-based. Routing decisions are distributed throughout the network and performed by individual routing nodes. The only information used for making routing decisions at a node is its location, the location of the destination, and sometimes the locations of its neighboring nodes. In the context of the IoT where networks are usually wireless, the locations of nodes typically correspond to their network connectivity (radio signal strength is inversely proportional to the geographic distance), which makes geographic routing a natural approach for routing in the IoT. Moreover, using only locations, geographic routing provides an efficient way to route packets, since the state required to be maintained at each node and the control overhead required to be exchanged among neighboring nodes are minimized. As packets are always forwarded towards their destination, there is no need for route discovery and maintenance before and during sending data packets. Geographic routing also responds fast to network dynamics, since local changes of network topology can be quickly adapted to (e.g., via a simple Hello protocol). For these reasons we choose geographic routing as the approach for our proposed routing algorithms for the IoT.



(a) UDG Model **(b) Quasi-UDG Model** **(c) RIM Model**

FIGURE 3.2: The wireless link models

A typical geographic routing scenario considers a network of a (large) number of nodes (IoT devices) that are deployed on a (large) geographical area (e.g., a WSN deployed in a forest to detect fire). Routing in the network is multi-hop because data packets must be forwarded from one relay node to another until they reach their destination. Since there are multiple routing paths between a pair of source and destination, the network has a mesh topology. The dynamicity of the topology is due to the causes discussed in Sec. 3.1.1.2.

Before reviewing the major forwarding techniques that are used in geographic routing in Sec. 3.2.2, we present in Sec. 3.2.1 the common models for defining if two nodes in the network do share a wireless link, since this is essential for designing and studying any routing protocol. When the two nodes are connected by a link, they can directly and bidirectionally exchange data packets with each other, and the wireless link is said to be *symmetric*.

3.2.1 Wireless Link Models

Perhaps the most commonly used model for representing the wireless link between a pair of nodes is the *unit disk graph* (UDG) model, which assumes that all nodes in the network have the same transmission range r. As illustrated in Fig. 3.2-(a), the transmission region of node A is modeled as the circular region with radius r, that is centered at A, i.e., (x_A, y_A). Node B is said to share a link with and is a neighbour node of A if and only if $d_E(A, B) \leq r$, where $d_E(A, B)$ is the Euclidean distance between the two nodes and is given by

$$d_E(A, B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2} \tag{3.1}$$

This definition is binary, since two nodes do either share or not share a link. With this definition, nodes A and B share a link, whereas nodes A and C do not. The UDG model is used pervasively during the design of geographic routing protocols, as its simplicity and symmtry are ideal for studying the theoretical behaviour of the routing protocol in design, as well as for performing mathematical analysis and inference. The disadvantage of UDG is, however, that the model does not capture the inherent irregularity of the wireless medium, thus may lead to significant difference between the theoretical and the practical performance of the routing protocol.

To address this disadvantage, the work in [111] proposes a relaxation of the UDG model called the *Quasi-UDG* or *QUDG* model. Given a parameter $r' \in [0, r]$, any two nodes A and B of the network are said (i) to share a link if $d_E(A, B) \leq r'$; or (ii) to not share a link if $d_E(A, B) > r$. This definition does not specify whether two nodes A and B having distance $d_E(A, B) \in (r', r]$ share a link. Such a link may or may not be there depending on a probabilistic model, hence the name "quasi". Usually, a fixed probability is used to model such quasiness. Clearly, the UDG model is a special case of the QUDG with $r' = r$. The advantage of QUDG is that it approximates the irregularities of the wireless medium, thus is more realistic than the UDG model, while is still easy for theoretical analysis. An illustration of the QUDG model is given in Fig. 3.2-(b), where the irregularity or "quasiness" of the wireless medium is illustrated as the shaded area. Due to the given probability to model the quasiness, the two nodes A and E share a symmetric link while A and D do not, even though both D and E are located in the shaded area. On the other hand, the two nodes A and B definitively share a symmetric link, while A and C do not share one.

An alternative to the QUDG model is the Radio Irregularity Model (RIM) proposed in [112]. The RIM model is actually the QUDG model from a practical point of view. This means the irregularity of the wireless medium of RIM is not mathematically modeled as is in QUDG, but is derived from empirical data taken from wireless sensor devices (see Fig. 3.2-(c)), thus is the most realistic wireless link model. The drawback of RIM is that it is not mathematically modeled, therefore a performance analysis of a RIM-based routing protocol would have to be based on simulation.

3.2.2 Forwarding Techniques

The general forwarding principle of geographic routing is "*greedy* when possible and *recovery* when not", i.e., the next relay of a data packet should be selected such that it is closest to the destinations among other neighbor nodes (the definition of closeness is based on a distance metric such as the Euclidean distance). When a next relay cannot be found greedily (e.g., due to a void), a recovery mechanism is used to forward the data packet until the greedy principle can be applied again (e.g., after the void is passed). The implementations of this principle is, however, vary. We call such an implementation a forwarding technique. In the following, we review the most common forwarding techniques that a node could employ to select the next relay for the data packet. For each technique we refer the readers to the original work that proposes it and describe how it works. For a complete and thorough list of techniques and their variants, we refer the reader to [113].

In addition to the assumption that every node of the network knows their own location, all forwarding techniques assume that the nodes also know the location of the destination of the data packets. Due to the easiness of the UDG model in theoretical analysis, we use it to present the forwarding techniques.

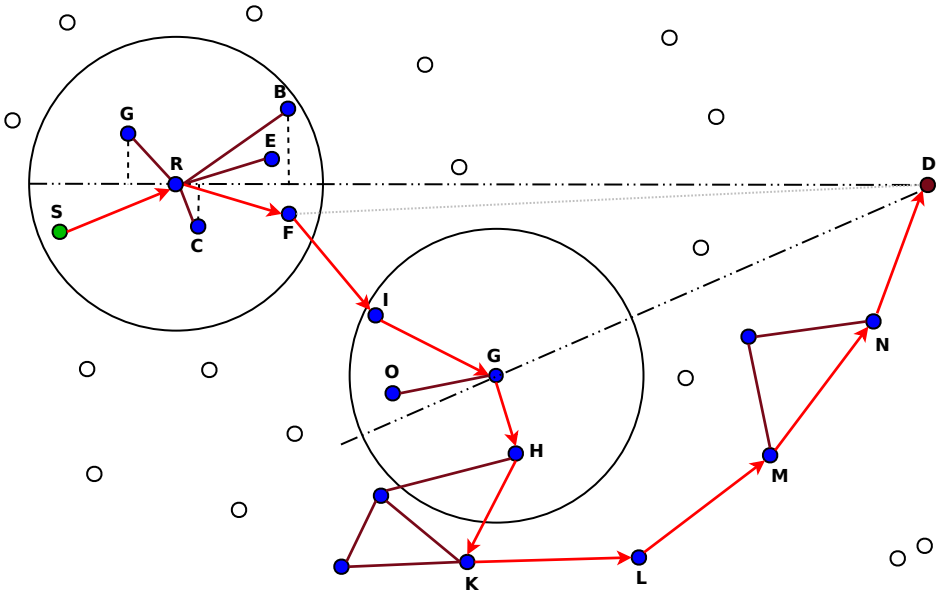


FIGURE 3.3: The forwarding techniques

Without loss of generality, we consider a network that is abstracted as the graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ shown in Fig. 3.3, where $\mathbb{V} = \{S, D, R, G, ...\}$ and $\mathbb{E} = \{\overline{RG}, \overline{RF}, \overline{FI}, ...\}$ are the set of vertices and edges of the graph, respectively. Each vertex in $\mathbb{V}$ represents a node in the network, and each edge in $\mathbb{E}$ represents the existence of a symmestric wireless link between two nodes in the network. Suppose that the source S has sent a data packet p to the destination D, and the routing path over which p has been routed is $route_{R,F,I,G,H,K,L,M,N}$. Note that, vertices that are not filled represent nodes that were not involved in the routing of p.

3.2.2.1 Greedy Techniques

We focus on the relay node R in Fig. 3.3. R has just received p from S, and will forward p towards D. To do that, R will select one of its neighbour nodes (i.e., those that are located within R's transmission region) as the next relay of p and forward p to the relay node. We present now the greedy forwarding techniques that help R select a relay.

- *MFR*: With the MFR [114] technique, the neighbour node that has the longest projection on the line connecting R and D is selected as the next relay, e.g., node B in Fig. 3.3.
- *NFP*: In contrast to MFR, the NFP [115] technique selects the neighbour node whose projection on the line $\overline{RD}$ is shortest as the next relay, e.g., node C.
- *DIR or Compass Routing*: Proposed in [116], this technique selects a neighbour, i.e., node E, as the next relay such that the angle formed by E, R, and D is the minimum among all neighbour nodes.

- *GEDIR*: The neighbour selection of the GEDIR [117] technique is based on the geographic distance between the neighbour and the destination nodes. The neighbour node with the minimum distance is selected as the next relay, i.e., node F.

- *Random Forwarding*: With this technique, any neighbour node that has a positive progress towards D can be selected as the next relay with an identical probability. A neighbour node has positive progress if the length of its projection on the line segment $\overline{RD}$ is greater than zero, e.g., B and C can be chosen, but not G. Examples of this technique are [118] and the references therein.

- *Resource-aware*: The technique in [119] computes the theoretically optimal locations of R's neighbour nodes such that if p is forwarded to nodes at these locations, the power consumption is minimized as well as the geographical progress to the destination is maximized. Obviously, there is no guarantee that R's neighbour nodes are located at those locations in practice. However, the next best selections would be those neighbour nodes that are located closest to those locations. This technique differs from the above techniques in that it takes power consumption into account in addition to geographical progress. The technique, hence, can be seen as a representative for a family of techniques that, in addition to geographical progress, take network resources (i.e., energy, memory, wireless bandwidth) into account. We call this the *resource-aware* family of techniques.

The common and main goal of the above greedy forwarding techniques is to deliver a data packet as fast as possible to its destination. However, due to the characteristics of the wireless medium and the density of the network, the performance of the techniques varies, and none of them is always superior than the others. For example, MFR minimizes the number of hops in a dense network as the higher the network density the higher the probability that there is a neighbour that is located close to both D and the line RD. However, if we assume that the transmit power can be adjusted to the transmission range, a long transmission range in a dense network would increase packet collisions, leading to retransmission which consumes energy and time. In this case, NFP is recommended instead of MFR [118]. The choice for a relatively dense network could be the DIR technique, as the hop count is somewhat higher than that of MFR or GEDIR but with potentially less packet collision. Besides fast packet delivery, other goals such as energy saving, load balancing, or loop-freedom are also important. MFR and GEDIR are known to be loop-free. The random forwarding technique can be used for equally distributing routing load across the network. And the resource-aware technique can be used for resource optimization (including energy saving).

3.2.2.2 Recovery Techniques

When a greedy forwarding technique is used, it is possible that a relay node fails to select the next relay of p, due to none of its neighbours satisfying the conditions of the forwarding

technique. We call such node a *dead-end* node. For example, if GEDIR is used, node G in Fig. 3.3 is a dead-end node. In the following, we present recovery techniques that are used to bypass the dead-end problem.

- *Flooding*: The simplest technique is to have the dead-end node (e.g., G) broadcast p to all of its neighbours [120] (e.g., O and H). The dead-end node also includes in p an indication that it is a dead-end node so that its neighbours record it and will not forward data packets to it in the future. After broadcasting, neighbours that are also dead-ends (e.g., H) repeat this process and reject further copies of p. Non-dead-end neighbours (e.g., K) continue to forward p according to the greedy technique. The disadvantages of flooding are redundancy and excessive resources usage, due to the presence of multiple copies of p in the network, and nodes are required to store a list of dead-end nodes in their memory.

- *Route Discovery*: With this technique, any route discovery scheme can be empolyed to find a path from the dead-end node to the destination, e.g., from G to D. Once a path has been discovered, p is only forwarded along that path to the destination, i.e., not using geographic routing any more. Examples of route discovery scheme are [121], [108], [90]. Although this approach is more sophisticated and incurs less overhead than the flooding technique, route discovery still does not scale well, since it follows the state-based design approach (see Sec. 3.1.3.3).

- *GFG*: This technique is illustrated in Fig. 3.3. The dead-end node (e.g., G) constructs a planar graph using the location of its own and its neighbours, and includes in p an indication that p should be forwarded in recovery mode. The node, then, selects a neighbour as the next relay of p according to the *right hand rule* [122], i.e., node H is selected but not node O. If the neighbour is a dead-end node (e.g., H), it repeats the same process as G did. Otherwise, e.g., node K is node a dead-end, it changes the indication in p to be greedy mode, and forwards p according to the greedy forwarding technique. The resulted routing path in the illustration figure is $RP_{H,K,L,M,N}$. The GFG technique and its variants are the mostly used recovery techniques in geographic routing, as they are localized and low-overhead, therefore are robust and scalable.

3.3 Multi-region Geocast Routing for the IoT

In this section we present a routing algorithm called "Recursive Multi-region Geocasting" (RMG). RMG follows the geographic routing approach and is targeted to the (very) large scale of the IoT (see challenge $C3$ in Sec. 3.1.1.3).

3.3.1 Motivation

While originally most sensor network deployments are rather small with tens or few hundreds of nodes [123], there is a recent trend towards much larger scale deployments with several thousands of nodes being deployed over large geographical areas. In particular, the IoT envisions globally interconnected sensor networks, and in the "smart cities" application domain we are witnessing first actual large-scale deployments. For example, the FastPrk smart parking solution[2] relies on a mesh network of several thousand parking spot occupancy sensors that are deployed over parking areas of the city of Barcelona, to help drivers find empty parking spots to minimize search traffic and time. For a similar purpose, the SFPark project[3] has deployed sensor nodes on 7000 out of 28.800 parking spots of the city of San Francisco, USA. At an even bigger scale, the U-City project [124] is being deployed in South Korea with the ultimate vision of creating a ubiquitous society where the urban environment is soaked with ubiquitous sensor networks and RFID systems.

In such systems, there is often a need to send a message to nodes that are located in multiple geographic regions. With smart parking systems, for example, a user would send a request for a free parking spot to all parking spot sensors located in certain streets, where each street defines a geographic region. In a smart city, a request to locate a lost object (equipped with a wireless tag whose presence can be detected by close-by sensor nodes or RFID readers) would be sent to nodes in multiple geographic regions where the user usually spends time (i.e., home, office, streets on the way from home to office, gym, restaurants) [125]. These examples also demonstrate the relationship between the two services of routing and sensor searching in the IoT, since the search service would need to communicate with all sensors in the geographic regions to find out the sensors that match the search query (e.g., the occupancy sensors that are reading "empty"). The routing service, thus, would be required to realize this communication.

The underlying problem in these examples is *geocasting a message from a source to multiple geographic regions*, respectively to all nodes located in one of those regions. Although geocasting in general is a well-studied problem, most existing work focuses on geocasting to either a *single destination node* at a given location, to *few destination nodes* where the location of each destination node is given, or to a *single geographic region* respectively all nodes located in this single region. Although one could invoke those protocols repeatedly to send the same message to multiple regions, this would not be efficient, especially in networks of IoT devices (e.g., WSN) with their severely limited energy, networking bandwidth, and computational resources.

In this section, we therefore study the problem of multi-region geocast routing to a set of remote regions in geographically large-scale networks of IoT devices. We call such a geographically remote region a *destination region*. Our contribution is two-fold. Firstly, we design

[2] www.worldsensing.com
[3] http://sfpark.org/about-the-project/

the Recursive Multi-region Geocasting (RMG) protocol to address the multi-region geocast routing problem. RMG is tailored to large-scale networks and large numbers of destination nodes. Secondly, we evaluate RMG and compare it to state-of-the-art protocols. We find that RMG (i) minimizes the total number of forwards needed to successfully deliver a packet, thus saving network bandwidth and energy; (ii) minimizes the length of the routing paths between the source node and all destination regions; and (iii) minimizes the computation overhead at relay nodes along a routing path.

3.3.2 Related Work

We structure the discussion of related geocasting approaches according specification of destinations: a set of nodes, a single region, and finally – the focus of our work – multiple regions.

3.3.2.1 Geocasting to a Set of Nodes

Approaches in this class support the delivery of a message from a source node to a set of destinations nodes where the geographical location of each destination node is given. Specific protocols have been designed where the destination nodes are a set of base stations [126], actuators [127], or other sensors [128], [129]. The GMR protocol in [126] is somewhat similar to our work as it divides the destination group into subgroups. For each relay node, a minimal subset of the node's neighbours that promises most geographical progress towards the destinations is selected as the next relay of the packet. The selection is performed based on the so-called cost-over-the-progress ratio. A drawback of GMR is that the computation of such a minimal subset must be performed at all intermediate relay nodes along the paths from the source to all destinations. This is expensive especially when the network density is high and the routing paths are long (e.g., in geographically large scale WSN). Our approach, in contrast, performs a lighter computation only when a particular condition is violated, therefore saving processing resources.

However, all of the above protocols have been designed for small-scale networks and for a small set of destinations (whose locations all have to be included in the header of the message). In contrast, the multi-region geocast problem does not consider individual destination sensors, but geographic regions each of which contains many nodes. Specifically, our solution is tailored for regions located remotely from the source in geographically large-scale networks. Comparison results in Sec. 3.3.5 between our protocol RMG and the above GMR protocol show that RMG excels in such settings.

3.3.2.2 Geocasting to a Single Region

Single region geocast routing deals with the problem of delivering data packets from a source node to all destination nodes located in a particular destination region. The protocol in [130] uses flooding with restricted flooding zone to deliver a packet to the destination region. Although flooding zones reduce bandwidth usage when compared to conventional flooding, it does not scale in geographically large-scale networks, where the distance between the source and the destination region can be long, resulting in a large forwarding zone, thus excessive bandwidth usage. Moreover, the protocol will fail in the presence of network partitions within the flooding zone.

To improve scalability and reliability, geographic unicast routing is used. The packet is unicasted to a node located inside the destination region, from which it is further disseminated to all other nodes in the destination region. If a void is encountered during unicasting, a recovery forwarding technique (e.g., [131]) is invoked to detour the packet around the void, thus guaranteeing the arrival of the packet at the destination region. Representatives of this approach are [132] and [133], which differ from each other in the way they disseminate the packet inside the destination region, i.e., [133] uses restricted flooding while [132] recursively divides the destination region into 4 subregions and unicasts the packet to each subregion.

In-region packet dissemination is challenged by network partitions (e.g., due to sparse topology or obstacles). To overcome this issue, [133], [134], and [135] propose to include out-region nodes in packet dissemination. [133] uses the GFG recovery technique on the planar faces intersecting the border of the destination region. [135] proposes to route the packet to the entrance zone which is an internal border ring of the destination region, to make sure the packet reaches every side of the destination region thus all partitions (if any) are reachable. The idea in [134] is to repeatedly merge all faces intersecting with the destination region to obtain a large virtual surrounding face that covers the destination region. The nodes on this face are then traversed for disseminating the packet into the destination region.

3.3.2.3 Geocasting to Multiple Regions

The protocols reviewed above mainly focus on one destination region. For multiple destination regions there are few works including [136], [137], [138], [139]. The work in [137] relies on flooding to discover routes to the destination regions using “route discovery” and “route reply” messages. This approach clearly does not scale in large-size and dense networks. Also relying on flooding but with a hierarchical approach, [138] groups nodes into clique-clusters. A super cluster head is elected among these cluster heads. The packet is sent to super cluster head which then floods it to all clique-cluster heads. Each clique-cluster head then forwards the packet to its members if they belong to one of the regions. This approach is supposed to be energy efficient, but it comes with the extra overhead of management and maintenance of the clusters. The protocol in [139] geographically partitions the deployment area of the

network into disjoint and equally sized cells, and performs geocast routing on top of these cells' managers. Again, cell management and maintenance should be considered as extra overhead for this protocol.

The closest work to ours is the GGP protocol in [136]. GGP employs the concept of Fermat point, which is the point within a triangle from which the sum of distances to the vertices of the triangle is minimized. To route a packet to a pair of destination regions, the packet is first greedily forwarded to the pre-computed Fermat point of the triangle formed by the packet's source and two centres of the regions. The packet is, then, duplicated and forwarded to the two regions. If more than two destination regions are given, e.g., for three regions A, B, C, GGP computes the Fermat point F_1 for the triangle formed by the source and the centres of A and B, then computes F_2 for the triangle formed by the source, C's center, and F_1. The packet is routed to F_2 first where it is duplicated and routed to F_1 and C. When the packet reaches F_1 it is duplicated again and routed to A and B. The same principle is applied for a larger number of destination regions. We can see that GGP delivers a packet to the destination regions in a sequential fashion, which may result in a long routing path for the packet to reach all regions. In contrast, our approach is to group closely located regions and forward the packet along a forwarding line towards all group's members in parallel, thus requiring much shorter routing paths. Simulation results in Sec. 3.3.5 validate this claim.

3.3.3 Assumptions and Approach

We outline the basic approach of our multi-region geocast routing algorithm RMG using an analogy with the real world, after presenting basic assumptions and models underlying RMG.

3.3.3.1 Network Model

We study the multi-region geocast routing problem in geographically large-scale networks of IoT devices. For the ease of exposition, we first model the wireless link using the UDG model, and assume that each node has a fixed transmission range, which is identical for all nodes. The radio link is therefore perfectly bidirectional, i.e., two nodes that lie within each other's transmission range can exchange data without packet loss. The dynamicity of the network topology is the consequence of the causes presented in Sec. 3.1.1.2 (the routing challenge $C2$). Note that our routing algorithm will still work with relaxation of these assumptions, as we will discuss in detail in Sec. 3.3.5.

We also assume that all nodes are aware of their locations, e.g., through GPS receivers, or by employing a distributed location discovery algorithm, such as in [105]. Each node also knows the locations of its neighbours via a simple Hello protocol. Moreover, the information about the shape of the destination regions are known to the source node before the data packet is sent out.

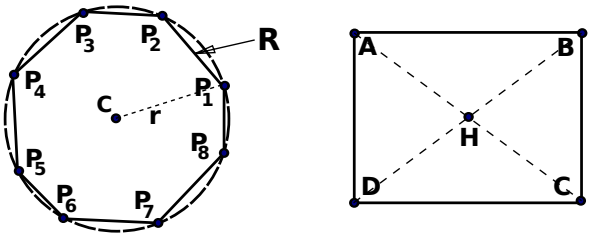


FIGURE 3.4: Description of destination regions

3.3.3.2 Shape of Destination Regions

Many single-/multi-region geocasting protocols such as in [130], [133], [136], either treat the shape of the destination regions as a convex closed polygon (square, rectangle, circle), or as in [134], as a concave closed polygon shape. The authors implicitly assume that a closed polygon is described by a set of points and some extra information, e.g., a circle is given by its center and radius, which are included in the packet header.

There is a trade-off between how detail a destination region can be described and the amount of information included in the packet header, which consumes network bandwidth resource. For example, the region R in Fig. 3.4 could either be more accurately described by 8 points $P_1, .., P_8$, which requires a storage overhead of 16 real numbers. Alternatively, R can be less accurately described by the circle centring at C whose radius is r, which requires only 3 real numbers. The choice for this trade-off depends on the particular applications. Our proposed routing algorithm is flexible and supports any type of convex geometric shape as long as the description of the destination regions and a formula to compute their centres are given.

Throughout this section we use rectangles to model the destination regions, since they provide a good trade-off between detail and overhead, as well as geometric flexibility. In real life, many application scenarios can benefit from this modeling, e.g., a building, a subregion in a forest, a coast region by a sea, etc, because their shape can naturally be decomposed into a set of rectangles.

We assume that the source node of the data packet can consult a service that resolves a destination region into a set of rectangles described by their width, height, 1 corner, and the polar angle formed between the polar axis and its width. For example, the rectangle $ABCD$ in Fig. 3.4 is described by its corner (x_A, y_A), its width $|AB|$, its height $|AD|$, and a polar angle of 0. Thus, 5 real number are required to describe the region. Multiple destination regions will then be represented as a set of regions of rectangular shape, which are included in the packet header before the packet is sent out. To send a packet to a region, e.g., $ABCD$, we forward the packet towards the region's center (x_H, y_H). Thus from now on, we refer to the region's center as the destination of the data packet.

3.3.3.3 The Recursive Forwarding Approach

We illustrate the operation of our routing algorithm with the following analogy. Imagine you live in Berlin and are visiting your friends living in Paris. In Berlin, you do not see any detail of Paris because the city is too far away. To you, Paris looks just like a point, thus it does not make much sense for you to spend time and effort to calculate the paths to each an every friend in Paris at departure in Berlin, because the travel distance is dominated by the distance between the two cities. So you decide to travel to Paris first, before you plan the visit of your friends. The shortest way to get to Paris is obviously the straight line connecting the two cities. When arriving at the entrance of Paris (the city gate that you enter from the highway), you know in which districts your friends live. Since the districts are still far apart, again you decide to travel to the districts before planning the visit of friends. The shortest path to a district is again the connecting line between its center and the entrance of Paris. This strategy is recursively repeated until you can directly see the house of a friend (e.g., from an end of the street where your friend's house resides). Now that you can see the house, you approach it and knock on the door.

Consider a group of destination regions that are located closely to each other, and a source node is transmitting data to the nodes inside these regions. The distance from the source node to the group of regions is much larger than the average distance between the member regions of the group. Applying the spirit of the above analogy, we forward data packets along the straight line connecting the source node and a *division point*, which we define, similarly to the entrance of Paris in the analogy. A division point is the point where new routing decisions have to be made. At this point, the destination group is divided, and the packet is forwarded to the sub-groups in the same manner. We call that straight line the *forwarding line*.

The advantage of this approach is two-fold. Firstly, it is lightweight because we only have to compute the division points and divide the destination group at some intermediate nodes during the delivery of the data packet. Secondly, it saves bandwidth because instead of sending a packet separately towards each an every destination (i.e., n transmissions), we only send the packet once along the forwarding line towards all the destinations. The approach, however, raises three questions: (i) how to compute the forwarding line; (ii) how to calculate a division point; and (iii) how to divide a group of destinations into sub-groups. We will address these questions in Sec. 3.3.4.

3.3.4 Recursive Multi-region Geocasting

In this section, we describe in detail the main two elements of RMG: the computation of the forwarding line and the division of a group of destination regions into subgroups. Finally, we outline how these two elements are integrated into a complete algorithm.

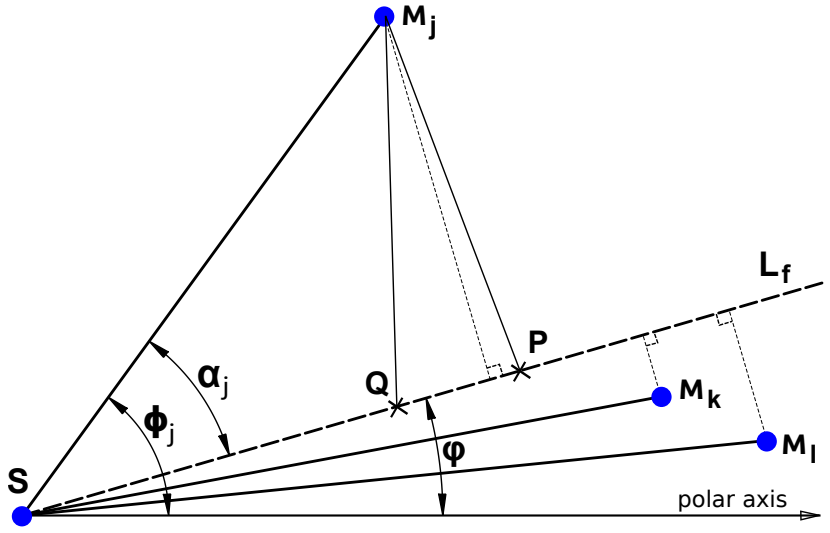


FIGURE 3.5: The recursive forwarding approach

3.3.4.1 Forwarding Line & Division Point

To understand the computation of the forwarding line and the division point, we take a look at an illustration of our approach in Fig. 3.5, where the source node S is sending data packets to a remote group of destination regions $G = \{M_j, j = 1..m\}$, where M_j is a center point of a rectangular destination region. The dashed line L_f connecting the source S and the division point P is the forwarding line along which data packets are sent. To compute P, we need to compute φ and $d_E(S, P)$, where $d_E(\cdot)$ stands for Euclidean distance.

Consider a destination M_j. If we assume that the cost of forwarding a data packet is proportional to $d_E(S, M_j)$, then the minimum forwarding cost we could achieve by sending the packet along L_f is when L_f coincides with the line $S\vec{M}_j$, i.e., $|\phi_j - \varphi| = 0$. To minimize forwarding cost to all M_j by using only L_f, we need to find a φ such that $\sum_{j=1}^{m} |\phi_j - \varphi|$ is minimized. Hence

$$\varphi = \frac{1}{m} \sum_{j=1}^{m} \phi_j \tag{3.2}$$

Since we want to use only L_f to send a data packet to all M_j to minimize the total forwading cost, we want to place the division point P on L_f to be as close as possible to all M_j. This means to find a P, such that $\sum_{j=1}^{m} |PM_j|$ is minimized. However, if the data packet is routed via P on L_f to all M_j, the individual forwarding cost to each M_j is higher than sending the packet along SM_j. We define the individual forwarding cost of the transmission of the data packet from S to M_j via P as

$$\gamma_P^j = 1 - \frac{d_E(S, M_j)}{d_E(S, P) + d_E(P, M_j)} \tag{3.3}$$

Consider a point $Q \in L_f$. Due to the triangles inequality, if $d_E(S,Q) < d_E(S,P)$ then $\gamma_Q^j < \gamma_P^j$, which means that reducing individual forwarding cost would increase total forwarding cost, and vice versa. A good trade-off would be to place P at a position on L_f such that P is closest to all M_j and S, i.e., to minimize the sum of the distances from P to all M_j and S. Such position can be found by minimizing:

$$d_E^2(S,P) + \sum_{j=1}^{m} d_E^2(P,M_j) \tag{3.4}$$

There are two reasons why we use the square instead of the absolute value in this situation. First, square is continuously differentiable, therefore is helpful when we want to find a minimum. Second, square emphasizes larger differences, thus an asymmetric minimum would be avoided.

Now according to the law of cosines we have:

$$d_E^2(P,M_j) = d_E^2(S,M_j) + d_E^2(S,P) - 2d_E(S,M_j)d_E(S,P)\cos\alpha_j \tag{3.5}$$

Supplying Eq. (3.5) into Eq. (3.4) and expanding, we obtain:

$$(m+1)d_E^2(S,P) - 2d_E(S,P)\sum_{j=1}^{m} d_E(S,M_j)\cos\alpha_j + \sum_{j=1}^{m} d_E^2(S,M_j) \tag{3.6}$$

Taking the first derivative of Eq. (3.6) with respect to $d_E(S,P)$ and setting it to zero, we have:

$$2(m+1)d_E(S,P) - 2\sum_{j=1}^{m} d_E(S,M_j)\cos\alpha_j = 0$$

Since $2(m+1) > 0$, Eq. 3.4 is minimized when

$$d_E(S,P) = \frac{1}{m+1}\sum_{j=1}^{m} d_E(S,M_j)\cos\alpha_j \tag{3.7}$$

Equation (3.7) says that instead of separately forwarding the data packet from S to each M_j, we can achieve a good trade-off between individual and total forwarding cost by forwarding the packet from S, along L_f until P, then separately forwarding the packet to each M_j.

The point P is our division point and acts as the "entrance" of the city in our Berlin-Paris example, while the group of destination regions acts as the city.

3.3.4.2 Group Division

We now discuss when and how we divide a group of destination regions into subgroups. According to the philosophy of our approach, we use the group's forwarding line to forward a

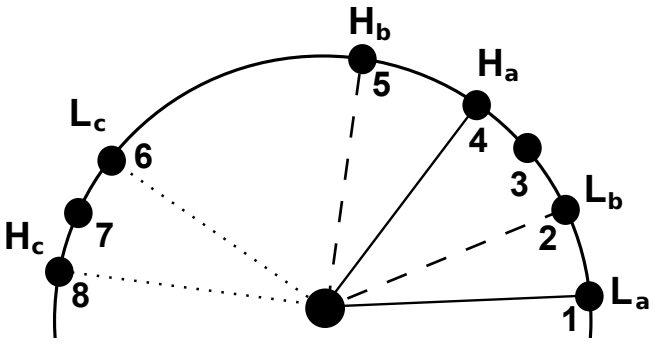


FIGURE 3.6: The GMGD algorithm

data packet as long as the group still looks "small", until the group looks "big" such that it cannot be considered a point destination any more. The group therefore needs to be divided.

Considering the Berlin-Paris analogy, the size of Paris as perceived by a traveller depends on the ratio between the diameter of the city and the distance from the traveller to the city. Similarly, to quantify how small a group G looks from the perspective of a relay node, we denote $\delta_G \in [0, 1]$ as the *smallness* of G such that

$$\delta_G = \frac{2}{\pi} \max_{j \in G} \alpha_j \tag{3.8}$$

Where α_j is the angle enclosed by the forwarding line and the line towards destination region M_j. Now given an application-defined threshold $\delta_{th} \in (0, 1]$, G is said to look small if

$$\delta_G < \delta_{th} \tag{3.9}$$

Based on the condition (3.9), we propose the greedy multi-geocast group division (GMGD) algorithm. The idea is to divide G into a minimum number of subgroups, each of which has the maximum number of destination regions that meet the condition (3.9).

We explain the algorithm by illustration (see Fig. 3.6). We first sort the members of G in increasing order of their polar angles and assign an integer number corresponding to their order. Then we pick the member whose polar angle is smallest (number 1) and iterate over G in increasing order of polar angles, until a member (number 4) such that condition (3.9) is violated i.e., $\delta_{G_{\overline{1,4}}} \geq \delta_{th}$ (where $G_{\overline{1,4}}$ is the subgroup consists of the just visited 4 members). We insert this pair of indices, i.e., $(L, H) = (1, 4)$ into an index list $iList$ that is sorted by decreasing number of members between the pair (i.e., nodes with polar angles greater than or equal to L's and smaller than H's). Note that each index pair (L, H) will define a subgroup. We repeat this process for each member until all 8 members of the group are picked.

After the $iList$ is built, we iterate over the entries of $iList$ and add a pair (L_i, H_i) $(i = a, b, c, ...)$ to a final list $fList$ if the pair does not overlap with any pair that is already in $fList$. For example in Fig. 3.6, two pairs (L_a, H_a) and (L_b, H_b) are overlapped but (L_a, H_a) and (L_c, H_c) are not. Note that since $iList$ is sorted by decreasing subgroup size, biggest

subgroups are always added to $fList$ first. The iteration is done when all entries of $iList$ have been visited.

To build the list of subgroups, we iterate all entries (L_j, H_j) $(j = a, b, c, ...)$ of $fList$. Each entry corresponds to a subgroup whose members are the members of G that are between L_j and H_j.

The detail of GMGD is given in Algorithm 1 (written in pseudo Java programming language). The function $violateFrom(L)$ returns the smallest index H such that the subgroups formed between two indices L and H would violate the condition (3.9). The function $overlap(L, H, fList)$ checks if the pair (L, H) overlaps any pair of indices in $fList$. The notation $|G|$ stands for the number of members of the group G.

Algorithm 1 The GMGD algorithm

```
1: Sort members of G (using Quicksort algorithm)
2: for (L = 0; L < |G|; L++) {
3:   H = violateFrom(L);
4:   iList.insertInDecreaseOrder(L, H − 1);
5: }
6: while (!iList.isEmpty()) {
7:   (L, H) = iList.get(0);
8:   if(!overlap(L, H, fList))
9:     fList.add(L, H);
10:  iList.remove(0);
11: }
```

To investigate the optimality of GMGD, we compare its performance against an exhaustive algorithm that finds the minimum number of subgroups. The result in Fig. 3.7 is the average of 100 experiments where the destination regions are randomly distributed over a square area whose width is 30 times the transmission range. The source node is placed at the center of the area. The figure shows that GMGD on average is within 20% of the optimal solution.

Theorem 1: The worst case complexity of GMGD is $O(m^2)$, where m is the number of destination regions that the considered node is responsible for.

Proof. The worst case complexity of the Quicksort algorithm is $O(m^2)$. The worst case complexity of the *for* loop would be $O(m^2)$ when there is no violation of the condition (3.9). In that case, line 3 would need $m - 1$ comparisons and line 4 would need 1 comparison for m iterations in total.

The complexity of the *while* loop is dominated by line 8. In the worst case, the index list would consist of m disjoint pairs of indices of length |H-L|=1, thus over m iterations of the *while* loop, line 8 would need $1 + 2 + .. + m = \frac{m^2+m}{2}$ comparisons to check if there is an overlap.

In total, the worst case number of comparisons is $\frac{5m^2+m}{2}$, which gives us an algorithm with $O(m^2)$ comparison steps. □

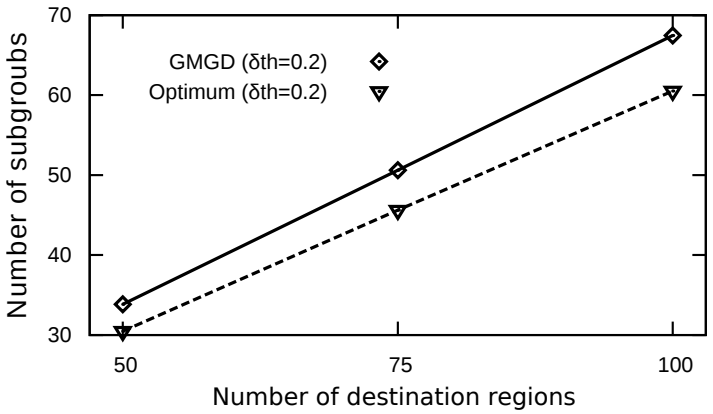


FIGURE 3.7: GMGD: Optimality investigation

We can see in the proof that the complexity of GMGD is dominated by the number of destinations (number of members of the geocast group). GMGD is therefore less complex than the merging algorithm in [126] which is $O(mk\min(m,k)^3)$ (k is the number of neighbours of the current node), especially when both the number of destinations and network density increase. Moreover, we perform GMGD only when the condition in Inequ. (3.9) is violated which further reduces the overhead of our algorithm in comparison with the one in [126], since the latter is run at every relay node. Given that the worst case is highly improbable, the average complexity of GMGD can be expected to be much smaller. This prediction is verified by simulation in Sec. 3.3.5.

3.3.4.3 The Recursive Multi-region Geocasting Algorithm

Based on the discussion in previous sections, we present the Recursive Multi-region Geocasting (RMG) algorithm (see Algorithm 2). The algorithm is completely localized and performed on a per-packet basis by individual nodes in the network. At the source of the packet or the node whose transmission range covers and is closest to the division point among its 1-hop neighbours, the group of destination regions that the node is responsible for is divided into subgroups using GMGD, if the condition (3.9) is violated. The forwarding line and division point of each subgroup are computed using equations (3.2) and (3.7). For each subgroup, the next relay of the packet towards the subgroup is selected using a greedy forwarding technique (e.g., MFR). If a routing void is encountered, a recovery forwarding technique (e.g., GFG) is used. The packet is then broadcasted to 1-hop neighbours.

On receiving the packet, a relay node strips out from the packet's header all but the information about the subgroup that it is responsible for, and forwards the packet along the subgroup's forwarding line towards the division point. Note that, in reality nodes do not necessarily fall on a forwarding line. Thus, the relay nodes select those neighbours that are closest to the forwarding line as the next relays of the data packet. Also, routing voids encountered during forwarding are circumvented by a recovery forwarding technique. This

process repeats until all destination regions are reached, i.e., a node that is inside a destination regions is reached. At this point, restricted flooding is performed to disseminate the packet to all nodes inside a destination region.

Algorithm 2 The RMG Algorithm

```
1: if Node does NOT cover Division Point then
2:    Forward the packet towards division point.
3: else
4:    if Condition (3.9) is violated then
5:       Perform GMGD algorithm.
6:    end if
7:    repeat
8:       Pick a sub-group.
9:       Calculate φ and P for the sub-group.
10:      Select next relay node r for this sub-group.
11:   until All sub-groups has been visited.
12:   Add all φ, P, and r to packet's header.
13:   Broadcast the packet to 1-hop neighbours.
14: end if
```

3.3.5 Evaluation

We study the performance of our RMG algorithm, where the message is first routed to the center points of all destination regions as described before and constrained flooding is then used to disseminate the message within each region. We use simulation to compare RMG with two related routing algorithms: GGP [136] and GMR [126]. GGP is selected because it also addresses the multi-region geocasting problem, and its approach to compute a branching point to fork the routing tree during forwarding is similar to our approach. Although GMR does not support multiple regions but only multiple destination nodes, it can be applied to multiple regions by first geocasting to the center points of all destination regions and then flooding the messages within each region similar to RMG.

We consider the following comparison metrics:

- ◇ *Relay load*: The ratio between the number of forwards needed to successfully deliver the data packet, and the total number of network nodes. The lower this ratio, the lesser network resources (energy and bandwidth) are consumed for the delivery of the packet.
- ◇ *Average path length overhead*: Path length overhead is the ratio between the length of the shortest path (in terms of number of hops) from the source node to the center point of a destination region, and the actual path length (also in terms of number of hops) that the packet took to reach to that region. Suppose the packet has traversed k hops from the source to the center point of a destination region M_j then the path

length overhead for M_j is $\varepsilon_j = 1 - \frac{d_{shortest}(S,M_j)}{k}$. The average path length overhead is given by $\overline{\varepsilon} = \frac{1}{m}\sum_{j=1}^{m}\varepsilon_j$.

- ◇ *Computation time*: The total execution time it takes to successfully deliver a packet from the source node to all nodes in all destination regions. As a relay node receives a packet, the routing engine implemented in the node is invoked to compute the next relay(s) for the packet, which takes a certain period of time. In order to eliminate the impact of background activities, we run this routing engine for 1000 times and take the average of those time periods as the execution time of the routing engine. This approach for measuring execution time was suggested in [140]. For every nodes in the routing path that the packet has traversed from the source node to all nodes in all destination regions, we record the execution time and add them up to get the total execution time.

The simulation setup consists of nodes randomly placed in a square deployment area. The source node is placed at the center of the area. To achieve varying geographical scale of the network, we vary the width of the deployment area in terms of the number of hops, i.e., $gscale = \frac{AreaWidth}{TransmissionRange}$. We vary the mean number of neighbours per node ($meanNB$) to achieve different network densities. Configuration detail will be presented in the subsequent subsections. Our results for each configuration are the average over 500 simulation runs.

To summarize our results, our proposed RMG algorithm achieves the best performance across all comparison metrics i.e., computation overhead, relay load, and path length overhead, compared to GMR and GGP. Moreover, RMG adapts well to various application requirements in terms of relay load and path length overhead by tuning the value of δ_{th} accordingly, which is not supported by the other algorithms.

3.3.5.1 Choice For δ_{th}

Although δ_{th} is given by the application, we are interested in its best experimental value in an average case. We investigate relay load and average path length overhead for values of δ_{th} ranging from 0.1 to 0.9. The investigation is performed with a $gscale$ of 40, number of destination regions ($nregion$) is 50, and a $meanNB$ of 20 neighbours per node. After 100 simulation runs for each value of δ_{th}, and taking averages, we found that the best experimental value is $\delta_{th} = 0.2$. We will use this value to run our RMG algorithm in comparison with GMR and GGP.

