Distributed Algorithms for Networks Formation in a Scalable Internet of Things

by

Ahmed Jedda

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for the Doctorate in Philosphy degree in Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

The Internet of Things (IoT) is a vision that aims at inter-connecting every physical identifiable object (or, a *thing*) via a global networking infrastructure (e.g., the legacy Internet). Several architectures are proposed to realize this vision; many of which agree that the IoT shall be considered as a global network of networks. These networks are used to manage wireless sensors, Radio Frequency IDentification (RFID) tags, RFID readers and other types of electronic devices and integrate them into the IoT. A major requirement of the IoT architectures is scalability, which is the capability of delivering high performance even if the input size (e.g., number of the IoT objects) is large. This thesis studies and proposes solutions to meet this requirement, and specifically focuses on the scalability issues found in the networks of the IoT. The thesis proposes several *network formation algorithms* to achieve these objectives, where a network formation algorithm is an algorithm that, if applied to a certain network, optimizes it to perform its tasks in a more efficient manner by virtually deleting some of its nodes and/or edges.

The thesis focuses on three types of networks found in the IoT: *1) RFID readers coverage networks*; whose main task is to cover (i.e., identify, monitor, track, sense) IoT objects located in a given area, *2) readers inter-communications networks*; whose main task is to guarantee that their nodes are able to inter-communicate with each other and hence use their resources more efficiently (the thesis specifically considers inter-communication networks of readers using Bluetooth for communications), and *3) Object Name Systems (ONS)* which are networks of several inter-connected database servers (i.e., distributed database) whose main task is to resolve an object identifier into an Internet address to enable inter-communication via the Internet. These networks are chosen for several reasons. For example, the technologies and concepts found in these networks are among the major enablers of the IoT. Furthermore, these networks solve tasks that are central to any IoT architecture. Particularly, the thesis *a)* studies the data and readers redundancy problem found in RFID readers coverage networks and introduces decentralized *RFID coverage* and *readers collisions avoidance* algorithms to solve it, *b)* contributes to the problem of forming multihop inter-communications networks of Bluetooth-equipped readers by proposing decentralized time-efficient *Bluetooth Scatternet Formation* algorithms, and *c)* introduces a geographic-aware ONS architecture based on Peer-To-Peer (P2P) computing to overcome weaknesses found in existing ONS architectures.

Acknowledgements

My sincere thanks go first to my supervisor Dr. Hussein T. Mouftah. My fellow colleagues in the laboratory and myself have always found him when needed, and he has been always generous in sharing his experience and providing help whether it is technical or non-technical. Dr. Mouftah is a flame of energy, a confidence booster, an extremely inspirational character, and a very kind person and teacher. I have been very lucky to work with him during my PhD.

Among the most enlightening experiences I had during this PhD is working with Dr. Guy-Vincent Jourdan. A lot of what I have learned during these years is due to my collaboration with him. Dr. Jourdan continued to follow my progress even after our research collaboration finished. More importantly, Guy-Vincent is a good friend. I appreciate all the help he has given me (and he knows what I am talking about). I would like to thank Dr. Arnaud Casteigts who has been working with me on Bluetooth Scatternet Formation algorithms. He has also helped me in most of the other problems of this thesis. Dr. Casteigt is a very bright researcher whose strong technical skills simplified and deciphered many of my ideas and writings. I would also like to thank Dr. Mazen G. Khair who has been working with me on RFID reader networks. He is a very knowledgeable researcher in his field with many interesting and creative ideas. I would like to thank Dr. Nicola Santoro for introducing me to the field of distributed algorithms through his courses; principles of distributed computing and advanced topics in distributed computing. He is among the best teachers I have ever had.

I would like to thank my father; Hassine - the best role model of a successful self-made man through hard work, my mother; Naziha - the kindest and sweetest woman ever, and my sisters; Jihene, Kawther and Sara - and their wonderful families. This thesis, however, would have never been finished without the love, support and patience of my beautiful wife, Hafsa.

Contents

List of Tables

List of Figures

Part I

Introduction

Chapter 1

Introduction

In the early 1990's, Mark Weiser predicted in [Wei91] that computers will be invisibly surrounding humans everywhere. These *ubiquitous computers* are supposed to help humans to perform their everyday tasks with less effort. A few decades before Weiser's predictions, Karl Steinbuch similarly predicted that "computers will be interwoven in almost every industrial product" [1]. These predictions have been almost realized nowadays, as computers are found embedded in smart-phones, TV's, tablets, cars, laundry machines and almost every modern industrial machine. A vision that extends further these predictions is the *Internet of Things* (`IoT`).

The `IoT` vision calls for the interconnection of every *thing* via a global networking infrastructure (e.g., the Internet). A *thing*, in this context, is defined as any identifiable physical object (e.g., river, tree, screw, bed, coffee machine, fridge and others) [Hal10]. Thus, the `IoT` *things* are also called `IoT` *objects*. There is no common definition of the `IoT`. The most suitable definition, according to this thesis, is found in [r108], where `IoT` is defined as a world wide network that consists of interconnected objects uniquely addressable and is based on standard communication protocols (see [AIM10] for more definitions). Many of these standard communication protocols already exist and have reached a maturity level (e.g., Bluetooth, ZigBee, IP, TCP, UDP, and others). As the legacy Internet, the `IoT` is a network of networks, called *edge networks* [TW10, Fur08]. The main objectives of these networks is to interconnect the `IoT` objects and integrate them into a global networking infrastructure. Due to the expected ultra large number of `IoT` objects, a major challenge for the `IoT` is its scalability, which is the ability of

[1] A brief history of the Internet of Things: `http://postscapes.com/internet-of-things-history` (last fetched 19-11-2013).

the IoT to handle a large number of objects without having its performance negatively and substantially affected. This thesis studies and proposes solutions to some of the scalability issues that occur in the IoT networks. A brief background knowledge on the IoT is given before getting into the details of the issues that the thesis studies.

1.1 Background Knowledge: The Internet of Things

The Enabling Concepts of the IoT

According to the IoT definition given above and other definitions, a *thing* must be provided with the following basic properties in order to be part of the IoT. First, the *thing* must have a unique identifier (i.e., *identifiability*), since it is not possible to differentiate between the *things* otherwise. Second, the *thing* must be able to communicate with other *things* (i.e., *communication*). Third, an IoT *thing* must be *intelligent*. The intelligence level of a *thing* may vary from basic intelligence such as storing simple information about the *thing* (e.g., identifier, status, location etc ...) to a more complex level of intelligence (e.g., recognizing complex external events and making decisions). See [WMZA02] [MFH09] for more details about objects intelligence levels [2]. An IoT *thing* may be provided with these properties by embedding a computer in it. This embedded computer must have a communication interface. However, equipping all IoT *things* with embedded computers is not feasible, at least at the current time. To overcome this issue, the Auto-ID center [3], and independently the Ubiquitous ID Center [4], introduced two concepts that are regarded as the enabling concepts of the IoT. These concepts are:

1. *Remote intelligence* (or, intelligence by the network [MFH09]), which means that an IoT *thing* may delegate a remote server to perform all its intelligent tasks. The remote server shall be easily accessed via the Internet.

2. *Automatic identification*, which may be implemented using various technologies. Among these technologies, Radio Frequency IDentification (RFID) showed to be

[2] Intelligence may be seen as the capability of processing information. The level of intelligence is increased as the amount of knowledge increases. This may be achieved using *sensing* technologies. More benefits may be obtained from Intelligence with the use of *actuators*. Thus, sensing and actuating are optional parts of intelligence.

[3] The Auto-ID center is known currently as the Auto-ID labs, http://www.autoidlabs.org/page.html (last fetched 1-1-2014).

[4] Ubiquitous ID Center: http://www.uidcenter.org (last fetched 1-1-2014).

very promising and attracted much attention recently. RFID is actually seen as a major enabler for the `IoT`.

Radio Frequency IDentification (RFID)

RFID, in its general definition, is a term used for any technology that transmits identifiers via radio [Rou06]. More commonly, an RFID system refers to a system that consists of at least an *RFID tag* and *RFID reader*. The RFID tag is an electronic circuit that contains at least a memory and a radio transceiver. An RFID tag may be equipped with its own power source, and hence it is called an *active tag*. RFID tags that are not equipped with power supplies are called *passive tags*. An RFID reader interrogates a nearby tag by sending a radio signal to it. Upon the reception of the reader's signal, the tag sends back its memory content (or some of it). If the tag is passive, the received signal's energy is used to empower the tag and sends the reply back to the reader. Thus, a passive RFID tag does not initiate communication with the reader.

The memory content of an RFID tag may contain a unique code that serves as a unique identifier of the tag. The tag's unique identifier may be used to identify an object which has the tag attached to it. Attaching tags to objects of various sizes and costs is becoming common as the cost and size of RFID tags are decreasing. For example, a company called Nonatec introduced small-sized RFID tags that can be attached to ants. Kodac introduced ingestible RFID tags that can be attached to drug pills, and can be used thus to monitor patients health. These small-sized tags are commercially available with an acceptable cost. Contrary to wireless sensor networks, the RFID technology does not depend on *guessing* to identify nearby objects [PROR99] [LPT06] [ZLN11]. Moreover, RFID does not depend on a line of sight to identify nearby objects as in the case of barcodes. Thus, robust identification of objects of different sizes is becoming possible with the use of the RFID technology, and this is basically why RFID attracted much attention recently.