3.3.5.2 Computation Time

We evaluate the computation time (in μs) of the algorithms for three cases, namely varying $nregion$ (see Fig. 3.8), varying $gscale$ (see Fig. 3.9), and varying $meanNB$ (see Fig. 3.10). The general result shows that RMG requires the least computation time among all three

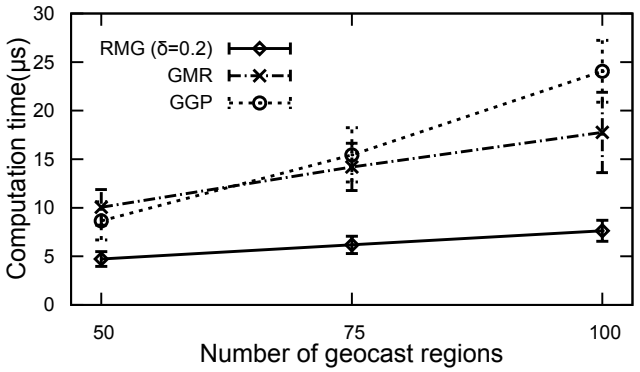


FIGURE 3.8: Computation time evaluation with varying *nregion*

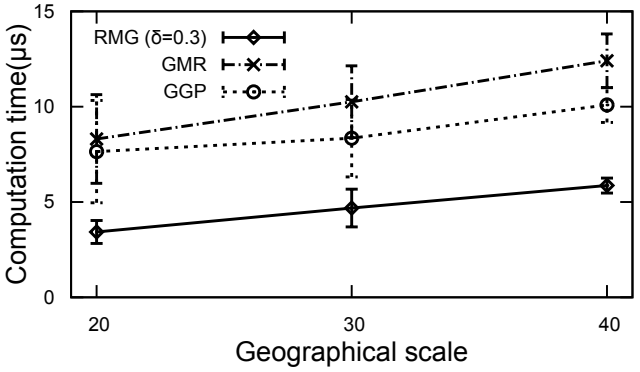


FIGURE 3.9: Computation time evaluation with varying *gscale*

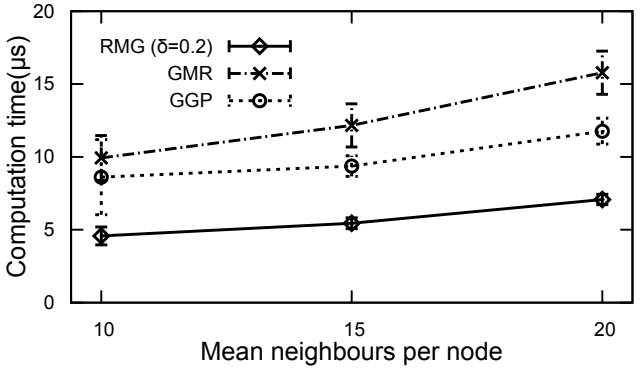


FIGURE 3.10: Computation time evaluation with varying *meanNB*

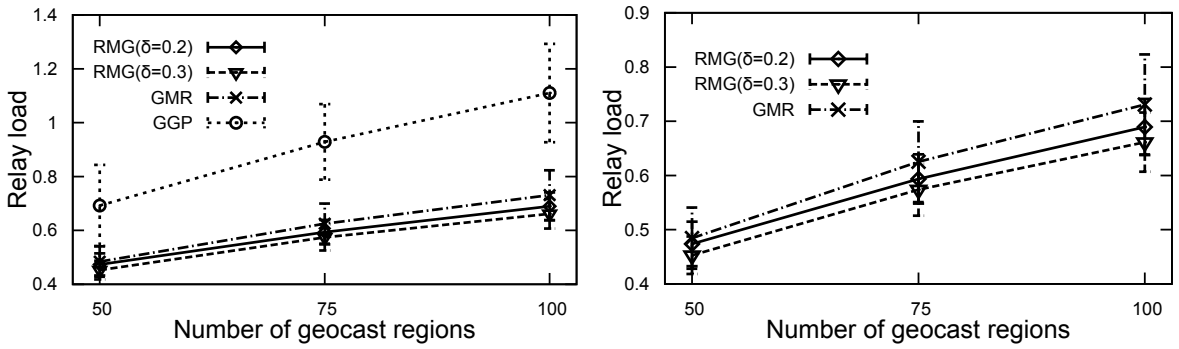


FIGURE 3.11: Relay load evaluation with varying *nregion*

algorithms. The reason why the computation time of RMG is less than that of GGP in spite of the fact that an $O(m^2)$ algorithm (GMGD) is executed is two-fold: (i) First, GMGD benefits from the distribution of the geocast regions: In the best case where all regions reside no farther than δ_{th} from each other, GMGD does not even have to be performed. Moreover, GMGD is only run when a group of regions needs to be divided, which is expected to be rare in the average case; (ii) Second, the computation time of both RMG and GGP depends on the greedy neighbour selection algorithm, because the packet is greedily forwarded by RMG to a division point and by GGP to a Fermat point. Thus the packet's total travelling distance dominates the computation time of the two algorithms. With GGP, the packet has to go through all Fermat points which increases travelling distance for geocast regions at the beginning of the Fermat chain. With RMG, the packet always progresses directly towards a subgroup of destination regions, which results in a much shorter total travelling distance when compared to GGP. The evaluation results in subsection V.D confirm this argument.

The computation time of GMR is expected to be high because it depends on GMR's exhaustive neighbour selection algorithm (an $O(mk \min(m,k)^3)$ algorithm) which is executed every time the data packet is forwarded. This expectation is confirmed in the two cases of varying *gscale* (Fig. 3.9) and varying *meanNB* (Fig. 3.10) as we can see the computation time of GMR grows faster than RMG's and GGP's. There is an exception in the case of varying *nregion* where the computation time of GMR grows slower than GGP's (Fig. 3.8). An explanation for this case is that as *nregion* increases the total travelling distance of the packet with GGP increases to be much larger than with GMR, thus GGP's greedy neighbour selection algorithm is invoked many more times than GMR's exhaustive neighbour selection algorithm.

3.3.5.3 Relay Load

We compare the relay load that each algorithm exerts on the network under varying *nregion* and *gscale*. Specifically, *nregion* is 50, 75, and 100 (see Fig. 3.11-left) and *gscale* is 20, 30, and 40 (see Fig. 3.12-left). Comparison results are in line with our expectation that GGP incurs high relay load on the network because data packets have to go through all the Fermat

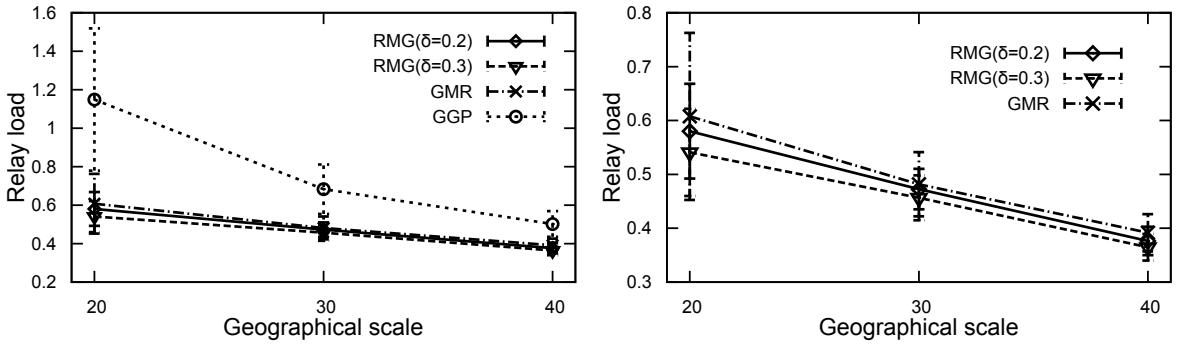


FIGURE 3.12: Relay load evaluation with varying *gscale*

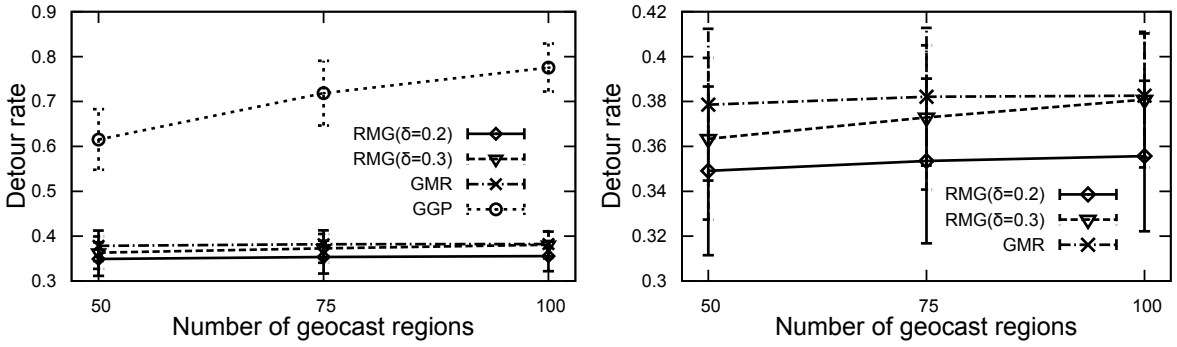


FIGURE 3.13: Average path length overhead evaluation with varying *nregion*

points before they reach all geocast regions, which creates unnecessary detours for regions at the begin of the Fermat chain.

The performance of RMG is slightly better than that of GMR, which surprises us because we expect that the exhaustive neighbour selection algorithm of GMR that ensures that the next set of selected relays is minimal with regard to the cost-over-progress metric, would result in minimal relay load to be exerted on the network. An explanation for this is that our greedy group division algorithm (GMGD) tries to group as many as possible geocast regions into one transmission which reduces network relay load. A closer look at the performance comparison between RMG and GMR can be found in Fig. 3.12-right and Fig. 3.12-right.

3.3.5.4 Average Path Length Overhead

In this experiment we consider the same setup as in the above relay load experiment. The comparison results in Fig. 3.13-left and 3.14-left agree with our expectation that there will be a decent superiority of path length overhead of RMG over GMR's, and a huge jump from GGP's. This is because RMG restricts path length overhead according to δ_{th} while GMR's main goal is to maximize cost-over-progress ratio which does not necessarily decrease path length overhead. The GGP algorithm only forwards data packets to Fermat points in spite of the actual distribution of geocast regions, thus creating extra path length overhead. A zoomed version of Fig. 3.13-left and Fig. 3.14-left is given in Fig. 3.13-right and 3.14-right, respectively.

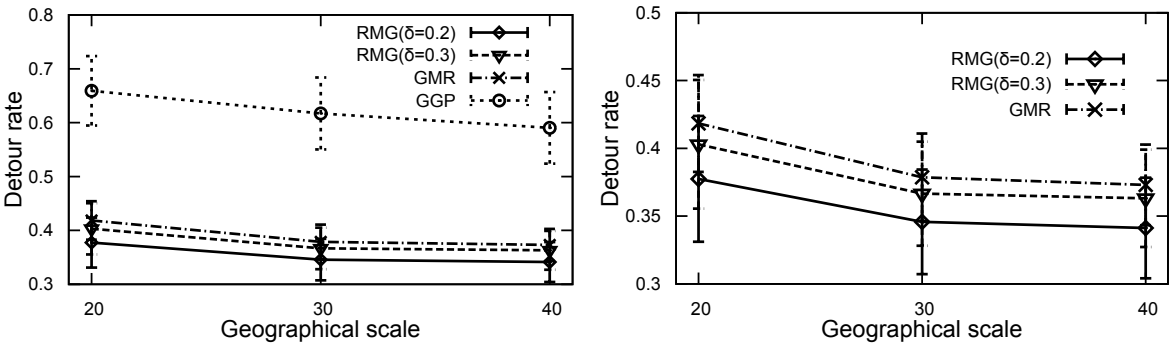


FIGURE 3.14: Average path length overhead evaluation with varying *gscale*

3.3.5.5 Flexibility

We consider the effect of value of the parameter δ_{th} on the performance of RMG. If we take a closer look at the relay load and average path length overhead evaluation results given in Fig. 3.11-right and Fig. 3.12-right as well as in Fig. 3.13-right and Fig. 3.14-right, we can see that increasing δ_{th} reduces relay load but at the same time increases path length overhead and vice versa. More specifically, with $\delta_{th} = 0.3$, RMG achieves further reduced relay load compared to that of GMR but at the same time still achieves lower path length overhead than that of GMR in case of varying *gscale* (see Fig. 3.14-right). This observation shows that RMG is flexible in tuning the value of δ_{th} to adapt to varying application need in terms of network relay load or path length overhead. For a particular application with specific relay load or path length overhead requirements, an appropriate value for δ_{th} that best fits the application requirements could be selected.

3.3.5.6 Non-UDG Wireless Link Model

We elaborate here our statement in Sec. 3.3.3.1 that our proposed RMG algorithm will still work with non-UDG wireless link models. We mentioned in Sec. 3.2.1 that the work in [112] proposes the RIM wireless link model, that is based on empirical data from real sensor devices. The model introduces realistic Degree of Irregularity (DOI) [141] values for simulation purpose thus it provides a good approximation of radio irregularity for simulations. We use this model as a relaxation for our UDG assumption in Sec. 3.3.3.1. Each node determines its transmission region using the RIM model (with $DOI = 0.004$) at the beginning of a simulation, and uses this transmission region throughout the simulation. This means each node has a completely different irregular geometric shape of the transmission region. An example of such a shape is given in Fig. 3.2-(c).

The evaluation results given in Fig. 3.15-left, Fig. 3.16-left, Fig. 3.17-left, and Fig. 3.18-left are the average of 500 simulation runs. As we observe, RMG and GMR are not much affected by the RIM model as they both achieve a similar performance of relay load and average path length overhead when compared to the UDG model (see previous subsections). However, a

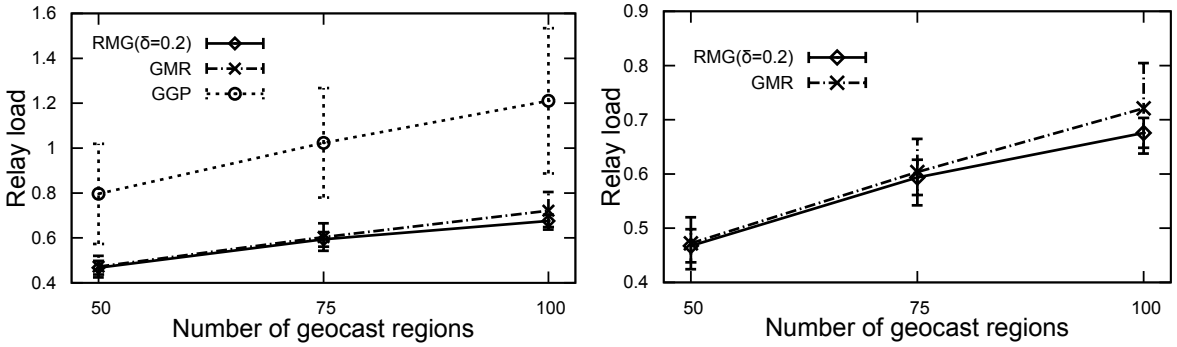


FIGURE 3.15: Relay load evaluation with varying *nregion* (using RIM model)

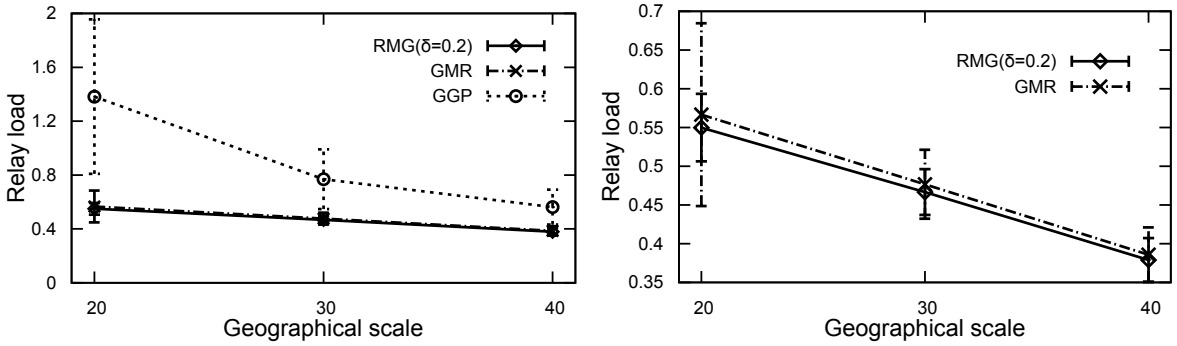


FIGURE 3.16: Relay load evaluation with varying *gscale* (using RIM model)

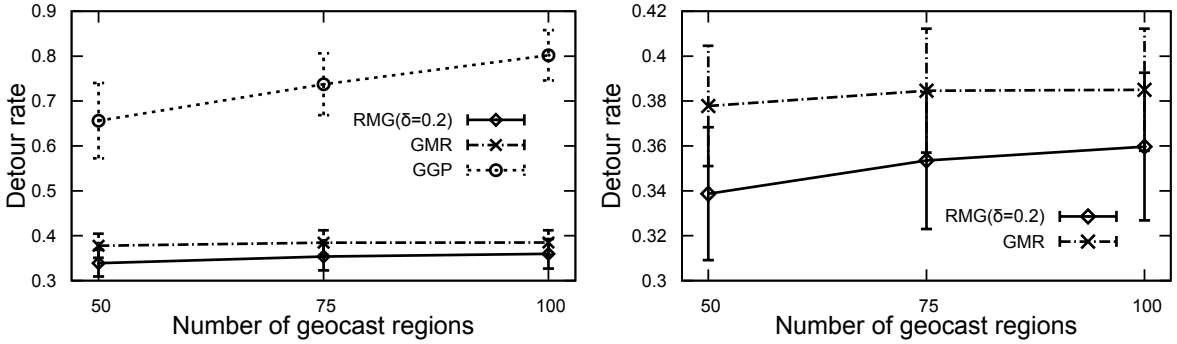


FIGURE 3.17: Average path length overhead evaluation with varying *nregion* (using RIM model)

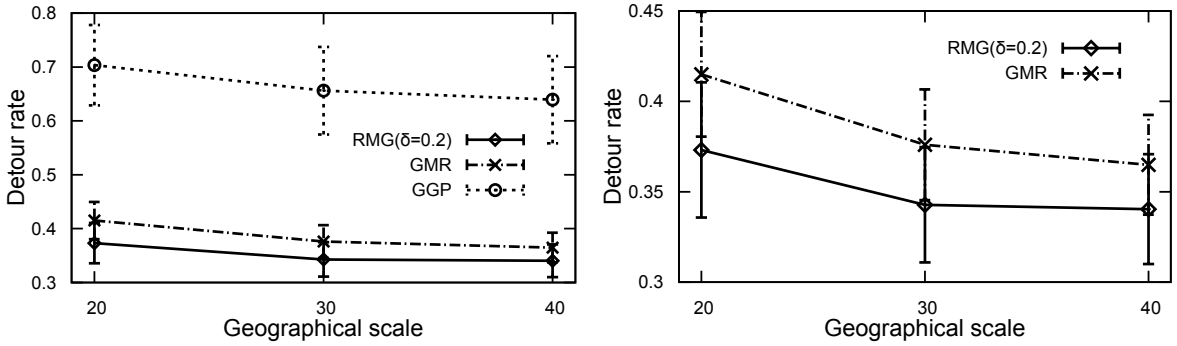


FIGURE 3.18: Average path length overhead evaluation with varying *gscale* (using RIM model)

closer look at the relay load comparison between RMG and GMR is given in Fig. 3.15-right and Fig. 3.16-right showing that RMG still outperforms GMR under the RIM model.

In contrast, GGP exhibits a performance degradation in both relay load and path length overhead under the RIM model. We know that in GGP the data packet must traverse the main route connecting all Fermat points in order to reach all geocast regions. The irregularity of the transmission range of the nodes on that route makes the route become more zigzag and longer than under the UDG model, thus incurring extra relay load and path length overhead.

3.3.6 Conclusion

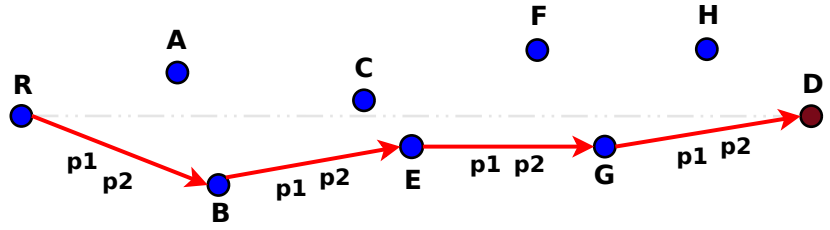
We have presented a novel multi-region geocast routing algorithm called Recursive Multi-region Geocasting (RMG), which addresses the problem of delivering data from a source to multiple remote geocast regions in large-scale networks of IoT devices. We compared the performance of RMG with two state-of-the-art protocols, namely GMR and GGP, using simulation. The comparison has shown that RMG outperforms the other protocols in all comparison metrics including computation time, network relay load, and transmission latency overhead. Furthermore, RMG is flexible with respect to application needs due to its capability of tuning the parameter δ_{th}. A performance evaluation of RMG under a more realistic wireless model [112] reveals that our proposed algorithm also works well with irregular transmission regions.

3.4 Stochastic Routing in the IoT

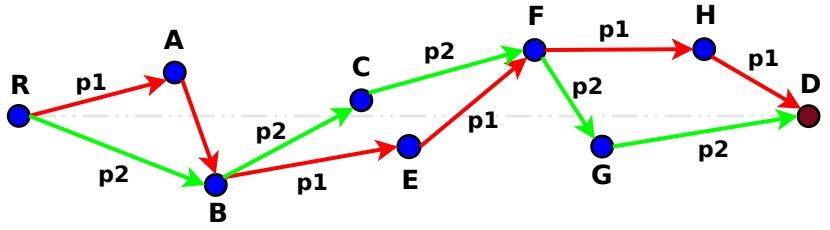
In this section we present a routing algorithm called "Stochastic Forwarding-based Routing" (SFR), which follows the geographic routing approach. The routing algorithm is targeted to the routing property $P2$ (see Sec. 3.1.2), i.e., to maintain a fair distribution of the routing load in terms of energy saving across the network, so that the life time of the network can be increased.

3.4.1 Motivation

Conventional geographic routing protocols in networks of IoT devices (e.g., WSN) usually send data packets over a single *deterministic* routing path from a source node to a destination node. A routing path is deterministic if the relay nodes included in the path were selected deterministically using a forwarding technique. An illustration of a deterministic routing path is given in Fig. 3.19-(a), where the GEDIR forwarding technique always results in the routing path $RP_{B,E,G}$ for 2 data packets ρ_1 and ρ_2 that need to be sent from S to D. This is due to the selection of the next relay for every ρ_i, at every intermediate relay nodes being deterministic following the GEDIR principle.



(a) Deterministic routing path



(b) Different routing paths for different data packets

FIGURE 3.19: Load balancing vs. Shortest path usage.

Iterative use of the same routing paths (e.g., $route_{B,E,G}$) increases the stress on the relay nodes that lay on the paths (e.g., nodes B, E, and G), and eventually uses up the energy budget of the nodes. This may cause partitions in the network since some of those relay nodes may be the only links between different parts of the networks, rendering the network useless as data packets cannot be exchanged between nodes in different partitions. For example, in a forest fire monitoring application, the detection of a fire may not be delivered to the base station due to network partitions.

In order to mitigate this problem, we propose to use different routing paths for sending different data packets, therefore resulting in a fairer routing load distribution across the network. Fig. 3.19-(b) illustrates our proposal, where the two data packets ρ_1 and ρ_2 are sent using two different routing paths $route_{A,B,E,F,H}$ and $route_{B,C,F,G}$. The routing load is distributed to 7 nodes A, B, C, E, F, G, H instead of 3 nodes B, E, G as shown in Fig. 3.19-(a).

The use of multiple routing paths, however, has the drawback that data packets may be forwarded along routing paths that are longer than the shortest routing path (e.g., $route_{A,B,E,F,H}$ is longer than $route_{B,E,G}$ in Fig. 3.19-(b)). This results in a longer delay in delivering data packets, which should be avoided in time-critical applications. For example, in the above forest fire monitoring application, the detection of a fire must be reported to the base station within a hard time bound of 2 seconds, which implies that the data packets should be sent over shortest or, at least, near-shortest routing paths, whenever it is possible.

Our goal in designing SFR is to find a trade-off between these two apparently conflicting objectives of load balancing and shortest path usage. To achieve the former, we follow the random forwarding technique (see Sec. 3.2.2.1) to randomly select the next relay for each data packet. We call this approach the *randomized routing* approach. To achieve the latter, we design the neighbour selection process such that the resulting routing paths do not deviate

too much from the shortest path. In the following, we present related work and the details of our SFR algorithm.

3.4.2 Related Work

Randomized routing is actually a well-studied approach in the literature, however it is mostly used in non-geographic-routing domains (e.g., flooding-based, state-based routing, etc). Many randomized routing protocols, such as [142–145], mostly consider a scenario where the location of destination is not known and employ a pure random walk to discover the destination. Among those, [144] provides an analysis of pure random walks on sensor networks with regular deployment, such as triangular, hexagonal, or square-based topologies. In [142], the authors also consider unbiased random walk on a regular deployment of nodes, forming a hexagonal lattice pattern.

Another use of unbiased random walks is presented in [143] in order to detect outlier data in sensor networks. Zhang et al. [145] utilize the random walk approach in a rather different context in sensor networks, i.e., in order to enhance the source-location privacy by introducing *phantom* sources in between the actual data source and the sink. A random walk is initiated at the actual source and terminated after a predefined hop count, where the phantom source is created. Data is then sent to the sink by the phantom source using a given routing protocol. An overview and comparison of different random walk strategies for ad hoc networks is given in [146], which covers random walk with memory, random walk with look-ahead, random walk using highest degree, random walk proportional to the degree, and random walk using minimum link weight. None of these approaches consider a given destination location, hence there is no bias toward a target in the probabilities of the random walk.

Geographical Random Forwarding (GeRaF) [147] introduces a new concept of *receiver contention* for packet forwarding. In this scheme, the relaying node does not specify the next hop but the receiving neighbors decide which one should relay the packet based on the location information, similar to the greedy approach. The paper presents an analysis of this scheme, but does not fully address how contention among receivers is resolved in a distributed manner. Probabilistic Geographic Routing (PGR) [148] is similar to our approach in the sense that it assigns probabilities to a few candidate relaying nodes, but the assignment is uniform along the routing path, unlike our approach where the probability assignment scheme (bias) changes as packets get closer to the destination. Barrett et al. introduce a family of routing protocols based on probabilistic flooding in [149]. Our approach can be viewed as an additional member to this family, albeit as one not utilizing flooding in order to ensure low power operation.

3.4.3 Assumptions and Approach

3.4.3.1 Network Model

We consider a large multi-hop wireless network consisting of m stationary nodes, where each node has a fixed circular transmission region, determining the set of nodes it can communicate with directly, i.e., we follow the UDG wireless link model. We assume that all nodes are aware of their geographical locations at network deployment time (e.g. through GPS receivers), or shortly after deployment by employing a distributed location discovery algorithm, such as in [105]. The locations of the neighbours of a node can be obtained using a simple Hello protocol. Moreover, the location of the destination node is assumed to be known to the source node.

3.4.3.2 The Stochastic Forwarding Approach

Existing geographic routing protocols exploit the fact that the shortest routing path between a source node and a destination node in a network gets increasingly closer to the straight line connecting the two nodes in the Euclidean space as the node density increases. In networks of IoT devices where limited battery power is a vital resource, frequent use of such "straight line" routing paths would quickly exhaust the energy of the relay nodes that lie on the paths. Thus, critical applications should employ a load balancing strategy for relaxing the energy consumption in order to maximize the network life-time, while keeping the length of the routing paths for active flows at a reasonable level compared to the shortest routing path.

In this thesis, we present a novel forwarding approach called *stochastic forwarding* (SF), that aims at fairly distributing the routing load of a transaction between a pair of source and destination among network nodes such that the network life-time is maximized. We model the movement of each packet sent during the transaction as a random walk [150], whose transition behavior is influenced by intermediate relay nodes along the routing paths from the packet's source to destination. Upon receiving a packet to be forwarded, a relaying node assigns transition probabilities to all of its 1-hop neighbors and randomly forwards the data packet to a neighbor based on those probabilities.

The novelty of our approach is the assignment of the transition probabilities of the random walk, which we formulate as a set of requirements:

- *Load distribution*: In the first steps of the random walk, the data packet should tend to discover many different routing paths, rather than biasing too much toward the shortest path.
- *Convergence*: In later steps, the bias should be increased toward the nodes that are closer to the destination.

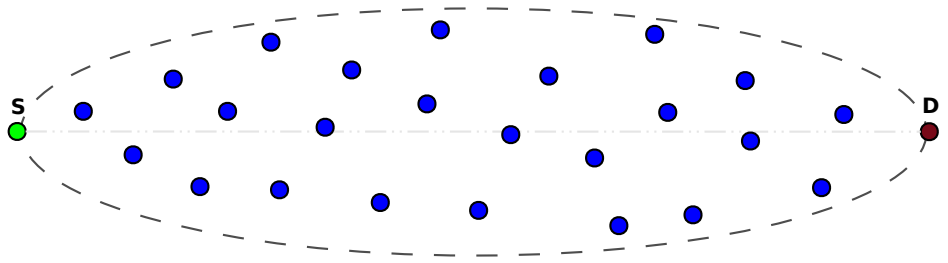


FIGURE 3.20: The stochastic forwarding approach.

An illustration of our approach is given in Fig. 3.20, where the routing of multiple data packets from the source S to the destination D results in a routing load distribution among nodes that are located inside the ellipse region whose transverse diameter is the line segment $\overline{SD}$. Note that this set of requirements describes a generic stochastic forwarding method that can be realized in different ways, according to the particular assignments of the transition probabilities. In this thesis we present one such assignment (see Sec. 3.4.4) and evaluate its performance. Note that, hereafter we interchangebly use the two terms "transition probability" and "forwarding probability" as they convey the same assignment of probability, and only differ semantically in that "transition" is used for random walk theory while "forwarding" is used for routing.

3.4.4 A Heuristic for the SF Approach

We conceptualize and demonstrate our SF approach with a heuristic for assigning the forwarding probabilities, which we call the *SF heuristic 1* (SFH1). Consider a source or relaying node R in a transmission session with packets destined to a destination D. Let $d_E(X, Y)$ represent the Euclidean distance between the two nodes X and Y and π_R the set of neighbours of node R. For every neighbour N_i of R, i.e., $N_i \in \pi_R$, we assign a weight w_{N_i} such that

$$w_{N_i} = \frac{d_E(R, D) + \max\limits_{N_k \in \pi_R} \{d_E(R, N_k)\} - d_E(N_i, D)}{d_E(N_i, D)} \tag{3.10}$$

Note that, it can be easily proven, using the triangle inequality, that $\forall N_i, w_{N_i} > 0$. The weights are then normalized according to

$$p_{N_i} = \frac{w_{N_i}}{\sum\limits_{N_k \in \pi_R} w_{N_k}} \tag{3.11}$$

We use the normalized weights as the forwarding probabilities to be assigned to the neighbours of node R.

Fig. 3.21 illustrates the probability assignment for two nodes, the source node S, and a relay node R that is close to the destination D. Each node has three neighbors at identical relative positions. We observe that all neighbours of the source are assigned relatively similar forwarding probabilities, i.e., the difference among them is relatively small. In contrast,

FIGURE 3.21: The SFH1 heuristic for assigning forwarding probabilities.

such difference is much greater among the forwarding probabilities that are assigned to the neighbours of R. This observation illustrates the first two requirements of our approach, i.e., load balancing and convergence. The third requirement, greedy forwarding, is also observed. For both S and R, neighbours that are located closer to D are assigned with greater forwarding probabilities, i.e., $p_B > p_A > p_C$ and $p_F > p_E > p_G$.

3.4.5 The Stochastic Forwarding-based Routing (SFR) Algorithm

We present now the SFR algorithm (see Algorithm 3), given that an assignment for our SF approach has been derived, e.g., the above SFH1 heuristic. The algorithm is implemented on every nodes in the network and uses only local information, i.e., the location of neighbour nodes, for computing the forwarding probabilities.

Algorithm 3 The SFR algorithm run on a relay node R holding the data packet p.

1: **if** $D \in \pi_R$ **then**
2: Forward p to D.
3: **else**
4: **for** each $N_i \in \pi_R$ **do**
5: Calculate p_{N_i} using Eq. (3.11).
6: **end for**
7: $rand\_val \leftarrow$ a random value $\in [0, 1]$
8: $bound_1 \leftarrow 0$
9: $bound_2 \leftarrow 0$
10: **for** each $N_i \in \pi_R$ **do**
11: $bound_2 \leftarrow bound_1 + p_{N_i}$
12: **if** $rand\_val \in [bound_1, bound_2]$ **then**
13: Forward p to N_i.
14: Terminate the process.
15: **else**
16: $bound_1 \leftarrow bound_2$
17: **end if**
18: **end for**
19: **end if**

3.4.6 An Analytical Framework

We model our stochastic forwarding approach using the Markov chain theory [151] so that its behaviour can be theoretically studied and analyzed.

A Markov chain can be specified by a set of *states*, $C = \{c_1, c_2, \ldots, c_n\}$, and a *transition probability matrix* $\mathbb{P}$, whose entry $p_{ij} \in \mathbb{P}$ represents the probability of the chain to be in state c_j at the next step given that it is currently in state c_i. A state c_i of a Markov chain is called *absorbing* if it is not possible to leave it once reached (i.e., $p_{ii} = 1$). A Markov chain is *absorbing* if it has at least one absorbing state and if from every state it is possible to reach an absorbing state. In an absorbing Markov chain, a state which is not absorbing is called a *transient state*.

Given the above definitions, we model the routing process of a data packet between a source S and a destination D as an absorbing Markov chain, where network nodes represent the set of states of the Markov chain, the destination D corresponds to an absorbing state c_d, and the source S corresponds to a transient state c_s. The expected number of hops for the data packet to travel from S to D corresponds to the number of steps for the Markov chain, starting from state c_s, to be absorbed at c_d. For the routing process, we construct the transition probability matrix $\mathbb{P}$ of the Markov chain by assigning each entry p_{ij} the probability for the data packet to be forwarded from a relay node I to another relay node J.

In order to compute the number of steps for the modelled absorbing Markov chain to be absorbed, we first write the matrix $\mathbb{P}$ in its *Canonical form* as in Eq. (3.12), where $\mathbb{I}$ is the identity matrix, $\mathbb{Q}$ is an $(m-1) \times (m-1)$ matrix, $\mathbb{R}$ is a column vector with $m-1$ entries, and m is the number of nodes (or states).

$$\mathbb{P} = \begin{pmatrix} \mathbb{Q} & \mathbb{R} \\ 0 & \mathbb{I} \end{pmatrix} \tag{3.12}$$

We, then, compute the *fundamental matrix* $\mathbb{N}$ of $\mathbb{P}$ as

$$\mathbb{N} = (\mathbb{I} - \mathbb{Q})^{-1} \tag{3.13}$$

According to the Markov chain theory, an entry n_{ij} of the fundamental matrix $\mathbb{N}$ represents the expected number of times the Markov process is in the transient state c_j, given that it started in state c_i. Thus, adding all entries in the row i of $\mathbb{N}$ yields the expected number of steps required before the Markov chain is absorbed, given that it started in state c_i. This addition can be written as

$$\mathbb{T} = \mathbb{N} \times c \tag{3.14}$$

where c is a column vector whose all entries are 1. The entry t_i of $\mathbb{T}$ gives the expected number of steps until the Markov Chain reaches the absorbing state c_d.

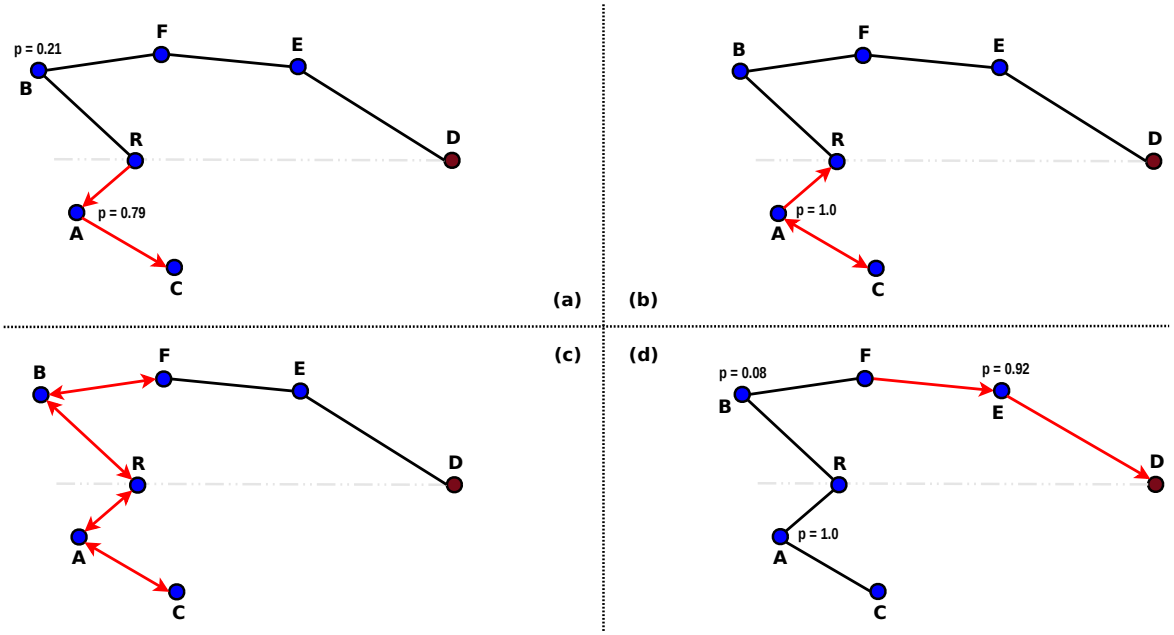


FIGURE 3.22: Packet delivery guaranteed.

Therefore, the expected number of hops for data packets to be routed from the source S to the destination D is given by t_s and the expected number of forwards that a relay node J has to perform in the routing process is given by n_{sj}. This Markov chain-based framework allows for performing analytical evaluation of any practical assignment of forwarding probabilities, as well as any randomized routing approach (see Sec. 3.4.2).

3.4.7 Guaranteed Packet Delivery

An interesting side effect of our approach is that packet delivery can be guaranteed, although in certain cases it may take very long time. Network voids may emerge in the network especially when there is a sparse deployment of nodes. In some cases this may result in a poor selection of routing paths by a forwarding technique, which is a common problem for geographic routing protocols. With pure greedy forwarding techniques, the destination may not even be reached unless there is a recovery mechanism to overcome the voids problem (e.g., the GFG technique). In our approach, non-zero forwarding probability is assigned to all neighbours ensures that the data packet will arrive (probably after very long time) at the destination, as long as the network is connected.

To illustrate this property, we look at Fig. 3.22. The relay R is a dead-end node as it is closer to the destination node D than its neighbours A and B. Due to our forwarding approach that the forwarding probabilities are distributed to all neighbours such that sum of them is exact 1, both A and B receive non-zero, significant values of 0.79 and 0.21, respectively. Since $p_A > p_B$, A is more likely selected as the next relay of the data packet p, which leads to the dead-end node C (see Fig. 3.22-(a)). From C, however, A is selected as the next relay with forwarding probability of 1. At this point, there might be a temporary loop where p

traverses back and forward between A and C. The loop will eventually end because of the random neighbour selection, and p is forwarded to R (see Fig. 3.22-(b)). This process goes on until p reaches F (see Fig. 3.22-(c)). At F, since $p_E = 0.92 >> 0.08 = p_B$, it is very likely that p is forwarded to E, and eventually to D (see Fig. 3.22-(d)).

The guaranteed packet delivery property can be easily theoretically proven using the above Markov chain-based analytical framework. We model the network as an absorbing Markov chain, with the destination node being the absorbing state and other nodes being transient states. Due to the network being connected and SFH1, every relay nodes will be assigned with a non-zero forwarding probability, which means the data packet *always* be forwarded, until it reaches the destination.

The guaranteed packet delivery property does not only hold true for SFH1, but for any SF heuristic that assigns a non-zero forwarding probability to all neighbours of a relay node. In general, this means our SF approach can guarantee packet delivery, although it may take long time in certain cases depending on specific forwarding probability assignments.

3.4.8 Evaluation

In order to evaluate our SF approach, we define the following two metrics that correspond to our two design objectives of load balancing and shortest path usage. There is an apparent trade-off between these two objectives. On the one hand, repeated use of shortest paths provides fast routing with a poor forwarding load balancing. On the other hand, using many alternative paths provides a fairer load distribution while requiring longer paths.

As radio communications constitute the major consumption of energy in networks of IoT devices, we use the number of forwards that a relay node R_i has performed as a measure of the energy consumption, and denote it as c_{R_i}, where $i = 1..m$ and m is the number of nodes in the network. Given that a data packet p has been successfully delivered from a source to a destination, the average energy consumption of all nodes in the network except the destination is denoted as c_{avg} and is given by

$$c_{avg} = \frac{1}{m-1} \sum_{i=1}^{m-1} c_{R_i} \tag{3.15}$$

The standard deviation from the average energy consumption is given by

$$\delta_c = \sqrt{\frac{1}{(m-1)} \sum_{i=1}^{m-1} (c_{R_i} - c_{avg})^2} \tag{3.16}$$

We use δ_c as the measure of the load balancing property of a routing protocol, since the smaller δ_c, the smaller the difference between the number of forwards that network nodes have performed.

We define how efficient p has been routed between a pair of source and destination as the number of relay nodes required to deliver p, which should be small to be efficient. To measure this efficiency, which we denote as e, we use the ratio between the length of the actual routing path that has been used and the length of the shortest path. That is

$$e = \frac{l_{ap}}{l_{sp}} \tag{3.17}$$

where l_{ap} and l_{sp} are the length of the actual routing path and the shortest routing path, respectively. The value of l_{sp} can be computed using the Dijkstra algorithm.

For an integrated approach that addresses both objectives, we introduce a linear combination of the two metrics, controlled by a parameter $\alpha \in [0, 1]$. For a given value of α, a routing protocol should aim at *minimizing* the objective function f as given below

$$f = \alpha \cdot \delta_c + (1 - \alpha) \cdot e. \tag{3.18}$$

3.4.8.1 Near-Optimal Forwarding Probability Assignment

In this section we present a centralized technique to derive the forwarding probability assignment that minimizes the objective function f, i.e., a near-optimal assignment. This optimal solution is helpful as it provides a means to quantify the performance of an SF heuristic such as the SFH1 heuristic.

Given a network, a data source S, a destination D, an assignment of the forwarding probabilities for each node of the network such that the sum of all probabilities is exactly 1, and a value for α, we can theoretically compute the objective function f in Eq. (3.18), using the Markov chain-based analytical framework that is presented in Sec. 3.4.6.

To do so, the network is modeled as an absorbing Markov chain with D being the only absorbing state. Since the entry t_S of the matrix $\mathbb{T}$ contains the expected length of the routing path between S and D, the term l_{ap} in Eq. (3.17) can be replaced by t_S. Moreover, the expected number of forwards that a relay node R in the network has to perform can be obtained from the entry n_{sr} of the fundamental matrix $\mathbb{N}$, thus the term c_{R_i} in Eq. (3.15) can be replaced by n_{sr}.

Using these replacements, we develop a *genetic algorithm* to compute the optimal assignment that minimizes Eq. (3.18). We define a *gene* g_R as the set of assigned forwarding probabilities for the relay node R such that the sum of these probabilities is exactly 1. Thus, g_R corresponds to the row vector $\overrightarrow{p_{r1}, .., p_{rm}}$ in the transition probability matrix $\mathbb{P}$ of the Markov chain model. $\mathbb{P}$ is, therefore, the set of the genes of all nodes in the network, which we call a *genome* (or an individual) according to the language of genetic algorithm. In this sense, finding the optimal forwarding probability assignment is equivalent to finding the optimal genome.

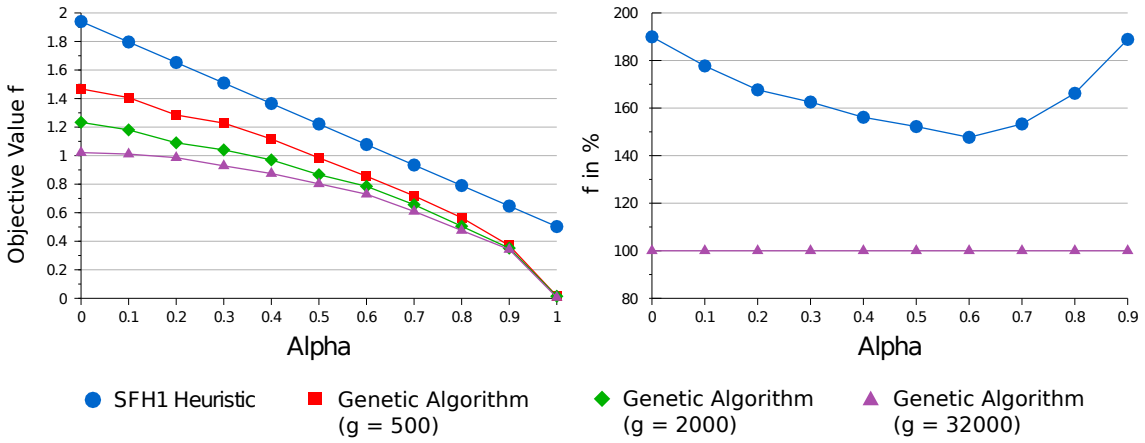


FIGURE 3.23: Performance comparison between SFH1 and the genetic algorithm.

Our genetic algorithm consists of the following steps:

1. *Initialization:* Create the first generation of genomes by randomly generating their genes.
2. *Comparison and Selection:*
   - Compare the quality of all genomes using the objective function f in Eq. (3.18).
   - Select a specified number of genomes for the reproduction phase based on the comparison.
3. *Reproduction:*
   - Create a new generation of genomes by combining the genomes of the parent generation.
   - Apply some random modifications (mutations) to the genes of the new generation.
4. *Iteration:* Repeat steps 2 and 3 until the termination condition is reached, which in this case is the number of iterations.

3.4.8.2 Near-Optimal Assignment vs. SFH1

In this section we compare the performance of the SFH1 heuristic with a near-optimal forwarding probability assignment obtained by our genetic algorithm. Since the genetic algorithm requires high computation time for obtaining near-optimal results, we use a moderate network size of 100 nodes for the comparison. Nodes are randomly scattered in a unit square area with a transmission range of 0.2 units. The source and destination nodes are selected such that they are apart from each other and located at the opposite edges of the network. The source sends 100 data packets to the destination. Using this configuration, each result data point is the average of 100 generated networks.

Fig. 3.23 shows the comparison results for varying α values over the range of [0,1]. Note that α does not affect the operation of the SFH1 heuristic, but the objective function value is naturally affected by different values of α. For the genetic algorithm, each selected α value requires a separate optimization process. The parameter g in the figure represents the number of generations used for the genetic algorithm. The diagram on the left shows the absolute value of f depending on the selected value for α. It also illustrates the convergence of the genetic algorithm results towards the optimal solution, which is still not reached after 32000 generations, but is close enough to provide good benchmark values comparable to the optimal solution. Note that, the optimal assignment which linearly minimizes f *cannot be obtained* in practice due to the contradiction between our two design objectives.

The diagram on the right shows the same result values, but normalized with respect to the best known solution, such that the results of the genetic optimization with $g = 32000$ are fixed at 100%. We observe that the performance of the SFH1 heuristic is at its best compared to the near-optimal solution when α is around 0.5 and 0.6. This demonstrates that the SFH1 heuristic is effective in simultaneously addressing both load balancing and shortest path usage. Furthermore, it is important to note that, for all values of α, the SFH1 heuristic always achieves a performance that is less than twice that of the best known solution, which can only be obtained using the centralized genetic algorithm.

3.4.8.3 Performance Evaluation of SFH1

We now investigate the performance of the SFH1 heuristic in more detail using different network topologies with respect to parameters such as network size, node density, and network diameter. Two scenarios are evaluated for the results in this section:

- A varying number of nodes with fixed network dimensions to analyze the influence of node density.
- A growing network size (in terms of both network dimensions and the number of nodes) while keeping node density constant, in order to investigate the scalability.

In the first scenario, we use a unit square as the network dimensions while the number of nodes is varied between 25 and 800. The second scenario uses a fixed node density with the network dimensions being varied by setting the edge size to values between 0.5 and 2.5 units (which results in network sizes between 25 and 625 nodes). In both scenarios, every data point is an average over 100 random networks. For each random network, the source node and the destination node are selected apart from each other so that the distance between the nodes scales with the network diameter, and the source sends 100 data packets to the destination.

Fig. 3.24 shows the performance of the SFH1 heuristic for the first scenario, i.e., networks with constant dimensions and varying number of nodes. For small node densities (number

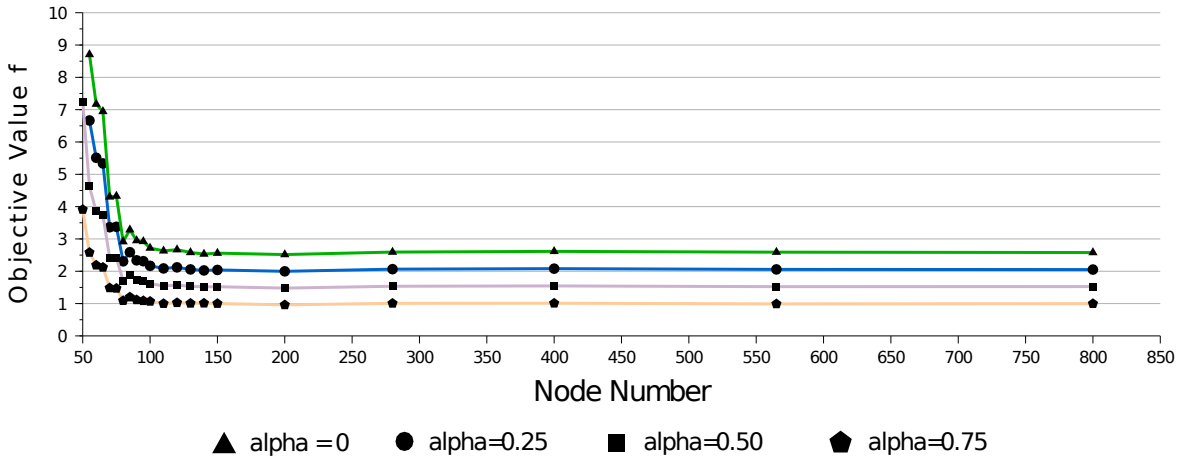


FIGURE 3.24: SFH1 vs. Increasing node density in a fixed network area.

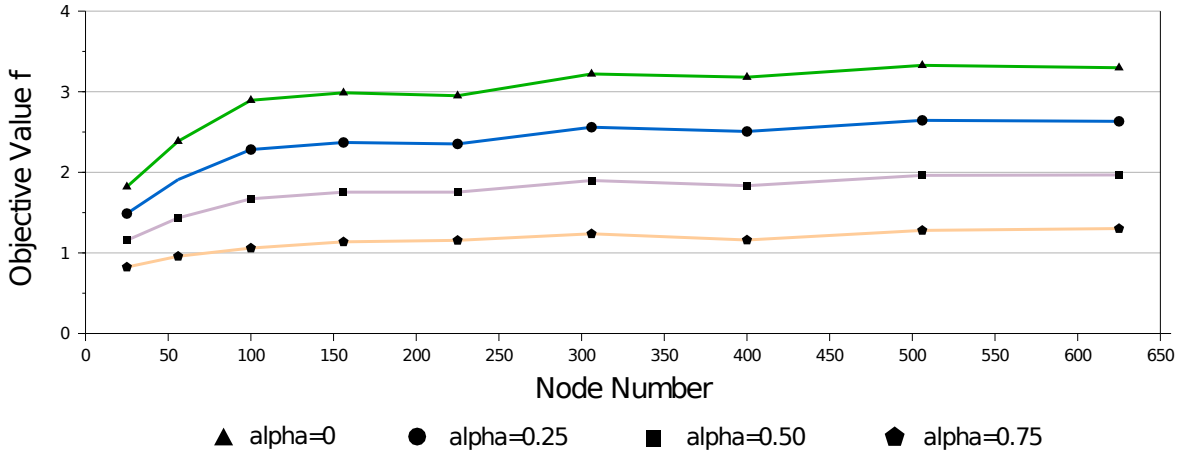


FIGURE 3.25: SFH1 vs. Different network sizes and a fixed node density.

of nodes < 100), we observe a rather poor performance. However, when the node density reaches some threshold (number of nodes is in between 100 and 150), the performance of the heuristic stabilizes and stays constant regardless of the network size or density. This result suggests that the SFH1 heuristic requires a minimal degree of connectivity in order to perform well. For networks which are too sparse, a deterministic recovery forwarding technique can be adopted to bypass network voids.

Fig. 3.25 show the performance of SFH1 while we proportionally increase the network dimensions and the number of nodes together to maintain a certain constant node density. At a first glance, we see that SFH1 maintains relatively stable performance as the value of the objective function f does not increase much with the increasing of the number of nodes and the varying value of α. At a closer look, we can observe that SFH1 performs very well in terms of load balancing ($\alpha = 0.75$), performs good with both the load balancing and shortest path usage objectives ($\alpha = 0.5$), and performs relatively good in terms of shortest path usage ($\alpha = 0.25$ and $\alpha = 0$).

3.4.9 Conclusion

We have introduced a novel geographic routing algorithm for the IoT, that is targeted to two conflicting objectives: (i) fairly distributing the routing load across a network to improve the network life time; and (ii) minimizing the number of routing hops for fast data packet delivery. At the heart of the algorithm, we introduced a new stochastic forwarding approach and proposed a heuristic to implement this approach. We presented an analytical framework based on the Markov chain theory for modeling and analyzing our approach. The framework can also be used generally for analyzing any other randomized routing protocol. Using this framework we proved that our stochastic forwarding approach guarantees packet delivery. To evaluate our work, we formulate the two design objectives into an integrated objective function which should be minimized. We proposed and implemented a genetic algorithm to find near-optimal forwarding probability assignments that minimize this objective function. Using the best near-optimal assignment, we found that our heuristic performs well for both the design objectives. Furthermore, we provided numerical results for the heuristic, demonstrating that our proposed heuristic scales well with respect to network size and density.

3.5 Summary

We presented in this chapter a background review on the generic routing problem and the specific geographic routing approach in the IoT. After that, we proposed two geographic routing algorithms for the IoT, namely the Recursive Multi-region Geocast (RMG) and the Stochastic Forwarding-based Routing (SFR) algorithms. While RMG is more scenario-driven in the sense that it is targeted to a class of routing scenarios where the geographical area on which the network is deployed is (very) large, SFR is more network-driven as it is designed to improve routing load balancing, thus prolonging the operational life time of the network. The two algorithms can be combined, since SFR can be used as a greedy forwarding technique by RMG to forward data packets from one division point to another, creating an efficient and load-balanced routing protocol for large-scale IoT application scenarios.

Chapter 4

Searching The Real World Via The IoT

One of the most essential and mostly used services in the traditional Web is *search*, since it enables Internet users to quickly find information of interest among the massive amount of information available on the Web. Given a description of what a user wants to find, a search service quickly returns matching information, which can be of any form such as web-pages, multimedia, services etc. Similarly, in the WoT where billions of Things are connected to the Internet and publish their data (i.e., real-world states perceived by their embedded sensors) on the Web in real time, we believe that *sensor search* will be an essential service for many IoT applications. A sensor search service that allows for finding Things (i.e., their embedded sensors) with certain properties (i.e., the states of the physical world measured by the embedded sensors) in real time would enable the users to search the real world for objects and places via the IoT.

For example, in order to efficiently reproduce a newly developed type of corn, an agriculture scientist may want to find places with similar climatic conditions to the current place where he has successfully planted the new type of corn during the last agricultural season. Since climatic sensory data are published in the WoT, the scientist can simply search for climatic sensors whose published data are similar to the published data of the climatic sensor at the current place. The location of the climatic sensors returned by the search service is what the scientist is looking for. In another example, to find an empty parking spot in his proximity, a car driver can simply search for near-by parking-spot sensors whose output at the moment is "empty". For later reference, we label the first and second applications as $Ex1$ and $Ex2$, respectively.

In these applications, the underlying problem is *finding sensors in the WoT whose output measurements match certain search criteria.* This chapter is devoted to addressing this problem[1]. To focus our work on the core sensor search problem, we assume in this chapter

[1] This chapter is based on our work in [6] and [7].

that any search query for Things, e.g., places in $Ex1$ or parking spots in $Ex2$, can be translated to a search query for embedded sensors, e.g., climatic sensors or parking-spot sensors.

This chapter is structured as follow. In Sec. 4.1, we discuss the general sensor search problem in the context of the IoT and introduce a specific sensor search problem that we consider in this thesis. In Sec. 4.1.3, we present a set of challenges to our specific sensor search problem and derive a set of requirements for solving it. We outline a generic architecture and an approach in Sec. 4.1.4, given those challenges and requirements. In the next two sections, we present two sensor search algorithms for the sensor search problem in the WoT. In particular, the sensor similarity search algorithm is presented in Sec. 4.2, and the content-based sensor search algorithm is presented in Sec. 4.3. Finally, we conclude the chapter in Sec. 4.4.

4.1 The Sensor Search Problem

The general sensor search problem refers to *the process of finding sensors with specified properties in a search space (i.e., a collection of sensors)*. In this section, we will discuss this problem in detail, which serves as the background for our proposed sensor search services in this chapter. We first discuss the essential components of the sensor search problem in Sec. 4.1.1. Based on this discussion, we introduce in Sec. 4.1.2 the specific sensor search problem that we consider in this chapter, which focuses on a subset of parameters of the essential components. In Sec. 4.1.3, we identify a set of challenges and derive from it a set of requirements for solving this specific sensor search problem in the context of the IoT. Finally, we present our approach and outline a distributed architecture for this problem in Sec. 4.1.4.

4.1.1 Essential Components

As implied by its definition, the essential components of the general sensor search problem are *sensor, sensor property, search space, search query, and search approach*. In the following we discuss each of them in detail.

4.1.1.1 Sensor

In the context of the IoT, a sensor is integrated into an IoT device such as a sensor node or an RFID tag which, in turn, is embedded into real-world entities (e.g., a person, a chair, a plant), i.e., Things. The mission of a sensor is to provide measurements about the environment so that Things can perceive their state (e.g., whether a chair is being sat on). Commonly, a sensor is uniquely identified by an absolute address such as a URI/URL, which serves as an access point to the measurements of the sensor.

4.1.1.2 Sensor Property

Based on its properties, a sensor can be found among a collection of sensors. There are 3 types of sensor property. Static properties of a sensor refer to the static descriptions of the sensor such as its type (e.g., light, humidity), its date of production, and its production number. These descriptions are usually stored on the sensor during the production by its manufacturer and do not change over time. "Quasi-dynamic" properties of a sensor refer to those descriptions that may slowly change over time. For example, the ownership of the sensor, its location, and its deployment context. Dynamic properties of a sensor refer to its output measurements which usually change (quickly) over time. For example, the temperature of a room changes continuously over the course of 24 hours.

4.1.1.3 Search Space

A search space is a (very large) collection of sensors. A search space can be characterized by the following factors:

- *Close vs. open*: In a closed search space, the number of sensors is known and fixed, i.e., there will not be replacements of existing sensors with new sensors. In contrast, the number of sensors is not known in an open search space as well as sensors can be added to or removed from the search space. The IoT is an example of an open search space.
- *Structured vs. unstructured*: In a structured search space, sensors are organized based on either their identification or their properties, such that the search process can be speeded up significantly (e.g., a binary search algorithm performs much faster than a linear search algorithm on a sorted list of sensors). Usually, it is easier to structure a closed search space, since everything is known. For example, sensors of the same type could be grouped together, and within each group sensors located in the same place could be further grouped and so on. An open search space, however, is difficult to be structured due to its dynamicity and unknown size. The IoT is an example of an unstructured search space.
- *Small vs. large*: This factor refers to the size of the search space in terms of the total number of sensors within it, which can be small or large. Obviously, performing a sensor search in a small search space would, in most cases, be faster than in a large one, except that the latter being well-structured while the former being not. The IoT is considered to be a very large search space.

4.1.1.4 Search Query

A search query is a description of sensor properties and their relationships. Based on this description, one or more sensors in the search space are found. In the following, we present query types that are supported by many sensor search systems:

- *One-time vs. continuous*: A one-time search query will be completed after a search result (i.e., a (set of) sensor(s)), while a continuous query returns changing search results over a period of time before its completion.
- *Time-constrained vs. time-independent*: A time-constrained search query demands that the search result must be returned immediately or within a hard time bound. A time-independent search query, in contrast, does not specify a time bound for returning search results.
- *Description-based vs. output-based*: This parameter refers to the types of sensor properties to be specified in a search query, which can be description-based, i.e., based on the static/quasi-dynamic descriptions of sensors (e.g., sensor type or location), or output-based, i.e., based on the output measurements of sensors (e.g., the current temperature), or the combination of both.
- *Raw output vs. abstracted output*: An output-based search query can refer directly to raw sensor measurements (e.g., "find rooms whose temperature at the moment is below 16°C") or some high-level states derived from them (e.g., "find rooms that are cold at the moment", where "cold" is a high-level state of rooms, which is derived from the measurement values of their attached temperature sensor being lower than 16°C). While abstracted output is close to human language which eases the user in creating the search query, it requires proper domain expertise to derive high-level states from raw sensor measurements (e.g., what temperature is considered as "cold") as well as confines the search to specific applications (e.g., smart home application). Searching for sensors based on their raw measurements, however, does not have these limitations.

4.1.1.5 Search Approach

A search approach is the use of a set of strategies in developing and implementing a solution for the sensor search problem. In the following we present a set of fundamental techniques [152] that are used in many existing sensor search systems.

- *Push and pull*: In a push technique, changes in a search space such as new sensor measurements or modified sensor descriptions are proactively pushed to the sensor search engine and stored at its local storage, so that search queries can be resolved using only this data. In a pull technique, these changes are stored at the sensors and are pulled by the search engine when resolving search queries. In the IoT where the

number of sensors is larger than the number of users and sensors typically produce changes more frequently than users produce search queries, the pull technique might be preferred as it incurs lower communication overhead.

- *Divide-and-conquer*: Due to the large size of the search space, e.g., the IoT, the execution of the search can be distributed and parallelized. Upon receiving a search query, the search engine would forward the query to a number of search points (a computer that runs the search algorithm) each of which maintains an aggregated view of sensors in a (disjont) region of the search space. All search points perform search in parallel and send their search results back to the search engine where they are aggregated and presented to the user. This technique can be implemented in a hierarchical fashion (e.g., a hierarchy of search points, where a search point has an aggregate view of its sub-search-points).

- *Indexers*: An indexer provides a structure for a search space such that looking up for sensors in the search space is optimized. Sensors are indexed (i.e., structured) according to their properties, i.e., their description or output. Depending upon how dynamic the sensor properties are, sensors may be reindexed over time. Indexers are usually desired in large and open search spaces such as the IoT.

- *Crawlers*: A crawler is responsible for discovering changes in a search space and giving them to an indexer so that an up-to-date view of sensors can be maintained. The changes can be the (dis)appearance of existing/new sensors or changes in the properties of already indexed sensors. Crawlers are an essential component of any sensor search system designed for the IoT due to its large scale, openess, and dynamicity.

- *Compression*: Compression is usually used to reduce the amount of data to be communicated and stored in a sensor search system. For example, changes in the search space can be compressed before being pushed/pulled or before indexed.

- *Models*: Models are used to infer information about sensors without having to actually communicate with them (e.g., the model of an occupancy sensor in a meeting room may predict that the meeting room is "unoccupied" at 23:00 PM, which is normally the case). Accurate models, thus, can reduce communication cost. Models of sensors are usually built using their past measurements.

- *Scoring and ranking*: Scoring refers to assigning a sensor a scalar value proportional to its importance. For example, given a search query, the more it is relevant to the query, the higher score it is assigned. Ranking is concerned with sorting sensors according to their score. Scoring and ranking can be used to present the user with top-ranked (i.e., most relevant) sensors. The two techniques can also be used by an indexer to optimize the search space for one or more sensor properties that are important to the search engine.

4.1.2 The Specific Sensor Search Problem

The specific sensor search problem that we consider in this thesis is characterized as follow:

- *Sensor*: All sensors that are and will be connected to the IoT.
- *Sensor property*: Dynamic property (i.e., sensor measurements).
- *Search space*: The IoT which is (very) large, open, dynamic, and unstructured.
- *Search query*: Output-based and raw output, i.e., we focus our work on searching for sensors based on their output measurements.
- *Search approach*: Our proposed approach, which is a combination of all fundamental search techniques presented in Sec. 4.1.1.5.

More specifically, we consider two sub-problems, namely *sensor similarity search* and *content-based sensor search*. The first sub-problem is concerned with finding sensors whose recent measurements are similar to that of an example sensor, thus is essentially dealing with the similarity between streams of historical measurements of sensors. The second sub-problem is concerned with finding sensors whose latest measurements fall in a given value range.

The reason for our choice for this specific sensor search problem, respectively its two sub-problems, is two-fold: (i) these two search services are useful because they enable the user to search the real world for objects and places via their embedded sensors with either a given example object or place (e.g., see $Ex1$ at the beginning of this chapter), or a given "state" defined by a range of values (e.g., see $Ex2$ at the beginning of this chapter); and (ii) the two search services are novel.

4.1.3 Challenges and Requirements

We present in this section a set of challenges to our proposed specific sensor search problem and derive from it a set of requirements that need to be met in designing solutions for this problem. The challenges will are given below. For each challenge, we include a brief discussion in *italics*.

- *C1*: Wireless communication dominates power consumption of sensor nodes. *With state-of-the-art sensor node hardware, the radio consumes about 100 mW in active mode, draining an AAA battery within a few days. In order to achieve lifetimes in the order of months or years on a single battery, wireless communication needs to be minimized. This is one of the fundamental research challenges in WSNs specifically and in the IoT generally.*

- *C2*: The statistical properties of sensor measurements may change over time. *The states of the real world, i.e., physical objects and processes monitored by sensors, may change substantially over time, even if the sensor is immobile. For example, an occupancy sensor that is fixed on the wall of a lecture hall produce significantly different measurements between during semester and during semester break.*
- *C3*: The computational and memory resources of sensor nodes are severly constrained. *Typical hardware platforms offer a simple micro controller with few MIPS and few kilobytes of RAM. In particular, there is typically no support for floating point operations, such that complex math needs to be avoided (see Sec. 2.5.2).*
- *C4*: Raw sensor measurements are inherently imperfect. *Sensor data typically suffers from noise, jitter, outages, or outliers (e.g., due to environmental influences or hardware malfunctions), and is non-uniformly sampled (i.e., different sampling rates at different times across different sensors).*
- *C5*: It is anticipated that billions of sensors will be connected to the IoT [153].

The following requirements are drawn from the above challenges, which must be met by a sensor search algorithm designed for the WoT:

- *R1*: In order for sensors to be discovered and matched, their identification and properties must be indexed, which implies that the sensor search algorithm should analyze sensors' measurements to extract their properties. Furthermore, such analysis should be performed on a regular basis as dictated by $C2$. And it is not feasible for regularly downloading raw sensor measurements from sensor nodes, due to $C1$. Thus, sensor nodes should compute a compact *sketch* from their time series of sensor measurements, such that only the compact sketch is downloaded from the sensor nodes to minimize wireless communication overhead and thus energy consumption. The computed sketches should contain the properties of the sensors.
- *R2*: Sketches of already indexed sensors need to be regularly updated due to $C2$.
- *R3*: The time interval between two consecutive downloads of sketches from a sensor node should not be two short due to $C1$.
- *R4*: Computation of a sketch on a sensor node should be efficient and lightweight due to $C3$.
- *R5*: A sensor search algorithm should be scalable to a large number of sensors due to $C5$.
- *R6*: Matching of a sensor search request against indexed sensors should be fast and efficient due to $C5$.
- *R7*: The accuracy of a search result should be minimally impaired by the imperfectness and non-uniformity of raw sensor data due to $C4$.

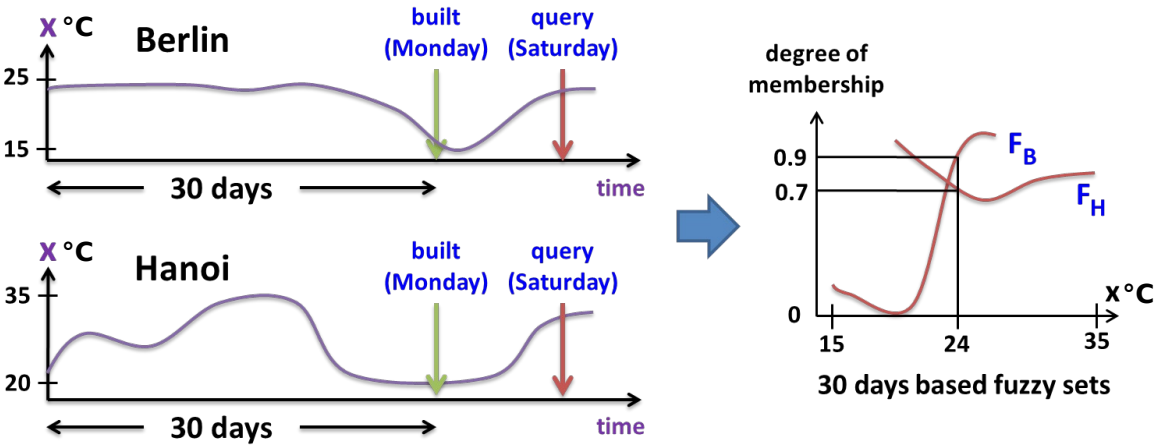


FIGURE 4.1: Fuzzy Set Illustration.

4.1.4 Architecture and Approach

Given the above challenges and requirements, we propose here an approach based on fuzzy set theory for solving our proposed sensor search problem, and outline a generic sensor search architecture to support the approach. As will be presented, the proposed architecture and approach follow a set of fundamental search techniques discussed in Sec. 4.1.1.5, including pull, divide-and-conquer, indexing, crawling, modelling, and scoring and ranking techniques.

4.1.4.1 A Fuzzy Set-based Approach

Our approach to solving the specific sensor search problem is based on fuzzy set theory. Fuzzy set theory [154] deals with uncertainties using approximate reasoning (rather than fixed and exact reasoning) to arrive at definite decisions. The imperfectness and non-uniformity of raw sensor data, thus, make fuzzy sets a natural approach for our problem.

A fuzzy set is an extension of a crisp set which allows partial membership rather than only binary membership. A fuzzy set F on a universe of discourse $\mathbb{U}$ is defined by its membership function $mf_F(x), x \in \mathbb{U}$ such that $mf_F(x) \in [0, 1]$. The closer the value of this function is to 1 for a given x, the more x belongs to the fuzzy set F. With this definition, an element $x \in \mathbb{U}$ can be *member of more than one* fuzzy set at a time, with *different* degrees of membership.

Fig. 4.1 illustrates this key concept of fuzzy set theory. We assume that on Monday we had built two fuzzy sets from the measurements of two temperature sensors (Fig. 4.1-left) during the last 30 days, one is located in Berlin and the other in Hanoi. The formal descriptions of the two fuzzy sets are $F_{Berlin} = \{(x, mf_{Berlin}(x))|x \in \mathbb{R}\}$ and $F_{Hanoi} = \{(x, mf_{Hanoi}(x))|x \in \mathbb{R}\}$, as illustrated in Fig. 4.1-right. Note that we assume the set of real numbers $\mathbb{R}$ as the universe of discourse to represent scalar measurements of sensors. We also assume the measurement ranges of the sensor in Berlin and the sensor in Hanoi are from 15°C to 25°C and from 20°C to 35°C, respectively.

Suppose that 5 days have passed since the fuzzy sets had been built and the present time is Saturday. Now if we search for a temperature value $x = 24°\text{C}$, the membership functions of F_{Berlin} and F_{Hanoi} will tell us the degree of membership of x in the two fuzzy sets, which is $mf_{Berlin}(x) = 0.9 > 0.7 = mf_{Hanoi}(x)$. We, therefore, expect that the sensor in Berlin is more likely currently reading 24°C than the sensor in Hanoi. Likewise, the sensor located in Berlin is expected to be currently reading values within a range of $[22, 25]°\text{C}$ if the sum of the degree of memberships of all temperature values within $[22, 25]°\text{C}$ in F_{Berlin} is greater than that in F_{Hanoi}. That is

$$\sum_{x \in [22,25]} mf_{Berlin}(x) > \sum_{x \in [22,25]} mf_{Hanoi}(x) \tag{4.1}$$

The design of our two sensor search algorithms in Sec. 4.2 and Sec. 4.3 will be based on this key idea of our fuzzy set approach. In particular, we will implement sketches of raw sensor data as a fuzzy set associated with a membership function, and use this membership function to compute a matching score given a set of search criteria. To satisfy the requirements presented in Sec. 4.1.3, the fuzzy set will be small in size and the computation of a matching score given a fuzzy set (i.e., a sketch) and a set of search criteria will take linear time and therefore will be fast and efficient.

4.1.4.2 Sensor Search Architecture

Fig. 4.2 shows the overview of our architecture. We have the *local sensor search* (LSS) component that is part of multiple sensor gateways that are distributed across the Internet. Each LSS component is responsible for the local sensor networks that are managed by the sensor gateway. Each LSS component is also associated with a database that contains indexed *identifications* (id) and fuzzy sets of sensors. The network of LSS components forms a distributed fuzzy set database system in the Internet.

Each sensor node in the WoT computes by itself a fuzzy set from its output measurements over a time window. Periodically, the LSS component crawls sensors to download and index those fuzzy sets in the distributed fuzzy set database. Each fuzzy set has a memory footprint of few tens of bytes and can thus be efficiently downloaded from the sensor nodes.

To perform a search, the user specifies a set of search criteria and hands it to the *global sensor search* (GSS) component via a user interface (GUI). As all sensor gateways register with the GSS component, the latter will forward the set of search criteria to relevant LSS components, where the search criteria are matched against the indexed fuzzy sets to compute a *matching score* (mc) for each indexed sensor. After that, an LSS component forms a list of sensors together with their identifications ranked by the matching scores and sends the list to the GSS component.

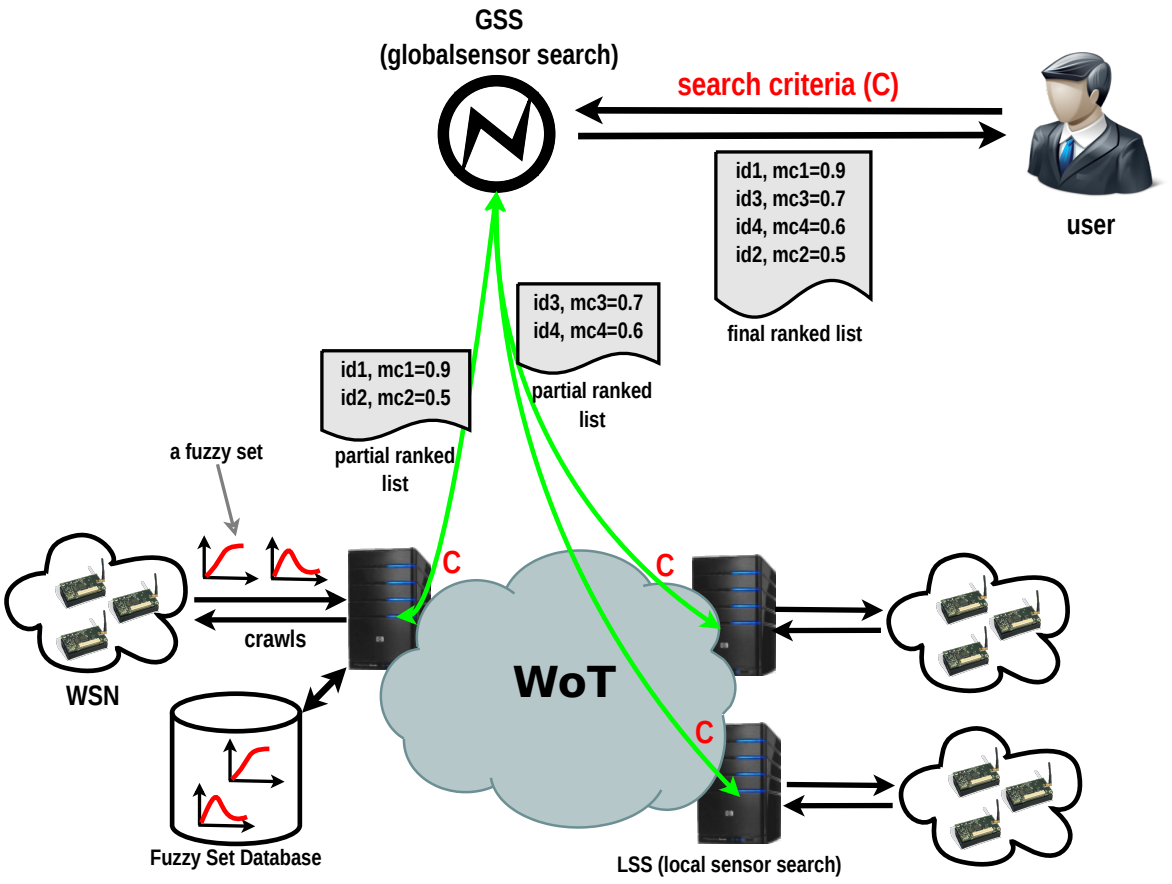


FIGURE 4.2: A generic architecture for the problem of sensor search in the WoT.

After receiving all partial ranked lists from LSS components, the GSS component merges the partial lists together to form the final ranked list that is sorted in descending order of the computed matching scores. If needed, the GSS component also verifies sensors in this list (via the LSS) whether they actually match the search query, and removes unmatched sensors from the list. The (filtered) final ranked list is presented to the user as the search result.

The flow of information in this architecture demonstrates the relationship between the sensor search service and the routing service. That is, the routing service is required to enable the efficient and reliable communication between an LSS and one or multiple sensors in a local WSN, e.g., crawling for fuzzy sets or verifying sensors.

4.2 Sensor Similarity Search in the WoT

In this section we propose, implement, and evaluate a sensor search algorithm for the IoT, that addresses the sensor similarity search problem.

4.2.1 Motivation

Most of current sensor search services define the search criteria based on the textual meta-data of the sensors. For example, existing directories of online sensors such as Cosm, GSN [25], or Microsoft SensorMap [155] support search for sensors based on textual meta-data that describes the sensors (e.g., type and location of a sensor, measurement unit, object to which the sensor is attached) and which is manually entered by the person deploying the sensor. Other users can then search for sensors with certain metadata by entering appropriate keywords. This approach, unfortunately, does not work well in practice, as humans make mistakes when entering metadata, different users use different terms to describe the same concept, or important metadata is not entered at all. For example, in [156] a user study is described where 20 participants were asked to enter metadata for a weather station sensor using a simple user interface. Those 20 persons made 45 mistakes in total.

An approach to address this problem is to store some metadata of a sensor such as sensor type on the sensor during production using so-called Transducer Electronic Data Sheets (TEDS) as defined by IEEE 1451, for example. However, most of the relevant metadata of a sensor depends on the deployment and use of the sensor (e.g., logical location of the sensor, object to which the sensor is attached) and cannot be provided by the producer of a sensor.

Another approach is to use standardized ontology for describing sensors. For example, SensorML[2], SSN[3], and RDF[4] represent the meta-data of a sensor as a knowledge graph (e.g., sensor1 *is-in* room1), where ontologies define the meaning of the vertices (sensor1 and room1) and edges (is-in) of the graph. However, these ontologies and their use are rather complex and end users likely won't be able to provide correct meta-data for sensors and their deployment context without help from experts.

Note that, the same problem also applies to search for multimedia items on the traditional Web such as images and videos, which typically can only be found if appropriate textual descriptions are provided by users. One interesting alternative approach that avoids the use of metadata altogether is searching by example. For example, Google Images includes an option to find images that are similar to another image. There are even specialized search algorithms such as TinEye[5] which find images that are similar to a given image.

In this thesis, we adopt this search-by-example approach to sensors, i.e., a user provides a sensor, respectively a fraction of its past output measurements as an example, and requests sensors that produced similar output in the past. As the result, a list of sensors is returned to the user, sorted in descending order of the values of a *similarity score* metric. A similarity score value is computed using a *similarity model*. We call this the *sensor similarity search* service. The set of search criteria, therefore, is defined as the past output measurements

[2] http://www.opengeospatial.org/standards/sensorml
[3] http://www.w3.org/2005/Incubator/ssn/ssnx/ssn
[4] http://www.w3.org/RDF/
[5] www.tineye.com

of the example sensor. This search service could be used for different purposes. Firstly, it could be used to find places with similar physical properties (e.g., the above agriculture scientist example). Secondly, it could be used to assist users with the formulation of a metadata description of a newly deployed sensor. A user would deploy a new sensor and then search for sensors with similar output, fetch the metadata of the found sensors, and reuse appropriate fractions of the metadata for the new sensor.

The rest of this section is strutured as follow. In Sec. 4.2.2, we discuss related work. In Sec. 4.2.3, we formulate the sensor similarity search problem. We then explain how the architecture presented in Sec. 4.1.4.2 can be used for the sensor similarity search service in Sec. 4.2.4. In Sec. 4.2.5, we explain how a similarity score is computed using the fuzzy set approach. After that, we explain how a similarity model is constructed and how a research problem related to it is addressed in Sec. 4.2.6 and Sec. 4.2.7. In Sec. 4.2.8, we present how to reduce the size of a similarity model. In Sec. 4.2.9, evaluations of our proposed search service are given. And finally, we conclude this section in Sec. 4.2.10.

4.2.2 Related Work

We put our work into the context of prior work that is concerned with search for sensors and similarity of sensors. During discussion, we will point out why a particular related work is not suitable for the sensor similarity search problem as it either violates a subset of assumptions or does not meet a subset of requirements that are presented in Sec. 4.1.3.

4.2.2.1 Search based on Metadata

In the context of the IoT, a number of systems support search for sensors based on meta data (i.e., textual annotations), for example Cosm, GSN, or Microsoft SensorMap. However, that requires that each sensor has already been annotated with appropriate meta descriptions. In [157], the authors propose a systematic design for a search application in the WoT. Their design is based on clustering of sensors with similar semantic descriptions. Our work differs as we are aiming at searching for sensors in the WoT that have produced similar output measurements to the search criteria, i.e., the example time series of measurements.

4.2.2.2 Search based on Sensor Measurement

This type of search refers to the problem of finding a sensor that outputs a given value at the time of a query and is investigated in [30] and [158]. The key idea is to exploit the periodicities in sensor output (e.g., a meeting room is occupied every Monday from 8 to 10), or correlations between sensors (e.g., parking spots close to the entrance of a building are often all occupied, whereas spots further away are often free) to build prediction models that predict which sensors would output the sought value at the time of the query. However, in

this section we investigate the statistical similarity of sensors over a longer time window, which is a fundamentally different problem.

4.2.2.3 Search based on Similarity of Data Streams

Computing a score that quantifies the similarity of two data streams is a fundamental problem that has been studied in different contexts. In traditional multimedia systems, similarity of audio data, images, and video streams is considered (e.g., [159], [160]). However, these methods are often not tailored to typical sensor data we are interested in but multimedia data such as audio and video. These solutions typically exceed the resources of low-power sensor networks by far ($R4$).

Non-multimedia, general time-series data The authors in [161] address clustering of data streams in general based on their similarity by developing an online version of the K-means algorithm which involves discrete Fourier transform (DFT) and pairwise distance computation of data streams on-the-fly. This technique is too resource-demanding for sensor nodes, in particular with respect to computational overhead ($R4$).

Also based on DFT, [162] proposes to obtain and index the first few coefficients of the DFT transform of every data stream of a database using R^*-trees. When a data stream is to be sought, it needs to be DFT transformed, then the first few Fourier coefficients are used to compute a distance to all indexed coefficients in the database. Those data streams in the database whose computed distance is smaller than a user-defined threshold are considered similar to the sought data stream. The DFT transformation, however, requires that data streams are uniform, i.e., their data points are sampled at identical rates, which is impractical in the sensor world as sensors usually have different sampling rates. Moreover, this work also assumes that all data streams have the same length, i.e., the same number of data points, which is not easy to achieve for sensor data streams ($C4$). The approach in [163] is close to our work in the sense that it also supports similarity search for data streams of different lengths. They propose to represent a data stream by a small set of rectangles in a feature space, which then are indexed using R^*-trees to be used for future search. A rectangle is the minimum bounding box of a set of points in the 2-dimensional plane that is formed by taking the first 2 Fourier coefficients of the DFT transform of the data stream. Comparison of two data streams means to compare their sets of representative rectangles. Again, DFT is used here which is too heavy for sensor nodes ($C3$). Furthermore, approximation of data using rectangles is rather crude as dissimilar sensor data streams might also be included in the search result.

The work in [164] also investigates the notion of "user perceived similarity" between data streams like we do, i.e., a broader notion of similarity which does not only include exact Euclidean distance between two data streams, but also their scaling and shifting transformations. For example, the stock price data streams of two companies may have identical

fluctuations, but one's stock is worth twice as much as the other at all times; or the temperature on two different days may start at different values but then gooes up and down in exactly the same way. The authors propose that two data streams are similar if one can be obtained from the other by means of affine transformations. This approach, however, cannot handle the inherent small differences in sensor data, i.e., the slight differences between a pair of sensor data streams, which is supported by our approach ($C4$).