The Minimalist `IoT` Architecture

Remote intelligence and automatic identification are combined in an architecture that is called in this thesis the *minimalist `IoT` architecture*. This architecture is inspired by two similar architectures independently introduced by the Auto-ID Center (or more specif-

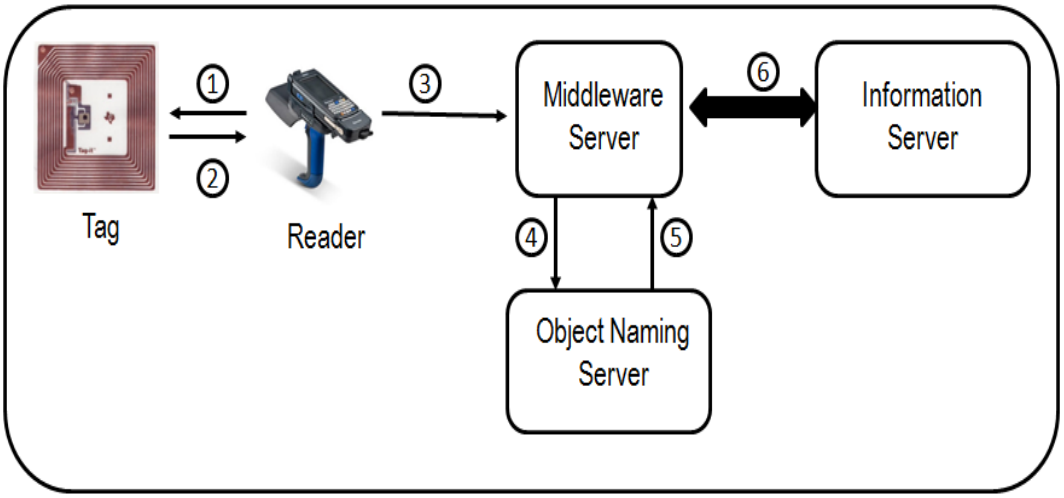


Figure 1.1: Minimalist `IoT` architecture inspired by the EPCglobal architecture framework [ABD+07] and the Ubiquitous ID architecture [uid06]. The tasks executed by the components of this architecture are the following: 1) The *reader* interrogates the object's *tag*. 2) The tag sends its identifier to the reader. 3) The reader sends the identifier to the *middleware server*. 4) The middleware server queries the *object naming server (ONS)* for the address of the *information server* of the interrogated object, using the object's identifier. 5) The ONS sends the address of the information server back to the middleware server. 6) The middleware server and the information server communicate with each other.

ically, the EPCglobal organization) [5] [ABD+07] and the Ubiquitous ID Center [uid06]. The architecture, illustrated in Figure 1.1, can be seen as the first step towards realizing the `IoT`. The components of this architecture are: 1) a *tag* (attached to an object and stores a unique identifier), 2) a *reader*, 3) a *middleware server*, 4) an *information server* (that is, the remote server delegate to handle the intelligent tasks of an object), and 5) an *object naming server (ONS)* (which is a global database that stores mappings between each object identifier and its information server address). The tasks executed by these components are explained in Figure 1.1.

Note that the components of the minimalist `IoT` architecture are not necessarily separate physical entities. For example, the tag may be an embedded computer that stores a unique code and has radio communication interface. This embedded computer

[5] *Historical note*: the Auto-ID Center is an organization that had substantial contributions to RFID and the Internet of Things. It is originally a research center at Massachusetts Institute of Technology, that was later divided into a collaboration of several international labs, called Auto-ID labs, and an international organization, called the EPCglobal. The Auto-ID labs main focus is on developing technologies that enable the Internet of Things, whereas EPCglobal focuses on standardizing and marketing technologies that are part of the Internet of Things. Several protocols have been introduced and standardized by the Auto-ID labs and EPCglobal, many of them focus on RFID since it is a major enabler of the `IoT`.

may be attached to an object, and the unique code may serve as a unique identifier of the same object. Thus, the embedded computer is an RFID tag. Note that this does not conflict with the general definition of RFID. This implies that any implementation of RFID may fit with this architecture. Also, note that the embedded computer may as well be the information server of the object it is attached to, since the intelligent tasks of the object may be delegated directly to the embedded computer.

This thesis categorizes the functions of the minimalist `IoT` architecture into three layers, which are:

1. *Objects coverage layer*: this includes all functions between the tag and the reader. That is, this includes all functions required to identify, track, monitor objects. This also includes all functions required to sense the environment (e.g., using sensors). Wu et al. in [WLL+10] and Liu and Zhou in [LZ12] called this layer as the *perception layer* of the `IoT`.

2. *Data processing and transportation layer*: this includes all functions between the reader and the middleware server. Basically, this includes the transportation of data from the reader to the middleware. This also includes storing and processing the data. This thesis, however, focuses on functions related to data transportation mainly.

3. *Objects names resolution layer*: this includes all functions between the middleware server and the object name server. Basically, this includes the linkage between an object identifier and its information server address.

These layers are the central layers of any `IoT` architecture. Every other function is either considered in an *application layer*, or is handled by the Internet layers (i.e., OSI model or TCP/IP model layers). EPCglobal standardized several solutions for functions found in each of the central layers. At the object coverage layer, EPCglobal focused mainly on integrating the RFID technology. This is because of the important role played by RFID in the `IoT`, and due to the lack of research and standards for RFID. Among the standards presented by EPCglobal are: 1) unique identifiers standards for identifying objects, 2) air interface protocols for interrogating RFID tags, 3) protocols for readers management, 4) protocols for recording interrogation events, and 5) protocols of object names resolution. Thiesse et al. in [TFH+09] gives a brief description of these standard

protocols. The Ubiquitous ID architecture focuses more on identifying and modeling the objects (see [uid06] for more details).

1.2 Thesis Motivations

The IoT attracted significant interest from governmental and industrial organizations. For instance, the IoT was among the interests of the 2005 United Nations World Summit on Information Society in Tunis. A report published by the summit stated that the IoT 'will radically transform our corporate, community, and personal spheres.' The US National Intelligence Council classified the IoT as one of six disruptive civil technologies that may potentially impact the US national power [nic08]. Also, the European Union is investing heavily in the IoT by funding leading research projects, such as CASAGRAS [6], that focus primarily on realizing the IoT. The Chinese government is also intensively promoting the IoT [Liu13]. Cisco Systems Inc., the giant computer networking corporation, is investing heavily on the IoT, and it estimates that global profit of the IoT will be about $14.4 trillions over the next decade. Other giant corporations such as IBM, Oracle, Wal-Mart, and others are also involved in the IoT. However, there are many issues to be solved before the IoT is totally realized. Among these issues is the scalability of the IoT.

Statistics in [Eva11] estimate that the number of the IoT objects will reach about 50 billion in 2020. This ultra-large number of objects causes several challenges for the IoT. The most important and general of which are: 1) the processing and transportation of the excessive amounts of data generated by the IoT objects and networks, and 2) the enormous amounts of energy required to power the IoT and its objects. There are suspicions about whether the data and energy required by the IoT can be handled by the currently existing networking, processing, or power-supply infrastructures. Another important challenge is the management of the objects and networks of the IoT. As a result, a main requirement of the IoT is to be scalable [AIM10] [MSDC12] (that is, to be able to handle the large number of IoT objects without having its performance negatively affected). In fact, Wu et al. in [WTJ+11] considered achieving scalable networking in IoT as the 'most critical challenge' for IoT. This thesis studies some of the challenges that face the realization of a scalable IoT; with a special focus on challenges found at

[6] CASAGRAS: Coordination And Support Action for Global RFID-related Activities and Standardization - http://www.iot-casagras.org/ (last fetched 6-11-2013).

the networking part of IoT. The general definition of *scalability* in this thesis is given in the following:

Definition 1.2.1. Scalability *is the ability of a system to handle a large sized input without having its performance negatively and substantially affected.*

For example, a network of nodes is said to be *scalable* if the performance of running a certain algorithm by its nodes is not substantially negatively affected if the number of nodes increases. To improve the scalability of the network is to improve the complexity of the algorithm with respect to the number of nodes.

1.3 IoT Scalability

The problem of designing a scalable IoT has been tackled using several general approaches. This section surveys these approaches and leads the reader to the approach followed in this thesis to handle the problem.

Remark 1.3.1. *The thesis studies several subproblems that help in solving the IoT scalability, which is the general problem of the thesis. The solutions of these subproblems are all categorized under one of the approaches surveyed below (namely, optimizing the IoT networks using network formation algorithms). Related work for each of these subproblems are surveyed in-depth in the chapters where these problems are studied.*

Approach I: Improve the components

This means that the components forming the IoT (such as, IoT objects, the Internet infrastructure, or the storage and mining facilities) shall become more capable in handling large amounts of data and be more efficient in energy consumption. For example, storage infrastructures shall be scalable and more energy efficient as suggested by [BS11, TSH09]. Similarly, the electronic devices used to inter-connect the IoT objects shall be designed to consume as little energy as possible [ZGLB10, MJ11, r108], use renewable energy [MJ11], or even harvest their own energy [r108] [7]. This is of great importance especially as these devices must be functional for long period of times with the least possible human

[7] See also Cymbet energy harvesting products in: http://www.cymbet.com/design-center/energy-harvesting.php (last fetched 20-11-2013).

interaction. Fortunately, the research area of energy-efficient electronic devices has been noticeably active during the last decade.

One of the IoT components that has been subject to many recommendations for modifications is the Internet itself. There is significant work on introducing and standardizing routing and network management protocols that meets the new requirements of the IoT (see for example, [WT09, HCC09]). Likewise, there is an important school of thought that calls for a radical redesign of the Internet on the form of a *future Internet* [She95, BBDC11]. This redesign shall not meet only the requirements of the IoT, but also the requirements of other technologies that will change the nature of the current Internet, such as multimedia streaming, service-oriented architectures, smart grids, secure transactions, and others.

Approach II: Resource re-utilization (or simply, efficient utilization)

Service-oriented architectures were proposed in [HKS09, BS11, OH12] as a way to manage the IoT objects on a higher and more abstract level. Service-oriented architectures are software architectures that see each IoT object as a *provider* of *services* (e.g., sensor readings, real-time video streaming, an RFID reader interrogations of RFID tags in proximity). A service-oriented application does not depend on the object itself, but on the services provided by the object. Thus, if another object provides the same service (or a similar one), then the application may indirectly connect to that object. Moreover, an IoT object may indirectly provide services to more than one application. These procedures and others are accomplished by a *middleware layer* (a software layer that manages the inter-connection between the service providers and the applications) [AIM10]. Thus, software-oriented architectures enables the re-utilization of the resources of IoT [OH12].

An application of software-oriented architectures is crowd-sensing, which also focuses on re-utilizing resources. Crowd-sensing can be defined as the action of sensor nodes volunteering to collaboratively execute certain tasks by sharing sensing and processing capabilities, where these tasks are required by a crowd-sensing application. See [OH12, GYL11, HCCL13] for more details about crowd-sensing. There are suggestions for extending this idea by making the sensor nodes contributions rewardable [OH12]. Crowd-sensing may be generalized to crowd-monitoring, crowd-tracking, crowd-processing and other similar applications.

Approach III: Processing at the edges

The IoT objects generate excessive amounts of data. These data can be on the form of sensor readings, events captured by RFID readers (e.g., detection, monitoring, tracking of RFID tags), or other types of data. A general approach followed in the IoT to handle this large scale of data is to reduce the amount of data transmitted via the Internet by transmitting data only when necessary. This can be achieved by filtering irrelevant data [HKS09], transmitting data only when important events occur [HKS09], smarter data fusion (e.g., by averaging or compressing the data) [EGHK99, KKK+08], or even by mining the data at the edge networks [GBA+13]. Other than avoiding the overloading of the Internet with - perhaps unnecessary - excessive data, this approach also decreases the overall energy consumption in the IoT [AAE12]. For example, Baliga et al. found in [BAHT11] that data transport dominates the total power consumption at cloud storage considering a high level usage (about 58%) - (see [BAHT11] for more details).

Approach IV: Optimizing the networks

One way to use the resources of the IoT objects more efficiently is to create local networks of these networks by enabling the the IoT objects to communicate with each other in order to perform certain *tasks* using distributed algorithms. However, the large scale of the IoT objects may lead to an increase in the number of nodes in these networks, which as a result degrades the performance of their distributed algorithms. A network can overcome this issue by optimizing its distributed algorithms. For instance, a network shall avoid *centralization* and use only *decentralized algorithms* [EGHK99, HKS09], where a decentralized algorithm is a distributed algorithm that its execution does not depend on a centralized node. This is because decentralized algorithms are more scalable compared to centralized algorithms. More preferably, a network shall perform its tasks, if possible, using *local distributed algorithms* [Suo13, EGHK99], which are decentralized algorithms in which the final decision taken by every node v depends on v itself and on neighbors of v found within a constant number of hops from v (more details on local distributed algorithms are found in Section 1.6). Local distributed algorithms are not only scalable, but also fault tolerant. Another approach to handle scalability in the IoT networks is to use *network formation algorithms*. This thesis follows this approach to tackle the IoT scalability problem, and hence it is discussed in more details in the following section.

Networks Formation Algorithms:

The main objective of a network formation algorithm is to organize the network such that it exploits its resources efficiently, and hence execute its tasks with better performance. These algorithms, also called *network planning algorithms* and *topology construction* algorithms, can be formally defined as follows:

Definition 1.3.2 (Network Formation Algorithms)**.** *Given:*

- *a network $G = (V, E)$, where V is the set of nodes and E is the set of edges.*
- *a* task $\mathbf{t}$ *that shall be executed by the nodes of G.*
- *a performance metric for executing $\mathbf{t}$ over the network G, denoted $\mathsf{p}(G, \mathbf{t})$.*

A network formation algorithm *is an algorithm that forms a subnetwork $G' = (V', E')$ of G such that: 1) $V' \subseteq V$, 2) $E' \subseteq E$, and 3) $\mathsf{p}(G', \mathbf{t}) \succ \mathsf{p}(G, \mathbf{t})$ (that is, the performance of executing $\mathbf{t}$ over G' is improved compared to its execution over G.*

Definition 1.3.2 suggests that the available resources at a given network are used more efficiently if a network formation algorithm is employed. Note that the performance metric $p(G, \mathbf{t})$ can be a combination of multiple performance metrics depending where the network formation algorithm is applied.

Examples of Network Formation Algorithms This section gives a few examples found in the literature of the `IoT`. A detailed survey of network formation algorithm for wireless ad-hoc networks, which is an important part of the `IoT`, is found in [Wan08].

1. *Sensor coverage (found in wireless sensor networks)*: Let $G = (V, E)$ be a wireless sensor network whose main task is to cover (i.e., sense) every point in a given area. In some situations, the coverage task may be executed by a smaller number of sensor nodes, denoted V'. Thus, all sensor nodes in the set $\{V \setminus V'\}$ are considered *redundant*. As a result, these redundant sensor nodes consume unnecessary energy. Thus, the coverage task is executed more efficiently over the induced subnetwork G', which consists of the nodes V' and the edges between them. See [Wan11] for a survey of sensor coverage algorithms.

2. *Transmission power reduction (found in wireless ad-hoc networks)*: Let $G = (V, E)$ be a wireless ad-hoc network. The neighbors of a node $v \in V$, denoted $N(v)$, is the set of nodes which are within r_c units from v. The energy to transmit a message from v to a neighbor u depends on the distance r_c. Taking this into consideration, the objective of this problem is to let each node v connect only to a subset of its neighbors $N'(v)$. This forms a network G', which shall also be connected. Ramanthan and Rosales-Hain achieved this objective in [RRH00] by letting each node adjust its transmission range. Algorithm `Span` [CJBM02] achieves this objective by creating a connected backbone of the sensor nodes. Every sensor node not in the backbone may switch itself off without affecting the connectivity of the network, and hence it reduces the energy consumption.

3. *Network planarity for geographic greedy routing (found in wireless ad-hoc networks)*: The well-known greedy routing algorithm of [BMSU01] requires that the network be *planar* [8]. To achieve a correct routing result, a network formation algorithm that forms a planar subnetwork is executed before the routing algorithm. An example of a network formation algorithm that performs this task is *Gabriel Graph* construction [GS69], or localized Delaunay triangulation construction [LCW02].

Note that some collision avoidance algorithms and bandwidth assignment algorithms for wireless sensor networks can also be considered as network formation algorithms by defining the edges of the networks as functions of time or frequency.

There are many other network formation algorithms found in the literature that are not necessarily related to the `IoT`. For instance, a *minimum spanning tree (MST)* or a *dominating set* is often built to improve broadcasting at a given network (see [San06, SSZ02]). Moreover, building a *Steiner tree* is an approach frequently used to improve multicasting (i.e., broadcasting a message to a specific subset of nodes in a network) [KPP93].

The interaction of the `IoT` objects and the different types of the `IoT` objects create several types of networks. Some of these networks have been rarely studied in the literature. Thus, there are many undiscovered and unsolved network formation problems introduced by the `IoT`. Among the `IoT` network formation problems studied in this thesis are: 1) the *RFID coverage problem*, 2) the *Bluetooth scatternet formation problem*, and

[8] A planar network or a planar graph is a graph that can be drawn on the plane such that such that no edge crosses another.

3) the *geographic-aware P2P overlay construction problem*. Each of these problems is found in one of the layers of the minimalist `IoT` architecture. More details about these problems are given in the following.

1.4 Thesis Objectives

RFID and wireless sensor/actuator networks are two technologies that play a major role in generating a large number of `IoT` objects. These technologies will constitute most of the edge networks of the `IoT`. The scalability of wireless sensor/actuator networks have been studied intensively in the last decades. RFID has been less studied, especially from a networking perspective. Unfortunately, there is not much research done on designing network architectures that are specifically designed to integrate the large number of RFID-equipped objects into the `IoT`. This issue is noticed in the minimalist `IoT` architecture, which is designed to cover one tag at a time using one reader as explained in Figure 1.1. Interestingly, this issue remains, although the minimalist `IoT` architecture has been the target of continuous research to standardize its functions. Given its importance and its wide acceptance, this thesis **objective** is the extension of the minimalist `IoT` architecture in a way that makes it handle the large number of RFID-equipped `IoT` objects.

Extending, or scaling-up, the minimalist `IoT` architecture affects many functions in its three layers. The following elaborates more on this point. A large number of objects cannot be covered simultaneously by a single reader. Thus, coverage shall be collaboratively executed by a network of readers. Such a network is called an *RFID reader coverage network*. The readers of this network shall be able to inter-communicate with each other at the local scale. This inter-communication, which forms a *reader inter-communication network*, is important for better exploitation of the readers resources. These two scenarios show the advantage of using networks to handle the scalability issue, since organizing nodes into networks allows them to exploit their resources more efficiently. Similarly, a scalable object name resolution shall be performed collaboratively by a network of distributed database servers, forming a *distributed database*. The use of networks is not sufficient to handle the scalability issue, but instead the networks shall be optimized to be more scalable, and to exploit their resources more efficiently. Network formation algorithms are perfectly suitable for achieving these objectives.

As a result, this thesis suggests the use of network formation algorithms to overcome

the scalability issue found in the IoT, with special focus on RFID reader coverage network, reader inter-communication network, and ONS distributed database. This would, as a consequence, lead to an extension of the minimalist IoT architecture.

1.5 Thesis Contributions

This thesis studies three main problems; each of which is found in one of the central layers of the minimalist IoT architecture. The thesis introduces decentralized algorithms for each studied problem. These problems are summarized in the following. A summary of the contributions is found in Table 1.1.

1. *The RFID Coverage Problem* (Objects Coverage): The input of this problem is a network of RFID readers, and a set of RFID tags such that each tag is covered by at least one RFID reader. The objective is to cover all the tags with the minimum possible readers. This would eliminate readers redundancy and hence reduce the network energy consumption. Another objective is to eliminate data redundancy. This is achieved by assigning for each tag an *owner* reader that is the sole responsible for interrogating data.

   The thesis studies two versions of the RFID coverage problem. The first version assumes that readers are only allowed to *indirectly* communicate with each other by writing and reading the memory of RFID tags in their proximity (called, the *reader-tag* RFID coverage problem). The tags used in this type of algorithms are called *writable RFID tags* and are standardized in [tag11]. This problem is first introduced in [CRK+09]. The second version of the RFID coverage problem assumes that the readers communicate by exchanging messages via a wireless communication interface (called, the *reader-reader* RFID coverage problem). The thesis introduces several decentralized algorithms (deterministic and randomized) for each version of the RFID coverage algorithms; some of which are local distributed algorithms. Table 1.1 lists the names of these algorithms. The performance of the proposed algorithms is shown to outperform existing major RFID coverage algorithms with respect to different quality metrics.

2. *The Readers Collisions Avoidance Problem* (Objects Coverage): this problem is related to the reader-reader RFID coverage problem, and is applied in all RFID

reader networks where direct communication readers between the readers is allowed. The objective of the problem is to eliminate all signal collisions that are caused by readers interrogation signals and communication signals. It is shown how reader-reader RFID coverage algorithms may be used to solve this problem. The thesis introduces a new reader collision avoidance algorithm called `IOB-(RCA+COV)`, which is the first deteminstic decentralized algorithm for the reader collisions avoidance problem that does not necessitates any geographical location knoweldge.