Most of the above work assumes Euclidean distance as the underlying similarity measure. In [165], a dynamic time warping (DTW) technique is used to measure the similarity between two data streams. DTW, however, has complexity $O(N \times M)$, where N and M are the lengths of two data streams, respectively. To narrow down the search space, the authors proposed a filtering technique called "FastMap" that is based on DFT to map the data stream of interest into a multi-dimensional space and use Euclidean distance to filter out those data streams in the database whose indexed mapping (using FastMap) is dissimilar to that of the data stream being sought. The remaining data streams are compared with the sought one using the DTW technique. This approach is not suitable for sensor networks because it assumes that all raw data streams are stored in a database, as well as extra storage is required for FastMap filtering (due to $C1$). We, in contrast, do not assume that, and our approach has only complexity $O(N)$ with N being the length of the sensor data stream being sought ($R6$).

Sensor data In the domain of sensor networks, the work in [166] aims at identifying similarities between data streams generated by neighbouring nodes in a sensor network so that the streams can be aggregated to save bandwidth. A data stream is divided into multiple chunks and these chunks are compared against the chunks of another data stream to find similarities using the Jaccard similarity function. Our goal is, however, different as we do not seek for similar parts of two data streams, but we want to determine if the two entire streams are similar.

The authors in [29] cluster sensors based on the similarity of their data streams. Their approach has a high memory footprint and communication overhead as all sensor measurements need to be stored either at a central processing point or at sensors. Storing sensor measurements at sensor nodes requires high memory capacity which is typically not possible on sensor nodes ($C3$), while sending them over the wireless medium for storage at the central processing point consumes a lot of communication bandwidth and energy ($C1$ and $R1$). We, in contrast, only send sketches of data streams over the wireless medium, thus minimizing communication overhead and energy consumption. Further, their approach also requires two sensor data streams to have the same number of measurements which we do not assume in our approach ($C4$).

The works in [27] and [28] explore secure pairing of devices if their sensors show similar output data (e.g., two objects shaken together would experience similar acceleration patterns). The technique used to compute similarity of sensor data streams is based on a coherence function to measure the correlation in the frequency domain between data streams. This requires

sensors to exchange raw data streams which is clearly not scalable to the magnitude of the IoT ($C5$ and $R5$).

4.2.3 Problem Formulation

We introduce here formal models for the sensor similarity search problem. We model a sensor S that monitors a physical entity (i.e., a physical object or process) by the function

$$S : \mathbb{T} \mapsto \mathbb{R} \tag{4.2}$$

where any $t \in \mathbb{T}$ denotes a point in time and $S(t) \in \mathbb{R}$ denotes the measurement of the sensor at time t. The measurements of sensor S over time are represented by the time series

$$ts_S = S(t_1), S(t_2), .., S(t_n) \tag{4.3}$$

where $n \in \mathbb{N}$ is the number of measurements, and $t_i < t_{i+1}, \forall i \in \mathbb{N}$.

We define a fuzzy set from a (long) time series ts_S of S as a *similarity model* sm_S, that can be computed directly by S and is given by:

$$sm_S : TS \mapsto SM \tag{4.4}$$

where $TS = \{ts\}$ is the space of time series of sensors and $SM = \{sm\}$ is the space of similarity models of time series of sensors. With that, a similarity score between a time series and a similarity model of a time series can be computed:

$$sc : TS \times SM \mapsto [0, 1] \tag{4.5}$$

The higher the similarity score for the two time series, the more similar to each other the two sensors are considered.

We define the set of already indexed sensors as

$$IS = \{(sm_{S_1}, id_{S_1}), (sm_{S_2}, id_{S_2}), ...\} \tag{4.6}$$

where sm_{S_i} is a similarity model of a time series obtained from sensor S_i and id_{S_i} is the unique identification of sensor S_i.

With the above notation, we formulate the sensor similarity search problem as follows:

Given:

- The set of already annotated sensors IS
- An unannotated sensor V and its time series ts_V

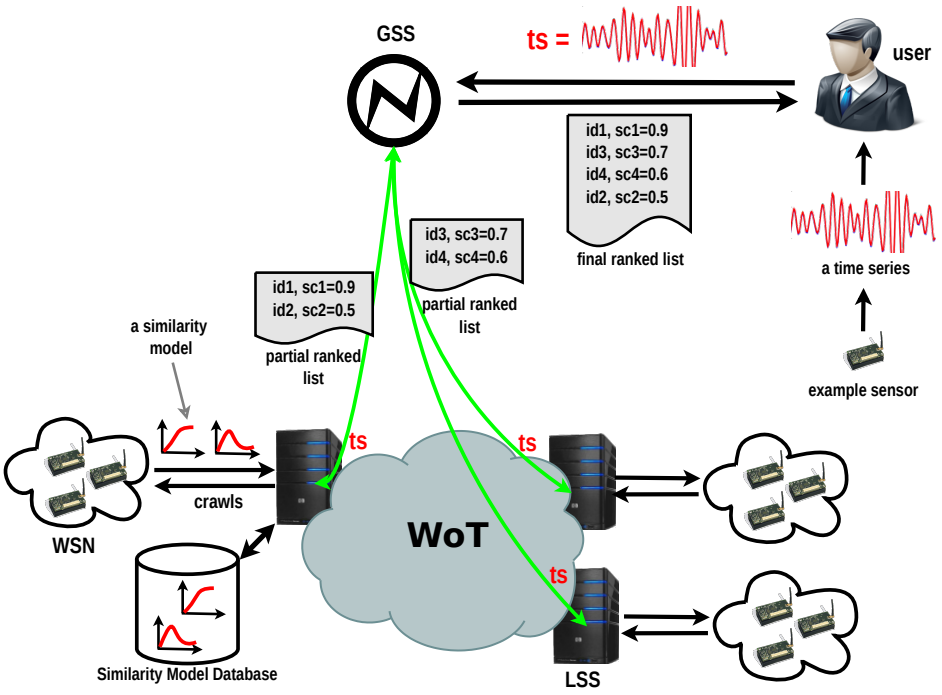


FIGURE 4.3: Sensor Similarity Search: Architecture.

Find: The list of similar sensors rl, which is defined as

$$rl = [(id_1, sc_1), (id_2, sc_2), ..., (id_k, sc_k)] \tag{4.7}$$

where $k \in \mathbb{N}$ is a user-defined parameter specifying the requested number of similar sensors, id_i are the identifications of the similar sensors from IS, and $sc_i \in [0, 1]$ are similarity scores indicating how well sensor S_i matches sensor V. The result list rl shall be ranked by matching score, that is, $sc_i \geq sc_{i+1}$. In other words, the result list shall contain an entry (id, sc) if and only if there exists an entry (sm, id) in IS such that $sc = sc(ts_V, sm)$ is among the k-highest scores over all id.

4.2.4 Sensor Similarity Search Architecture

The generic architecture outlined in Sec. 4.1.4.2 can be used for the sensor similarity search service with minor modification. As illustrated in Fig. 4.3, a fuzzy set of a sensor is used as a similarity model for the sensor, a matching score for the sensor is used as a similarity score (sc) for it, a time series of measurements (ts) obtained from a given example sensor is used as the set of criteria, and the sketch database at each sensor gateway is implemented as a similarity model database.

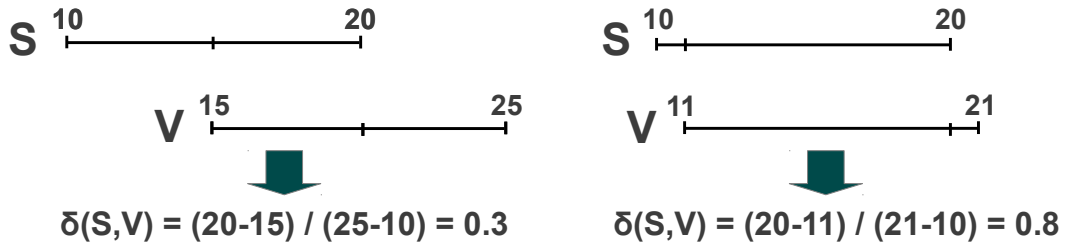


FIGURE 4.4: The measurement range difference.

4.2.5 Similarity Score Computation

We use the reasoning in Ineq. (4.1) for computing a similarity score. Assuming that we have two sensors S and V, and the similarity model sm_V is already constructed. Given a time series ts_S of S, the similarity score for S and V is computed as

$$sc(ts_S, sm_V) = \delta(S,V)\frac{1}{|ts_S|}\sum_{x \in ts_S} sm_V(x) \tag{4.8}$$

We call $\delta(S,V)$ the *range difference* between two sensors S and V that is normalized to $[0,1]$ and is given by

$$\delta(S,V) = \frac{\min\{q_3^S, q_3^V\} - \max\{q_1^S, q_1^V\}}{\max\{q_3^S, q_3^V\} - \min\{q_1^S, q_1^V\}} \tag{4.9}$$

where q_1^S, $q_3^S \in ts_S$ and q_1^V, $q_3^V \in ts_V$ are the first and third quartiles of the time series of measurements of sensors S and V, respectively. The quartiles of a set of ordered values are the three points that divide the set into four equal groups, each representing a fourth of the population of the values.

Consider sensor S, to obtain the quartiles of the time series of measurements of S we first sort the measurement values in decreasing order. The first quartile of ts_S, denoted by q_1^S, is the maximum among the smallest 25% of measurements of ts_S. The second quartile, or also called median, is the value that cuts ts_S in half, i.e., 50% of the measurements of ts_S are smaller than this value. The third quartile, denoted by q_3^S, is the minimum among the largest 25% of measurements of ts_S. The use of the quartiles makes sure that influence of outliers is eliminated because outlier measurements would be located outside the interquartile range $[q_1^S, q_3^S]$.

We call $[q_1^S, q_3^S]$ and $[q_1^V, q_3^V]$ the measurement ranges of sensor S and V. Fig. 4.4 shows an example for measurement ranges of two sensors S and V. The small overlap in the left between the two ranges implies a small $\delta(S,V)$, whereas the big overlap in the right implies a big $\delta(S,V)$.

The aim of the range difference is two fold: (i) to quickly rule out sensors of different types or sensors monitoring different physical entities because those sensors would produce very different ranges of measurement values; and (ii) if there is no clear distinction between sensors

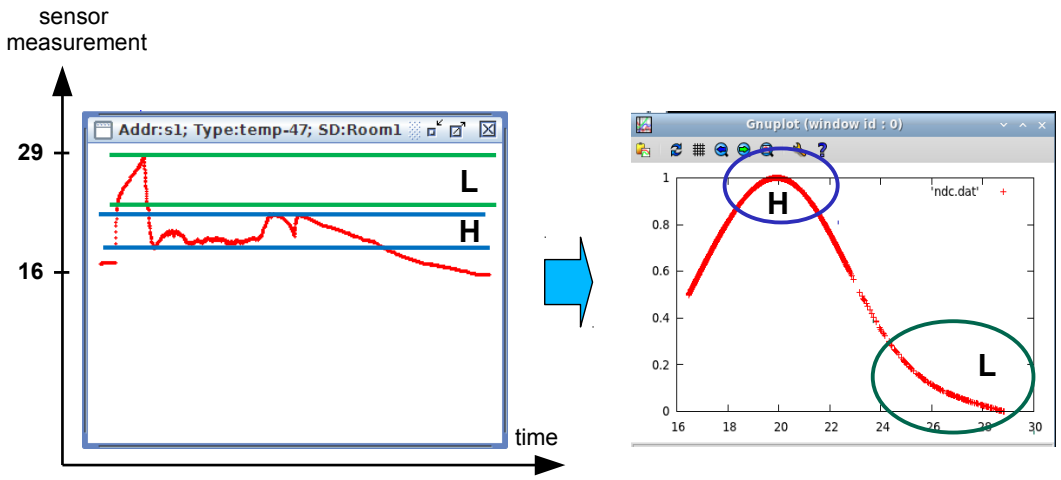


FIGURE 4.5: Construction of a similarity model.

(types, or deployment contexts), then sensors that produce measurement values within the same or very close ranges would have highest similarity scores. Thus, prior to examining the structure of the measurement curve of sensors, a potentially large number of sensors are already ruled out to narrow down the search space.

In summary, the similarity score is the higher, the more the measurement ranges overlap and the more the measurements of sensor S belong to the similarity model defined by the output of sensor V. Note that the similarity score of sensors with disjoint measurement ranges is zero.

4.2.6 Similarity Model Construction

We elaborate here how a similarity model is constructed from a given time series of sensor measurements. Consider Fig. 4.5. The left image shows the time series of measurements of a temperature sensor S over a certain time period, i.e., ts_S, and the graph in the right shows the constructed similarity model sm_S, which is the membership function of a fuzzy set that is defined by the measurements of S. We want to find a sm_S that best captures the shape of the curve created by the series of measurements of the sensor over time.

We denote x^S_{min} and $x^S_{max} \in ts_S$ the minimum and maximum values among the measurements of sensor S, respectively. Considering a real value $r_d \in (0, \frac{x^S_{max} - x^S_{min}}{2}]$ and the interval $dd_x = [x - r_d,\ x + r_d] \subset [x^S_{min},\ x^S_{max}]$, we are interested in how many measurements $x \in ts_S$ fall into $[x - r_d,\ x + r_d]$ over time because this captures the behaviour of the physical entity that the sensor is measuring: "does temperature tend to be within the range dd_x?". Put it another way, the density of the population of sensor measurements in dd_x describes the likelihood of temperature to be within dd_x over time. By sliding dd_x over $[x^S_{min},\ x^S_{max}]$ we can calculate the likelihood for each temperature value x in the measurement range, and call it the *neighborhood density* of x. $sm_S(x)$ is then defined as this neighborhood density of $x \in [x^S_{min},\ x^S_{max}]$.

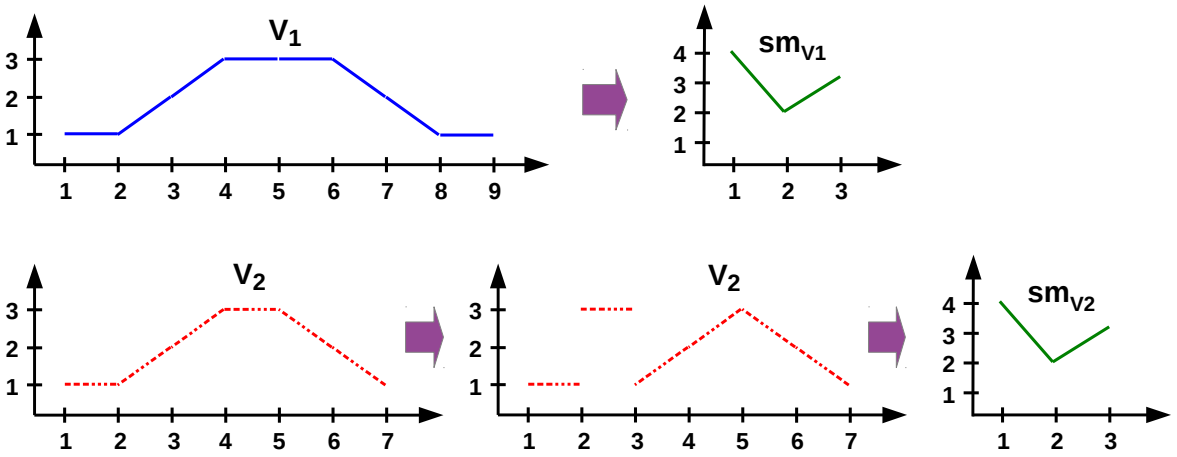


FIGURE 4.6: "Jump" reordering of sensor measurements.

We evaluate the neighborhood density of sensor measurements within dd_x around a measurement x by calculating the distances between x and every sensor measurement $y \in [x^S_{min}, x^S_{max}]$, computing these distances' weight as the exponential decay function (i.e., values with larger distances have an exponentially smaller weight), and sum up the computed weights. We then normalize the sum to $[0, 1]$ by dividing it by the number of sensor measurements $|ts_S|$. Formally, the neighborhood density sm_S for a sensor measurement x is given by

$$sm_S(x) = \frac{1}{|ts_S|} \sum_{i=1}^{|ts_S|} e^{-\frac{|x - S(t_i)|}{r_d}} \tag{4.10}$$

Due to the exponential function, sensor measurements which are outside of $[x - r_d, x + r_d]$ have little influence on $sm_S(x)$. The obtained fuzzy set therefore is $F_S = \{(x, sm_S(x)) | x \in ts_S\}$.

For a visual explanation of the construction of a similarity model, consider Fig. 4.5 again, which shows the neighborhood density function of the temperature sensor in the right. In the right graph, the peak in region "H" results from a dense distribution of temperature measurements within region "H" in the sensor data, while the low values in region "L" in the right graph are explained by a sparse distribution of temperature measurements within region "L" in the sensor data.

4.2.7 Injective Mapping Problem

Although the neighborhood density function in Eq. 4.10 is able to represent the series of measurements of a sensor S over time by a compact similarity model (i.e., a fuzzy set), it does not guarantee an injective mapping between the series of measurements and the computed similarity model. This issue may lead to exceptional cases where two dissimilar sensors are considered similar due to them having the same computed similarity model.

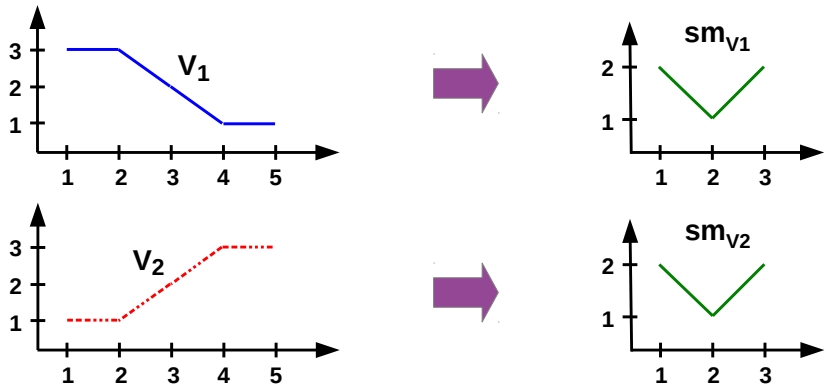


FIGURE 4.7: Reordering by flipping the sensor measurement curve.

We observe that by reordering measurement values of a sensor on the time axis, one will obtain the same similarity model. Some of these reorderings will probably not appear in reality because the resulting measurement curve would have "jumps" which do not reflect real-world phenomena as they are typically "smooth", i.e., they do not suddenly change from one state to another distant state but do gradually change between close states. For example, the temperature within an office typically does not suddenly jump from 18 to 50 degree Celsius. Such "jumps" often indicate some fault (e.g., physical damage, battery depletion, etc). Fig. 4.6 illustrates this effect where a reordering in the measurement values of sensor V_1 results in an identical similarity model to the computed similarity model of sensor V_2.

There are reorderings that preserve smoothness. Therefore, they may very well occur in practice, e.g., by flipping the sensor measurement curve over a line parallel to the y-axis. Fig. 4.7 illustrates this, where the measurement curves of sensor V_1 and V_2 look different but one of them could be obtained by flipping the other over the line $x = 3$. The resulting similarity models of the two sensors are identical.

However, reordering the measurement values of a time series on the time axis typically changes the discrete derivative of the time series. Motivated by this fact, we propose to incorporate information about the discrete derivative into the construction of the similarity model to deal with the reordering effect.

We define

$$V'(t_i) = \frac{V(t_{i+1}) - V(t_i)}{t_{i+1} - t_i} \tag{4.11}$$

as the discrete derivative of V at t_i. We then denote

$$ts_{V'} = V'(t_1), V'(t_2), .., V'(t_{|ts_{V'}|-1}) \tag{4.12}$$

as the time series of discrete derivatives of a sensor V. The similarity model of the discrete derivatives of V can be obtained using Eq. 4.10 and is denoted as $sm_{V'}$, thus the corresponding fuzzy set of the discrete derivatives of V is $V_{S'} = \{(x', sm_{V'}(x'))|x' \in ts_{V'}\}$.

We redefine the similarity score of the sensor S with respect to the sensor V in Eq. 4.8 as

$$sc(ts_S, sm_V) = \delta(S,V)\frac{1}{|ts_S|}\sum_{i=1}^{|ts_S|} sm_V(S(t_i)) \times sm_{V'}(S'(t_i)) \tag{4.13}$$

Eq. 4.13 says that the more the measurements and discrete derivatives of sensor S belong to the fuzzy sets F_V and $F_{V'}$ of sensor V, respectively, the higher is the similarity score. In the language of fuzzy logic, this is equivalent to the "and"-operator [154]. Thus, our proposal helps mitigate the reordering effect because of the following two reasons:

- For the same time series of measurements ts_V, most of reorderings would generate different discrete derivative, thus resulting in different derivative similarity models and in different similarity scores.
- For two different time series of measurements ts_{V_1} and ts_{V_2} that have the same discrete derivative, there is a very good chance that the resulting similarity models of the two time series are different, thus resulting in different similarity scores from Eq. 4.13.

There is, however, an exceptional case in which Eq. 4.13 produces the same similarity score even though the two time series of measurements as well as the two corresponding discrete derivatives are different. Considering two sensors V_1 and V_2, this case happens if and only if $sm_{V_1} = sm_{V_2'}$ and $sm_{V_1'} = sm_{V_2}$ due to commutative property of the multiplication in Eq. 4.13. In reality, however, it is extremely unlikely that both the two following cases happen at the same time:

- ts_{V_1} can be obtained from $ts_{V_2'}$ by a form of reordering
- $ts_{V_1'}$ can be obtained from ts_{V_2} by a form of reordering

Thus, we have an effective heuristic approach for an injective mapping of sensor time series to a pair of similarity models.

4.2.8 Similarity Model Approximation

Since the storage overhead for a similarity model of a sensor S is proportional to the size of ts_S, communication and storage for the similarity model may be expensive. We observe that the curve of the similarity model function is typically smooth due to the exponential weighting. Thus, we propose to represent this function by a set of line segments that approximate its curve. As each line segment can be defined by two float values, the memory footprint is small and typically in the order of few tens of bytes.

We use the similarity model function $sm_S(x)$ for explanation. The illustration of our approach is given in Fig. 4.8, where in the left we have the similarity model whose linear

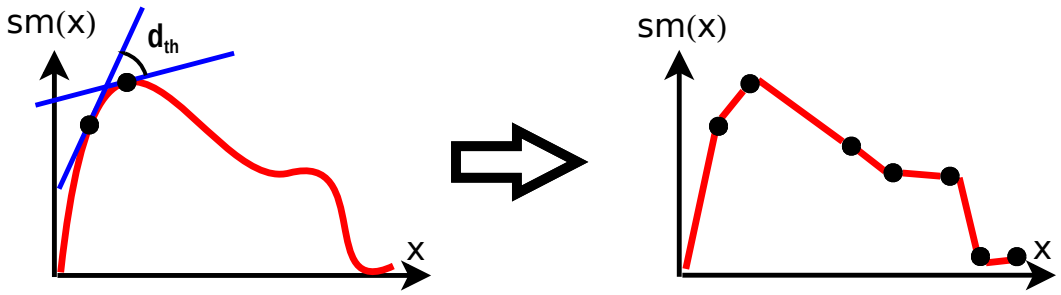


FIGURE 4.8: Approximation of a similarity model (sm).

approximation is shown in the right side of the figure. We first define a derivative threshold d_{th}, compute $sm_S(x)$'s first discrete derivative d_1 at x_1 (x_1 is the second smallest value in the measurement range of S), and mark the point $A_1 := (x_1, sm_S(x_1))$. We then iterate over points $(x_i, sm_S(x_i))$ on the curve and compute $sm_S(x)$'s first discrete derivative d_i, until $d_i - d_1 > d_{th}$. We assign $x_2 := x_i$, $A_2 := (x_2, sm_S(x_2))$, and store the line A_1A_2 as the approximation of $sm_S(x)$ for the interval $[x_1, x_2]$. After that, we assign $A_1 := A_2$ and $d_1 := d_i$, and continue to iterate over points on the curve in the same fashion until we reach the point $(x^S_{max}, sm_S(x^S_{max}))$. The resulting set of line segments is the desired approximation of sm_S.

4.2.9 Evaluation

In this subsection, we evaluate the performance of our sensor similarity search algorithm. As a result for a search request (i.e., an example sensor), a list of sensors with their identifications ranked by decreasing similarity score is obtained. Similar sensors should be ranked highly (i.e., on top of the list), while dissimilar sensors should be ranked low (i.e., at the bottom of the list). A list that has many similar sensors ranked on top is said to have a high degree of accuracy.

In order to enable a quantitative evaluation, we manually group sensors based on their location and name a group after each location. All sensors belonging to a group are manually annotated with the location name of the group. For example, all temperature sensors in a meeting room may thus form a group, and are annotated with "meeting-room".

This way, we obtain groups of sensors $G_1, G_2, ...$ We now pick a sensor S from a group G_i, take a time series of it over a certain time window, and feed the time series to the sensor similarity search algorithm. We would expect to receive a result list of sensors, where all sensors belonging to G_i are ranked highest. However, the result may be imperfect, i.e., sensors from G_i might be ranked lower than sensors from other groups. Therefore, we need a metric to quantify the accuracy of a rank list, which we describe next before we present the evaluation setup and results.

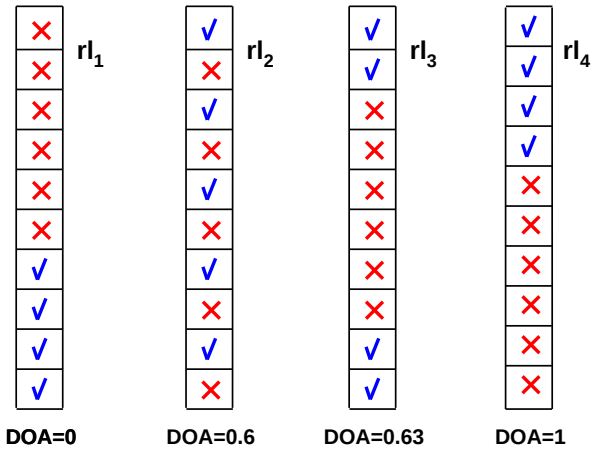


FIGURE 4.9: Illustration of the *doa* metric.

4.2.9.1 Degree of Ranking Accuracy

Figure 4.9 shows possible rank lists obtained as a result when searching for a sensor S from a group G_i. The check marks indicate matching sensors, i.e., sensors from the same group G_i, while crosses indicate non-matching sensor from other groups. The best possible result is list rl_4 as all matching sensors are ranked highest. The worst result is list rl_1.

We now define a metric that maps a rank list to a scaler value between 0 (worst result) and 1 (best result). For each matching sensor, we compute the ranking error, i.e., the number of non-matching sensors ranked higher. We then compute the average ranking error of all matching sensors, which equals 0 in the best case, and equals the number of non-matching sensors in the worst case. To normalize to the range to the interval $[0, 1]$, we devide by the number of non-matching sensors. By subtracting the resulting value from 1, we obtain the desired metric. Thus, we define the degree of ranking accuracy (doa) of a rank list rl as follows:

$$doa(rl) = 1 - \frac{1}{C_{rl}(N_{rl} - C_{rl})} \times \sum_{i=1}^{N_{rl}} e_{rl}(i) \tag{4.14}$$

where N_{rl} is the length of rank list rl, C_{rl} is the number of matching sensors in rl, and $e_{rl}(i)$ is the ranking error of a matching sensor at rank i, i.e., the number of non-matching sensors ranking higher than i. If i is a non-matching sensor, then $e_{rl}(i) := 0$. Fig. 4.9 shows the value of the metric for different rank lists.

4.2.9.2 Experiment Setup

We evaluate our sensor similarity search algorithm using simulation with realistic sensor data. The advantage of using simulation is two-fold. First, we need to repeatedly run our mechanism on the same set of sensor data to investigate the mechanism's behavior and

FIGURE 4.10: Grouping of sensors in the NOAA data set.

improvement. This is impossible in a real testbed as there are always variations between two successive collections of sensor data. Second, the time needed to collect sufficiently large amount of sensor data for testing is often too long, e.g., collecting 24 hours of temperature data takes exactly 24 hours waiting time.

We develop a simulation tool in Java that is able to replay recorded measurements of multiple sensors, execute search operations over these sensors, and compute the resulting ranking accuracy according to the above metric.

We use three data sets with recorded sensor values from real-world deployments for the evaluation. As described earlier, we group sensors in each of the data sets based on their location, such that sensors in a group should observe similar (but not identical) output measurements.

The first is the NOAA data set[6] which contains the output of sensors monitoring ocean and athmosphere (e.g., barometric pressure, wind speed, air temperature, conductivity, water velocity) that are deployed along the coast lines of various places in North America. We use 23 barometric-pressure sensors from this data set and group them into 5 groups, namely Alaska (3 sensors), West-Coast, Great-Lakes, East-Coast, and Hawaii (5 sensors each) as shown in Fig.4.10.

The second is the IntelLab data set[7] which contains recorded measurements of 54 sensor nodes equipped with four different sensors, namely temperature, light, battery voltage, and humidity (i.e., 216 sensors in total). These sensors were deployed in the Intel Berkeley Research Lab between February 28th and April 5th, 2004. We select a set of 12 humidity sensors and group them into three groups, namely Lecture-Hall (4 sensors), Dining-Room (4 sensors), and Meeting-Room (4 sensors) as shown in Fig. 4.11.

[6]http://tidesandcurrents.noaa.gov/gmap3
[7]http://db.csail.mit.edu/labdata/labdata.html

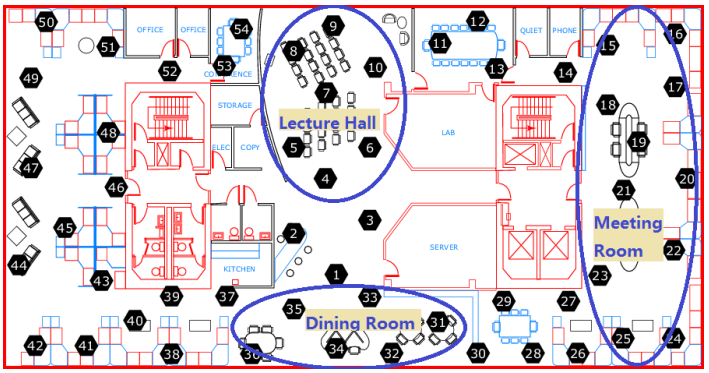


FIGURE 4.11: Grouping of sensors in the Intel Lab data set.

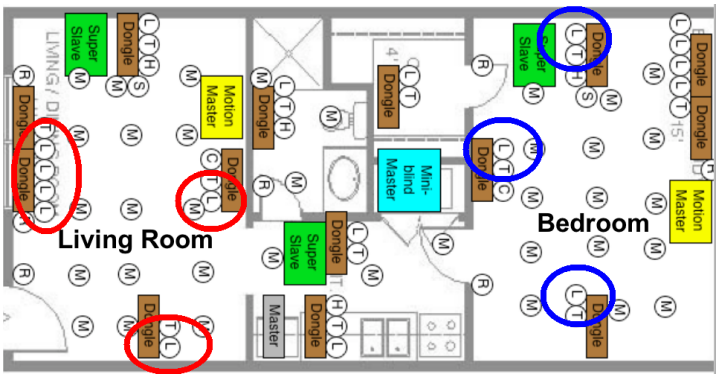


FIGURE 4.12: Grouping of sensors in the MavHome data set.

The third is the MavHome sensor data set[8] that contains recorded measurements of sensors monitoring daily living activities of people at home. The sensor types include light, humidity, heat, and motion sensors. The data set was recorded from January 3 to February 2, 2005. We select a set of 8 light sensors and group them into 2 groups based on their location: "Living-Room" (5 sensors) and "Bedroom" (3 sensors) (see Fig.4.12).

To perform the evaluation, we sequentially pick one sensor after another from the selected sets of sensors, take a time series from it over the time window of 24 hours, feed the time series to the search algorithm, obtain a rank list, and compute the *doa* value. We call this series of operations a search trial. For each sensor, we use the last 24 hours of measurements because it is representative, since the sensor data tends to repeat every day. This, for example, approximately equals 1500 data points in the IntelLab data set and 200 data points in the NOAA data set.

Note that, as we outlined in Sec. 4.1.4, sensors are periodically crawled by the LSS component to download their constructed fuzzy sets, which in this case are similarity models. Thus, sensor similarity search algorithm should be evaluated over a certain period of time rather than just once. We perform the evaluation described in the above paragraph for several simulated days, during which new fuzzy sets are periodically updated once per day while requests for similarity search arrive at the GSS component randomly at any time of a day.

[8] http://ailab.wsu.edu/mavhome/index.html

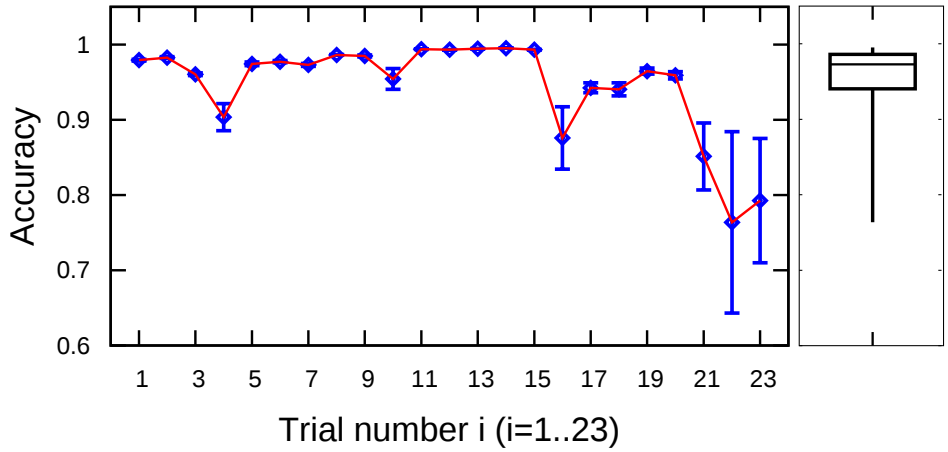


FIGURE 4.13: Average degree of accuracy (NOAA).

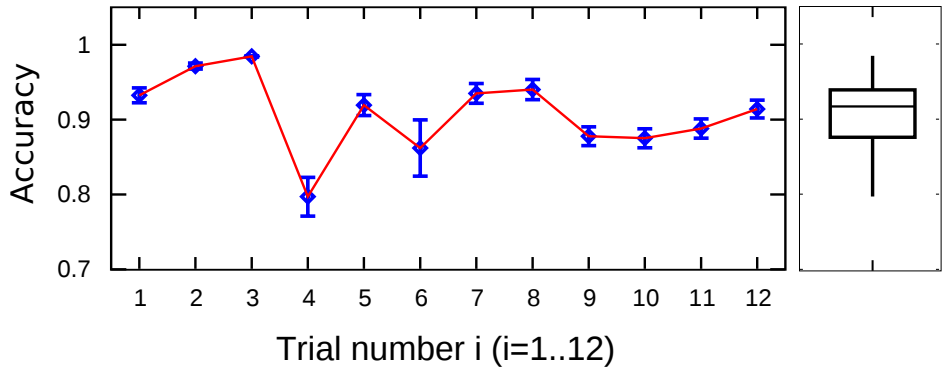


FIGURE 4.14: Average degree of accuracy (IntelLab).

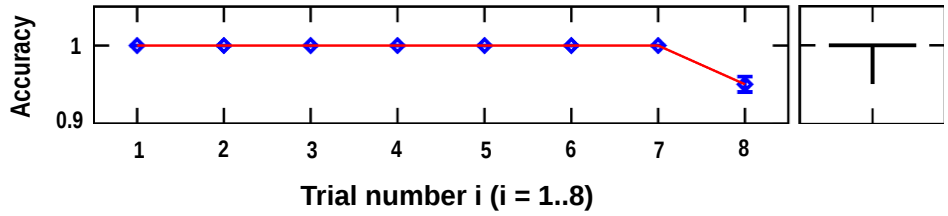


FIGURE 4.15: Average degree of accuracy (MavHome).

4.2.9.3 Numerical Results

Fig. 4.13, Fig. 4.14, and Fig. 4.15 show the resulting average *doa* values when searching for each of the sensors in the NOAA, IntelLab, and MavHome data sets over the course of 20 days, respectively. Also, a box plot aggregating the results is shown next to each figure.

As observed in the figures, our sensor similarity search obtains a high degree of accuracy as the average *doa* is above 0.97 for NOAA and MavHome data sets, and above 0.94 for IntelLab data set. The boxplots show a stable performance of our approach with a small inter-quartile range, i.e., 0.025 for the NOAA data set, 0.083 for the IntelLab data set, and 0.0 for the MavHome data set.

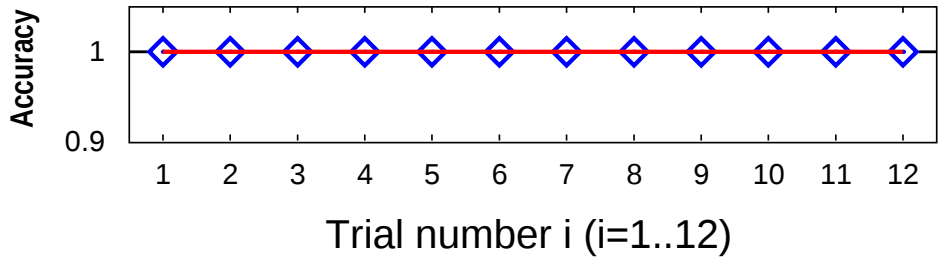


FIGURE 4.16: IntelLab data set: Best case.

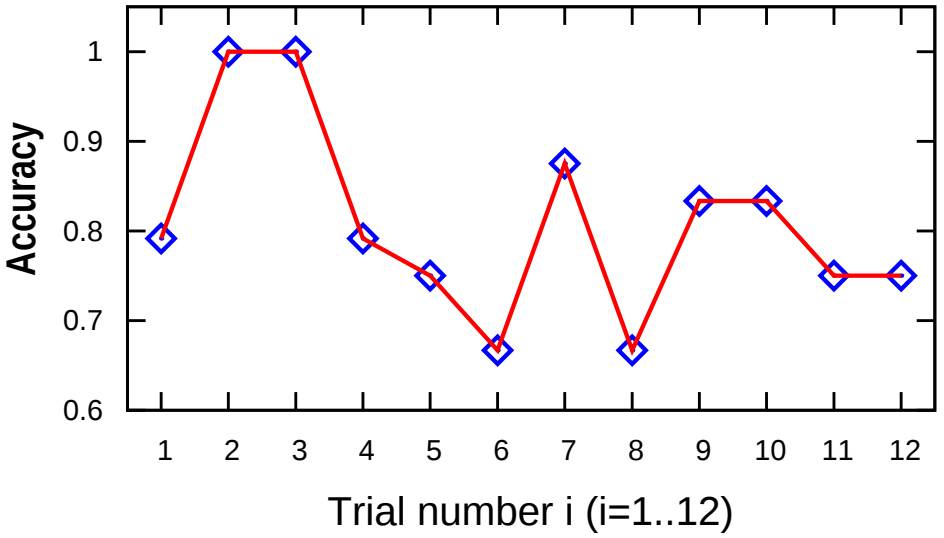


FIGURE 4.17: IntelLab data set: Worst case.

There are, however, a few outliers such as search trials number 22 and 23 in Fig.4.13, and number 4 in Fig.4.14. The reason for this is that environmental conditions change over time, and even though sensors in each group are deployed close to each other, they may experience significant variations due to micro-climates (in the NOAA data set) or due to sensors being close to the heating or air conditioner (in the IntelLab data set).

To further investigate how environmental conditions affect performance, worst case and best case performance are included in Fig.4.16 and Fig.4.17 for the IntelLab data set. Our search algorithm performs the best (100% accuracy) when there are clear climate differences among the regions, and performs worst when (micro)climate conditions are similar among different regions. For example, the Intel Lab is an indoor environment with no clear boundary between regions, therefore humidity in the meeting room and the lecture hall is sometimes very similar, thus causing a low degree of accuracy.

The light usage in a smart home, however, does not only depend on environmental conditions but on habits of the people living in the house, which differs clearly between living room and bedroom. This explains why evaluation of MavHome data set results in the highest accuracy among the three data sets (see Fig.4.15).

4.2.9.4 Performance and Scalability

We investigate the time needed to compute a matching score for a pair of sensors as this is the fundamental operation performed by our sensor similarity search algorithm. We use the approach in [167] to minimize the impact of garbage collection and just-in-time compilation in the Java VM on computation time measurement.

For a matching score computation, we obtained an average computation time of $222\mu s$ for the IntelLab data set, $28\mu s$ for NOAA, and $70\mu s$ for MavHome. The difference stems from the fact that the number of measurements per day in NOAA (200) and MavHome (500) is much smaller than for the IntelLab data set (1500). That is, we can compare against 4505 to 35714 sensors per second. The computer used in our experiment has an Intel Core i5 CPU that runs at a clock rate of 2.4Ghz.

It is worth noting that, even though the number of measurements per day of NOAA and MavHome sensors is much smaller than that of IntelLab sensors, the accuracy is high for all three data sets as can be seen in Fig.4.13, Fig.4.14, and Fig.4.15. This can be exploited to reduce the computation overhead by incrementally computing matching score of increasing accuracy. As a search request arrives, we first compute an approximate matching score according to Eq. 4.13 for a small subset of the measurements U_S of the given sensor, for example, by only using every 10th sample from the time series. In a second pass, more samples are added, say every 5th sample, and so on. This way, a first approximate search result can be very quickly presented to the user which is continuously refined the longer the user waits.

We applied this approach to the IntelLab data set by computing the matching score on a subset of the date that contains only every 6th sample, (i.e., 250 data points per day). This results in exactly the same *doa* as the *doa* obtained for the full set of 1500 data points per day, but reduces the computation overhead to one sixth.

Finally, please note that similarity search can be efficiently parallelized on a cluster of computers (as used by many Internet search companies) by partitioning the set of sensors, respectively their indexed fuzzy sets, and distributing them to the computers in the cluster.

4.2.10 Conclusion

We identified that a fundamental service in the forming WoT is search for sensors. Instead of relying on manual annotations (which are often incorrect, inconsistent, or incomplete), we propose sensor similarity search service, where based on the past output of a sensor, sensors with similar output are found. We designed an efficient mechanism to compute a similarity score for a pair of sensors. All sensors compute similarity models that represent their past output using only few tens of bytes. These similarity models are indexed in a database. Given the output of another sensor, similarity scores are computed for each indexed sensor,

sensors are ranked by this similarity score and returned to the user. Using sensor data from three real-world deployments, we could show the high accuracy of our search algorithm.

4.3 Content-based Sensor Search in the WoT

In this section we propose, implement, and evaluate a sensor search algorithm for the IoT, that addresses the content-based sensor search problem.