3. *The Bluetooth Scatternet Formation Problem* (Data processing and transportation): intercommunication between readers increases the efficiency of reader networks, as it enables collaboration between the readers. The Bluetooth technology is found to be suitable to implement such inter-communications. Other than its technical advantages, the use of Bluetooth is motivated mainly by its wide availability nowadays. For instance, statistics in [Nok11] estimate that about 95% of today's mobile phones are Bluetooth-enabled. Given this availability, studies such as that in [abi13b] considers Bluetooth as a main enabler of the `IoT` along with RFID. However, the use of Bluetooth for networking introduces a problem called the Bluetooth Scatternet Formation problem. The objective of this network formation problem is to form Bluetooth networks that 1) meet the requirements of the Bluetooth standards, and 2) are efficient with respect to a number of quality metrics (including energy consumption). Two time-efficient algorithms are introduced, namely `Eliminate` and `BSF-UED`. Theoretical analysis and simulation experiments show that these algorithms outperform major competitor algorithms with respect to several performance metrics.

4. *The Geographic-Aware Object Name Resolution Problem* (Objects name resolution): the object name resolution problem of the ONS is the problem of linking the unique identifier of an `IoT` object to its address or the address of its remote server (i.e., the server delegated to execute intelligent tasks on behalf of the objects). The address is used to accomplish communication with the object. The address is possibly an IP address, a web address, or others. A major component in the minimalist `IoT` architecture is the Object Name System (ONS) which is a database that returns the address of an object if provided with its unique identifier. The ONS shall be a global distributed database. That is, it is a network of several database servers; each of which is responsible for a portion of the stored

Table 1.1: Summary of the main contributions of each chapter

Problems Studied	Contributions
Reader-Tag RFID Coverage	Randomized algorithms (`RANDOM`, `RANDOM`$^+$, and `MAX-MIN`) Deterministic algorithms (`GDE`, and `LIMITED-GDE`)
Reader-reader RFID Coverage	Deterministic local distributed algorithm (`OB-COVERAGE`) Deterministic algorithm (`IOB-COVERAGE`)
Readers Collisions Avoidance	Deterministic algorithm (`IOB-(RCA+COV)`)
Bluetooth Scatternet Formation	Time-efficient deterministic algorithm (`Eliminate`) Time-efficient deterministic algorithm (`BSF-UED`) Comparative study from a time-efficiency perspective
Geographic-aware ONS	A geographic aware content addressable network (`GCAN`)

data. An approach to improve the performance of the ONS distributed database is to organize its servers on a Peer-To-Peer (P2P) overlay. This allows the ONS to be more scalable, fault-tolerant and free of single points of governance (i.e., components in the ONS that may restrict its management and control to a few or only one organization). The thesis introduces a *geographic-aware* P2P overlay network over which the ONS may be implemented. This is because geographic-awareness at the servers of the ONS increases the overall performance of the ONS. Note that geographic awareness is defined briefly as the property of taking into consideration the geographic locations of the servers to improve the overall performance of the system.

1.6 Performance Analysis Methodology

This thesis introduces distributed algorithms for all of the problems studied. The performance of these algorithms is studied using two methods: empirical analysis and theoretical analysis. These methods are described in the following.

Empirical analysis: is done using simulation experiments. The complexity of the simulator depends on the performance metrics to be analyzed. The simulation experiments settings shall match the settings found in similar empirical studies. The experiments shall emulate realistic scenarios as much as possible. Most of the results are elaborated in graphs. However, the details of the results are elaborated in tables in Appendix B. The appendix also elaborates how the confidence intervals are calculated.

Theoretical analysis: Every distributed algorithm is executed over a *network* modeled as a graph $G = (V, E)$, where V is the set of *nodes* and E is the set of *edges*. The set of nodes may be called *vertices*, *readers*, *devices*, *peers* or *servers* depending on the context; however, the term nodes is the most frequently used. A pair of nodes v and u are *neighbors* if the edges (v, u) is in E. A pair of neighbors are able to *communicate* with each other, where communication herein may follow various forms. Except if otherwise mentioned, the links are assumed to be bidirectional. That is, if $(v, u) \in E$, then $(u, v) \in E$. The set of neighbors of a node v is denoted $N(v)$. These notations are used throughout this thesis, except if otherwise mentioned.

The theoretical analysis of any algorithm shall follow two steps. First, the *correctness of the algorithm* shall be formally proven. Then, the *complexity of the algorithm* shall be formally analyzed. As this thesis introduces distributed algorithms solely, the complexity measures that will be used are the *message complexity* and the *time complexity*. The message complexity counts the number of messages exchanged by all the network's nodes during the execution of the algorithm. A broadcast message sent by a node to k neighbors is counted as k messages. Time complexity is measured as the total delay required to execute the algorithm, assuming that each message is transmitted from one node to its neighbor (see [San06] for more details). Time complexity is measured in some references as the number of *communication rounds* required by a distributed algorithm. A communication round is a round in which every node 1) performs local computation, 2) sends a message to all its neighbors, and 3) receives a message from all its neighbors. (See [Lyn96] for more details). Both definitions of time complexity are used in this thesis; however, the default used definition of time complexity is the first. A distributed algorithm that executes only a constant number of communication rounds is called a *local distributed algorithm*. These algorithms have important theoretical and practical properties as they are scalable and fault-tolerant [Suo13]. Lastly, complexity analysis is done using the standard asymptotic notation, also called Landau notation (see [LRSC01] for more details).

1.7 Thesis Organization

This thesis is organized into parts; object coverage, data transportation, and object name resolution. Each part is independent and not required for the understanding of the other parts. Each part studies at least one problem. A detailed survey of related work for each

Table 1.2: Terms and notations frequently repeated in the thesis

Term	**Definition**
$G(V, E)$	A network (or, graph) consisting of a set of nodes V and a set of edges E
$N(v)$	Neighbors of a node v in a network (or, graph) G
$\|S\|$	The size of any given set S (i.e., the number of S elements)
$S \setminus T$	Set difference of any given sets S and T. That is, $S \setminus T$ denotes the set that contains all the elements that are in S but not in T (formally, $S \setminus T = x \mid x \in S, x \notin T$).
Lexicographical comparison	Given any two sets of comparable elements $S = s_1, ..., s_n$ and $T = t_1, ..., t_n$, S is lexicographically larger than T if there is $sj \succ tj$ for $1 \leq j \leq n$, while $si = tj$ for each $i < j$.

studied problem is given in the chapters where the problems are studied.

Chapter 2 studies the reader-tag RFC problem. Chapter 3 gives a comparative study of major reader-tag RFC algorithms using simulation experiments. Chapter 4 studies the reader-reader RFC problem and the RCA problem. Chapter 5 studies the BSF problem and introduces algorithm `Eliminate`. Chapter 6 introduces a BSF algorithm called `BSF-UED`, and gives a comparative study of major BSF algorithms using simulation experiments. Chapter 7 surveys existing ONS architectures, and introduces a geographic-aware content addressable network called `GCAN`. Chapter 8 concludes and proposes future research directions.

List of Publications

[1] Ahmed Jedda, Mazen G. Khair, and Hussein T. Mouftah, *Distributed algorithms for the RFID coverage problem*, IEEE International Conference on Communications (ICC'13), pp. 1758–1762, IEEE, (2013).

[2] Ahmed Jedda, Mazen G. Khair, and Hussein T. Mouftah, *Decentralized RFID coverage algorithms using writable Tags*, ACM Transactions on Sensor Networks (under review), ACM, (2013).

[3] Ahmed Jedda and Hussein T. Mouftah, *Decentralized RFID coverage with applications for the Reader Collision Avoidance Problem*, IEEE Transactions on Emerging Topics in Computing (under review), IEEE, (2013).

[4] Ahmed Jedda, Guy-Vincent Jourdan, and Hussein T. Mouftah, *Time-efficient algorithms for the outdegree limited Bluetooth scatternet formation problem*, IEEE Symposium on Computers and Communications (ISCC'12), IEEE, pp. 132–138, (2012).

[5] Ahmed Jedda, Guy-Vincent Jourdan, and Nejib Zaguia, *Towards better understanding of the behavior of Bluetooth networks distributed algorithms*, International Journal of Parallel Emergent Distributed Systems **27**, no. 6, 563–586, Taylor & Francis, (2012)

[6] Ahmed Jedda, Arnaud Casteigts, Guy-Vincent Jourdan, and Hussein T. Mouftah, *Bluetooth scatternet formation from a time-efficiency perspective*, Wireless Networks, 1–24. Springer, DOI: http://dx.doi.org/10.1007/s11276-013-0664-z, (2013).

[7] Ahmed Jedda, Arnaud Casteigts, Guy-Vinvent Jourdan, and Hussein T. Mouftah, *BSF-UED: a new time-efficient Bluetooth Scatternet Formation*

based on Unnecessary-Edges Deletion, IEEE Symposium on Computers and Communications (ISCC'13), IEEE, (2013).

[8] Ahmed Jedda, and Hussein T. Mouftah, *Adding geographic-awareness to P2P-based Object Naming Service architectures*, IEEE Global Communication Conference (Globecom), IEEE, (2014).

Part II

Objects Coverage

Chapter 2

RFID Coverage Using Reader-Tag Communication

The main tasks of *RFID reader coverage networks* (or simply, *reader networks*) is the identification, monitoring and tracking of RFID tags. These tasks, simply called *objects coverage*, are performed collaboratively by the readers of the reader network. This chapter studies the *RFID Coverage (RFC)* problem, whose main task is to optimize the energy consumption of a reader network by eliminating unnecessary redundancy at the readers layer. This chapter introduces two sets of RFC algorithms. The first set consists of three randomized algorithms, called `RANDOM`, `RANDOM`$^+$, and `MAX-MIN`. The second set consists of deterministic algorithms, called `GDE` and `LIMITED-GDE`. All of these algorithms are *decentralized* and assume that the only way of communication between the readers is by writing and reading the memories of the RFID tags in proximity. That is, the tags are assumed to be *writable*, which are standardized in [tag11]. These algorithms are therefore called *reader-tag RFC algorithms*. Chapter 3 gives a comparative study of the reader-tag RFC algorithms studied in this chapter.