4.3.1 Motivation

With the formation of the WoT, we believe that another important functionality of a sensor search service would be to allow the users to find real-world entities (i.e., physical objects and places) that are currently exhibiting a certain state (e.g., the car driver example mentioned in the beginning of this chapter, i.e., wheather a parking spot is "free" at the moment or not). This means the search criteria should be defined based on the current output measurement of sensors (which are embedded into real-world entities).

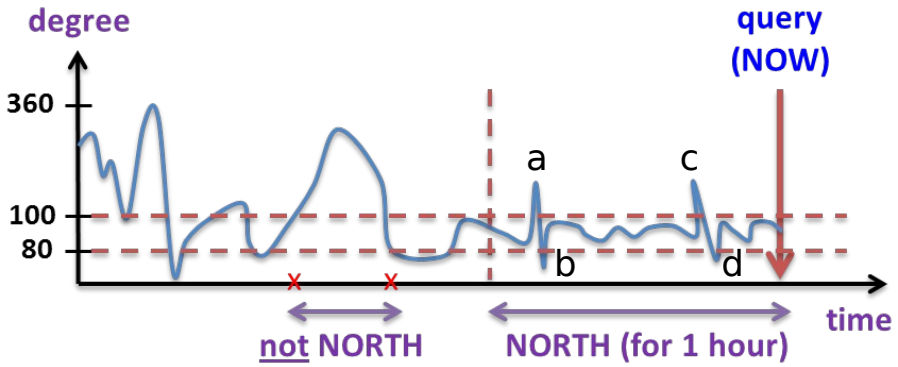


FIGURE 4.18: Direction measurements of a wind sensor.

However, sensor measurements usually fluctuate over time as the states of the monitored physical processes change often quickly but marginally. Hence, searching for sensors with a given single measurement value at an instant of time is of limited benefit. To illustrate this, we take an example of a wind sensor (see Fig. 4.18) that is located on top of a hill to measure the wind direction. Due to the nature of the wind, direction measurements may change rapidly over time, making it hard to tell which direction the wind is blowing by just considering the latest sensor measurement. However, if we consider sensor measurements during the last 1 hour, statistics show that the wind has mainly blown towards the North as measurements fell mostly within 80° and 100°. Thus, a human would say that the wind has been blowing north (for the past hour).

Inspired by this observation, we propose to search for sensors that *have been producing sensor measurements in a certain range of values for a certain amount of recent time*. In particular,

the majority of the sensor measurements in that time interval should fall into the desired value range. A search query would then consist of a range of values and a recent time window. We call this service the *content-based search* service.

There are, however, challenges to the contented-based search service. Firstly, sensor measurements are very dynamic as the real-world states observed by the sensors change frequently. This dynamicity makes current search algorithms designed for the traditional Web not suitable for the WoT, as they are usually designed for relatively static content and meta-data tags of Web documents (e.g., Web pages, video and audio titles and tags, etc). Secondly, the anticipated huge number of sensor nodes already connected, being connected, and to-be-connected to the WoT [153] implies that a search service that needs to communicate with all available sensors in the WoT when processing a user's search query is impratical, due to the tremendous communication overhead incurred. Thus, any search algorithm designed for the content-based search service must be able to minimize the communication overhead, and at the same time deal with the dynamicity of sensor measurements.

To reduce communication overhead, our approach is to only communicate with those sensors that are *likely to match* the search query to verify if they actually match the query, and return the correctly verfied sensors to the user as the search result. To determine if sensors are likely to match the query, we propose to construct a *prediction model* from past output measurements of each sensor in the WoT (based on statistical properties of the measurements), and index this model in a distributed database system in the Internet, that can be accessed by the search algorithm with low overhead. These prediction models, in response to a search query, will estimate the probability that a given sensor has recently been producing sensor measurements that match the query. The search algorithm computes a list of sensors sorted in descending order of these probabilities and contacts top-ranked sensors for query verification until a sufficient number of matches has been found. Among the contacted sensors, only those sensors, that are correctly verified, are returned to the user as a list of results. We call such a list the *rank list*. This way we can drastically reduce not only communication, but also time overhead to process the query.

To adapt to the frequent changes of sensor data, we propose to periodically construct a new prediction model of an indexed sensor. The newly constructed prediction model is then merged with the existing one in such a way that recent changes in sensor data are reflected, and at the same time, the information from past sensor data is also preserved according to a certain *fading factor*. We call this process the *adaptation process*.

The rest of this section is organized as follows. In Sec. 4.3.2, we discuss related work. The content-based sensor search problem is formulated in Sec. 4.3.3. In Sec. 4.3.4, we explain how the content-based sensor search service is fitted into the sensor search architecture outlined in Sec. 4.1.4. In Sec. 4.3.5, we present in detail our solution to the content-based sensor search problem. A thorough evaluation of the proposed solution is given in Sec. 4.3.6. And finally, we conclude the section in Sec. 4.3.7.

4.3.2 Related Work

Previous work in the literature that is related to our work mostly falls in two categories: related work on time-series data forcasting and related work on sensor search. The former category is considered because we also predict future data given past data. However, while related work uses prediction models to forecast *what value* the data point at a future time t would take, this value, e.g., v, is given in our work and prediction models are used to estimate the probability of the data point at t being v. In the following, we present some representative work on time-series data forecasting before discussing related work on sensor search in the IoT.

4.3.2.1 Time-series Data Forecasting

Time-series data forecasting is a well-established research field, which has applications in many domains such as financial data analysis or Internet traffic analysis. To forecast future data, data mining techniques are usually used to extract features (e.g., statistical properties) from training data sets, such that they can be used to build prediction models.

The work in [168] uses the LS-SVMs (Least Squares Support Vector Machines) method within the Bayesian evidence framework [169] to extract from past data of financial time series nonlinear prediction models. The goal of this work is to not only predict future data points but also to calculate the error associated with them, such that people may make optimal financial investments using the extracted models' predictions and their potential risk. In [170], a mechanism called ART (AutoRegressive Tree) models is proposed as prediction models for a time series of data. An ART is a decision tree with an AR (AutoRegressive) model at its leaves, is constructed over a training data set (i.e., past data of a time series), and is refined (in terms of the tree's structure and parameters) using a Bayesian technique. A limitation of this work is that it does not support predicting data points at multiple time-steps in the future, i.e., y_{t+n} can only be predicted if $y_1, .., y_{t+n-1}$ are given. A neural network based approach is proposed in [171], which uses a trained WP-MLP (Wavelet Packet Multi-Layer Pceptron) neural network for prediction. The initial weights of the network are generated using a clustering algorithm to reduce the training time that the network needs to converge to a good solution (in comparison with a random assignment of the initial weights). The authors in [172] follow a totally different approach. Instead of focusing on building a specific prediction model, they select a set of already existing ones and combine them into a "super" prediction model that has the best forecasting accuracy.

The common limitation of these approaches is that they require training the prediction models on sample data, which usually takes time and computational power. These approaches, therefore, may not be suitable for predicting online and real-time sensor data due to time constraint as well as for implemention on sensor nodes due to their constrained computational resources (see challenge $C3$). The work in [173] addresses this limitation by proposing

3 different forecasting approaches. The first one approximates historical data of a time series using polynomial curves and use these curves to predict future data, which may not be realistic as polynomial curves are a rather a crude approximation of reality. The second approach converts the historical data into DFT coefficients in the frequency domain, uses them to extrapolate the DFT coefficients of future data points, and construct them using their extrapolated DFT coefficients. Again, DFT may be too computation intensive for sensor nodes. The third approach is probably the closest to our approach, since they also assume that data appearing frequently in the past has a higher probability of occurring again in the future. They divide a past time interval into time segments of the same length, calculate the mean of data in each segment, and use these means for predicting the means of data in future time segments. The future raw data, thus, cannot be predicted. Our approach, which is based on fuzzy set, does not suffer from this problem.

4.3.2.2 Sensor Search in the IoT

Most of related work to the sensor search problem in the IoT has been discussed in Sec. 4.2.2, in which we reviewed techniques for searching sensors based on their meta-data description and based on the similarity between their output measurements.

Unfortunately, those techniques cannot be used to search for states of real-world entities e.g., "hot", "empty", which are dynamic and depend on the recent measurements of embedded sensors. For example, we could find a "room" but cannot find a "room" that is currently "empty". The works in [30] and [158] address this issue by allowing a user to search for sensors that output a given value at the query time. The key idea there is to exploit the periodicities in sensor output (e.g., a meeting room is occupied every Monday from 8 to 10), or correlations between sensors (e.g., parking spots close to the entrance of a building are often all occupied, whereas spots further away are often free) to build prediction models that predict which sensors would output the sought value at the time of the query.

These works, however, have limitations. Firstly, they assume that there is a class of sensors that exhibits a high degree of periodicity (e.g., people-centric sensors) within a considered time window. For example, the measurements of ocupancy sensors monitoring a lecture hall exhibit a dominant periodic pattern of the lecture hall being "occupied" from Moday to Friday and being "free" during the weekend. Secondly, the "sensor output value" considered is not a raw sensor value but a high-level state derived from raw sensor measurements (e.g., "free" or "occupied" correspond to sensor measurements being lower or higher than a certain threshold, respectively). This approach does not only require proper domain expertise to derive those states from raw data, but also confines the search algorithm to specific applications.

Our content-based search service is different as we support search for raw sensor data which requires no domain expertise and gives users the flexibility of defining their own application-specific search. Furthermore, we make no specific assumption regarding periodicity in sensor data beyond assuming that past output is statistically similar to future output.

4.3.3 Problem Formulation

We introduce here formal models for the content-based sensor search problem. To model a sensor and its output measurements, we use the same modelling method as presented in Sec. 4.2.3, i.e., we model a sensor S that monitors a physical entity by the function

$$S : \mathbb{T} \mapsto \mathbb{R} \tag{4.15}$$

where any $t \in \mathbb{T}$ denotes a point in time and $S(t) \in \mathbb{R}$ denotes the measurement of the sensor at time t. The measurements of sensor S over time are represented by the time series

$$ts_S = S(t_1), S(t_2), .., S(t_n) \tag{4.16}$$

where $n \in \mathbb{N}$ is the number of measurements, and $t_i < t_{i+1}, \forall i \in \mathbb{N}$.

We model a user's search query q that is submitted by the user at time $t_q \in T$ as the pair

$$q = ([a, b], h) \tag{4.17}$$

where $[a, b] \subset \mathbb{R}$, and $h \in \mathbb{T}$. The query q means that the user wants to search for sensors in the WoT that have been producing sensor measurements falling in the range $[a, b]$ over the time window $[t_q - h, t_q]$. We define $Q = \{q\}$ as the space of possible user queries.

We define the prediction model that estimates the probability that sensor S is producing sensor measurements that match a given query q as the function:

$$pm_S : Q \mapsto [0, 1] \tag{4.18}$$

We call the estimated probability the prediction score of S for the query q and denote it as psc_S^q. The higher the computed prediction score, the more likely S matches q.

We denote $PM = \{pm\}$ the space of possible prediction models and $TS = \{ts\}$ the space of time series of sensors. A prediction model pm_S for a sensor S is constructed from a (long) time series ts_S of S by the function

$$pmConst_S : TS \mapsto PM \tag{4.19}$$

We define the set of indexed sensors, i.e., those whose prediction models have been constructed and indexed in the database system, as

$$IS = \{(id_{S_1}, pm_{S_1}), (id_{S_2}, pm_{S_2}), ...\} \tag{4.20}$$

where pm_{S_i} is a prediction model constructed from a time series obtained from sensor S_i, and id_{S_i} is a unique identification of sensor S_i (e.g., an URI or a unique description of the sensor).

With the above notation, we formulate the content-based sensor search problem as follows:

Given:

- The set of indexed sensors IS
- A search query $q = ([a, b], h)$ at time t_q

Find: The rank list of sensors that match q, which is defined as

$$rl_q = [(id_1, psc_1), (id_2, psc_2), ..., (id_k, psc_k)] \tag{4.21}$$

where $k \in \mathbb{N}$ is a user-defined parameter specifying the requested number of matching sensors, id_i are the identifications of matching sensors from IS, and $psc_i \in [0, 1]$ are prediction scores indicating how well sensor id_i matches the query q. The rank list rl shall be sorted by prediction score, that is, $psc_i \geq psc_{i+1}$. In other words, the rank list shall contain an entry (id, psc) if and only if there exists an entry (id, pm) in IS such that $psc = psc(pm_S, q)$ is among the k-highest prediction scores over all id.

4.3.4 Content-based Sensor Search Architecture

Our content-based sensor search service can be fitted into the generic sensor search architecture outlined in Sec. 4.1.4, with appropriate modifications and additions (see Fig.4.19). In particular, a fuzzy set of a sensor S is used as a prediction model pm_S, a matching score for S is defined as the prediction score psc_S, a search criterion is defined by a content-based search query $q = ([a, b], h)$, and the fuzzy set database at each sensor gateway is replaced by a prediction model database.

After receiving all partial rank lists from LSS components, the GSS component merges the partial lists together to form the final rank list that is sorted in descending order of the computed prediction scores (see Eq. 4.21). The GSS component, then, processes the final rank list from top to bottom to verify if sensors are actually matching query q. Verifying a sensor is done by communicating with the sensor through the sensor gateways and asking if the sensor has been producing measurements that fall in the range $[a, b]$ over the time interval h prior to the query submission time of q. The verification process stops when a k actual

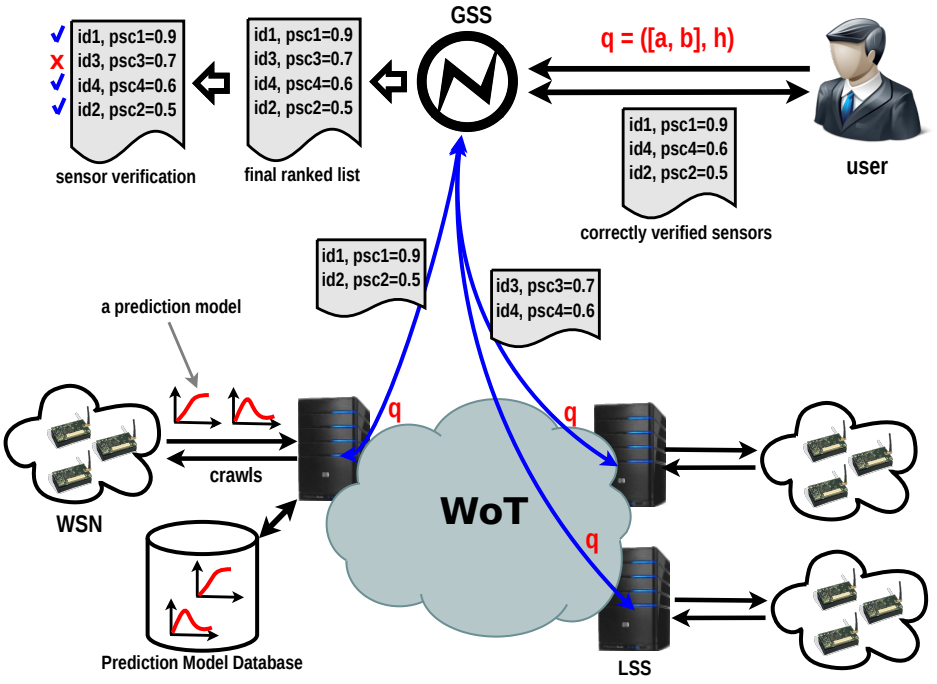


FIGURE 4.19: Content-based Sensor Search: Architecture.

matching sensors have been found. The GSS component presents these k matching sensors, together with their prediction score, to the user as the search result, sorted in descending order of the prediction scores.

4.3.5 Content-based Sensor Search

In this subsection, we present our content-based sensor search algorithm in detail, which includes construction of a compact prediction model from a time series of sensor data, adaptation of a prediction model to changes in recent sensor data, and computation of prediction scores for an indexed sensor given a search query.

Based on the key concept of fuzzy set theory that we presented in Sec. 4.1.4.1, we design a prediction model called time-independent prediction model (TIPM), which consists of two components, namely *sensor measurement density* and *sensor measurement stability*. We will present TIPM in the following steps: (i) how it is constructed at sensor nodes; (ii) how it is used to evaluate a search query; (iii) how it adapts to recent changes of the sensor's measurements at a LSS component; and (iv) how its size is reduced for efficient wireless communication.

We are going to construct TIPM for a sensor using its past measurements. For the sake of explanation, we consider a temperature sensor S monitoring a room, whose time series of measurements over the time period $[t_c - w, t_c]$ is ts_S, where t_c is the time at which the model is constructed, and w is a given time window. We denote x^S_{min} and $x^S_{max} \in ts_S$ as the minimum and maximum values among the measurements of sensor S over w, respectively.

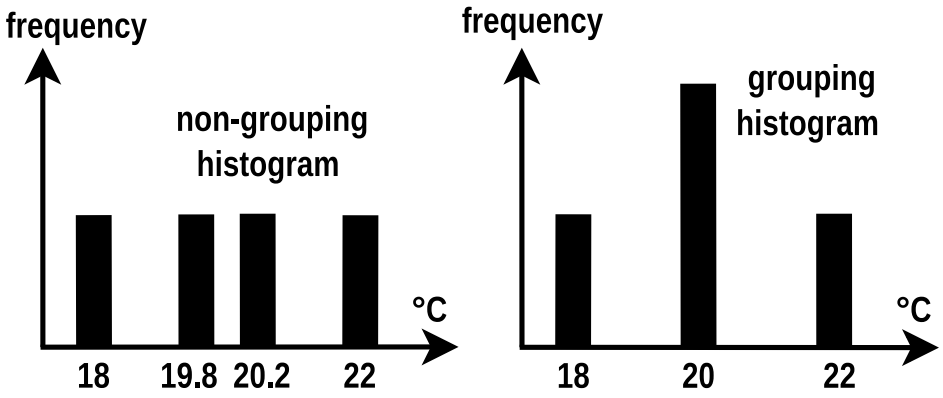


FIGURE 4.20: Histogram for temperature measurements.

4.3.5.1 Sensor Measurement Density

To estimate the probability that a given value is currently read by a sensor, a histogram of the past sensor measurement distribution could be used. For example, by looking at the histogram of measurements of a temperature sensor monitoring a room shown in Fig. 4.20-right, we notice that the room temperature was mainly 20°C during last 24 hours. Since we are predicting if S is reading a value x, it can be assumed that the more times S read x in the past, the higher the probability it is reading x at the current moment. Thus, a histogram of every temperature readings of S in the last w time units helps predicting current readings of S.

In reality, however, a simple histogram over sensor measurements is of limited benefit. The reason is that sensor measurements are inherently imperfect (see $C4$ and $R7$) due to the presence of jitter and noise. Moreover, we, as human beings, usually are not able to differentiate between slight changes in the state of a physical entity. For example, temperature values of 19.8° and 20.2° are, for many practical purposes, indistinguishable from 20°C by a human user (who is, for example, searching a well-tempered room).

Fig. 4.20 illustrates this limitation. In Fig. 4.20-left, the frequency of 18°, 19.8°, 20.2°, and 22° are the same, which means all those temperature values would be predicted with the same probability. However, if we group temperature measurements that are close to each other and compute a histogram value for the group as in Fig. 4.20-right where 19.8° and 20.2° are grouped, we can say that a temperature value close to 20°C is being read now with high probability as the frequency of the small range around 20° is twice as high as the frequency of the small ranges around 18° and 22°.

Inspired by this observation, our approach is not to count the exact number of occurences of distinct sensor measurements x, but to consider the *density* of sensor measurements *surrounding* x. Considering a real value $r_d \in (0, \frac{x^S_{max} - x^S_{min}}{2}]$ and the interval $dd_x = [x - r_d, x + r_d] \subset [x^S_{min}, x^S_{max}]$, the size of the population of sensor measurements in dd_x is proportional to the probability that x or sensor measurements y that are r_d-close to x (i.e., $|x - y| \leq r_d$) will

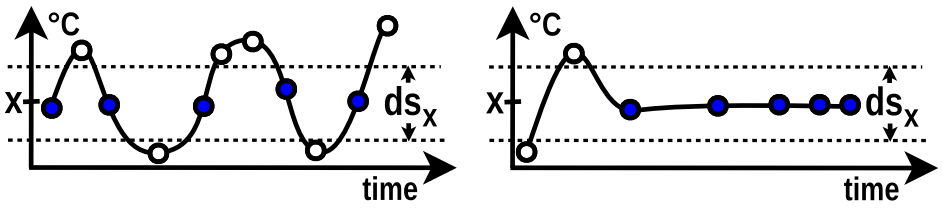


FIGURE 4.21: Stability illustration.

be read by S in the future. By sliding dd_x over $[x^S_{min}, x^S_{max}]$ we can calculate this density for each sensor measurement x in the measurement range $[x^S_{min}, x^S_{max}]$.

In Sec. 4.2.6, we introduce a formula in the Eq. (4.10) to evaluate the neighborhood density of sensor measurements within a given range of values. This formula could be used as a method for computing our desired density, which we denote as $md^w_S(x)$ for a sensor measurement x, and is given by

$$md^w_S(x) = \frac{1}{|ts_S|} \sum_{i=1}^{|ts_S|} e^{-\frac{|x-S(t_i)|}{r_d}} \tag{4.22}$$

Due to the exponential function, sensor measurements which are outside of $[x - r_d, x + r_d]$ have little influence on $md^w_S(x)$. We call $md^w_S(x)$ the *sensor measurement density* function. Note that, $md^w_S(x)$ is exactly the histogram value of x when $r_d = 0$. However, as mentioned above, r_d should be greater than zero to reflect human perception as well as the jitter and noise of raw sensor data.

4.3.5.2 Sensor Measurement Stability

As proposed in the beginning of this section and formulated in Sec. 4.3.3, we consider search queries of the form $q = ([a, b], h)$. This implies searching for sensors whose sensor measurements fall "almost continuously" within $[a, b]$ over the time interval $[t_q - h, t_q]$.

The adverb "almost continuously" is a reflection of the human perception that a physical entity is at a certain state, meaning it has been in that state for a certain period of time. A change of the perceived state of the entity normally corresponds to a significant change of measured sensor values from one range to another distinct or overlapping range. For example, in Fig. 4.18, if you are standing at the northern part of the hill and feeling that the wind blows towards you, that means the wind has been lately blowing north. In other words, the wind sensor measurements *continuously* fall between 80° and 100° in the last 1 hour. Although you may notice there are points in time (a, b, c, d in Fig. 4.18) when the wind did not blow north, *"almost"* all of the time during the last hour it did.

Considering our sensor S, we model a certain "perceived temperature" x of the room as the range $ds_x = [x - r_s, x + r_s]$, where $r_s \in [0, \frac{x^S_{max} - x^S_{min}}{2}]$. We are interested in how *continuous in time* are the temperature measurements that fall within $[x - r_s, x + r_s]$ during w. The closer in time those measurements are sampled one after another, the more *stable* we consider that

room temperature is at x during w. An illustration of stability is given in Fig. 4.21, where the distribution of 5 sensor measurements in Fig. 4.21-right is considered more stable than the distribution of other 5 sensor measurements in Fig. 4.21-left (sensor measurements fall within ds_x are depicted as filled circles, whereas ones do not are not filled).

To formally express stability, we denote the time series of temperature measurements that fall in ds_x during w as

$$ts_S^x = S(t_1), S(t_2), ..., S(t_n) | S(t_j) \in ts_S, j = 1..n \tag{4.23}$$

where $n = |ts_S^x|$ is the number of temperature measurements found between $[x - r_s, x + r_s]$, t_j is the time at which $S(t_j)$ was sampled. Note that the set of $\{t_j\}$ is a subset of the set of time stamps $\{t_i | i = 1..|ts_S|\}$, and we assume that $\forall j, t_j < t_{j+1}$.

We compute the stability ms_S^w of sensor measurements around x during w by summing up the exponentially weighted distances of adjacent timestamps t_j and t_{j+1} (i.e., larger distances have an exponentially smaller weight). This way, distributions like the one given in Fig. 4.21-right result in a much larger stability value than the one in Fig. 4.21-left, due to the exponential weighting. We then normalize the sum to [0, 1] by dividing it by the number of evaluated distances. Formally we obtain

$$ms_S^w(x) = \frac{1}{n-1} \sum_{j=1}^{n-1} e^{-\frac{t_{j+1} - t_j}{t_n - t_1}} \tag{4.24}$$

By sliding ds_x over $[x_{min}^S, x_{max}^S]$, we can calculate a stability value for every sensor measurement $x \in [x_{min}^S, x_{max}^S]$. We call $ms_S^w(x)$ the *sensor measurement stability* function.

4.3.5.3 Constructing TIPM

Our prediction model is based on the density and stability of sensor values as introduced above. In particular, the higher the product of the density and stability values of x, the greater the probability that S is currently reading x. Formally, we obtain the prediction model function $pm_S^w(x)$ for the time series ts_S during w:

$$pm_S^w(x) = md_S^w(x) \times ms_S^w(x) \tag{4.25}$$

With this TIPM defined, if we search for sensors that are reading x, those sensors S that have highest values of $pm_S^w(x)$ will be ranked highest. If we search for sensors that are reading either x or y, those S that have highest values of $pm_S^w(x) + pm_S^w(y)$ will be ranked highest. In general, if we search for sensors reading a value in the range $[a, b]$, those S that have highest

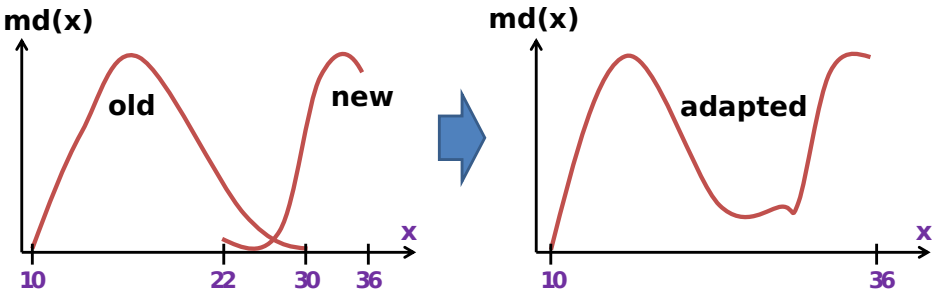


FIGURE 4.22: TIPM Adaptation.

values of

$$\sum_{x \in [a,b]} pm_S^w(x) \tag{4.26}$$

will be ranked highest. To avoid performing this summation for each query, we precompute and store the cummulative distribution function of $pm_S^w(x)$:

$$cdfpm_S^w(x) = \sum_{x_{min}^S \leq y \leq x} pm_S^w(y) \tag{4.27}$$

4.3.5.4 Query Resolution

When a search query $q = ([a, b], h)$ is submitted by a user, a rank list that contains matching sensors is presented to the user. This implies two actions: (1) a prediction score is computed for each indexed sensor; and (2) highly ranked sensors are contacted to verify if they actually match q. Action (2) is executed by the GSS component once the final rank list is completed. Action (1) is performed at LSS components using indexed prediction models. We present in the following how a prediction score is computed using TIPM.

Once the search query is directed to a LSS component, all indexed sensors V with $x_{max}^V < a < b$ and $a < b < x_{min}^V$ are ruled out to narrow down the search space. For the remaining sensors, we calculate for each sensor V its prediction score as the sum of the estimated probabilities for every $x \in [a, b]$ that x is being read at the query time. Since we already have the precomputed distribution function (Eq. 4.27), the prediction score psc_V^q is obtained by simply substracting the values of the cummulative fuction for b and a, and normalizing the value to the range $[0, 1]$ by dividing it by the total cummulative probability of V:

$$psc_V^q = \frac{cdfpm_V^w(b) - cdfpm_V^w(a)}{cdfpm_V^w(x_{max}^V) - cdfpm_V^w(x_{min}^V)} \tag{4.28}$$

4.3.5.5 TIPM Adaptation

With requirement $R2$ we specified that the prediction model of an indexed sensor should be regularly updated to reflect significant changes in the sensor's ouput. However, as we are using past information of the monitored physical entity to predict its future behaviour, it is

desired that this information is not totally discarded, but gradually fades away as time goes by. If we look at Eq. 4.22 and Eq. 4.24 again, we can see that the density and stability functions are additively constructed over the sensor's measurement range. Thus, we employ the exponentially weighted moving average technique to implement the adaptation process. A function is adapted by combining the old and the new versions of the function over the union of the two measurement ranges. Formally we have

$$md_S^w(x) = (1-\alpha) \times md_S^w(x)^{old} + \alpha \times md_S^w(x)^{new} \tag{4.29}$$

and

$$ms_S^w(x) = (1-\alpha) \times ms_S^w(x)^{old} + \alpha \times ms_S^w(x)^{new} \tag{4.30}$$

where the *old* and *new* indicators represent old and new density and stability functions of sensor S, respectively. The factor α is a *fading factor* assigned to the old and new functions of S's prediction model. Through α we can control how "important" are the recent measurements of sensor S in comparison with S's measurements in the more distant past.

As time goes by, the resulting union of measurement ranges describes the value domain whereas the adapted function predicts the sensor output. For example, in Fig. 4.22, the combined value domain of the density function of the sensor in consideration is $[10, 36]$, which is the union of the two value domains $[10, 30]$ and $[22, 36]$.

4.3.5.6 TIPM Size Reduction

Similar to reducing the storage overhead for a similarity model in Sec. 4.2.8, the storage overhead for TIPM can be greatly reduced by representing it as a set of line segments that approximate it. This is possible due to the observation that the curves of the component functions (i.e., density and stability) of TIPM are typically smooth due to the exponential weighting. Thus, we apply the same approximation process as presented in Sec. 4.2.8 to the density and stability functions, and perform Eq. 4.25 and Eq. 4.27 based on the approximated ones.

4.3.6 Evaluation

In this subsection we evaluate the performance of our content-based sensor search algorithm using simulation. We implement a simulation tool using Java that is able to replay recorded measurements of multiple sensors, execute search operations over these sensors, and compute the communication overhead incurred by the resulted rank list. Note that the search results presented to the user are always accurate, as the search algorithm verifies that sensors actually match the search query. Therefore, the rank list only influences the communication overhead but not the accuracy.

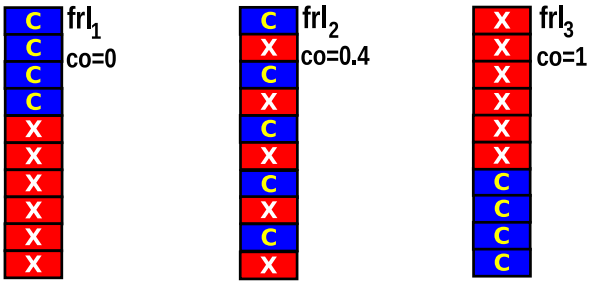


FIGURE 4.23: Communication overhead of a rank list.

4.3.6.1 Communication Overhead of a Rank List

To be able to assess the performance of our search algorithm, we need to define a method to measure the communication overhead incurred by the final rank list frl_q that is computed by the search algorithm based on a search query $q = ([a, b], h)$. In order to present the user the search result (see Eq. 4.21), the search algorithm has to process frl_q from top to bottom to find the top k-matching sensors (see Sec. 4.3.3). A sensor is matching the search query if its sensor measurements actually fell in the value range $[a, b]$ during the time interval $[t_q - h, t_q]$, else the sensor does not match.

Since sensor verification requires communication between the global search component and the sensor nodes, the optimal rank list would have all matching sensors ranked on top, thus minimizing communication overhead. Fig. 4.23 shows possible rank lists obtained after q is evaluated. The character "C" indicates matching sensors, while character "X" indicates non-matching ones. The best possible result is list frl_1 as all matching sensors are ranked highest. The worst result is list frl_3.

We now define a metric that maps a rank list to a scalar value between 0 (best result) and 1 (worst result). For each matching sensor, we compute the ranking error, i.e., the number of non-matching sensors ranked higher. We then compute the average ranking error of all matching sensors, which equals 0 in the best case, and equals the number of non-matching sensors in the worst case. To normalize to the interval $[0, 1]$, we devide by the number of non-matching sensors. The resulting value is the desired metric. Thus, we define the *communication overhead* (co) incurred by a rank list rl as follows:

$$co(rl_q) = \frac{1}{m(rl_q)\,[|rl| - m(rl_q)]} \times \sum_{i=1}^{|rl|} e_{rl_q}(i) \tag{4.31}$$

where $|rl_q|$ is the length of rank list rl_q, $m(rl_q)$ is the number of matching sensors in rl_q, and $e_{rl_q}(i)$ is the ranking error of a matching sensor at rank i, i.e., the number of non-matching sensors ranking higher than i. If i is a non-matching sensor, then $e_{rl_q}(i) := 0$. Fig. 4.23 shows the value of the metric for different rank lists.

4.3.6.2 Simulation Setup

We use three sensor data sets that were mentioned in Sec. 4.2.9 for our evaluation, namely IntelLab, NOAA, and MavHome data sets. For each data set, we select the size of the periodical time window w for constructing the fuzzy-based prediction model function to be 24 hours. This is a natural choice as the stae of physical entities tends to repeat day after day. We simulate the process of a human posing search queries by a Poisson process with $\lambda = 144$ over 24 hours (i.e., on average 1 query each 10 minutes).

When there is a query $q = ([a, b], h)$, the range $[a, b]$ is randomized within $[A, B]$ where A and B are the minimum and maximum values of the sensor readings of all sensors in the data set, respectively. The time interval h is fixed and smaller than w. The query time t_q is randomized between $[0, w]$. For each search query, we obtain a rank list whose communication overhead is determined by Eq. 4.31. The average of the communication overhead of all queries posed within a day, i.e., 24 hours, is used as the communication overhead of our search algorithm during that day. At the end of each simulated day, sensors compute the density and stability functions of their prediction model and send them to the gateway for adaptation and indexing. The simulation is run until one of the sensors of the data set has reached its last measurement. We repeat the simulation for 100 times and compute the average as well as the standard deviation of the communication overhead of our search algorithm.

We perform evaluation on 4 search spaces. The first is a set of 59 sensors of type humidity and temperature from the IntelLab data set. The second is a set of 80 sensors of types air temperature and water temperature from the NOAA data set. The third is a set of 23 sensors of types light, humidity, heat from the MavHome data set. And the last search space is the combination of the three search spaces. Evaluation on this combined search space gives us an idea of how our search algorithm would perform in real life where sensors have different types and diverse deployment contexts.

4.3.6.3 Tuning Parameters

There are several tunable parameters in the design of our content-based sensor search algorithm. By performing extensive evaluations with different values of the parameters we could experimentally gain more insight regarding the behaviour of the search algorithm. In the following, we briefly discuss these parameters, and in the next sub-section we will present numerical results obtained by varying the values of these parameters.

- The fading factor α is used to control the importance that we assign to recent sensor data with respect to the data from the more distant past. We conducted evaluations with varying values of α to investigate the effect of this parameter on the average communication overhead.

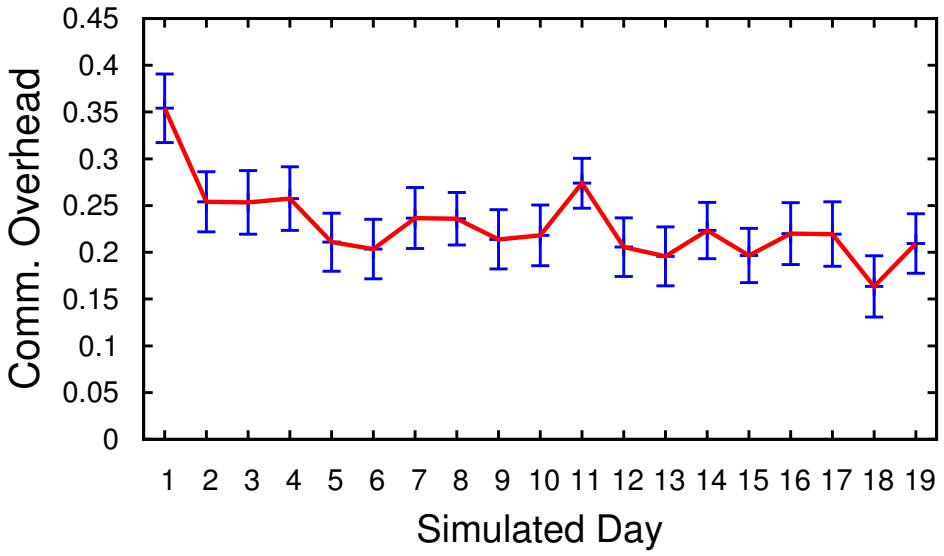


FIGURE 4.24: IntelLab: Communication overhead with $\alpha = 0.3$.

- The time interval h controls how long in the past from the query time backward we use sensor data for verifying if a sensor is matching the search query. We expect that the greater the value of h the lower the average communication overhead would be.
- As the result for a search query q, we receive a rank list rl_q sorted by prediction scores psc (see Eq. 4.28). We denote psc_{min} as the minimum psc of all sensors in rl_q. We would expect that the larger the number of sensors available in the IoT, the more likely sensors exist that match a query, and the more likely we obtain a rank list with a large psc_{min} value. However, in our experiments we only have a limited number of sensors (162 sensors). To get a feeling of how the TIPM would perform with a larger number of sensors (and thus with a higher psc_{min}), we also evaluate the communication overhead of those rl_q whose psc_{min} is greater than a threshold value. We performed evaluations with different threshold values and found that average communication overhead decreases when we increase the threshold value.

4.3.6.4 Numerical Results

In the following, we discuss in detail the evaluation results of all search spaces (the combined one also included) with 2 different values of α, namely 0.3, 0.8, to investigate its effect on the average communication overhead (ACO). Other tunable parameters are fixed, i.e., h=60 minutes and psc_{min}=0. Other values of these parameters will be presented and discussed later, for the combined data set.

Fig. 4.24 and Fig. 4.25 are evaluation results for the search space from IntelLab data set. We can see the common trend that the ACO decreases over time, which reflects the effect of our adaptation process. ACO on day 1 is around 0.35 and is much improved from day 2 to day 19 to be around 0.23 on average. A closer look at the figures reveals that $\alpha = 0.3$ results

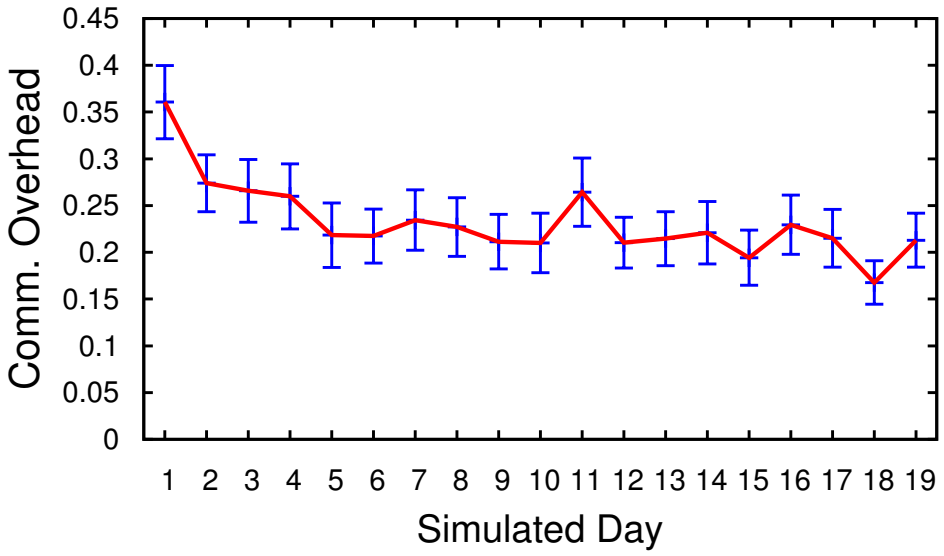


FIGURE 4.25: IntelLab: Communication overhead with $\alpha = 0.8$.

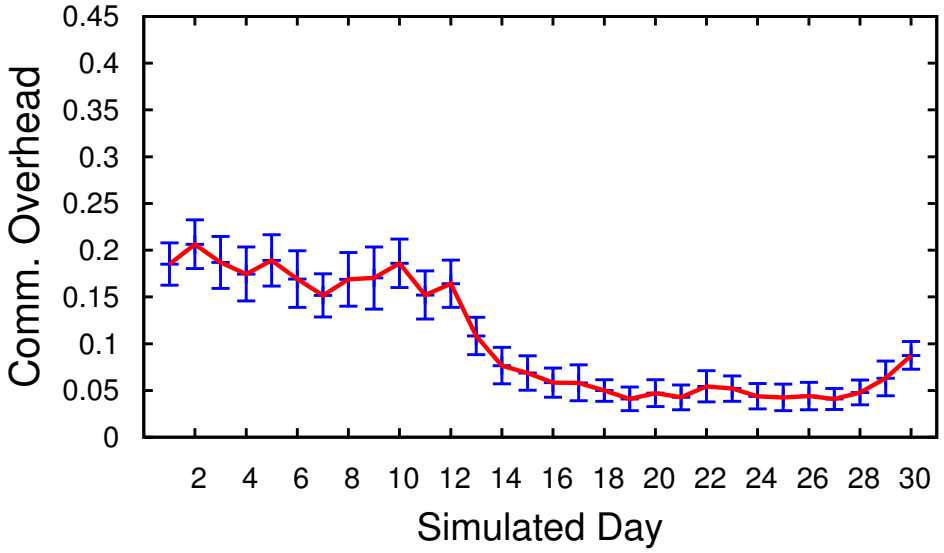


FIGURE 4.26: NOAA: Communication overhead with $\alpha = 0.3$.

in lower overall ACO than does $\alpha = 0.8$. This confirms our approach that not only recent historical sensor data but also data from the more distant past should be used for prediction (via the adaptation process).

Fig. 4.26 and Fig. 4.27 are evaluation results for the search space from NOAA data set. Similar to the IntelLab data set, we also observe here the trend of decreasing ACO over time. The ACO from day 14 onward is smaller than 0.1, which is very accurate. Also, we can see that $\alpha = 0.3$ results in a slightly smaller overall ACO than does $\alpha = 0.8$. So the same conclusion that sensor data from the more distant past should be used for prediction can again be drawn for the NOAA data set.

Similar observations are found for the search space from MavPad data set in Fig. 4.28 and Fig. 4.29, and for the combined search space in Fig. 4.30 and Fig. 4.31. That is, the ACO

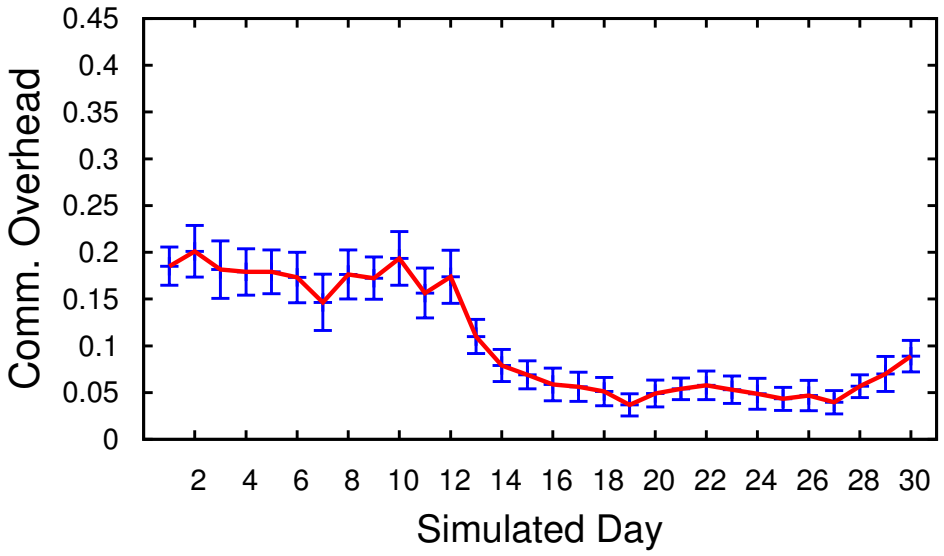


FIGURE 4.27: NOAA: Communication overhead with $\alpha = 0.8$.