Chapter 4 extends on the results of this chapter by introducing RFC algorithms that consider the case where the use of *writable* tags is not allowed. Instead, the algorithms of Chapter 4 assume that the readers communicate directly with each other via messages exchange using wireless communications interfaces. These algorithms are called *reader-reader RFC algorithms*. Chapter 4 also studies an important problem, called the *Readers Collision Avoidance (RCA)* problem. The objective of the RCA problem is to eliminate collisions caused by readers signals. An efficient solution of the RCA problem is essential

for a practical implementation of reader-reader RFC algorithms.

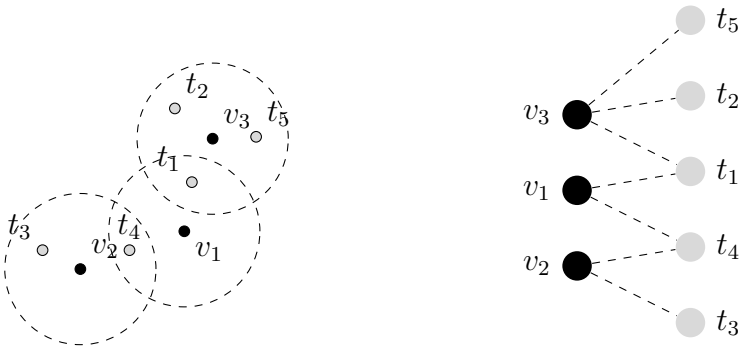
2.1 Introduction

An RFID system, consisting of an RFID *reader* and an RFID *tag*, is mainly used to identify, monitor, and track (or, simply *cover*) physical objects in proximity. This is done by attaching an RFID tag to the object to be covered. As the number of objects to be covered increases in a given environment, it becomes necessary to use a collaboration of readers to perform the coverage. This collaboration of readers is called an *RFID reader coverage network* (for short, *reader network*). Each reader in a reader network is responsible for covering a subset of the objects (or, tags) and report its readings to a special server that collects and processes the data gathered by all readers. An example of a reader network is given in Figure 2.1.

This chapter studies the *RFID Coverage* (RFC) problem, whose main objective is to increase the efficiency of a reader network by eliminating unnecessary redundancy in the reader network. A solution to the RFC problem eliminates two types of redundancies in reader networks: 1) *Data redundancy*, and 2) *Readers redundancy*:

1. **Data redundancy** occur in situations where more than one reader report the same readings about the same tag. This type of redundancies a) causes problems in processing and mining the data generated by a reader network [AH13], and b) causes an increase in the network traffic. Eliminating data redundancy can be done by assigning to each tag t an owner reader, such that only one reader in proximity of t (i.e., its owner) would report readings about this tag. Note that a tag does not need to know which reader owns it. Nevertheless, a reader must be aware of the tags it owns. This problem is called the *tag reporting* problem.

2. **Readers redundancy** occur if the tags of the reader networks can be covered by only a subset of the readers. Minimizing the number of readers in a given reader network, while preserving the tags coverage, improves the readers energy consumption. This decreases as well the reader network traffic, measured by the number of tag readings in the network. This procedure minimizes the number of readers that can cover all tags in the given reader network. This problem is called the *RFID redundant readers elimination* problem.



(a) Sample reader network consisting of three readers (black dots) and five tags (gray dots). The dashed circles represent the covering (or, interrogation) range of readers.

(b) Bipartite graph model for the reader network in (a). A reader is linked to a tag if it is within its interrogation range.

Figure 2.1: Example of a reader network that consists of three readers and five tags is illustrated in (a). The coverage relationships between readers and tags is usually modeled as a bipartite graph as shown in (b).

The RFC problem consists of the two subproblems defined above (i.e., the *tag reporting* problem, and the *RFID redundant readers elimination* problem).

The RFC problem is first introduced by Carbunar et al. in [CRK+09], who introduced an RFC algorithm, called `RRE` (Redundant Readers Elimination). The RFID readers in `RRE` are assumed to communicate *indirectly* with each other using *reader-tag communications*. That is, the readers communicate with each other by reading and writing the memory contents of the tags in their proximity. Therefore, `RRE` is called a *reader-tag RFC algorithm*. `RRE` implements reader-tag communications using *write/read rounds*. The procedures of write/read rounds is described in the following. The detailed definition of the RFC problem is given in Section 2.3.

A *write/read round* is a basic component in reader-tag RFC algorithms. Abstractly, a write/read round consists of a *write phase* and *read phase*. In the write phase, every reader v writes a set of bits, called the value of v, in the memory of all (or some of) its neighbor tags (i.e., tags that are covered by v). Every reader waits for specific period of time to allow every other reader to complete its write phase. In the read phase, the readers read the content of the memory of neighbor tags. At that time, the memory of a tag t contains the values of all the neighbor readers of t that wrote in t during the write phase.

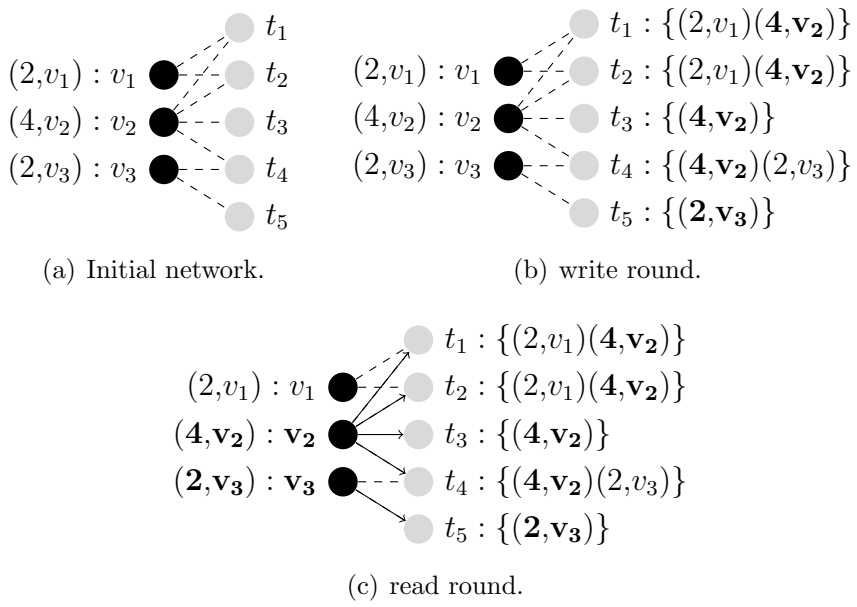
Carbunar et al. introduced in [CRK+09] a randomized implementation of write/read rounds that considers readers scheduling and collisions caused by the simultaneous write and read operations. The details of the implementation are found in [CRK+09].

To elaborate more on the idea of write/read rounds, a brief description of algorithm `RRE` is given in the following. `RRE` consists of a single write/read round. In the write phase, a reader v sets its value, denoted $\mathcal{W}(v)$, to $(|N_T(v)|, \mathbf{id}(v))$, where $N_T(v)$ is the set of neighbor tags of v, $\mathbf{id}(v)$ is the unique identifier of v and $|.|$ is the size of a set. The value of v is written in the memory of all neighbor tags of v. The readers read the memories of their neighbor tags after waiting for a specific period of time. A reader v is able then to decide whether it is an owner of a tag t by checking if it has the maximum value $\mathcal{W}(v)$ among all the readers that wrote in the memory of t. The comparison between the readers values is done in a lexicographical order. That is, the value of reader v_i is larger than the value of reader v_j, denoted as $\mathcal{W}(v_i) \succ \mathcal{W}(v_j)$, if v_i has more neighbor tags (i.e., $|N_T(v_i)|$). In case both readers have the same number of neighbor tags, then $\mathcal{W}(v_i) \succ \mathcal{W}(v_j)$ if the numerical values of the identifier of v_i is larger. An illustration of algorithm `RRE` is given in Figure 2.2(a). The non-redundant readers in the given example are v_2 and v_3. Reader v_2 owns tag t_1, t_2, t_3 and t_4. Reader v_3 owns tag t_5, whereas v_1 is a *redundant* reader.

A pseudocode of `RRE` executed at reader v is given in Algorithm 2.1. The notations used in Algorithm 2.1 are used to describe all reader-tag RFC algorithms. Procedure `write`(v,t, $\mathcal{W}(v)$) in line 3 indicates that reader v writes $\mathcal{W}(v)$ in the memory of a tag t. The for-loop in lines (2-3) represents a write phase of a write/read round. The `read`(v,t) procedure indicates that reader v reads the memory of a tag t, denoted $\mathcal{M}(t)$. The for-loop in lines (4-7) represents the read phase of the write/read round. The waiting period spent by readers after a write phase is omitted from the algorithm pseudocode to simplify its description.

Contributions

The performance of an RFC algorithm is evaluated by the number of non-redundant readers it generates, and by the number of the write/read rounds it executes. The number of write/read rounds gives an indication of the algorithm execution time. An algorithm with low execution time is more suitable for scenarios where the reader networks topology is subject to changes (e.g., due to mobility of readers and tags), or ad-hoc networks.



(a) Initial network. (b) write round.

(c) read round.

Figure 2.2: Example illustrating a write/read round in algorithm RRE. The pairs beside the readers are the values of the readers. Assume that $\mathbf{id}(v_3) \succ \mathbf{id}(v_2) \succ \mathbf{id}(v_1)$ according to some order, where $\mathbf{id}(v)$ is the identifier of v. For simplicity, assume that $\mathbf{id}(v) = v$. The sets of pairs beside the tags in (b) are the memory contents of the tags after the completion of a write round. The pair in bold is the maximum value written in the tag. For example, $(4, v_2)$ is the maximum value written in t_1 since v_2 has 4 neighbor tags while v_1 has only 2 neighbor tags. Readers v_2 and v_3 are the non-redundant readers selected by the algorithm. Reader v_2 owns t_1, t_2, t_3 and t_4, whereas v_3 owns t_5. The ownership relationships are represented by directed arrows in (c).

Algorithm 2.1 RRE at reader v

1: $\mathcal{W}(v) \leftarrow (|N_T(v)|, \mathbf{id}(v))$; where $N_T(v)$: set of neighbor tags of v, $\mathbf{id}(v)$: identifier of v.
2: **for** each $t \in N_T(v)$ **do**
3: `write`(v,t, $\mathcal{W}(v)$)
4: **for** each $t \in NT(v)$ **do**
5: `read`(v, t)
6: **if** $\mathcal{W}(v)$ is maximum in $\mathcal{M}(t)$ **then**
7: v owns t
8: **if** v does not own any tag **then**
9: v is redundant

This chapter introduces a number of RFC algorithms. The algorithms introduced in this chapter are decentralized. That is, they do not depend on any centralized or distinct node. The algorithms introduced in this chapter are:

1. Three randomized algorithms, called `RANDOM`, `RANDOM`$^+$ and `MAX-MIN`. Algorithm `RANDOM` runs in a single write/read round. `RANDOM`$^+$ is an extension of `RANDOM` that runs in a constant number of *iterations*; each of which consists of a single write/read round. `MAX-MIN` is a variant of `RANDOM`$^+$ that runs in two iterations. Each iteration of `MAX-MIN` consists of a single write/read round. More details are given in Section 2.5 and Section 2.6.

2. A deterministic algorithm, called `GDE` (*Greedy Decentralized Elimination*). The algorithm runs in at most $|\mathcal{R}|$ iterations, where $\mathcal{R}$ is the set of readers. Each iteration consists of two write/read rounds. More details are given in Section 2.7.

3. A deterministic algorithm, called `LIMITED-GDE`, which introduces an improvement on the number of write/read rounds of `GDE`. Algorithm `LIMITED-GDE` limits the number of iterations in `GDE` to a constant value. More details are given in Section 2.8.

Chapter Organization

More motivations for studying the RFC problem are given in Section 2.2. Section 2.3 formalizes the problem. Section 2.4 gives a survery of existing RFC algorithms. Algorithms `RANDOM` and `RANDOM`$^+$ are introduced in Section 2.5. `MAX-MIN`, `GDE`, and `LIMITED-GDE` are introduced in Section 2.6, Section 2.7 and Section 2.8 respectively. The correctness and worst case analysis of these algorithms accompany their descriptions. Section 2.9 concludes the chapter.

2.2 Motivations

This section discusses scenarios that lead to large scale RFID reader networks. Some issues caused by these networks are discussed afterward. Generally, these issues are: 1) *energy optimization*, 2) *readers and data redundancies*, and 3) *reliability of readings*. All of these issues can be solved using RFC algorithms.

2.2.1 Sources of Large Scale RFID Reader Networks

The Internet of Things (`IoT`): due to the low cost of RFID tags, it is foreseen that *things* equipped with RFID tags will be available in high quantities. Covering the

`IoT` things requires large-scale RFID reader networks. `IoT` is also calling for a complete integration between the RFID and wireless sensor networks in order to achieve the benefits of both. Such integration is called wireless RFID/sensor networks (see [LBNS08] for more details). This integration is leading toward higher availability of large scale RFID reader networks, especially as sensor networks and RFID are reaching a high level of maturity.

Large supply chains: advances in hardware manufacturing led to a significant decrease in the cost of RFID tags. This led, as a result, to wide acceptance and deployment of the RFID technology, especially among organizations with large supply chains. RFID received a boost when WalMart and the US Department of Defense decided to use RFID to manage their assets [HPP07]. Many other large organizations, such as Metro, Toyota, and others, showed interest in the RFID technology. The main reason behind this interest is that RFID allows these organizations to efficiently manage supply chains at a micro-scale level (i.e., each product is tracked and managed separately), and this creates financial benefits for the organizations. Thus, RFID may be used in large supply chains that consist of thousands or millions of to-be-covered products, which as a result creates scenarios where large scale RFID reader networks are needed. The tracking and monitoring of these organizations products generate enormous amounts of data. For instance, it is estimated that an organization such as WalMart generates in three days what is equivalent to the content of the US Congress Library if RFID is employed [Pal04]. This large amount of data causes an overhead on the networks that transmit this data, and complicates the processes of storing, mining, and analyzing the collected data [AH13, DOL07].

Ad-hoc placement of readers: a central problem in RFID reader networks is the organization of readers in specific fixed positions such that every tag in a given area is covered and such that certain performance qualities are met (e.g., minimum amount of readers, energy conservation, load balancing etc ...). A number of algorithms were proposed for this problem such as [AAH11a] and [WNR07]. The main disadvantage of these algorithms is their dependence on previous knowledge of the tags positions and/or the area dimensions. Another disadvantage is the high cost of readers installation. Ad-hoc placement of RFID readers is another approach that assumes that readers are randomly placed in the given area such that every tag is covered. A similar approach is followed

in the field of wireless sensor networks. *Crowd-sensing* and *vehicles clouds* depend on a similar approach as well [Sto12]. The approach of readers ad-hoc placement is motivated by the current low cost, low size and ubiquity of RFID readers. The disadvantage of this approach is that it may lead to unorganized and unoptimized reader networks.

Low interrogation range of passive RFID tags: the most basic and most commonly used tags are passive RFID tags. Passive RFID tags suffer from a low interrogation range, which is typically between 5 to 20 meters depending on the manufacturing features of the tag. The low reading range of passive tags causes an increase in the density and scale of reader networks. As a result, the levels of readers and data redundancy of reader networks are increased as well.

2.2.2 Challenges in Large Scale RFID Systems

A number of issues occur from using large scale RFID reader networks. The most important of these issues are *data redundancy* and *readers redundancy*, which were defined in Section 2.1. Data redundancy is reduced by tag reporting algorithms, and readers redundancy is reduced by RFID redundant readers elimination algorithms. An RFC algorithm is an algorithm that solves the tag reporting problem and the RFID redundant readers elimination problem and eliminate both types of redundancies.

Unreliability of readings: redundancy is not always a curse in RFID reader networks [VD08] as it can be used to increase the reliability of readings, which is a major issue in RFID systems [DOL07]. To overcome this weakness, each tag is assigned to exactly κ owner readers (for $\kappa \geq 1$) if possible. These κ readers are responsible for reporting the readings of the tag. This approach can be seen as a *control of the redundancy* in the reader network. This is similar to the κ-coverage problem in wireless sensors networks [Wan11]. As an example, assume that the network of Figure 2.3(a) is a subnetwork of a much larger reader network. A pair of readers (e.g., readers v_2 and v_4) can be assigned as owners of the tag t. Thus, the probability of failing to read t is decreased. A solution of the *unreliability of readings* problem would require a slight modification on the definition of the RFC problem such that each tag is owned by exactly κ readers (assuming that it is possible to be covered by κ or more readers). All RFC algorithms introduced in this thesis can be easily modified to solve this version of the problem.

RFC algorithms can be used in other applications as well. One application, inspired

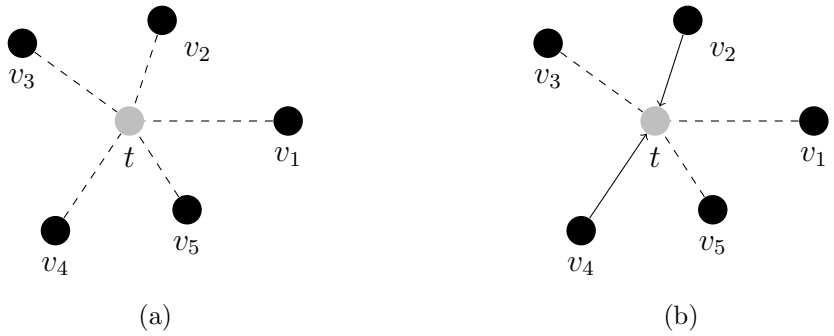


Figure 2.3: Sample reader network illustrating readers redundancy. A reader is linked to a tag by a dashed line if it covers it. A directed arrow between a reader and a tag represent an ownership relationship.

by the field of wireless sensor networks, is the grouping of the readers into subsets of readers $\{R_1, \ldots, R_k\}$. A subnetwork induced by a group R_i guarantees the coverage of all tags at a certain period of time. The readers subsets alternate in covering the network (that is, one subset of readers serves for coverage while all others are off). This technique, which is commonly used in wireless sensor networks, increases the network lifetime and fault tolerance of the network [Wan11].

2.3 Problem Formulation

This section defines the main terms used in this chapter, and gives a mathematical model of reader networks - according to reader-tag communication. A *reader network* consists of a set of readers $\mathcal{R}$ and a set of tags $\mathcal{T}$. A reader v is said to *cover* a tag t if v is able to read the memory content of t, and write the memory of t if t is *writable*. The memory content of a tag t is denoted $\mathcal{M}(t)$. These coverage relationships is modeled as a bipartite graph $\mathcal{G}_B = (\mathcal{R}, \mathcal{T}, E_s)$, where E_s is a set of *sensing edges* (or, *coverage relationships*).

The set of *neighbor tags* of a reader v, denoted $N_T(v)$, is defined as the set of tags covered by v (that is, $N_T(v) = \{t \mid (v,t) \in E_s\}$). The set of *neighbor readers* of a tag t, denoted $N_R(t)$, is defined as the set of readers that cover t (that is, $N_R(t) = \{v \mid (v,t) \in E_s\}$). A reader v is said to be a *coverage neighbor* (for short, **cov. neigh.**) of a reader u if both readers cover at least one common tag t (that is, if $N_T(v) \cap N_T(u) \neq \emptyset$). The set of all coverage neighbors of a reader v is denoted $N_{R_s}(v)$. In this chapter, the term

coverage neighbor reader may be interchangeably used with *neighbor reader* [1].