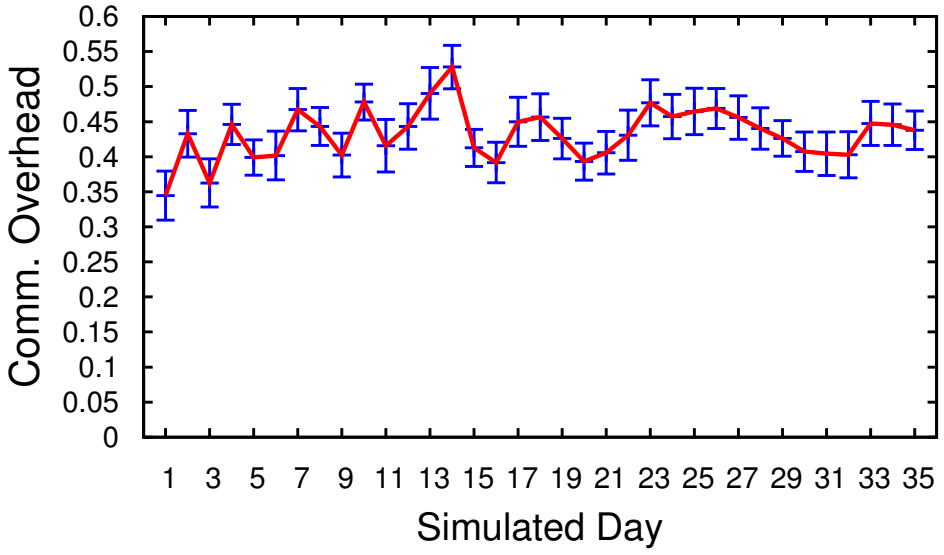


FIGURE 4.28: MavPad: Communication overhead with $\alpha = 0.3$.

decreases over time and $\alpha = 0.3$ results in a better overall ACO than does $\alpha = 0.8$.

With a general look at all results of the 4 search spaces, we notice the ACO decreases in order of MavPad, IntelLab, combined, and NOAA data sets. The reason is, probably, the nature of sensor data in different data sets. For example, the Intel Lab is a small, closed, and mostly static environment, so we expect a low level of heterogeneity among the time series of sensors of the same type in the IntelLab data set. In contrast, sensors of the NOAA data set are spread over a large geographical area along the coast line in the northern part of America. Therefore, weather sensors of the same type in the NOAA data set experience a high level of heterogeneity in their time series. Thus, we expect that searching in the NOAA data set would result in rank lists containing sensors with high *psc* value. Fig. 4.32 illustrates the low level of heterogeneity in the time series of 2 humidity sensors in the IntelLab data

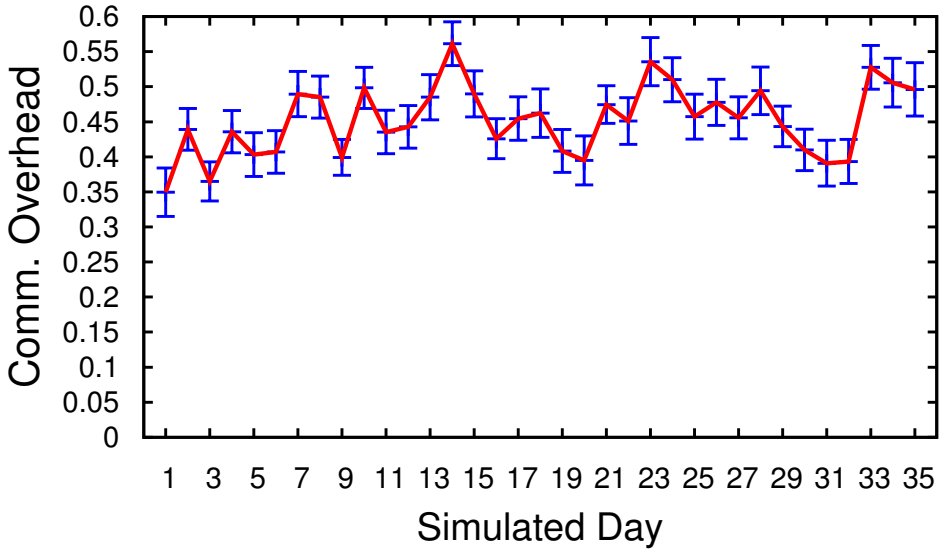


FIGURE 4.29: MavPad: Communication overhead with $\alpha = 0.8$.

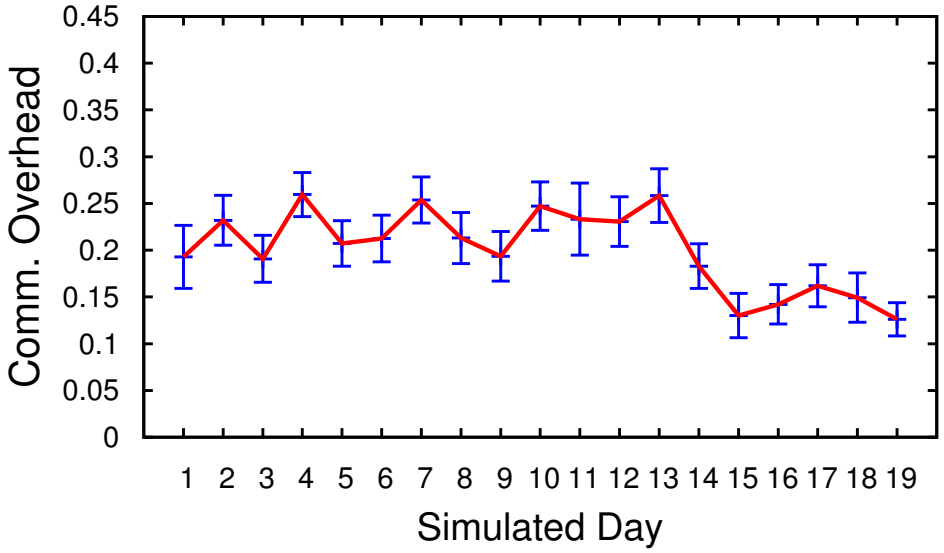


FIGURE 4.30: Combined: Communication overhead with $\alpha = 0.3$, $h = 60$, and $psc_{min} = 0$.

set while the opposite is observed in the time series of 2 air temperature sensors from the NOAA data set.

Besides α, the evaluation results also confirm our predictions on other tunable parameters. In particular, ACO decreases when increasing h and psc_{min}. For example, a comparison between $h = 60$ and $h = 120$ (α=0.3, psc_{min}=0) is presented in Fig. 4.30 and Fig. 4.33, where we can see a small improvement of ACO from $h = 60$ to $h = 120$.

Fig. 4.30, Fig. 4.34, and Fig. 4.35 show how accurate were the prediction models constructed during the simulation in terms of ACO. The evaluation was performed on the combined data set. We see that the ACO decreases when we increase the threshold value for psc_{min}, i.e., psc_{min}=0, psc_{min}=0.2, and psc_{min}=0.5, which confirms the accuracy of the proposed TIPM. This hints that the communication overhead decreases as the number of sensors in the IoT

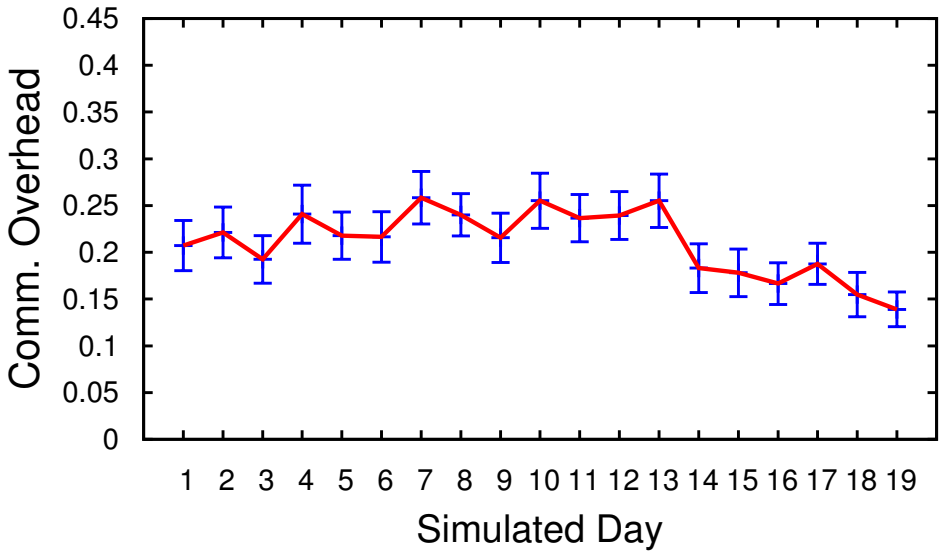


FIGURE 4.31: Combined: Communication overhead with $\alpha = 0.8$.

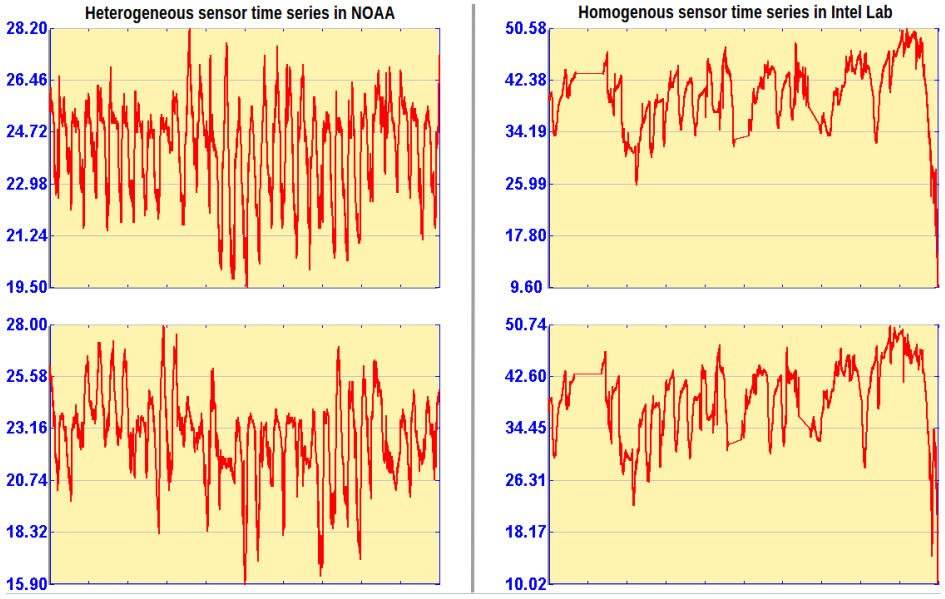


FIGURE 4.32: Heterogeneity VS. homogeneity in time series of sensors in different data sets.

increases. The reason is that in a real IoT, sensors usually have highly different types and diverse deployment contexts, thus resulting in highly heterogenous time series of sensor data. Therefore, the probability to obtain rank lists with small psc_{min} decreases with increasing number of sensors.

4.3.7 Conclusion

We proposed the content-based sensor search service for finding sensors in the WoT based on their current raw output, and developed a search algorithm for it. The novelty of our search algorithm lies in supporting search based on raw sensor data which requires no domain

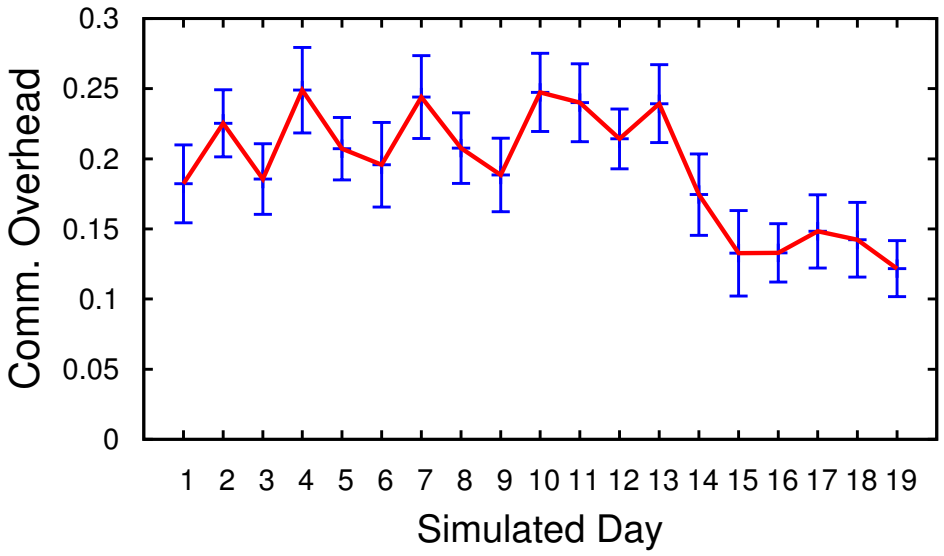


FIGURE 4.33: Combined data set: $h = 120$.

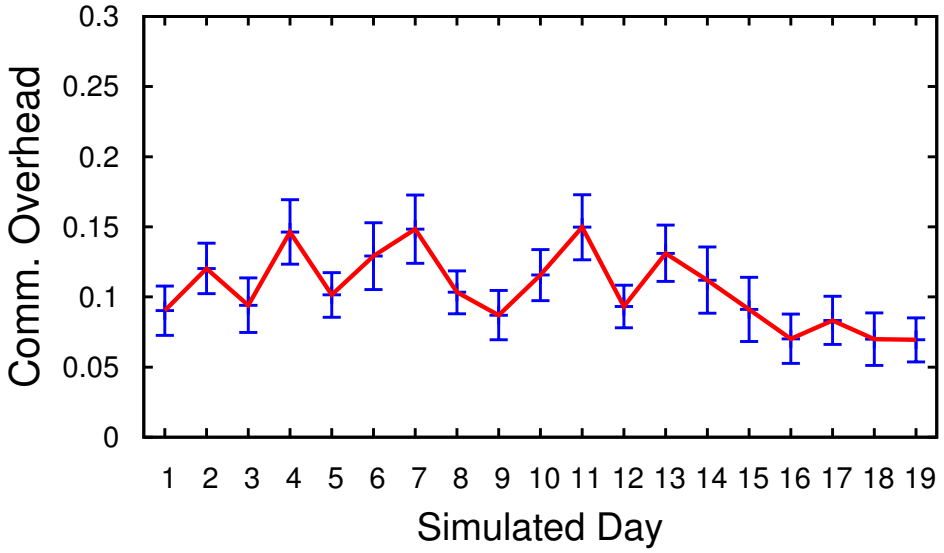


FIGURE 4.34: Combined Data Set: $psc_{min} = 0.2$.

expertise for deriving high-level states from raw sensor data and gives users the flexibility of defining their own application-specific search. The proposed hierachical architecture scales well with the number of sensors in the IoT as computations and storage are distributed over multiple sensor gateways. Moreover, since only compact prediction models are periodically downloaded from sensor nodes, wireless communication is minimized which is crucial for maximizing life time of battery-powered sensor nodes. The adaptation process that is performed for each periodical download of a prediction model helps our service to deal with the dynamicity of the states of the real world. At the same time, the fuzzy approach addresses the imperfections of sensor data obtained from low-cost sensor hardware. We performed an extensive evaluation of our search algorithm on a number of different data sets obtained from real-world deployments, and showed that the search result is accurate and therefore the search algorithm has low communication overhead.

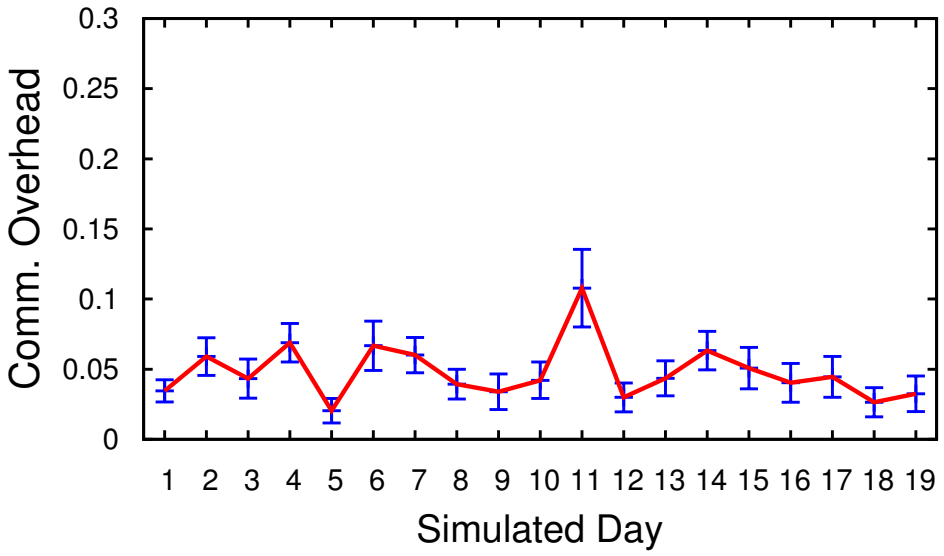


FIGURE 4.35: Combined Data Set: $psc_{min} = 0.5$.

4.4 Summary

In this chapter, we have discussed the need for sensor search in the WoT and proposed two novel sensor search services for the WoT. Firstly, the sensor similarity search service enables users to find sensors in the WoT whose output measurements are similar to the measurements of an example sensor. Secondly, given a search query that is composed of a range of sensor values and a time interval, the content-based sensor search service returns sensors that have been reading values in the given value range during the given time interval prior to the query submission time. We identified a set of challenges and derive the requirements that sensor search services and algorithms for the WoT should meet, and proposed a generic scalable architecture and an approach for both the services. For each sensor search service, we designed a sensor search algorithm, implemented, and evaluated it. Despite the severly constrained resources of embedded sensor nodes in the IoT, where wireless communication must be minimized and computational and storage capacities are several orders of magnitude more constrained than those of general purpose computers, our proposed algorithms offer scalable search and address the imperfection of sensor data obtained from low-cost platforms.

Chapter 5

A Prototypical Sensor Search Engine

In this chapter we present a prototypical sensor search engine to demonstrate the practical feasibility and usability of the sensor search services and algorithms that we proposed and developed in Chapter 4. The search engine supports searching the Web for sensors that are connected to and publishing their data on *Xively*[1], a cloud-based platform for deploying IoT devices and building IoT applications on top of them. Via a proper graphical user interface (GUI), users can find sensors that are similar to an example sensor (i.e., sensor similarity search service), or users can find sensors with certain properties (i.e., content-based sensor search service).

The structure of this chapter is as follows. In Sec. 5.1 and Sec. 5.2 we present the software architecture of the search engine and its implementation. In Sec. 5.3, we describe the GUI of our search engine. In Sec. 5.4, we present several demonstrations for the sensor search services. Finally, Sec. 5.6 concludes this chapter.

5.1 Software Architecture

The software architecture of our sensor search engine is given in Fig. 5.1 which consists of the *Global Sensor Search* (GSS), multiple *Local Sensor Search* (LSS) and multiple *Search Clients* (SC) that are distributed across the Internet. Each LSS registers with the GSS and is responsible for searching multiple *Sensor Data Sources* (SDS).

Inside an LSS, there are multiple *Crawlers* each of which is responsible for an SDS. Periodically, a Crawler requests its SDS to discover new sensors, download sensor data or fuzzy sets (i.e., similarity and prediction models) for both new and already indexed sensors, construct fuzzy sets from sensor data if required, index, and update the fuzzy sets in the *Fuzzy*

[1] https://xively.com/

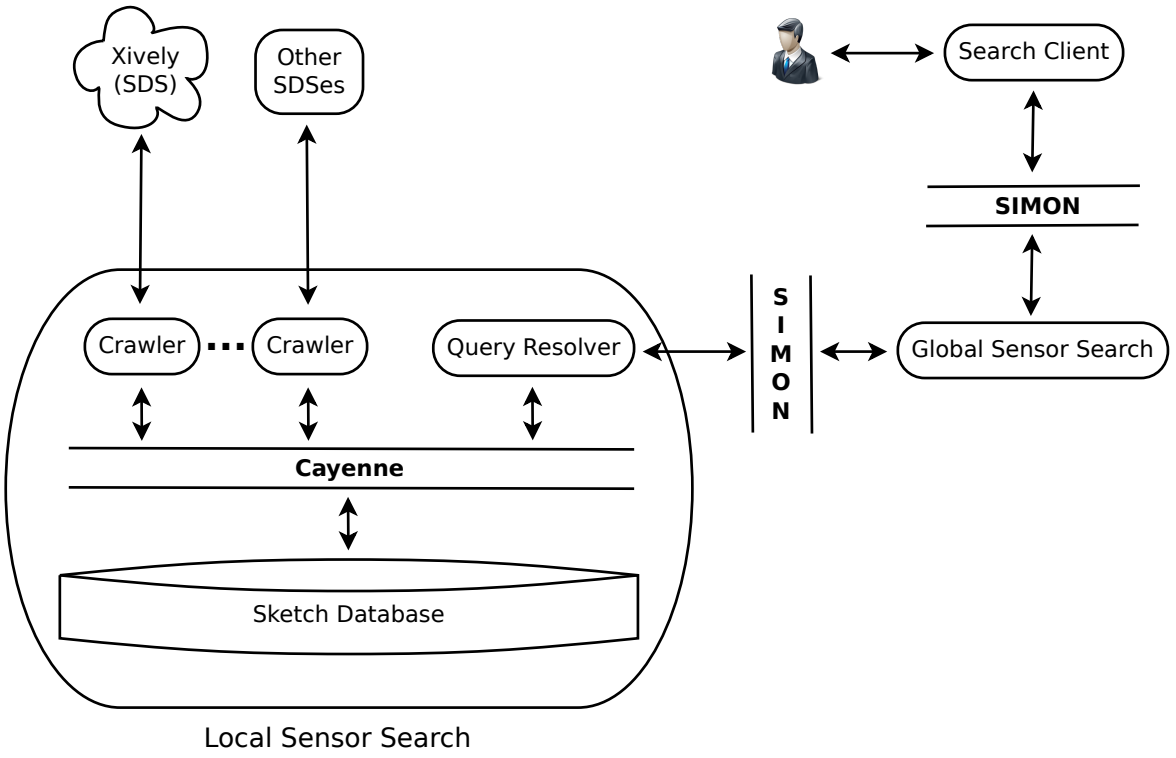


FIGURE 5.1: Prototypical search engine: Software architecture.

Set Database (FSDB). The *Query Resolver* is responsible for resolving search queries (including both sensor similarity search and content-based sensor search), building partial rank lists, and verifying if sensors from the SDSes actually match search queries as requested by the GSS. The FSDB is responsible for organizing storage and access to data generated by Crawlers.

The SC provides the user with a graphical user interface (GUI) for creating search queries for both sensor similarity search and content-based sensor search services and for presenting the user the search results.

The GSS is responsible for accepting search queries from an SC, forwarding them to LSSes, merging partial rank lists sent by LSSes, requesting LSSes for sensor verification if needed, and forwading the final search result to the SC.

5.2 Software Implementation

We implemented our sensor search engine using the Java programming language. In our current implementation, all components reside at and run on the same computer with an Intel Core i5 CPU clocked at 2.4 Ghz. However, they can be as well placed on different computers across the Internet by simply modifying a config file without touching the code.

At a high level, the implementation of the sensor search engine consists of 6 interdependent units corresponding to 6 separate Java projects. The *searchenginelib* project implements the

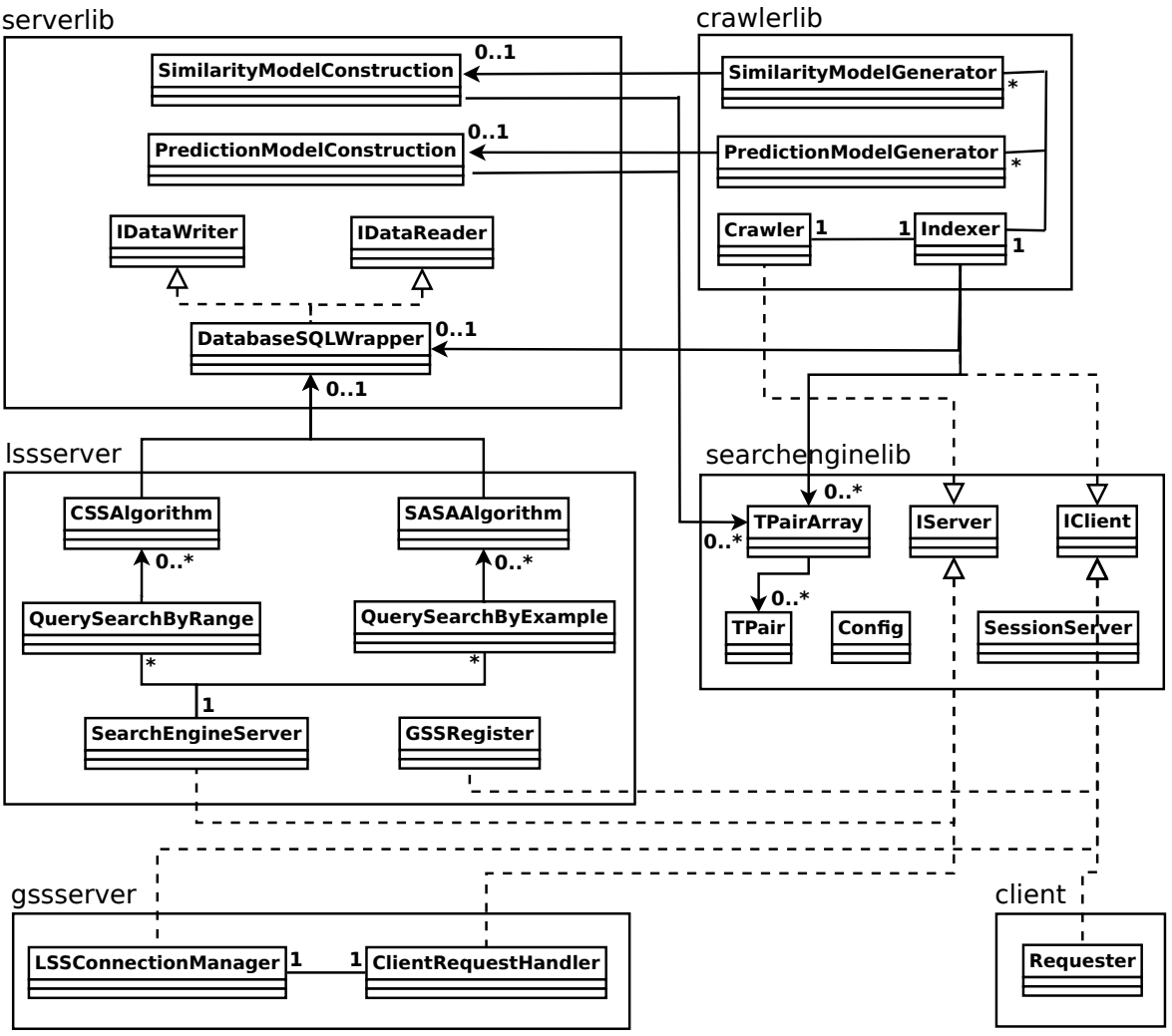


FIGURE 5.2: Protypical sensor search engine: UML class diagram

most essential functions for use by all other units, which include sensor data manipulation, client/server interfacing, client/server connection management, and system configuration. The *serverlib* project implements the basic functions such as similarity model construction, prediction model construction, and database organization and access, which are used by the Crawler and Query Resolver components. The *crawlerlib* project implements the functions for use by a Crawler including interfacing with the SDSes, sensor discovery, sensor indexing and caching, and sensor data downloading. The *lssserver* project implements the main function of the Query Resolver component (i.e., resolving sensor search queries), and other functions of the LSS component such as LSS registration. The *gssserver* project implements the GSS component and its functions. Finally, the *client* project implements the SC component, which includes a GUI and the communication between the SC and the GSS components.

The most important Java classes of these Java projects and the relationships between these classes can be described by the UML class diagram given in Fig. 5.2. In this figure, a Java

project is represented as a solid rectangle whose caption is the project's name, and the most important classes of the project is drawn within its representation rectangle.

In the *searchenginelib* project, a data point is defined by the class $TPair$ which represents a pair of real values. A time series of sensor data is defined by the class $TPairArray$ whose instances are basically an $ArrayList$ of instances of $TPair$. The manipulation of sensor data is implemented within this class. While the two Java interfaces $IClient$ and $IServer$ define the necessary methods for a client/server communication session, the $SessionServer$ class is responsible for managing multiple and simultaneous client/server communication sessions. The $Config$ class defines how the sensor search engine can be configured.

In the *serverlib* project, the two interfaces $IDataWriter$ and $IDataReader$ define the necessary methods for database access (e.g., writing and reading fuzzy sets to and from the database). The $DatabaseSQLWrapper$ class implements these two interfaces for the MySQL database engine. The two classes $SimilarityModelConstruction$ and $PredictionModelConstruction$ are responsible for constructing a similarity model and a prediction model from sensor data (i.e., a $TPairArray$ object), respectively.

In the *lssserver* project, the $SearchEngineServer$ class implements the $IServer$ interface and is responsible for accepting requests from the GSS component, including sensor similarity search (SSS) and content-based sensor search (CSS) requests. The two classes $QuerySearchByExample$ and $QuerySearchByRange$ are two managers for a pool of objects of the two classes $SASAAlgorithm$ and $CSSAlgorithm$, which implement the sensor lookup algorithms for the SSS and CSS services, respectively. These two classes interact with the database via the $DatabaseSQLWrapper$ class. The $GSSRegister$ class implements the $IClient$ interface so that it can register the LSS with the GSS by sending a register request to the $ClientRequestHandler$ class.

In the *crawlerlib* project, the $Crawler$ class implements the $IServer$ interface and is responsible for accepting requests for downloading sensor data from an SDS well as communicating with the SDS. The $Indexer$ class implements the $IClient$ interface so that it can request a Crawler for sensor data, generate similarity models (via the $SimilarityModelGenerator$) and prediction models (via the $PredictionModelGenerator$) from sensor data provided by the $Crawler$ class, and indexing them into the FSDB via the $DatabaseSQLWrapper$ class.

In the *gssserver* project, the $LSSConnectionManager$ class manages all LSSes that are registered with it, distributes requests from the SC component to the registered LSSes, and sends requests for sensor verification to the registered LSSes. The $ClientRequestHandler$ class implements the $IServer$ interface and is responsible for accepting search requests from and returning search results to the SC component. It also accepts other types of request such as LSS registration.

In the *client* project, the *Requester* class implements the $IClient$ interface and is responsible for sending search requests to the GSS component.

Note that, in the current implementation of our sensor search engine, we use several specific tools including SIMON for the communication among the GSS, LSS, and SC components, Cayenne for managing database access, and Xively as an SDS. However, while our sensor search engine works well with these tools, it is not limited to them as it will also work with any other set of tools that provide the same functionalities. In the following subsections we will explain these tools.

5.2.1 Xively

Xively is an IoT-focused cloud service that allows users to connect sensors to and publish sensor measurements on the Web, and to build their own IoT applications by mashing up these sensors, sensor measurements, and Web data and services, using the Xively API[2]. Currently, Xively already connects about 250 million sensors and has about 17 million users. Xively corresponds to the local WSNs in Fig. 4.2.

The base URL of the Xively API is *https://api.xively.com*, from which it is possible to securely read & write sensor data, read & write metadata, and read historical sensor data using HTTP methods such as GET, PUT, and POST. In Xively's terms, a *Feed* is an access point to multiple sensors, their associated metadata, and their time series of measurements. A time series of sensor measurements is represented as a *Datastream*. A Feed is usually a group of related sensors according to certain criteria, e.g., sensors owned by the same person or sensors deployed in the same location.

To get an idea of how the Xively API functions look like, we present in the following an example Xively API call to read a snapshot of a single Feed at the current time, i.e., the current sensor data value of each Datastream and the Feed's metadata. The snapshot can be returned in JSON, XML, or CSV format. The Xively API call for JSON is *https://api.xively.com/v2/feeds/FEED_ID.json*, where FEED_ID is the unique identification of a Feed. In response to this call, the Xively server sends the *200 OK* status code and a JSON body along with it, which looks like the code snippet below.

```
{
  "id": 121180,
  "title": "Example",
  "private": "false",
  "feed": "https://api.xively.com/v2/feeds/121180.json",
  "updated": "2013-08-13T03:25:48.686462Z",
  "created": "2013-04-29T12:50:43.394288Z",
  "creator": "https://xively.com/users/tduccuong",
  "version": "1.0.0",
  "datastreams": [
  {
    "id": "stream1",
    "current_value": "260",
```

[2] https://xively.com/dev/docs/api/

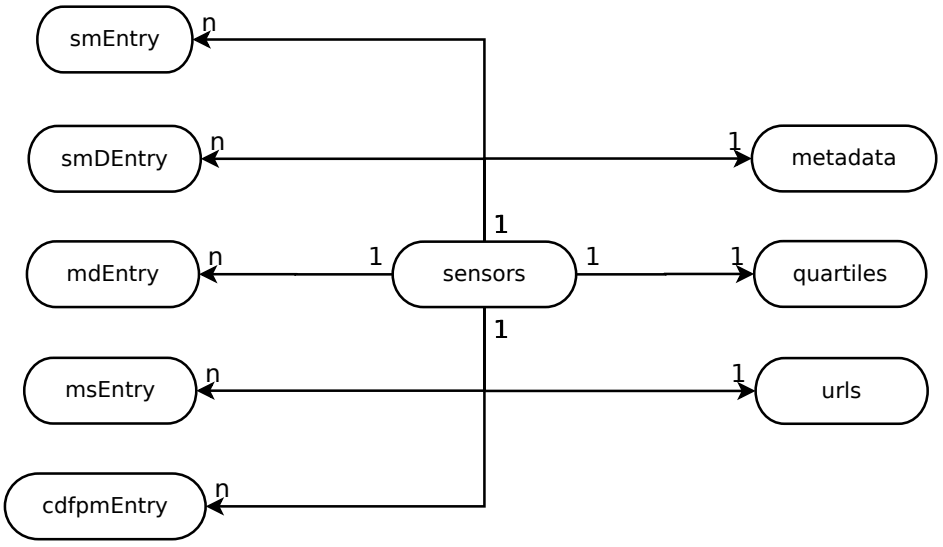


FIGURE 5.3: Database schema for the search engine.

```
    "at": "2013-08-23T01:10:02.986063Z",
    "max_value": "260.0",
    "min_value": "260.0"},
  {
    "id": "key",
    "current_value": "value",
    "at": "2013-08-23T00:40:34.032979Z"
  }],
  "location": {
    "domain": "ct"
  }
}
```

This JSON body contains the current value of the only one Datastream (stream1) of the Feed whose FEED_ID is 121180.

5.2.2 Cayenne

At the time of writing, our FSDB component contains already more than 20000 sensors (i.e., their URL, meta-information, a similarity model, and a prediction model). It contains data tables for both sensor similarity search and content-based sensor search services. In order to avoid dependence on any particular database engine, we use Cayenne[3] as an abstraction layer for managing and accessing a database. Cayenne is an ORM (Object Relational Mapping) framework that allows for seamlessly binding database schemas directly to Java objects, managing atomic database transactions (e.g., commit and rollbacks, joins, sequences), thus enabling programmers to work only with Java objects abstracted from a database. The underlying database engine that we use for the FSDB component is MySQL.

[3]http://cayenne.apache.org

In Fig. 5.3 we present the Cayenne objects (i.e., abstracted database tables) and the relationships among them. The `sensor` object plays the central role to which all other objects are connected to. The `*Entry` objects represent the models and functions built for the search engine (see Sec. 4.2.6, Sec. 4.2.7, and Sec. 4.3.5), including similarity model (`smEntry`), derivative similarity model (`smDEntry`), measurement density function (`mdEntry`), measurement stability function (`msEntry`), and TIPM model (`cdfpmEntry`). Ech instance of these objects contains a pair of values defining a data point of a particular model or function, hence the `1-to-N` relationships. The remaining objects, i.e., `metadata`, `quartitles`, and `urls`, contain the metadata descrbing the sensor (e.g., its type and location), the pair of first and third quartiles of its measurements, and the unique URL for communicating with it. The relationship between a sensor object and these objects is `1-to-1`.

5.2.3 SIMON

SIMON[4] is responsible for the communication between the GSS and SC components as well as between the GSS and LSS (i.e., its Query Resolver). SIMON is a tool that provides remote method invocations. While the function of SIMON is similar to that of Java RMI[5], SIMON requires at most one socket connection for both client-server and server-client communications, which serves as a secured communication tunnel. Java RMI, in contrast, requires at least one such socket connection. Furthermore, SIMON supports callback feature over the Internet through its secured communication tunnel, which makes it practically easier for network administrators to setup network firewalls/routers. With SIMON, we can access Java objects that are located in another JVM on the same computer or on another server computer somewhere on the Internet.

5.3 Graphical User Interface

The SC component provides the users with a GUI for creating search queries for both sensor similarity search and content-based sensor search services, and for presenting the users the search results. The context menu "Search" allows the users to switch between the two search services. In the following we describe the GUI for each servive.

5.3.1 Sensor Similarity Search: GUI

Fig. 5.4 shows the GUI for the sensor similarity search service. To specify an example sensor, respectively a fraction of its past measurements, a user selects from the dropdown list of available sensors on Xively a sensor (i.e., its URL). The up-to-date measurements of the selected sensor will be shown in a plot below the dropdown list. Now, the user specifies

[4] http://dev.root1.de/projects/2/wiki
[5] http://docs.oracle.com/javase/7/docs/api/java/rmi/package-summary.html

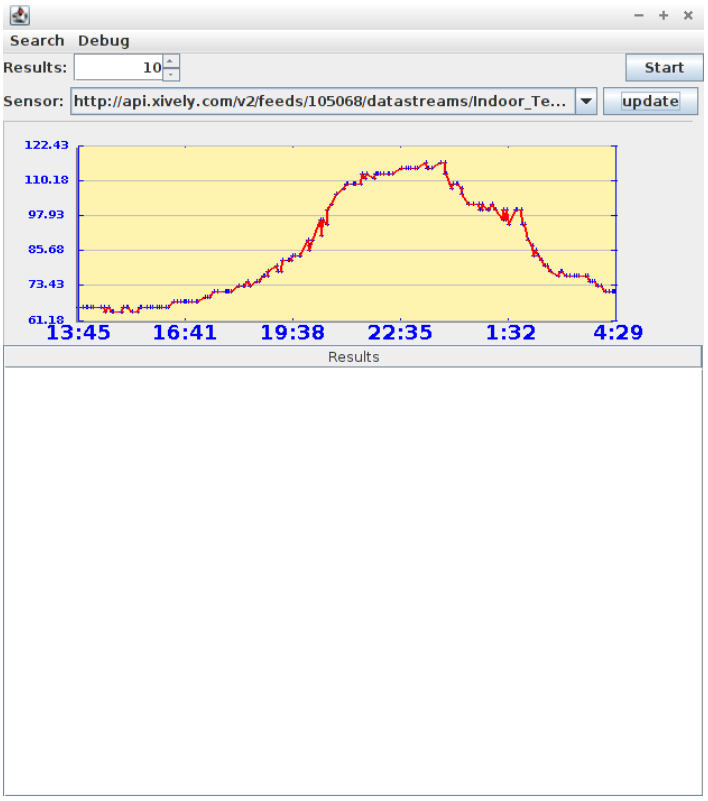


FIGURE 5.4: GUI: Sensor similarity search.

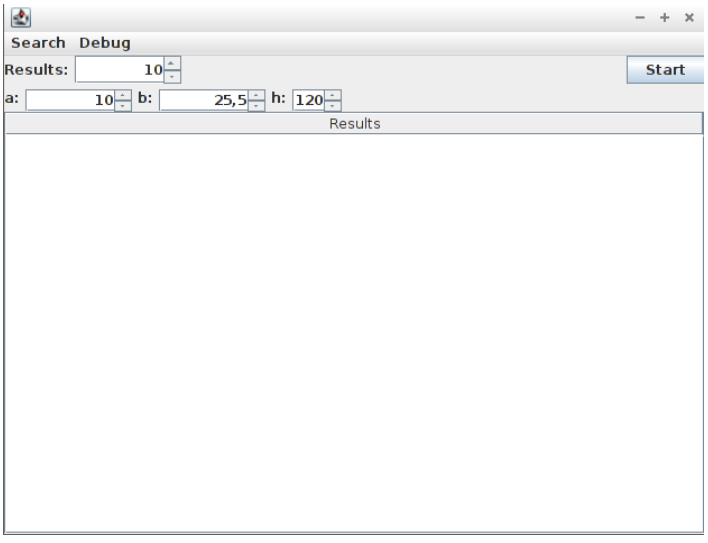


FIGURE 5.5: GUI: Content-based sensor search.

his desired number of result sensors that are similar to the selected one and presses the "Start" button to begin the search. After some time, a list of similar sensors will be shown in the "Results" window. For each result sensor, its measurements are plotted and its URL and metadata are shown.

5.3.2 Content-based Sensor Search: GUI

Fig. 5.5 shows the GUI for the content-based sensor search service. A user specifies the values for "a", "b", and "h" to create a content-based search query as described in Sec. 4.3.3, and click the "Start" button to begin the search. After some time, a list of matching sensors with their measurements and URLs is shown in the "Results" window.

5.4 Demonstration

In order to demonstrate our search engine, we will manually perform several series of search operations on each of the sensor search services and look at the results to see their actual performance, i.e., the degree of accuracy and the communication overhead of a final ranked list of sensors returned by the sensor similarity search and the content-based sensor search services, respectively.

5.4.1 Sensor Similarity Search

Since we do not have a ground truth to assess the performance of the sensor similarity search algorithm, we will randomly select 5 sensors from Xively, perform searches with them, present the search results (in the form of screenshots) here, and discuss the performance of the search engine based on these results. For each screenshot, both the data curve of the selected sensor and the search results are presented.

As can be observed in the 5 figures (Figs. 5.6, 5.7, 5.8, 5.9, and 5.10), our sensor similarity search engine can find "similar" sensors in Xively given an example sensor. This similarity is evident in two aspects: (1) the measurement ranges of the found sensors always largely overlap the measurement range of the example sensor; and (2) the data curve of the found sensors "appear" to resemble the data curve of the example sensor.