Assumptions: this chapter considers two models for reader networks that use reader-tag communication; RT-BASIC and RT-DISK. In the RT-BASIC model, an edge $(v,t) \in E_s$ is set arbitrarily between any reader v and tag t. The RT-DISK model sets an edge (v,t) in E_s if the Euclidean distance between v and t, denoted $d(v,t)$, is within a constant distance r_i (called the *interrogation range*). Note that r_i is common for all readers and tags. Both models share the following assumptions:

1. There are no communication links between the readers. This assumption will be relaxed in Chapter 4, which studies the reader-reader RFID coverage problem.
2. The readers of $\mathcal{R}$ are assumed to be able to write in the memory contents of their neighbor tags. However, a tag cannot transmit any message to a reader v without first being interrogated by v.
3. The readers have no previous knowledge of the reader network topology, and have no knowledge of their positions or their neighbor tags positions.
4. Each reader v is assumed to have a unique comparable identifier, denoted $\mathbf{id}(v)$. Each tag t is assumed to have a unique identifier, denoted $\mathbf{id}(t)$. Such feature can be guaranteed by the EPCglobal standards in [tag11].
5. There are no centralized nodes.

Problem Definition: a reader v *owns* a tag t if v is delegated to read tag t. The set of tags owned by a reader v is denoted as $S(v)$ (i.e., *slaves* of v). The readers that owns at least one tag are called *non-redundant readers* and denoted $\mathcal{C}$ (that is, $\mathcal{C} = \{v \mid S(v) \neq \emptyset\}$). If every tag is covered by at least one reader in $\mathcal{C}$, then $\mathcal{C}$ is called a *cover*.

The RFC problem consists of the following subproblems:

1. **Tag reporting**: for each reader v, find a set $S(v) \subseteq N_T(v)$ such that $\bigcup_{v \in R} S(v) = \mathcal{T}$ and $\bigcap_{v \in \mathcal{R}} S(v) = \emptyset$ (pairwise disjoint). That is, each tag has exactly one owner.

[1] Chapter 4 introduces the term *communication neighbor readers* which is given to any pair of readers that are able to communicate with each other using reader-reader communication.

Table 2.1: Terminologies used in Chapter 2

Term	**Definition**
$\mathcal{R}$	The set of readers
$\mathcal{T}$	The set of tags
$N_T(v)$	The set of neighbor tags of reader v
$N_R(t)$	The set of neighbor readers of tag t
$N_{R_s}(v)$	The set of coverage neighbor readers (or, **cov. neigh.**) of reader v
$S(v)$	The set of tags owned by reader v
$\mathbf{id}(v)$	The unique identifier of reader v
$\mathcal{C}$	The set of readers which owns at least one tag (non-redundant readers)
$\mathcal{M}(t)$	The memory content of tag t
r_i	The interrogation range of the readers

2. **Redundant readers elimination**: minimize the size of the set $\mathcal{C}$. That is, minimize the number of non-redundant readers in the network.

The combination of both objectives is called the *RFID coverage* problem (RFC problem). Minimizing the number of non-redundant readers turns the problem into an **NP Hard** problem, as the minimum disk coverage problem may be reduced from it in case the RT-DISK model is used [CRK+09]. A reduction to the minimum set cover problem is possible in case the RT-BASIC model is used [2].

Definition 2.3.1. *An algorithm* **A** *solves the the RFC problem if* **A** *solves* correctly *the* tag reporting *problem. This is because the requirement of eliminating all redundant readers turns the problem into an* **NP Hard** *problem.*

Lastly, the RFC problem may appear with additional constraints, such as multihop communication connectivity between readers, κ-coverage for improved fault-tolerance, handling faulty communication links, or achieving load balancing between readers. None of these constraints is considered in this chapter because:

1. There is a room for improvements in the unconstrained version of the RFC problem as will be shown in this chapter, and

2. Studying the problem without constraints provides a better understanding of it, which helps later in a better understanding of its constrained versions.

[2]The minimum set cover problem is defined in Section 2.4 as it is used extensively in the literature of RFC algorithms.

2.4 Related Work

This section gives a literature review of the RFC problem. RFC algorithms are categorized into *centralized* and *decentralized* algorithms. More details about these categories are given below.

2.4.1 Centralized algorithms

RFC centralized algorithms assume the existence of a programmable centralized node, which can be a dedicated server or an elected reader. Practically, even if a centralized dedicated server is available, it may not be possible to program this server for purposes other than what it is designed for. Other than that, the use of centralized nodes degrades the scalability of the network, increases its cost, and may lead to inefficient use of available resources. In RFC centralized algorithms, each reader is connected to the centralized node. Most centralized RFC algoritms do not specify how such connections are made. Each reader v sends its neighbor tags list $N_T(v)$ to the centralized node. The centralized node builds a complete view of the network, and executes a sequential algorithm that assigns to each reader the tags it owns.

Centralized algorithms reduce the RFC problem to the *unit-cost set cover* problem (or the set cover problem for short), defined as follows. Given a collection of sets $\mathcal{S} = \{S_1, \ldots, S_k\}$ and a universe of elements $U = \{e_1, \ldots, e_n\}$. Find a subcollection $\mathcal{S}' \subseteq \mathcal{S}$ such that $\bigcup_{S_i \in \mathcal{S}'} S_i = U$. A minimum sized set cover is called an *optimal set cover*. The problem of finding an optimal set cover is known to be **NP Hard** [LRSC01]. A solution of the set cover problem can be used to solve the RFC problem. It is sufficient to set U as $\mathcal{T}$, and set $\mathcal{S}$ as the collection of sets $\{N_T(v) \mid v \in \mathcal{R}\}$ (that is, the collection of tag neighbors sets for all readers).

The Centralized Greedy Approach: algorithm `GREEDY` [CRK+09] [YS11] is a centralized algorithm that uses the reduction given above. The algorithm is equivalent to a well-known greedy set cover algorithm, which is referred to in the following as the *standard greedy set cover algorithm*. This algorithm runs in iterations. The set (i.e., the reader) that covers the maximum number of not-yet covered elements (i.e., tags) is included in the solution set $\mathcal{C}$ in each iteration. This procedure continues until all the tags are covered. A reader included in the solution $\mathcal{C}$ owns all its neighbors tags that are not in $\mathcal{C}$ already (i.e., not-yet owned tags). This guarantees that every tag is owned

by exactly one reader. `GREEDY` is frequently used in this chapter given its important properties. For instance, `GREEDY` is known to have a $O(\log n)$ approximation ratio of the set cover problem, where n is the number of elements in the universe (or, the number of tags in this context) [3]. The pseudocode of `GREEDY` is given in Algorithm 2.2.

Algorithm 2.2 Algorithm `GREEDY`

1: $\mathcal{C} \leftarrow \emptyset$
2: $U \leftarrow \mathcal{T}$
3: **while** $U \neq \emptyset$ **do**
4: $v \leftarrow$ the reader that maximizes $|N_T(v) \cap U|$
5: $\mathcal{C} \leftarrow \{\mathcal{C} \cup v\}$
6: v owns all tags in $\{N_T(v) \cap U\}$ (that is, all tags that are not owned yet by any reader in $\mathcal{C}$).
7: $U = \{U \setminus N_T(v)\}$
8: **return** $\mathcal{C}$

`NTE` [AAH11b] is another centralized greedy algorithm. In each `NTE` iteration, a reader is included in the solution $\mathcal{C}$ if it is the reader with maximum ratio of *active* neighbor tags n_t (that is, the neighbor tags that are not owned yet at that iteration) to *active* **cov. neigh.** readers n_r (that is, the **cov. neigh.** readers that are not in $\mathcal{C}$ at that iteration).

The greedy approach is not the only approach to solve the set cover problem. However, the approximation ratio $O(\log n)$ of the *standard set cover greedy algorithm* can only be improved within a constant factor [LY94] [Fei98]. A survey of set cover problems is given in [CTF00]. Unfortunately, the surveyed algorithms are all sequential and cannot be implemented in a distributed manner without significantly affecting the execution time and the communication cost, which make them not practical for the RFC problem.

2.4.2 Decentralized algorithms

Decentralized RFC algorithms can be categorized into those that use *reader-tag communications* (reader-tag RFC algorithms), and those that use *reader-reader communications* (reader-reader RFC algorithms). Reader-tag RFC algorithms assume that the tags are writable (that is, a reader is able to write on the memory of its neighbor tags). Reader-reader RFC algorithms assume that a pair readers in proximity of each other

[3] An algorithm **A** is said to have a ρ-approximation ratio for a given problem **P** if it guarantees that any solution it outputs for $\mathcal{P}$ has a performance value within $\rho \times OPT$ in the worst case, where OPT is the optimal performance value of solving $\mathcal{P}$ using any algorithm.

can communicate with each other using wireless interfaces. The existing decentralized RFC algorithms are all based on reader-tag communications. Chapter 4 introduces new reader-reader RFC algorithms.

Carbunar et al. introduced in [CRK+09] algorithm `RRE`, which requires only one write/read round. Each reader v sets its value, denoted $\mathcal{W}(v)$, to $(|N_T(v), \mathbf{id}(v))$ in the memory of each neighbor tag t (i.e., $\mathcal{M}(t)$). The reader with maximum value written in $\mathcal{M}(t)$ owns it. A reader that own no tags is considered redundant. A more detailed description of `RRE` was given in Section 2.1 (see Figure 2.2). `RRE` guarantees that each tag t is owned by exactly one reader, and hence `RRE` is a correct RFC algorithm. This is because the reader that owns a tag t is the reader $v \in N_R(t)$ with maximum value $\mathcal{W}(v)$. There is necessarily one maximum value $\mathcal{W}(v)$ given that $\mathbf{id}(v)$ is unique for every reader v.

`LEO` [HCY07] (Layered Optimization Approach) is a decentralized algorithm based on a single write/read round. The algorithm uses a First-Write-First-Own approach. Each reader v writes its identifier in the memory of all its neighbor tags. A reader v owns a neighbor tag t if v is the first reader to write in its memory (i.e., in $\mathcal{M}(t)$). Similar to `RRE`, the readers that own no tags are redundant. The main issue in `LEO` is that it depends on the implementation of write/read rounds in the MAC (Multiple Access Control) layer. For instance, it depends on the order on which a reader contacts its neighbors tags, which can be significantly affected by the collisions. These important details are not considered by `LEO` in [HCY07]. Hsu et al. introduced in [HYC+12] an algorithm similar to `LEO`. The introduced algorithm is more sophisticated. The differences between the final results of `LEO` and the algorithm of [HYC+12] are not clear.