5.4.2 Content-based Sensor Search (CSS)

For this sensor search service, since it computes the search result based on predicting if sensors currently match a content-based search query $q = \{[a, b], h\}$ (i.e., if their generated measurements during $[now - h, now]$ fall within $[a, b]$), it can easily be concluded that the content-based sensor search algorithm will likely perform better with queries q whose $b - a$ is large when compared with those q whose $b - a$ is small. For example, in an extreme case the prediction that a sensor is generating measurements in $[-\infty, +\infty]$ would always be correct (assuming that the sensor has been actually generating measurements).

As Xively is a cloud-based platform for people to connect their sensors to, the sensors available on Xively are usually of common types such as temperature, humidity, light, voltage, gas,

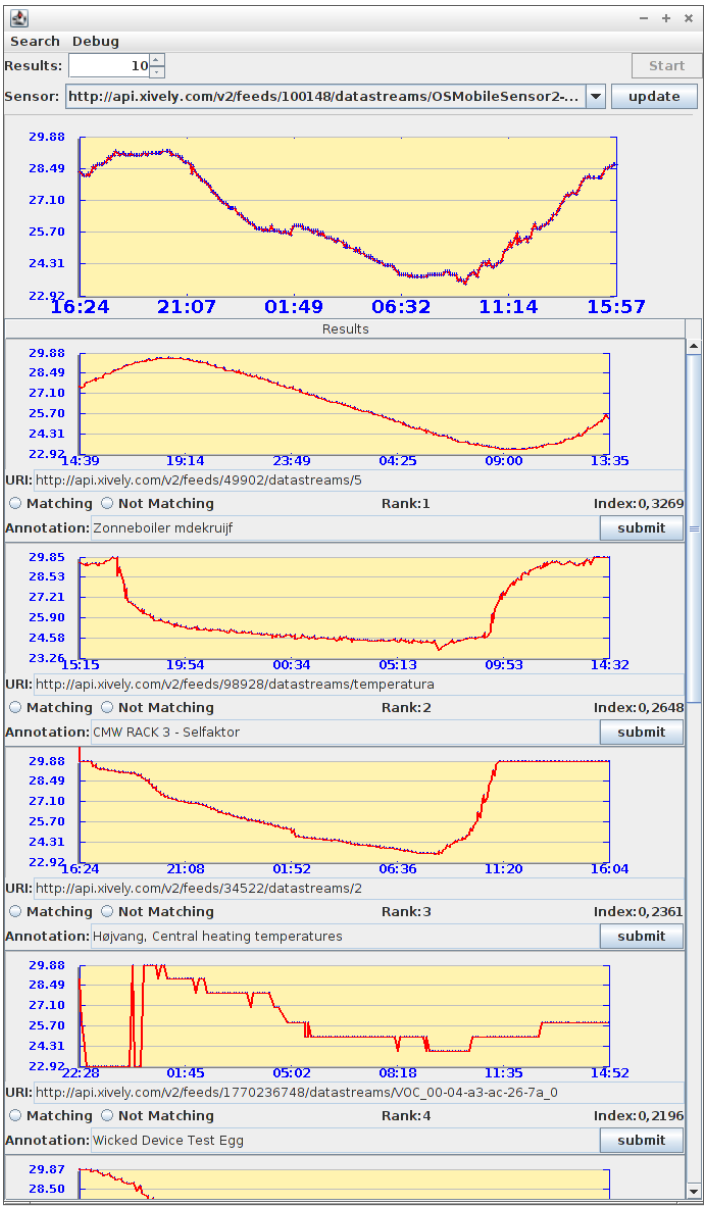


FIGURE 5.6: Demonstration of the sensor similarity search engine.

etc, that measure the common physical processes in our environment. Thus, as a rough estimation, the distribution of their generated measurements is likely to be concentrated between zero and a few hundreds or a few thoundsands (note that we ignore the measurement units as we care only about absolute measurement values).

Thus, we perform in this demonstration 4 series of search operations, each of which contains 20 search operations. For each series, all 20 searches are performed with a fixed length of $l = b - a$ but changing values of a and b (e.g., $q_1 = \{[10, 20], 60\}$ and $q_2 = \{[25, 35], 60\}$ where $l = 20 - 10 = 35 - 25 = 10$). Due to the above conclusion and estimation, we choose the specific values of l in the 4 series of search operations to be $l_1 = 50$, $l_2 = 10$, $l_3 = 5$, and $l_4 = 1$ for the first, the second, the third, and the fourth series, respectively.

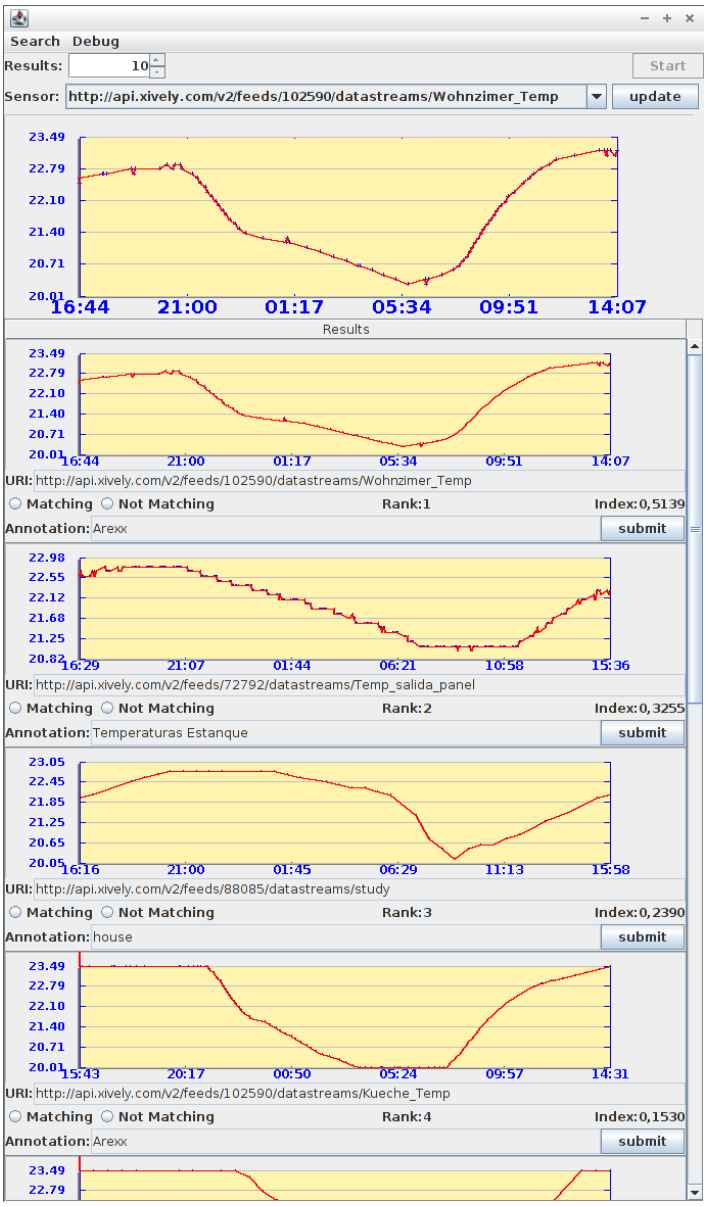


FIGURE 5.7: Demonstration of the sensor similarity search engine.

For each series of search operations $i = 1..4$, for each individual search number $j = 1..20$ we assign $a_j = b_{j-1}$ (i.e., the value of b of the individual search number $j - 1$) and $b_j = a_j + l_i$. We assign $a_1 = 0$. After that, we present a plot of the communication overhead of the search result of 20 individual searches j for the search series i. The communication overhead is computed according to Eq. 4.31.

The communication overhead of the 4 series of search operations are shown in Fig. 5.11, Fig. 5.12, Fig. 5.13, and Fig. 5.14 for $l = 50$, $l = 10$, $l = 5$, and $l = 1$, respectively. As we observe in the figures, the content-based sensor search algorithm performs well for all 4 series of searches as the communication overhead is zero in most individual search operations, and the average communication overhead of all series of search operations is well below 0.1 (see Fig. 5.15), which is very low.

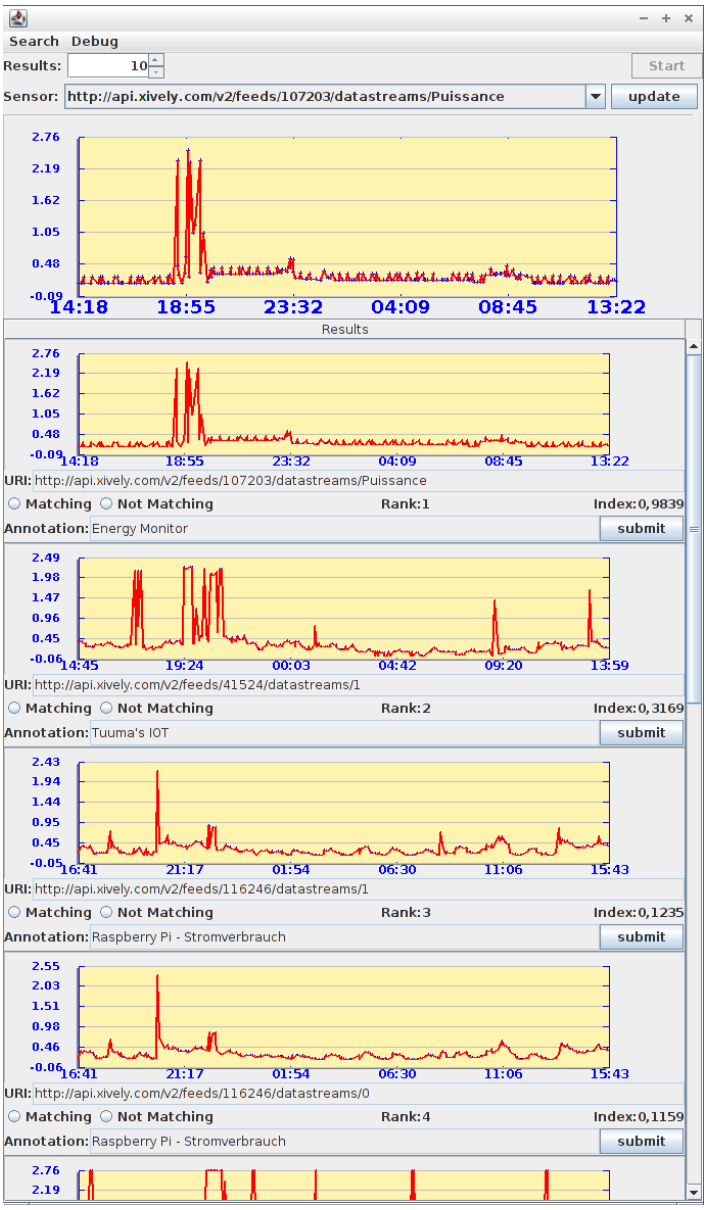


FIGURE 5.8: Demonstration of the sensor similarity search engine.

5.5 Performance

In order to give the reader an idea of how "heavy" our sensor search engine is, we briefly discuss in this section the performance of our sensor search engine, i.e., of its current implementation which runs on a computer with an Intel Core i5 CPU clocked at 2.4 Ghz. We are interested in the cost for resolving search queries in terms of time (i.e., the time period starting at the submission time of a search query and ending at the time the query's search result is returned), and of memory consumption (i.e., the amount of memory that each component of the search engine consumes). To obtain specific numbers, we performed serveral search operations, took note of these query resolution costs, and calculated the average of them.

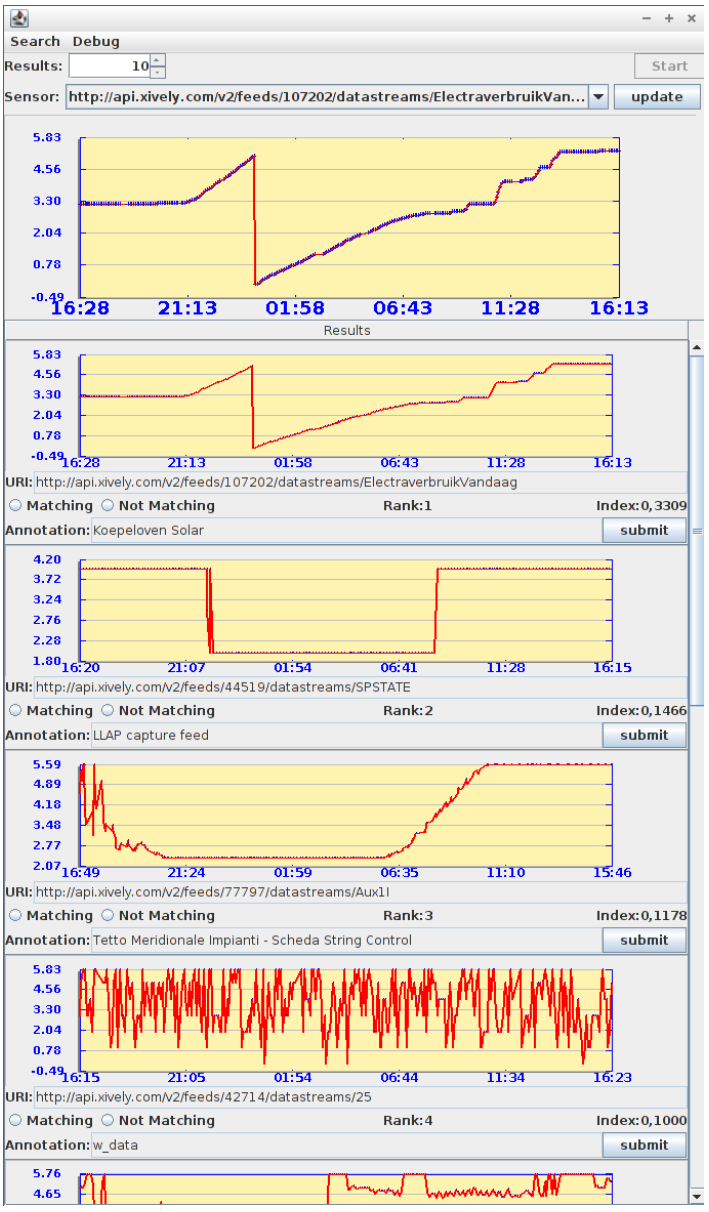


FIGURE 5.9: Demonstration of the sensor similarity search engine.

For query resolution time, the average query resolution time of the sensor similarity search service is about 45 seconds (i.e., performing Eq. 4.13 sequentially for about 20000 sensors in our current database), while that of the content-based sensor search service is only 2 seconds (i.e., performing Eq. 4.28 also sequentially for about 20000 sensors). The difference is due to Eq. 4.13 being more complex than Eq. 4.28. While Eq. 4.13 performs N calculations where N is the number of measurements contained in the sensor similarity search query, Eq. 4.28 always performs only one calculation.

To see the amount of memory that is consumed by our search engine while it runs, we use VisualVM[6], an "all-in-one Java trouble shooting" tool that allows the user to visually observe many factors (e.g., CPU, memory, running threads) during the run-time of a Java

[6]http://visualvm.java.net/

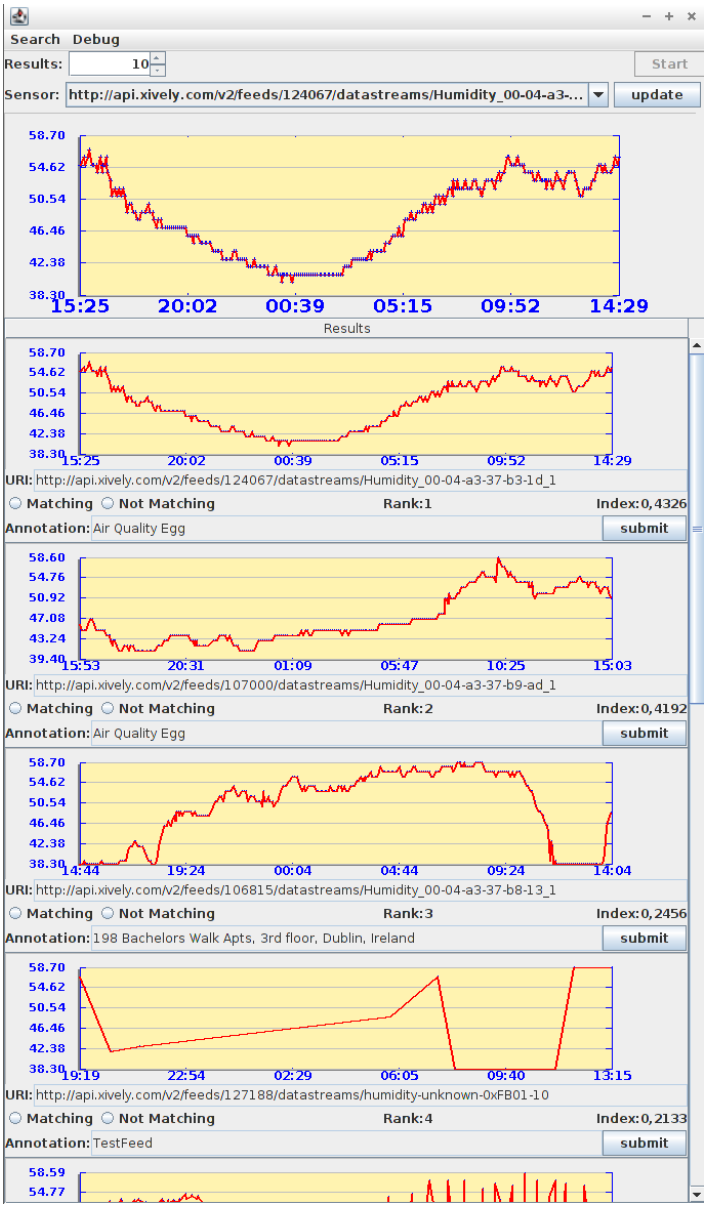


FIGURE 5.10: Demonstration of the sensor similarity search engine.

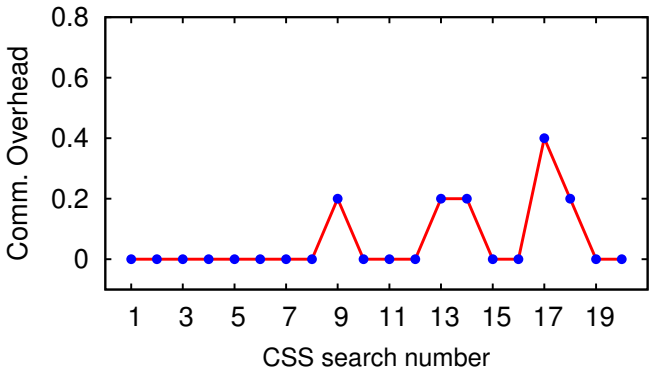


FIGURE 5.11: Evaluation of the content-based sensor search algorithm for $b - a = 50$

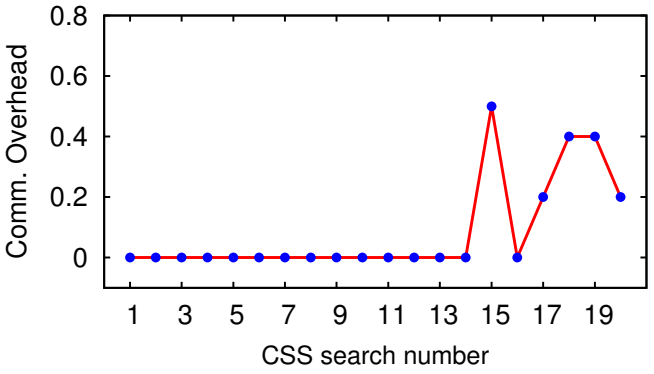


FIGURE 5.12: Evaluation of the content-based sensor search algorithm for $b - a = 10$

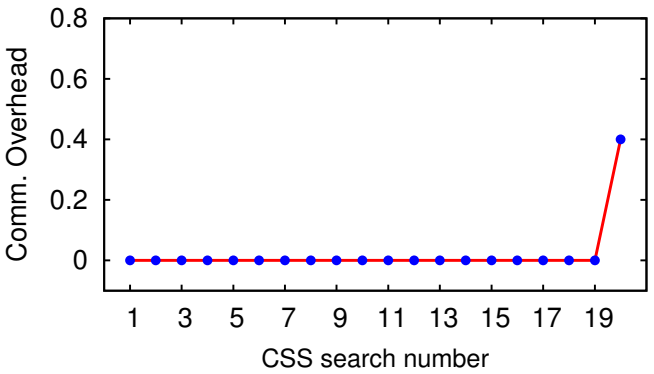


FIGURE 5.13: Evaluation of the content-based sensor search algorithm for $b - a = 5$

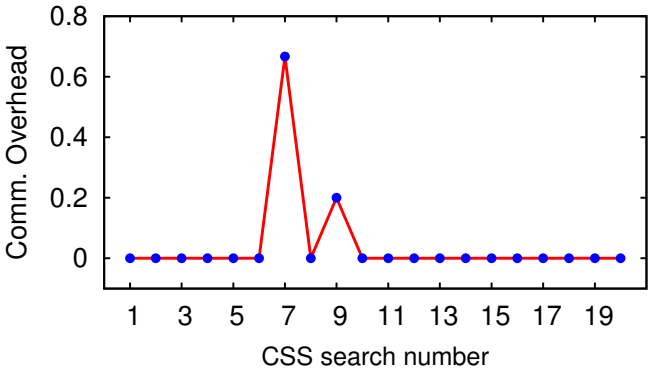


FIGURE 5.14: Evaluation of the content-based sensor search algorithm for $b - a = 1$

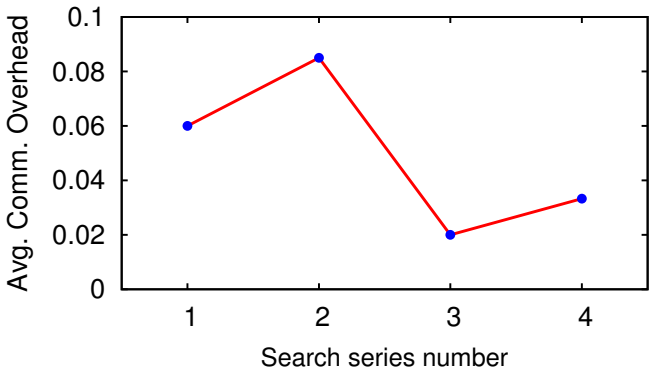


FIGURE 5.15: Average communication overhead for each search series

program. After we start all components of the search engine, i.e., GSS, LSS, Crawler, and Search Client, and let them run in the "idle" mode (i.e., wait for a search query), we observe that the amount of consumed memory of the Crawler is within the range of 30 Mb and 100 MB (see Fig. 5.16), and of each of the other 3 components is within 5 Mb and 30 Mb (see Fig. 5.17, and Fig. 5.18). The difference is because the Crawler periodically performs sensor discovery and indexing (if neccessary). As soon as a search query is submitted, the Search Client and GSS components do not consume more than 30 Mb of memory, as the Search Client simply forwards the search query to the GSS and the GSS simply forwards it to the LSS component. In contrast, the LSS component consumes much more memory, since it has to process the query. Below we discuss the memory consumption for resolving sensor similarity search query and content-based search query.

The amount of memory that the LSS component consumes for processing content-based search queries is well below 500 Mb, which includes the memory needed for caching the list of sensor URLs and prediction models that the LSS component retrieved from the FSDB for subsequent queries (around 300 Mb), and for processing the queries (on average 100 Mb). Fig. 5.19 shows this memory consumption, where we perform 5 search operations (search $i = 1..5$).

The consumed amount of memory for processing sensor similarity search queries is well below 750 Mb as we can observe in Fig. 5.20. This amount also includes memory required for query processing as well as caching. Fig. 5.20 also shows the memory consumption for 5 search operations (search $i = 1..5$) that we perform.

5.6 Conclusion

We presented in this chapter a prototypical sensor search engine that enables the user to find sensors that are currently available in the Web using our proposed sensor search algorithms, namely sensor similarity search and content-based sensor search. By performing serveral number of search operations for sensors that are available in Xively, an IoT-focused cloud

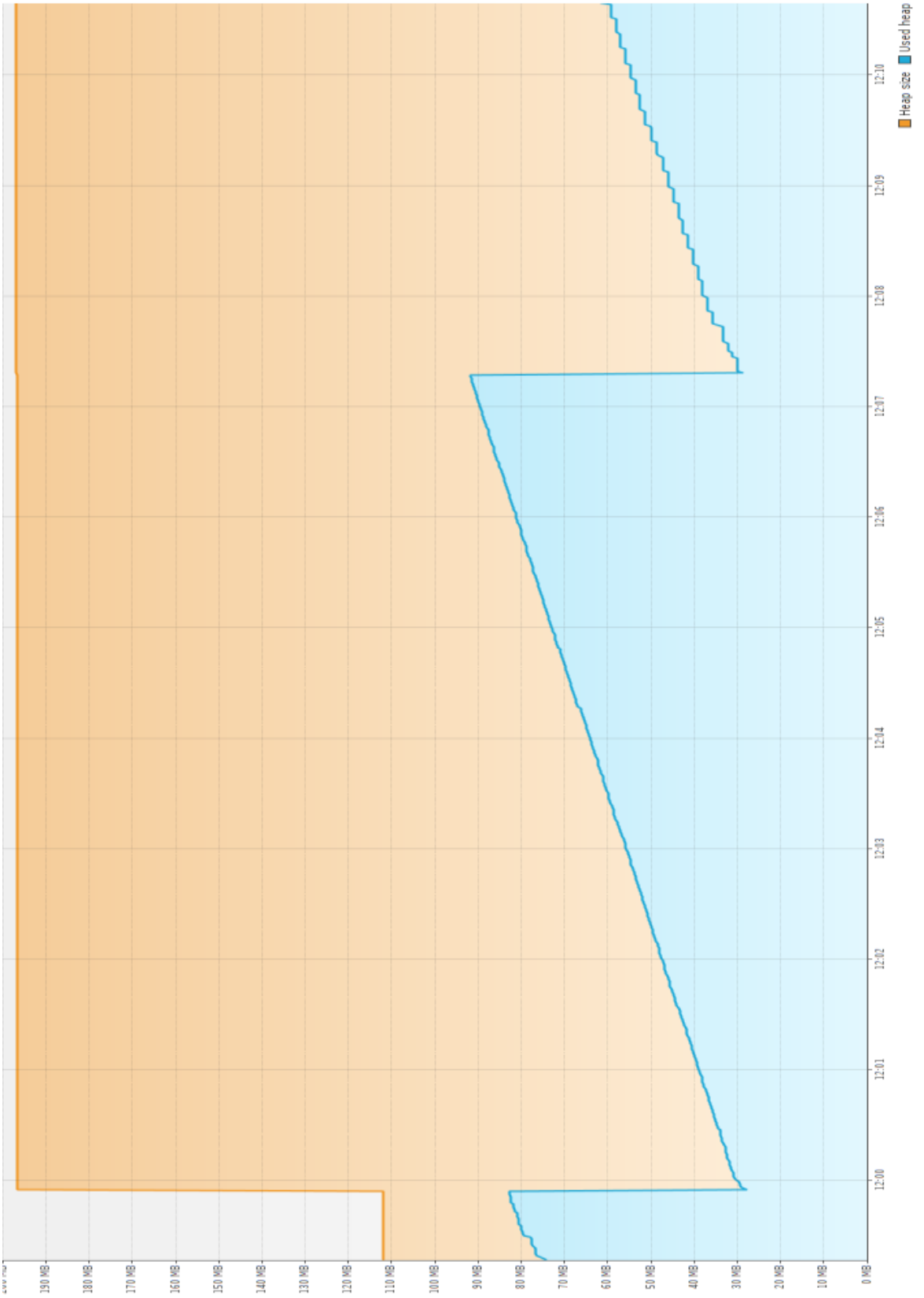


FIGURE 5.16: Memory consumption of the Crawler component

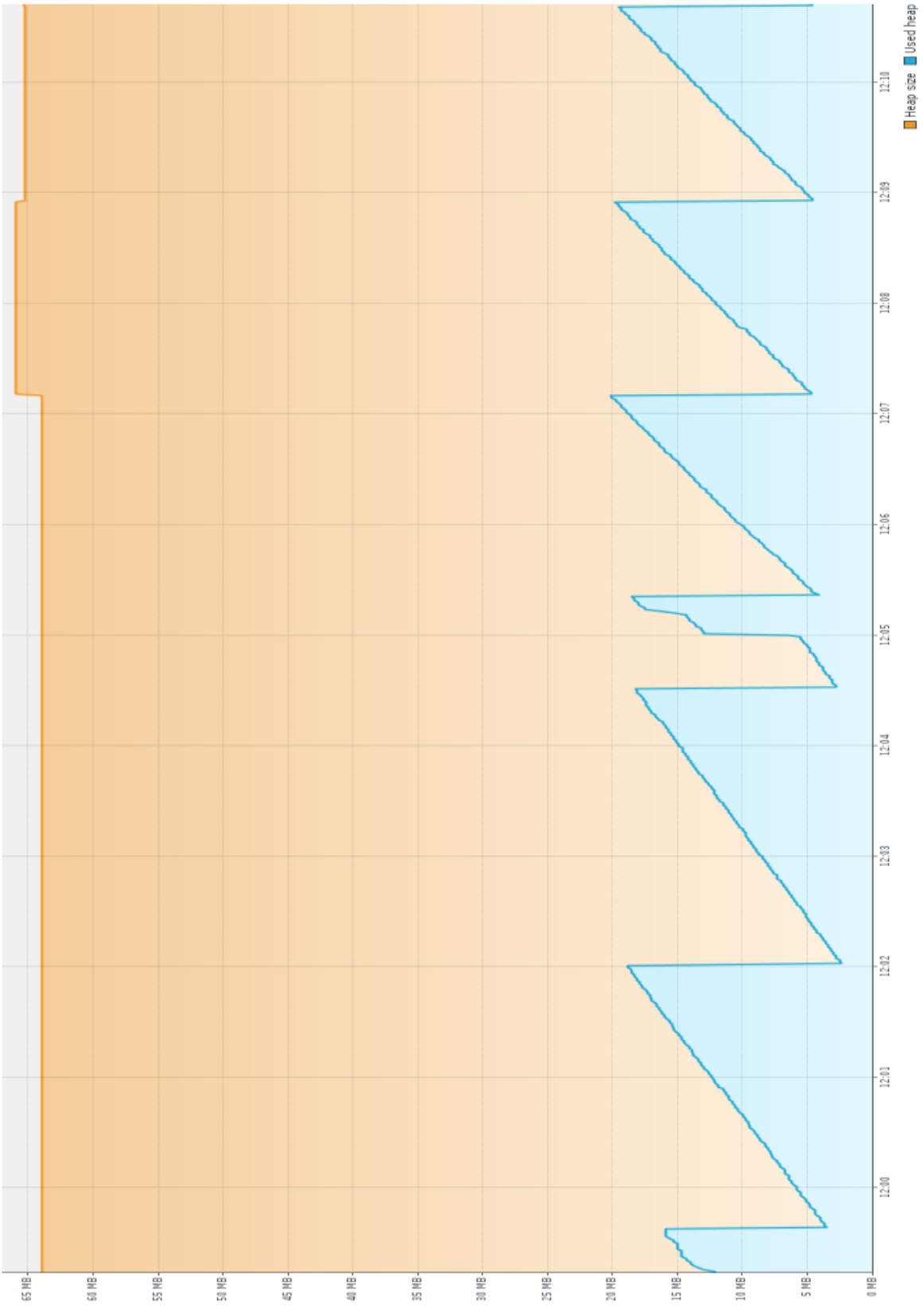


FIGURE 5.17: Memory consumption of the GSS component

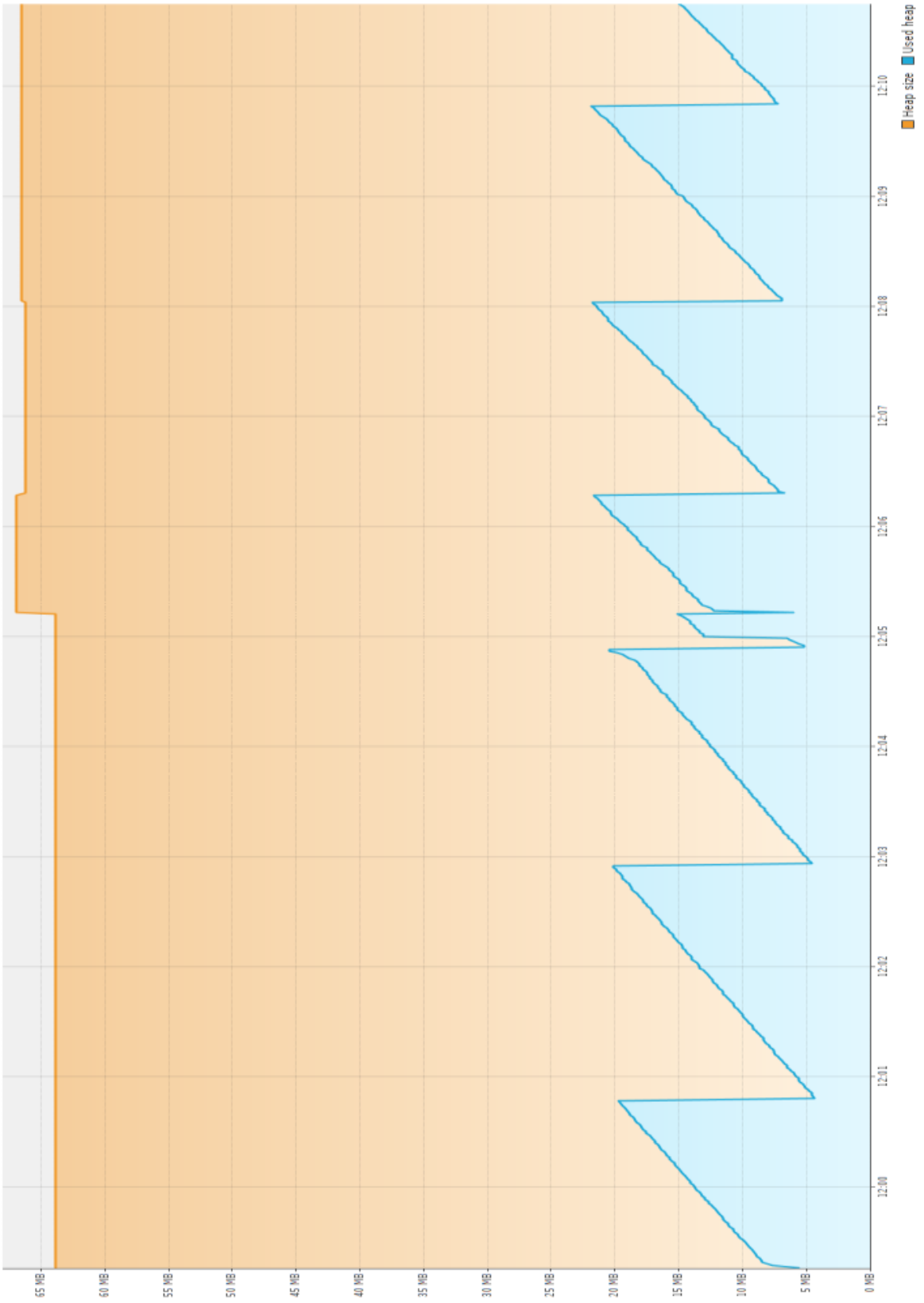


FIGURE 5.18: Memory consumption of the Search Client component

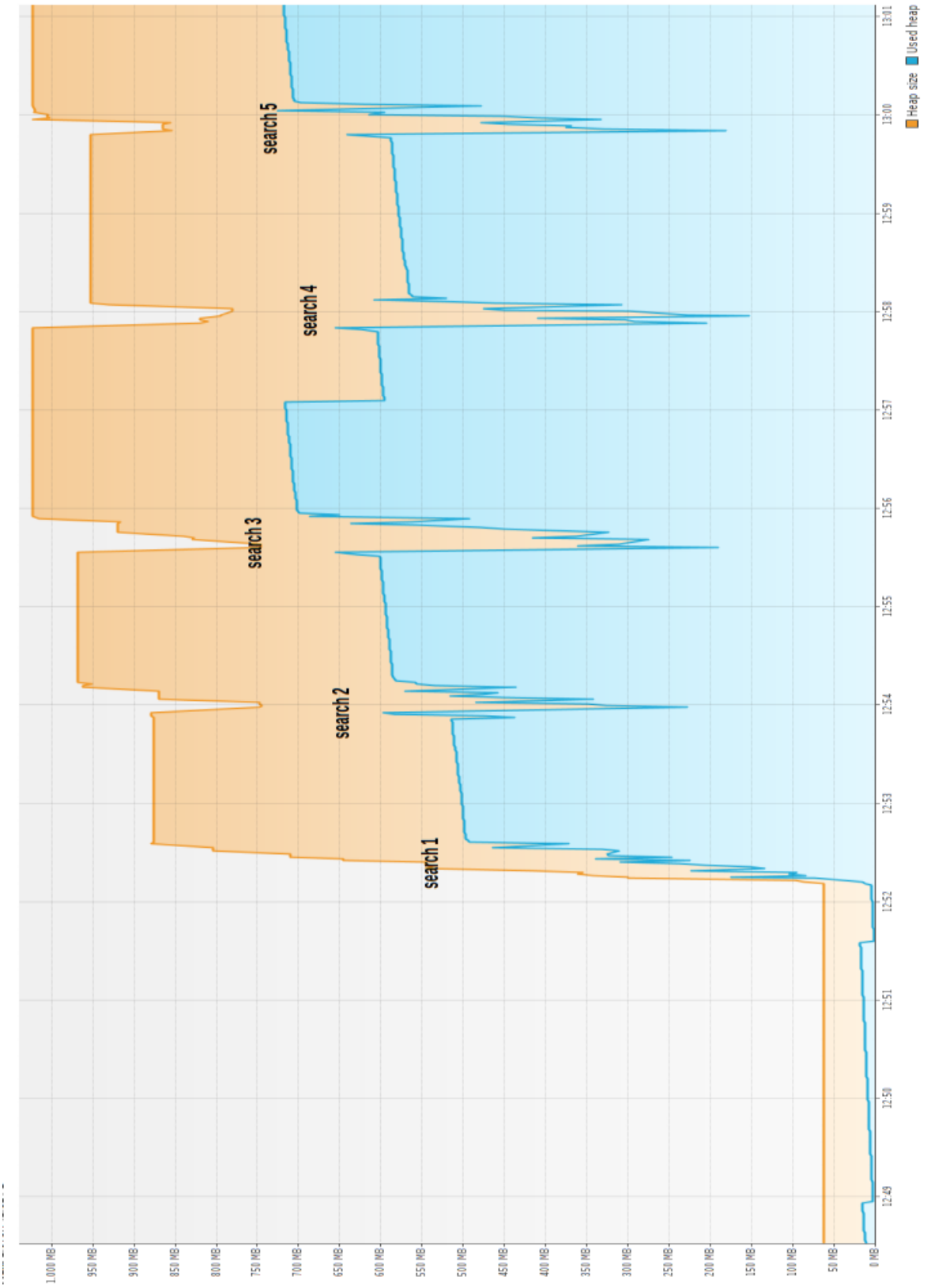


FIGURE 5.19: Memory consumption of the LSS component while processing content-based sensor search queries

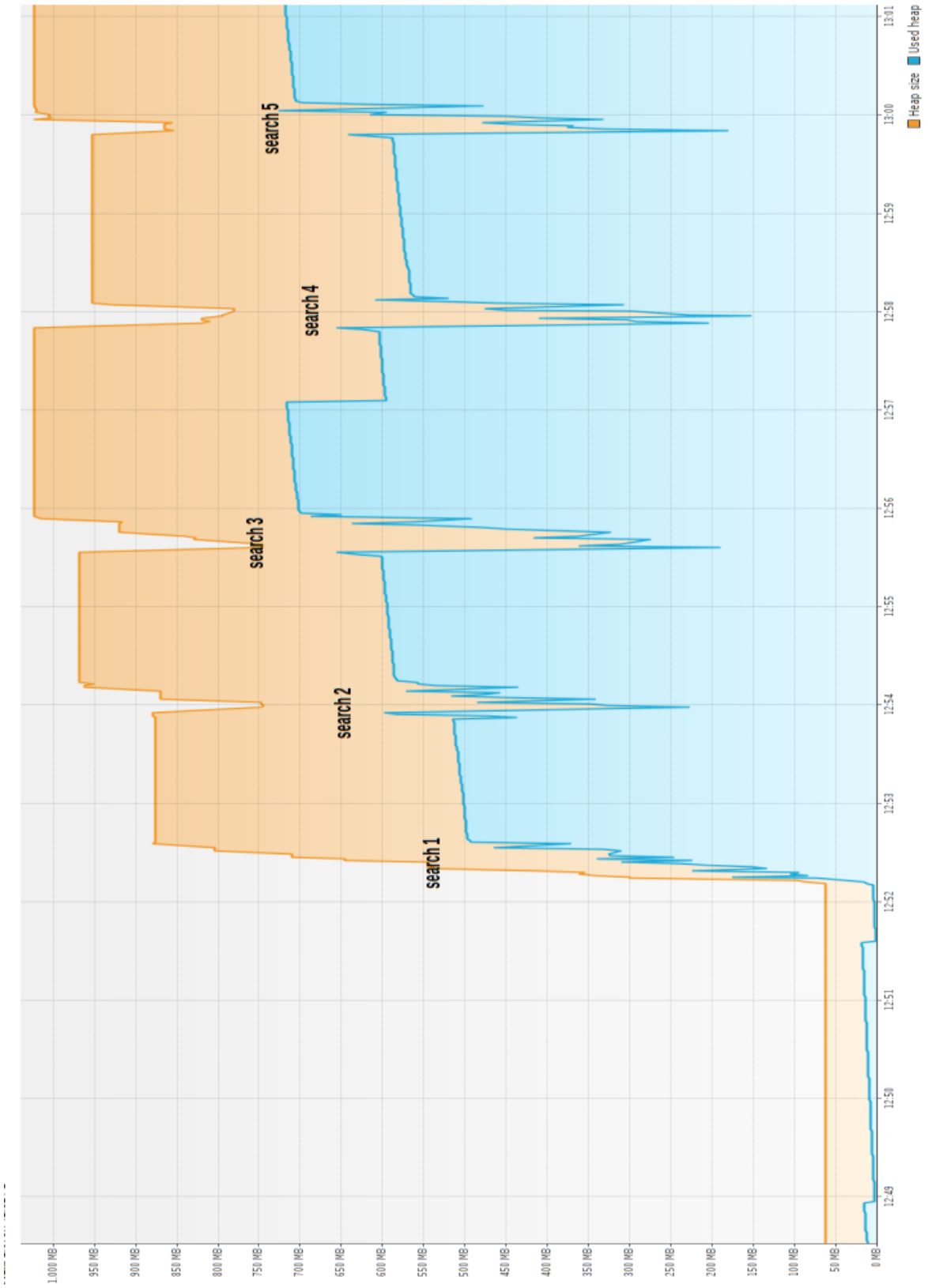


FIGURE 5.20: Memory consumption of the LSS component while processing sensor similarity search queries

service that allows users to connect their sensors to the IoT, we could demonstrate the feasibility and usability of our sensor search services.

Chapter 6

Conclusion and Future Work

The Internet of Things (IoT) allows real-world objects (e.g., people, plants, cars) and places (e.g., homes, offices, farm fields, forests) to be connected to the Internet, to publish their (real-time) states (perceived by embedded sensors/sensor networks), and to expose their functionalities on the Web, thus enabling them to be observed and controlled using Web technologies. This promises to reshape our society as entities of the physical world (i.e., objects and places) can be mashed up with data and services on the Web to create novel and valuabe IoT applications.

In this thesis we have studied the IoT in general and focused our work on two essential services for the IoT, namely sensor search and routing. In this final chapter, we summarize our contributions, discuss some limitations of our approaches, and sketch potential future work.

6.1 Contributions

Two of the main challenges to the provision of routing and sensor search services in the IoT are the large scale of the IoT and the resource limitations of Things (i.e., of their embedded IoT devices) connected to it. Our contributions in this thesis work showed that efficient solutions for the routing and sensor search services for the IoT can actually be provided, inspite of these challenges. We will summarize our contributions in the following subsections.

6.1.1 Routing

With respect to the routing service, we proposed *Recursive Multi-region Geocasting* (RMG) and *Stochastic Forwarding-based Routing* (SFR) for routing of information in WSN (which is a building block of the IoT).

The RMG algorithm is targeted to large-scale WSN where information needs to be delivered from a source to multiple geographic regions that are remotely located from the source, respectively to all Things, i.e., sensor nodes that are located therein. We showed that existing algorithms are not appropriately designed for this class of routing scenarios. In particular, they do not appropriately support large-scale WSN, both in terms of number of the network nodes and the geographic area that the WSN covers, since routing decisions are made at *every* intermediate node. In contrast, RMG makes routing decisions and duplicates a packet only at a few selected nodes on the routing path of the packet and is therefore scalable as processing and energy resources as well as wireless bandwidth are saved. This is confirmed by our evaluation results, which showed that RMG minimizes the total number of transmissions needed for the successful delivery of a data packet, while at the same time incuring little computation overhead on the network.

The SFR algorithm aims at improving the operational life time of WSN by balancing the energy consumption caused by the routing task across the network. Given a pair of source and destination, existing algorithms usually send data packets over a single deterministic routing path, which eventually uses up the energy budget of the nodes on the path. This may cause partitions in the network since some of those nodes may be the only links between different parts of the network. In contrast to this approach, we model the route of a packet as a random walk, such that different packets travel on different routing paths between the source and the destination. This way routing load is spread among multiple paths rather than concentrated on a single one. This, however, may increase the total time it takes to deliver all packets as they may travel on routing paths that are much longer than the shortest routing path (e.g., in terms of number of hops). To achieve a trade-off, we designed the random walk such that the ratio between the average length of the routing paths taken by all packets and the length of the shortest routing path (path length overhead) is small. Our evaluation results showed that SFR fairly balances the routing load across the network for a pair of source and destination while keeping the path length overhead small. Furthermore, SFR is scalable since routing decisions are made using only local information.

6.1.2 Sensor Search

With respect to the sensor search service, we proposed *sensor similarity search* and *content-based sensor search* for searching sensors on the Web with certain search criteria. These two sensor search services are novel and useful because they enable users to search the physical world for objects and places with a given state (i.e., the output measurements of their embedded sensors/sensor networks) via the IoT. We showed that state-of-the-art sensor search systems for the IoT do not support search based on the *similarity* between time series of output measurements of sensors and search based on the *recent* output measurements of sensors. There are, however, some systems that do support search based on the *current* output of sensors, with the assumption that an "output" is a high-level state abstracted from the raw measurements of sensors (e.g., "hot" and "cold" are abstracted from the temperature

being 35°C and 10°C, respectively). Complementary to these systems, our sensor search services support search based directly on raw measurements of sensors. In particular, the sensor similarity search service allows for finding sensors (i.e., Things) whose recent measurements (i.e., perceived states of Things) are similar to that of an example sensor (or Thing). The content-based sensor search service allows for finding sensors (i.e., Things) whose latest measurements (i.e., current state) fall in a given value range (which could be defined as a high-level state, e.g., temperatures being in the range of [15°C, 25°C] could be defined as "cool").

We proposed an architecture suitable for both two sensor search services, which is scalable because the search is distributed and executed in parallel among multiple sensor search servers across the Internet. Our fuzzy-set-based approach requires reasonable computation on IoT devices and incurs low storage and wireless communication overhead as the size of a fuzzy set is small (in the order of few tens of bytes), thus is suitable for resource-constrained IoT devices. The efficiency of our approach is confirmed by our evaluation results which showed that the proposed sensor similarity search algorithm is highly accurate, and the proposed content-based sensor search algorithm incurs low wireless communication overhead.

We demonstrated the practical feasibility of our solutions by implementing and integrating our proposed sensor search algorithms into a sensor search engine, and using it to search for sensors that are available on the Web based on our proposed sensor search services.

6.2 Limitations and Future Work

As our contributions to the routing and sensor search services in the IoT are focused on a subset of challenges (i.e., scalability and resource limitations) and targeted to certain classes of IoT applications (e.g., routing in geographically large-scale WSN, sensor search based on raw sensor measurements), it is natural that they are less suited for other challenges and application scenarios. In the following we discuss some of the limitations of each of our proposed algorithm and sketch some future work based on that.

6.2.1 Recursive Multi-region Geocasting Algorithm

Although RMG is efficient, it is not designed to handle network voids. In the current work, we combine RMG with a recovery mechanism (see Sec. 3.2.2.2) so that when a dead-end node receives the data packet, it initiates the recovery mechanism which will be used to forward the packet until it arrives at a non-dead-end node).

Another limitation of RMG is that we use the flooding approach to disseminate a data packet to all sensor nodes in a geographic region once it reaches a node located within the region. The flooding approach may fail in the presence of network partitions (e.g., a river dividing the region into two separate parts). To address this in a future work, the data packet may have

to enter the region at different crossing points on its border, such that it will be delivered to all nodes in all possible partitions.

6.2.2 Stochastic Forwarding-based Routing Algorithm

Again, SFR is not designed for handling network voids. There is, however, a side effect of our approach which guarantees that data packets will eventually be delivered, but this may take long time (see Sec. 3.4.7). A possible approach for this could be to adjust the forwarding probability of nodes that are on and close to the border of a network void such that data packets are likely to travel around but not towards the network void. For example, when a node discovers that it is a dead-end it notifies its neighbor nodes so that they lower the forwarding probability for it. These neighbor nodes, in turn, notify their neighbors in the same manner. The notification process can be limited by the number of hops so that only nodes in the proximity of the network void are notified. Depending on the dynamicity of the network topology, this process may need to be triggered time after time. An alternative approach is to combine SFR with a recovery mechanism as we did in the case of RMG.

6.2.3 Sensor Similarity Search Algorithm

To further improve the accuracy of this algorithm, the similarity models of sensors may need to be updated more frequently in order to incorporate latest sensor measurements into them. The limitation with this is that it imposes more computation on sensor nodes and the update of similarity models of billions of sensors that are (anticipated to be) available in the IoT takes time. To address the former, one would need to further simplify the computation of a similarity model, or put more computing power on sensor nodes (which is reasonable given the fast advancement in all technological fields), or a combination of both. A possible solution for the latter is to add more sensor servers into the sensor search architecture presented in Sec. 4.1.4.2.

Another possible limitation is the time it takes for computing a similarity score according to Eq. 4.13. Although it already is efficient, performing Eq. 4.13 for billions of sensors still takes a long time. For example, the average query resolution time of the sensor similarity search algorithm in our prototypical sensor search engine is about 45 seconds, i.e., performing Eq. 4.13 sequentially for about 20000 sensors in our current database in a computer with an Intel Core i5 CPU clocking at 2.4 Ghz. To address this, one could always parallelize the search using multiple computers. At the same time, for each computer, one could also apply a better search technique (i.e., instead of squentially evaluating all sensors) to reduce the searching time.

6.2.4 Content-based Sensor Search Algorithm

The query resolution time for a search in this algorithm is much shorter than in the sensor similarity search algorithm due to the simplicity of Eq. 4.28 (e.g., on average only 2 seconds as opposed to 45 seconds). However, this algorithm still faces the scalability problem as we discussed above, when the number of sensors is too large and the search is performed linearly. The same approaches outlined above also apply here.

6.3 Final Conclusion

In this thesis we studied routing and sensor search in the IoT, which are two essential services for the IoT. In order to facilitate the sensor search service, and also other IoT applications, the routing service is required to enable efficient communication among IoT devices and between IoT devices and Internet nodes.

We proposed, implemented, and evaluated novel solutions for these services, which address the unprecedentedly large scale of the IoT and the resource limitations of IoT devices. Our proposed routing algorithms are suited to specific IoT application scenarios, where information needs to be transmitted over long distances in geographically large-scale networks of IoT devices and the operatational time of these networks must be long. Our original sensor search algorithms enable searching the physical world for objects and places via the IoT. These solutions support our thesis that *despite the large scale of the IoT and the resource limitations of IoT devices, efficient solutions for the routing and sensor search services for the IoT can actually be provided.*

Bibliography

[1] Rob van Kranenburg, Alessandro Bassi, Dan Caprio, Sean Dodson, and Matt Ratto. The Internet of Things. *First Berlin Symposium on Internet and Society*, October 2011.

[2] Fraunhofer Research. The eGRAIN Project. URL http://cyberphysicalsystem.de/egrain-projekt/.

[3] B. A. Warneke, M. D. Scott, B. S. Leibowitz, L. Zhou, C. L. Bellew, J. A. Chediak, J. M. Kahn, B. E. Boser, and K. S. J. Pister. An Autonomous 16 Cubic mm Solar-Powered Node for Distributed Wireless Sensor Networks. In *IEEE International Conference on Sensors 2002*, pages 1510–1515, Orlando, USA, June 2002.

[4] Cuong Truong and Kay Römer. Efficient Geocasting to Multiple Regions in Large-Scale WSNs. *37th Int. Conf. on Local Computer Networks (LCN 2012)*, 2012.

[5] F. Sivrikaya, T. Geithner, Cuong Truong, M. A. Khan, and S. Albayrak. Stochastic Routing in Wireless Sensor Networks. *IEEE ICC 2009*.

[6] Cuong Truong, Kay Römer, and Kai Chen. Fuzzy-based Sensor Search in the Web of Things. *3rd Int. Conf. on Internet of Things (IoT-2012)*, 2012.

[7] Cuong Truong and Kay Römer. Content-based Sensor Search for the Web of Things. *IEEE Global Communications Conference (Globecom'13)*, 2013.

[8] INFSO D.4 Networked Enterprise & RFID INFSO G.2 Micro & Nanosystems. Internet of Things in 2020, Roadmap for the Future, Version 1.1. *In: Co-operation with the Working Group RFID of the ETP EPOSS*, 27 May 2008.

[9] Charu C. Aggarwal, Naveen Ashish, and Amit Sheth. The Internet of Things: A Survey From The Data-Centric Perspective. *Book: Managing and Mining Sensor Data*, pages 383–428, 2013.

[10] ITU Internet Reports. The Internet of Things. November 2005.

[11] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A Survey. *Computer Networks*, 2010.

[12] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Generation Computer Systems*, 29:1645–1660, September 2013.

[13] Adam Dunkels and JP Vasseur. Internet Protocol for Smart Objects. *White Paper*, 2008.

[14] Maarten Botterman. Internet of Things: An Early Reality of the Future Internet. *For the European Commission, Information Society and Media Directorate General, Networked Enterprise & RFID Unit (D4).*

[15] Adam Dunkels and JP Vasseur. IP for Smart Objects. *White Paper No. 1, IPSO Alliance*, July 2010.

[16] Neil Gershenfeld, Raffi Krikorian, and Danny Cohen. The Internet of Things. *Scientific American*, 2004.

[17] V. Bychkovskiy, S. Megerian, and D. Estrin. Collaborative Approach to In-place Sensor Calibration. In *2nd Intl. Conf. on Information Processing in Sensor Networks (IPSN'03)*, 2003.

[18] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven Data Acquisition in Sensor Networks. In *Proc. of the 13th Intl. Conf. on Very Large Databases (VLDB'04)*, 2004.

[19] C. C. Aggarwal and J. Han. Managing and Mining Sensor Data. *Springer*, 2013.

[20] J. Dean and S. Ghemawat. MapReduce: A Flexible Data Processing Took. *Communication of the ACM*, 53:72–77, 2010.

[21] T. White. Hadoop: The Definitive Guide. *Yahoo! Press*, 2011.

[22] H. Wang, C. C. Tan, and Q. Li. Snoogle: A Search Engine for Pervasive Environments. *IEEE Trans. on Parallel and Distributed Systems*, (8):1188–1202, 2010.

[23] C. C. Tan, B. Sheng, H. Wang, and Q. Li. Microsearch: When Search Engines Meet Small Devices. *Pervasive*, 5013:93–110.

[24] K. K. Yap, V. Srinivasan, and M. Motani. MAX: Human-centric Search of the Physical World. In *Proc. of the 3rd Intl. Conf. on Embedded Networked Sensor Systems (Sensys'05)*, New York, NY, USA, 2005.

[25] Ali Salehi, M. Riahi, S. Michel, and Karl Aberer. GSN, Middleware for Streaming World. *In Proc. 10th Int. Conf. on Mobile Data Management*, 2009.

[26] A. Kansal, S. Nath, J. Liu, and F. Zhao. SenseWeb: An Infrastructure for Shared Sensing. *IEEE Multimedia*, 14:8–13, 2007.

[27] Lars Erik Holmquist, Friedemann Mattern, Bernt Schiele, Petteri Alahuhta, Michael Beigl, and Hans-W. Gellersen. Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artefacts. In *Proc. of the 3rd Intl. Conf. on Ubiquitous Computing (UbiComp'01)*, 2001.

[28] Jonathan Lester, Blake Hannaford, and Geatano Borriello. "Are you with me?" - Using Accelerometers to Determine if Two Devices are Carried by the Same Person. In *Proc. 2nd Int. Conf. Pervasive Computing (PerCom'04)*, 2004.

[29] Matthias Gauger, Olga Saukh, Marcus Handte, and Pedro Jose Marron. Sensor-based Culstering for Indoor Applications. *SECON'08*, 2008.

[30] B. Maryam Elahi, Kay Römer, Benedikt Ostermaier, Michael Fahrmair, and Wolfgang Kellerer. Sensor Ranking: A Primitive for Efficient Content-based Sensor Search. In *Intl. Conf. on Information Processing in Sensor Networks (IPSN'09)*, San Francisco, CA, USA, 2009.

[31] Arne Broring, Johannes Echterhoff, Simon Jirka, Ingo Simonis, Thomas Everding, Christoph Stasch, Steve Liang, and Rob Lemmens. New Generation Sensor Web Enablement. *Sensors*, 11:2652–2699, 2011.

[32] Jin Cheng and Thomas Kunz. A Survey on Smart Home Networking. *Carleton University, Systems and Computer Engineering, Technical Report SCE-09-10*, September 2009.

[33] Abowd G and Mynatt E. Designing for the human experience in smart environments. *In: Cook D, Das S, editors. Smart Environments: Technology, Protocols, and Applications*, 2004.

[34] Rashvand Habib F. and Alcaraz Calero Jose M. Distributed Sensor Systems: Practice and Applications. *John Wiley & Sons., Ltd*, 2012.

[35] G. Michael Youngblood and Diane J. Cook. Data Mining for Hierarchical Model Creation. *IEEE Transactions on Systems, Man, and Cybernetics*, 37:561–572, July 2007.

[36] F. Doctor, H. Hagras, and V. Callaghan. A Fuzzy Embedded Agent-based Approach for Realizing Ambient Intelligence in Intelligent Inhabited Environments. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, 35:55–65, January 2005.

[37] Sumi Helal, William Mann, Hicham El-Zabadani, Jeffrey King, Youssef Kaddoura, and Erwin Jansen. The Gator Tech Smart House: A Programmable Pervasive Space. *Computer*, 38(3):50–60, 2005.

[38] Dipak Surie, Olivier Laguionie, and Thomas Pederson. Wireless Sensor Networking of Everyday Objects in a Smart Home Environment. *ISSNIP'08*, 2008.

[39] G. Broll, E. Rukzio, M. Paolucci, M. Wagner, A. Schmidt, and H. Hussmann. PERCI: Pervasive service integration with the Internet of Things. *IEEE Internet Computing*, 13:74–81, November 2009.

[40] D. Reilly, M. Welsman-Dinelle, C. Bate, and K. Inkpen. Just Point and Click? Using Handhelds to Interact with Paper Maps. In *Proc. of the 7th Intl. Conf. on Human Computer Interaction with Mobile Devices & Services (MobileHCI'05)*, September 2005.

[41] R. Hardy and E. Rukzio. Touch & Interact: Touch-based Interaction of Mobile Phones with Displays. In *Proc. of 10th Intl. Conf. on Human Computer Interaction with Mobile Devices and Services (MobileHCI'08)*, September 2008.

[42] SENSEI FP7 Project. Senario Portfolio: User and Context Requirements. URL http://www.sensei-project.eu.

[43] RSA Labotaries. RFID, a Vision of the Future. URL http://www.rsa.com/rsalabs/node.asp?id=2117.

[44] Rachael McBrearty. The Future of Retail Customer Loyalty RFID Enables Breakthrough Shopping Experiences. *Cisco's Whitepaper*, June 2011.

[45] H. Baldus, K. Klabunde, and G. Muesch. Reliable Set-Up of Medical Body Sensor Networks. In *Proc. of European Conference on Wireless Sensor Networks (EWSN 2004)*, Berlin, Germany.

[46] G. Schreier. Pervasive Healthcare via "The Internet of Medical Things". In *Proceedings of Medetel*, 2010.

[47] Xiao Ming Zhang and Cheng Xu. A Multimedia Telemedicine System in Internet of Things. *2nd International Conference on Information and Multimedia Technology*, 2010.

[48] Mikhail Simonov, Riccardo Zich, and Flavia Mazzitelli. Personalized Healthcare Communication in Internet of Things. *Proceedings of URSI, 2008.*

[49] H. Hawkeye King, Thomas Low, Kevin Hufford, and Timothy Broderick. Acceleration Compensation for Vehicle Based Telesurgery on Earth or in Space. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems.*

[50] Michael Stark, Tahar Benhidjeb, Stefano Gidaro, and Emilio Ruiz Morales. The Future of Telesurgery: A Universal System with Haptic Sensation. *Journal on Turkish-German Gynecological Association*, 2012.

[51] Sebastian Dengler, Abdalkarim Awad, and Falko Dressler. Sensor/Actuator Networks in Smart Homes for Supporting Elderly and Handicapped People. In *Proc. of 21st IEEE Intl. Conf. on Advanced Information Networking and Applications (AINA-07)*, 2007.

[52] Monica Tentori and Jesus Favela. Activity-Aware Computing for Healthcare. *IEEE Pervasive Computing*, 7(2):51–57, 2008.

[53] Universitat Politècnica de Catalunya. Advances In Medical Technology: What Does The Future Hold? *ScienceDaily, 16 Jun. 2009. Web. 30 Mar. 2013.*

[54] Future Trends in Medical Technology. URL www.medica.de.

[55] Ben-Jye Chang, Bo-Jhang Huang, and Ying-Hsin Liang. Wireless Sensor Network-Based Adaptive Vehicle Navigation in Multihop-Relay WiMAX Networks. *Advanced Information Networking and Applications*, 2008.

[56] Swarun Kumar, Shyamnath Gollakota, and Dina Katabi. A Cloud-Assisted Design for Autonomous Driving. *MCC'12, August 17, 2012, Helsinki, Finland.*

[57] Arun Hampapur, Lisa Brown, Jonathan Connell, Sharat Pankanti, Andrew Senior, and Yingli Tian. Smart Surveillance: Applications, Technologies and Implications. In *IEEE Pacific-Rim Conference On Multimedia*, 2003.

[58] Umakishore Ramachandran, Kirak Hong, Liviu Iftode, Ramesh Jain, Rajnish Kumar, Kurt Rothermel, Junsuk Shin, and Raghupathy Sivakumar. Large-scale Situation Awareness with Camera Networks and Multimodal Sensing. *Proceedings of the IEEE, Vol. 100, No. 4. (April 2012).*

[59] Justin Patton and Bill C. Hardgrave. RFID As Electronic Article Surveillance (EAS): Feasibility Assessment. *Information Technology Research Institute.*

[60] Indraveer Singh and Harshawardhan Patil. RFID: Dynamic Surveillance Approach. *IJCSI International Journal of Computer Science Issues*, 7(7), May 2010.

[61] Alessandro Oltramari and Christian Lebiere. Using Ontologies in a Cognitive-Grounded System: Automatic Action Recognition in Video Surveillance. *In Proc. of Semantic Technology for Intelligence, Defense, and Security, 2012.*

[62] Dae-Hyeong Kim et al. Epidermal Electronics. *Science*, 333:838–843, August 2011.

[63] University of Pennsylvania. Mind Reading from Brain Recordings? "Neural Fingerprints" of Memory Associations Decoded. *ScienceDaily, 26 Jun. 2012. Web. 28 Mar. 2013.*

[64] Austin Harney. Smart Metering Technology Promotes Energy Efficiency for a Greener World. *Analog Dialogue 43-01*, January 2009.

[65] KNX-Gebaeudesysteme. Smart Home and Intelligent Building Control Energy Efficiency in Buildings with ABB i-bus® KNX.

[66] Wenqi Guo, Willam M. Healy, and Mengchu Zhou. Wireless Mesh Networks in Intelligent Building Automation Control: A Survey. *International Journal of Intelligent Control and Systems*, March 2011.

[67] Siemens. Desigo Building Automation Energy-Efficient and Flexible: The Innovative System for Cost-Effective Buildings.

[68] Rodrigo Pantoni, Cleber Fonseca, and Dennis Brandão. Street Lighting System Based on Wireless Sensor Networks, Energy Efficiency - The Innovative Ways for Smart Energy, the Future Towards Modern Utilities. *Dr. Moustafa Eissa (Ed.)*, 2012.

[69] European Commission: Community Research. European SmartGrids Technology Platform: Vision and Strategy for Europe's Electricity Networks of the Future. *Technical Report*, 2006.

[70] Jorge Gil, Júlio Almeida, and José Pinto Duarte. The backbone of a City Information Model (CIM): Implementing a spatial data model for urban design. *City Modelling - eCAADe*, 2011.

[71] Bentley Systems. City Information Modeling for Sustaining Cities: Lessons Learned from Advanced Users. *Case Study Showcase*, August 2011.

[72] Hamid Gharavi and Reza Ghafurian. Smart Grid: The Electric Energy System of the Future. *Proceedings of the IEEE*, 99(6):917–921, June 2011.

[73] A. Illic, T. Staake, and E. Fleisch. Using Sensor Information to Reduce the Carbon Footprint of Perishable Goods. *IEEE Pervasive Computing*, 8(1):22–29, 2009.

[74] A. Dada and F. Thiesse. Sensor Applications in the Supply Chain: The Example of Quality-based Issuing of Perishables. In *Proc. of Internet of Things*.

[75] IBM. The smarter supply chain of the future: Insights from the Global Chief Supply Chain Officer Study.

[76] IoT-A Project. SOTA Report on Existing Integration Frameworks/Architectures for WSN, RFID and Other Emerging IoT Related Technologies. *Project Deliverable D1.1*, March 2011.

[77] Frank F. Kuo. The ALOHA System. *ACM Computer Communication Review*, 1995.

[78] Bill Glover and Himanshu Bhatt. RFID Essentials. *O'Reilly Media, Inc., 2006 ISBN 0-596-00944-5, pages 88-89*.

[79] EPC Global. Standards. URL http://www.gs1.org/epcglobal/standards/.

[80] MIT Auto ID Center. 3.56 MHz ISM Band Class 1 Radio Frequency Identification Tag Interference Specification: Candidate recommendation, Version 1.0.0. *Technical Report MIT-AUTOID-WH-002*.

[81] N. Kushalnagar, G. Montenegro, and C. Schumacher. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. *IETF Draft, RFC4919*.

[82] IEEE Task Group 4 (TG4). IEEE 802.15 WPAN Standard. URL http://www.ieee802.org/15/pub/TG4.html.

[83] Bluetooth SIG. Bluetooth Technical Information. URL http://www.bluetooth.com/Pages/Tech-Info.aspx.

[84] Maria-Gabriella Di Benedetto and GuerinoGiancola. Understanding Ultra Wide Band Radio Fundamentals. *Prentice Hall*, June 27, 2004.

[85] Moe Z. Win and Robert A. Scholtz. Impulse Radio: How It Works. *IEEE Communications Letters*, 2(1), Jan 1998.

[86] IEEE Task Group 4a (TG4a). IEEE 802.15 WPAN Low Rate Alternative PHY. URL http://www.ieee802.org/15/pub/TG4a.html.

[87] Kay Römer. Time Synchronization and Localization in Sensor Networks. *Ph.D. Dissertation*, 2005.

[88] K. Kim, S. D. Park, G. Montenegro, S. Yoo, and N. Kushalnagar. 6LoWPAN Ad Hoc On-Demand Distance Vector Routing (LOAD). *Internet Draft, work in progress, draft-daniel-6lowpan-load-adhoc-routing-03*, .

[89] K. Kim, S. D. Park, G. Montenegro, I. Chakeres, and C. Perkins. Dynamic MANET On-demand for 6LoWPAN (DYMO-low) Routing. *Internet Draft, work in progress, draft-montenegro-6lowpan-dymo-low-routing-03*, .

[90] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc On-Demand Distance Vector Routing. In *Proc. of 2nd IEEE Workshop on Mobile Computing Systems and Applications*, 1997.

[91] K. Kim, S. Yoo, J. Park, S. D. Park, and J. Lee. Internet Draft: Hierarchical Routing over 6LoWPAN (HiLow), . URL http://tools.ietf.org/html/draft-daniel-6lowpan-hilow-hierarchical-routing-01.

[92] IETF ROLL Working Group. RPL: The IP Routing Protocol Designed for Low Power and Lossy Networks. *Internet-Draft, RFC6550*.

[93] Jamal N. Al-karaki and Ahmed E. Kamal. Routing Techniques in Wireless Sensor Networks: A Survey. *IEEE Wireless Communications*, 11:6–28, 2004.

[94] S. Duquennoy, G. Grimaud, and J.-J. Vandewalle. Smews: Smart and Mobile Embedded Web Server. In *Intl. Conf. on CISIS*, 2009.

[95] I. Agranat. Engineering Web Technologies for Embedded Applications. *IEEE Internet Computing*, 2(3):40–45, 1998.

[96] O. Akribopoulos, I. Chatzigiannakis, C. Koninis, and E. Theodoridis. A Web Services-oriented Architecture for Integrating Small Programmable Objects in the Web of Things. *Development in E-systems Engineering*, pages 70–75, 2010.

[97] Kwang il Hwang, Jeongsik In, Nhokyung Park, and Doo seop Eom. A design and Implementation of Wireless Sensor Gateway for Effiecient Querying and Managing through World Wide Web. *IEEE Transactions on Consumer Electronics, 2003.*

[98] Vlad Trifa, Samuel Wiel, Dominique Guinard, and Thomas Bohnert. Design and Implementation of a Gateway for Web-based Interaction and Management of Embedded Devices. *In Proc. of IWSNE 2009.*

[99] Roy Thomas Fielding. REST: Architectural Styles and the Design of Network-based Software Architectures. *University of California, Irvine, Doctoral dissertation*, 2000.

[100] S. Duquennoy, G. Grimaud, and J.-J. Vandewalle. The Web of Things: Interconnecting Devices with High Usability and Performance. In *Proc. of ICESS'09*, May 2009.

[101] A. Jules. RFID Security and Privacy: a Research Survey. *IEEE Journal on Selected Areas in Communications*, 24:381–394, 2006.

[102] R. Acharya and K. Asha. Data Integrity and Intrusion Detection in Wireless Sensor Networks. *In Proc. of 16th Intl. Conf. on Networks, ICON'08, New Delhi, India*, 2008.

[103] C. M. Medaglia and A. Serbanati. An Overview of Privacy and Security Issues in the Internet of Things. *In Proc. of TIWDC, Pula, Italy, 2009.*

[104] V. Mayer-Schoenberger. Delete: The Virtue of Forgetting in the Digital Age. *Princeton University Press*, 2009.

[105] Seapahn Meguerdichian, Sasa Slijepcevic, Vahag Karayan, and Miodrag Potkonjak. Localized Algorithms in Wireless Ad-hoc Networks: Location Discovery and Sensor Exposure. *MobiHoc '01.*

[106] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). URL http://tools.ietf.org/html/rfc3626.

[107] E. Perkins Charles and Bhagwat Pravin. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *SIGCOMM'94*, London, August 1994.

[108] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. *Book: Mobile Computing*, pages 153–181, 1996.

[109] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva. Directed Diffusion for Wireless Sensor Networking. *IEEE/ACM Transactions on Networking*, 11:2–16, February 2003.

[110] Joanna Kulik, Wendi Heinzelman, and Hari Balakrishnan. Negotiation-based Protocols for Disseminating Information in Wireless Sensor Networks. *Journal on Wireless Networks, March-May 2002.*

[111] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Ad-Hoc Networks Beyond Unit Disk Graphs. In *Proc. of Joint Workshop on Foundations of Mobile Computing (DIALM-POMC '03)*, 2003.

[112] Gang Zhou, Tian He, Sudha Krishnamurthy, and John A. Stankovic. Impact of Radio Irregularity on Wireless Sensor Networks. In *Proc. of the 2nd Intl. Conf. on Mobile Systems, Applications, and Services (MobiSys'04)*, pages 125–138, 2004.

[113] Ana Maria Popescu, Gabriel Ion Tudorache, Bo Peng, and Andrew H. Kemp. Surveying Position Based Routing Protocols for Wireless Sensor and Ad-hoc Networks. *IJCNIS 2012*, 4(1).

[114] H. Takagi and L. Kleinrock. Optimal Transmission Ranges for Randomly Distributed Packet Radio Terminals. *IEEE Transaction on Communications*, 32:246–257, 1984.

[115] H. Takagi and L. Kleinrock. Transmission Range Control in Multihop Packet Radio Networks. *IEEE Transaction on Communications*, 34(1):38–44, 1986.

[116] E. Kranakis, H. Singh, and J. Urrutia. Compass Routing on Geometric Networks. In *Proc. 11th Canadian Conf. Computational Geometry*, August 1999.

[117] I. Stojmenovic and X. Lin. GEDIR: Loop-Free Location Based Routing in Wireless Networks. *Int'l Assoc. Science and Technology for Development (IASTED) and Conf. Parallel and Distributed Computing and Systems*, pages 1025–1028, Nov. 1999.

[118] R. Nelson and L. Kleinrock. The Spatial Capacity of a Slotted ALOHA Multihop Packet Radio Network with Capture. *IEEE Transaction on Communications*, 32(6): 684–694, 1984.

[119] I. Stojmenovic and X. Lin. Power-Aware Localized Routing in Wireless Networks. *IEEE Trans. Parallel and Distributed System*, 12(11), November 2001.

[120] Ivan Stojmenovic and Xu Lin. Loop-free Hybrid Single-path/Flooding Routing Algorithms with Guaranteed Delivery for Wireless Networks. *IEEE Transactions on Parallel and Distributed Systems*, 2001.

[121] Ivan Stojmenovic, Mark Russell, and Bosko Vukojevic. Depth First Search and Location Based Localized Routing and QoS Routing in Wireless Networks. *Book Title: Computers and Informatics*, pages 21–24, 2000.

[122] J. A. Bondy and U. S. R. Murty. Graph Theory with Applications. *Elsevier North-Holland*, 1976.

[123] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A Survey on Sensor Networks. *IEEE Communications Magazine*, pages 102–114, August 2002.

[124] Dong-Hee Shin. Ubiquitous city: Urban technologies, urban infrastructure and urban informatics. *Journal of Information Science*, 35:515–526, 2009.

[125] Christian Frank, Philipp Bolliger, Friedemann Mattern, and Wolfgang Kellerer. The sensor internet at work: Locating everyday items using mobile phones. *Pervasive and Mobile Computing*, 4(3):421–447, 2008.

[126] J. Sanchez, P. Ruiz, X. Liu, and I. Stojmenovic. GMR: Geographic Multicast Routing for Wireless Sensor Networks. In *Proc. of the 3rd Annual IEEE Conf. on Sensor and Ad Hoc Communications and Networks (SECON'06)*, 2006.

[127] Juan A. Sanchez, Pedro M. Ruiz, and Ivan Stojmenovic. Energy-efficient Geographic Multicast Routing for Sensor and Actuator Networks. *Computer Communication*, pages 2519–2531, 2007.

[128] M. Transier, H.Fueler, J. Widmer, Martin Mauve, and Wolfgang Effelsberg. Scalable Position-based Multicast for Mobile Ad-hoc Networks. In *First Intl. Workshop on Broadband Wireless Multimedia: Algorithms, Architectures and Applications (BroadWim '04)*, 2004.

[129] Shibo Wu and K. Selcuk Candan. GMP: Distributed Geographic Multicast Routing in Wireless Sensor Networks. In *Proc. of the 26th IEEE Intl. Conf. on Distributed Computing Systems (ICDCS'06)*, 2006.

[130] Young-Bae Ko and Nitin H. Vaidya. Geocasting in Mobile Ad Hoc Networks: Location-Based Multicast Algorithms. In *Proc. of the Second IEEE Workshop on Mobile Computer Systems and Applications (WMCSA'99)*, 1999.

[131] Prosenjit Bose, Pat Morin, Ivan Stojmenovic, and Jorge Urrutia. Routing with Guaranteed Delivery in Ad Hoc Wireless Networks. *Wireless Networks*, pages 609–616, 2001.

[132] Yan Yu, Ramesh Govindan, and Deborah Estrin. Geographical and Energy Aware Routing: A Recursive Data Dissemination Protocol for Wireless Sensor Networks. *UCLA Computer Science Department, Technical Report UCLA/CSD-TR-01-0023*, 2001.

[133] Karim Seada and Ahmed Helmy. Efficient Geocasting with Perfect Delivery in Wireless Networks. In *IEEE Wireless Communications and Networking Conference (WCNC'04)*, pages 2551–2556, 2004.

[134] Jie Lian, Kshirasagar Naik, Yunhao Liu, and Lei Chen. Virtual Surrounding Face Geocasting in Wireless Ad Hoc and Sensor Networks. *IEEE/ACM Transactions on Networking*, 17:200–211, February 2009.

[135] Ivan Stojmenivic. Geocasting with Guaranteed Delivery in Sensor Network. *IEEE Wireless Communications*, 2004.

[136] Young-Mi Song, Sung-Hee Lee, and Young-Bae Ko. FERMA: An Efficient Geocasting Protocol for Wireless Sensor Networks with Multiple Target Regions. In *Proc. of Intl. Conf. on Embedded and Ubiquitous Computing (EUC'05)*, 2005.

[137] Nassima Hadid and Jean Frederic Myoupo. Multi-Geocast Algorithms for Wireless Sparse or Dense Ad Hoc Sensor Networks. In *Fourth International Conference on Networking and Services (ICNS'08)*, 2008.

[138] Alain Bertrand Bomgni and Jean Frederic Myoupo. An Energy-Efficient Clique-Based Geocast Algorithm for Dense Sensor Networks. *Communications and Networks*, 2(2): 125–133, 2010.

[139] Chih-Yung Chang, Chao-Tsun Chang, and Shin-Chih Tu. Obstacle-Free Geocasting Protocols for Single/Multi-Destination Short Message Services in Ad-hoc Networks. *Wireless Networks*, 9:143–155, March 2003.

[140] Brent Boyer. Robust java Benchmarking: Part 1 and Part 2. *IBM's developerWorks, Technical Library*, 2008.

[141] T. He, C. Huang, B. M. Blum, J. A. Atankovic, and T. F. Abdelzaher. Range-Free Localization Schemes in Large Scale Sensor Networks. In *Proc. of the 9th Intl. Conf. on Mobile Computing and Networking (MobiCom'03)*, 2003.

[142] Issam Mabrouki, Xavier Lagrange, and Gwillerm Froc. Random walk based routing protocol for wireless sensor networks. In *Proc. of the 2nd Intl. Conf. on Performance Evaluation Methodologies and Tools (ValueTools '07)*, pages 1–10, 2007.

[143] K. Padmanabh, A.M.R. Vanteddu, S. Sen, and P. Gupta. Random Walk on Random Graph based Outlier Detection in Wireless Sensor Networks. *Wireless Communication and Sensor Networks, 2007. WCSN '07. Third International Conference on*, pages 45–49, Dec. 2007.

[144] Hui Tian, Hong Shen, and T. Matsuzawa. RandomWalk Routing for Wireless Sensor Networks. In *Sixth Intl. Conf. on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2005)*, pages 196–200, December 2005.

[145] Liang Zhang. A Self-adjusting Directed Random Walk Approach for Enhancing Source-Location Privacy in Sensor Network Routing. In *Proc. of the 2006 International Conf. on Wireless Communications and Mobile Computing (IWCMC '06)*, pages 33–38, New York, NY, USA, 2006.

[146] Santpal S. Dhillon and Piet Van Mieghem. Comparison of Random Walk Strategies for Ad Hoc Networks. In *Proc. of the Sixth Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net 2007)*, pages 196–203, Corfu, Greece, June 2007.

[147] M. Zorzi and R. R. Rao. Geographic Random Forwarding (GeRaF) for Ad Hoc and Sensor Networks: Multihop Performance. *IEEE Transactions on Mobile Computing*, 2 (4):337–348, 2003.

[148] M. Menzo T. Roosta and S. Sastry. Probabilistic Geographic Routing Protocol for Ad Hoc and Sensor Networks. In *Proc. Int. WorkshopWireless Ad Hoc Networks*, 2006.

[149] Christopher L. Barrett, Stephan J. Eidenbenz, Lukas Kroc, Madhav Marathe, and James P. Smith. Parametric Probabilistic Sensor Network Routing. In *Proc. of 2nd ACM Intl. Conf. on Wireless Sensor Networks and Applications, WSNA'03*, 2003.

[150] Karl Pearson. The Problem of the Random Walk. *Nature*, 72(1867), 1905.

[151] S. P. Meyn and R.L. Tweedie. Markov Chains and Stochastic Stability. 2005.

[152] Kay Römer, Benedikt Ostermaier, Friedemann Mattern, Michael Fahrmair, and Wolfgang Kellerer. Real-Time Search for Real-World Entities: A Survey. *Proc. of the IEEE*, pages 1887–1902, 2010.

[153] Dave Evans. The Internet of Things: How the Next Evolution of the Internet is Changing Everything. *White Paper, Cisco Internet Business Solutions Group*, April 2011.

[154] L. A. Zadeh. Outline of A New Approach to the Analysis of Complex Systems and Decision Processes. *IEEE Trans. on Sys., Man and Cybern.*, SMC-3:28–44, 1973.

[155] Suman Nath, Jie Liu, and Feng Zhao. SensorMap for Wide-Area Sensor Webs. *IEEE Computer*, 40(7):90–93, 2007.

[156] Arne Broering, Felix Bache, Thomas Bartoschek, and Corne P.J.M.van Elzakker. The SID Creator: A Visual Approach for Integrating Sensors with the Sensor Web. In *14th Int. Conf. on Geographic Information Science*, Utrecht, Netherlands, April 2011.

[157] Benoit Christophe, Vincent Verdot, and Vincent Toubiana. Searching the Web of Things. In *IEEE Intl. Conf. Semantic Computing*, 2011.

[158] Dennis Pfisterer, Kay Römer, Daniel Bimschas, Henning Hasemann, Manfred Hauswirth, Marcel Karnstedt, Oliver Kleine, Alexander Kroller, Myriam Leggieri, Richard Mietz, Max Pagel, Alexandre Passant, Ray Richardson, and Cuong Truong. SPITFIRE: Towards a Semantic Web of Things. *IEEE Communications Magazine*, Nov 2011.

[159] Petros Darasb, Theodoros Semertzidis, Lambros Makrisb, and Michael G. Strintzisa. Similarity Content Search in Content Centric Networks. In *Proc. Intl. Conf. on Multimedia (MM'10)*, 2010.

[160] Zhe Wang, Matthew D. Hoffman, Perry R. Cook, and Kai Li. Vferret: Content-based Similarity Search Tool for Continuous Archived Video. In *Proc. of the 3rd ACM Workshop on Continuous Archival and Retrival of Personal Experences (CARPE'06)*, 2006.

[161] Juergen Beringer and Eyke Huellermeister. Online Clustering of Parralel Data Streams. *Data and Knowledge Engineering*, 58:180–204, August, 2006.

[162] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient Similarity Search in Sequence Databases. *In Proc. of 4th Intl. Conf. on Foundations of Data Organization and Algorithms (FODO'93)*, pages 69–84, 1993.

[163] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast Subsequence Matching in Time-Series Databases. In *Proc. of the ACM Intl. Conf. on Management of Data (ACM SIGMOD'94)*, pages 419–429, 1994.

[164] Dina Q Goldin and Paris C Kanellakis. On Similarity Queries for Time-Series Data: Constraint Specification and Implementation. *Principles and Practice of Constraint Programming*, pages 137–153, 1995.

[165] Byoung-Kee Yi, H.V. Jagadish, and Christos Faloutsos. Efficient Retrieval of Similar Time Sequences Under Time Warping. *Proc. 14th Int. Conf. on Data Engineering*, 1997.

[166] Jacques M. Bahi, Abdallah Makhoul, and Maguy Medlej. Data Aggregation for Periodic Sensor Networks Using Sets Similarity Functions. In *7th IEEE Intl. Wireless Communications and Mobile Computing Conference (IWCMC 2011)*, Istanbul, Turkey, 2011.

[167] Brent Boyer. Robust Java benchmarking: Part 1 and Part 2. *IBM's developerWorks, Technical Library*, 2008.

[168] Tony Van Gestel, Johan A. K. Suykens, Dirk emma Baestaens, Annemie Lambrechts, Gert Lanckriet, Bruno Vandaele, Bart De Moor, and Joos Vandewalle. Financial Time Series Prediction using Least Squares Support Vector Machines within the Evidence Framework. *IEEE Trans. Neural Networks*, 2001.

[169] David J.C. MacKay. Bayesian Interpolation. *Neural Computation*, 4:415–447, 1991.

[170] Christopher Meek, David Maxwell Chickering, and David Heckerman. Autoregressive Tree Models for Time-Series Analysis. *Proc. 2nd SIAM Intl. Conf. Data Mining*, 2002.

[171] Kok Keong Teo, Lipo Wang, and Zhiping Lin. Wavelet Packet Multi-layer Perceptron for Chaotic Time Series Prediction: Effects of Weight Initialization. In *Proc. Intl. Conf. on Computational Science (ICCS'01)*, 2001.

[172] Hui Zou and Yuhong Yang. Combining Time Series Models for Forecasting. *International Journal of Forecasting*, 20:69–84, December 2003.

[173] Xiang Lian and Lei Chen. Efficient Similarity Search over Future Stream Time Series. *IEEE Transactions on Knowledge and Data Engineering*, 20:40–54, January 2008.