Single-round RFC vs. Multiple-round RFC: `RRE` and `LEO` execute a single write/read round. This property give these algorithms a practical advantage due to their shorter execution time and due to the difficulties of implementing multiple write/read rounds. The issue of these *single-round algorithms* is that they generate a large number of redundant readers. This issue is solved by introducing algorithms that execute more than one write/read round. Among these algorithms is `DRRE` [YYL08] (Density-based Redundant Readers Elimination). The algorithm states that the readers shall be compared by the number of their **cov. neigh.** readers. Each reader v uses a value $\mathcal{W}(v)$ set to $(|N_{R_s}(v)|, \mathbf{id}(v))$. A reader v shall execute an additional write/read round to build its **cov. neigh.** readers list $N_{R_s}(v)$. Therefore, algorithm `DRRE` requires two write/read

rounds in total.

Irfan and Yagoub introduced in [IY10] an algorithm similar to `RRE`. The weight $\mathcal{W}(v)$ of a reader v in [IY10] is calculated as follows:

$$\mathcal{W}(v) = \varepsilon \; w_t(v) + (1 - \varepsilon) \; w_r(v)$$

where $0 \leq \varepsilon \leq 1$, and:

$$\begin{aligned} w_t(v) &= \frac{|N_T(v)|}{\gamma \times \max\{|N_T(w)| \mid w \in \mathcal{R}\}} \qquad (2.1) \\ w_r(v) &= 1 - (\frac{|N_\zeta(v)|}{\max\{|N_\zeta(w)| \mid w \; in \mathcal{R}\}}) \end{aligned}$$

where γ is set to 3. A reader u is in $N_\zeta(v)$ if $d(v, u) \leq 2r_i$ (that is, the Euclidean distance between v and u is at most twice the interrogation range r_i). The algorithm of [IY10] assume that the positions of each reader is known. This is used to compute the set $N_\zeta(v)$ for each $v \in \mathcal{R}$. Moreover, the algorithm assumes the existence of a centralized node to compute the weight $\mathcal{W}(v)$ for each $v \in \mathcal{R}$. This is because the weight function $\mathcal{W}(v)$ depends on $\max\{|N_T(w)| \mid \forall \; w \in \mathcal{R}\}$ and $\max\{|N_\zeta(w)| \mid \forall \; w \; in \mathcal{R}\}$ which are the maximum values of $|N_T(v)|$ and $|N_\zeta(v)|$ for every $v \in \mathcal{R}$.

Executing RFC Algorithms in Sequence: Hsu et al. introduced in [HCY07] algorithm `LEO+RRE`. The algorithm consists of two write/read rounds. The algorithm `LEO+RRE` is an execution of of algorithm `LEO` [HCY07] followed by algorithm `RRE` [CRK+09]. In the first round, algorithm `LEO` [HCY07] is executed and generates a set of non-redundant readers $\mathcal{R}_1$. Algorithm `RRE` is executed over the set $\mathcal{R}_1$ instead of $\mathcal{R}$. Algorithm `LEO+RRE` is correct given that $\mathcal{R}_1$ is a cover (i.e., the readers of $\mathcal{R}_1$ covers every tag). The simulation experiments results of [HCY07] show that `LEO+RRE` outperforms `LEO` and `RRE`. The weakness of `LEO+RRE` is its dependency on `LEO`. The main advantage of `LEO+RRE` is the introduction of the novel approach of combining RFC algorithms (that is, executing a sequence of RFC algorithms to reduce the number of non-redundant readers). Alsalihy and Askar introduced in [AA13] an algorithm, called `LE`, that follows the same principle of `LEO+RRE`. Algorithm `LE` executes `LEO` in a first write/read round followed by an execution of the algorithm of Irfan and Yagoubi of [IY10] in the second iteration.

Two randomized algorithms are introduced in [DSS+08] with the objective of bal-

ancing the coverage load between non-redundant readers. The first algorithm runs in two write/read rounds. The second algorithm runs in multiple write/read rounds. The performance of the solution of the second algorithm is improved as the number of executed write/read rounds increases. Algorithm RANDOM$^+$, introduced in Section 2.5, uses a similar approach to improve its quality.

Handling Topology Changes: Dhas et al. introduced in [DMBG10] an interesting load balancing RFC algorithm that takes into consideration tags mobility. Each reader writes a timestamp in its neighbor tags. The reader writes other weights as well. The timestamp is rewritten continuously. The readers use their neighbor tags timestamps to change the ownership of a tag. Some factors that lead to ownership changes in a reader network are failure of a non-redundant readers or tag mobility (that is, a tag changed its location such that it is not covered by its owner anymore).

2.5 Algorithms RANDOM and RANDOM$^+$

This section introduces algorithms RANDOM and RANDOM$^+$, which are randomized decentralized RFC algorithms that use reader-tag communications. RANDOM consists of a single write/read round, whereas RANDOM$^+$ consists of ψ iterations, where ψ is a constant greater than 1. Each iteration of RANDOM$^+$ is an execution of algorithm RANDOM over the set of non-redundant readers generated in the previous iteration. The algorithms are presented first as a sequential algorithm, called SEQ. This sequential version of RANDOM and RANDOM$^+$ simplifies their description and analysis. An efficient decentralized implementation of SEQ is given afterward.

2.5.1 A Sequential Version of RANDOM and RANDOM$^+$

Decentralized RFC algorithms, such RRE and DRRE, can be described as the following sequential algorithm. First, the readers $\mathcal{R}$ are sorted by a *ranking function* $\pi(\mathcal{R})$, where the output of $\pi(\mathcal{R})$ is the set $\{v_{(1)}, \ldots, v_{(k)}\}$ where $v_{(i)}$ is the reader with the i^{th} ranking in $\mathcal{R}$ according to $\pi(\mathcal{R})$. For instance, $\pi(\mathcal{R})$ may order the readers according to their weights. That is, a reader $v_{(i)} \succ v_{(j)}$ if $\mathcal{W}(v_{(i)}) \succ \mathcal{W}(v_{(j)})$. As an example, a reader $v_{(i)} \succ v_{(j)}$ if $(|N_T(v_i)|, \mathbf{id}(v_i)) \succ (|N_T(v_j)|, \mathbf{id}(v_j))$ according to the ranking function $\pi(\mathcal{R})$ used by RRE. The sequential algorithm, called SEQ, passes by the readers in order

from the highest ranked to the lowest. A reader is included in the solution $\mathcal{C}$ if it covers at least one not-yet covered tag. The algorithm continues until $\mathcal{C}$ is a cover. A reader not included in $\mathcal{C}$ is redundant. The pseudocode of `SEQ` is given in Algorithm 2.3.

Algorithm 2.3 Algorithm `SEQ`

1: $\mathcal{C} \leftarrow \emptyset$
2: $U \leftarrow \mathcal{T}$
3: sort $\mathcal{R}$ according to a ranking function $\pi(\mathcal{R})$, as $v_{(1)}, \ldots, v_{(k)}$
4: **for** $i : 1$ to k **do**
5: **if** $\{N_T(v_i) \cap U\} \neq \emptyset$ (that is, if $v_{(i)}$ covers at least one not-yet covered tag) **then**
6: $\mathcal{C} \leftarrow \{\mathcal{C} \cup v_{(i)}\}$
7: $U \leftarrow \{U \setminus N_T(v_i)\}$
8: **return** $\mathcal{C}$

Remark 2.5.1. *There is a strong relationship between* ***SEQ*** *and* ***GREEDY****. Assume that the readers weights in* ***SEQ*** *is the same weight used in* ***GREEDY****. Both algorithms are greedy and terminates as soon as a cover $\mathcal{C}$ is created. However,* ***SEQ*** *sorts the readers only once at the beginning of the algorithm, whereas* ***GREEDY*** *re-sorts the readers at each iteration. This re-sorting is done to find the maximum reader in each iteration (i.e., the reader with maximum updated value after eliminating all readers already included in $\mathcal{C}$). Such continuous re-sorting performed by* ***GREEDY*** *decreases the size of the cover $\mathcal{C}$. The problem of this approach is that it is expensive to implement by a decentralized algorithm. The other extreme of this approach is to sort the readers only once, as* ***SEQ*** *does. This does not guarantee that at every iteration the reader with maximum value is chosen - considering all the readers already included in $\mathcal{C}$. An approach that comes in between both extremes is to select readers randomly at each iteration. This approach, as will be shown next, is easy to implement by a decentralized algorithm and it generates covers with smaller sizes in several practical environments. This approach is implemented by algorithm* ***RANDOM****, described in the following.*

Algorithm `RANDOM`

The sequential version of algorithm `RANDOM` assumes that the ranking function $\pi(\mathcal{R})$ shuffles the readers according to a uniform probability. That is, a reader $v_{(i)}$ is ranked as the i^{th} reader with a uniform probability $\frac{1}{|\mathcal{R}|}$. Therefore, algorithm `RANDOM` selects, in each iteration, a random reader v with uniform probability and inserts it into $\mathcal{C}$ if it covers at least one not-yet covered tag. This procedure is repeated until $\mathcal{C}$ is a cover.

An upper bound of the expected number of non-redundant readers generated by `RANDOM` is given in the following. It is sufficient to find an upper bound of the expected number of the *loop iterations* executed by algorithm `SEQ`[4]. Assume for simplicity that a tag t is covered by a reader v with a uniform probability p (that is, the edge (v,t) exists with a uniform probability p). Therefore, `RANDOM` generates a cover $\mathcal{C}$ with expected size $p^{-1}(c+1)\ln|\mathcal{T}|$ with high probability $1-1/|T|^c$ for $c \geq 1$. This is elaborated in the following.

Let $\mathsf{E}(t,m)$ be the event that a certain tag t is *not* covered after m loop iterations. Then the probability of $\mathsf{E}(t,m)$ is [5]:

$$\begin{aligned} \mathbf{P}[\mathsf{E}(t,m)] &= (1 - \sum_{r_i : t \in N_T(r_i)} 1/|\mathcal{R}|)^m \\ &= (1 - \frac{p|\mathcal{R}|}{|\mathcal{R}|})^m \\ &= (1-p)^m \end{aligned} \tag{2.2}$$

The probability that at least one of the tags is not covered after m iterations is:

$$\sum_{t \in \mathcal{T}} \mathbf{P}[\mathsf{E}(t,m)] = |\mathcal{T}|(1-p)^m \tag{2.3}$$

This is equivalent to the expected number of uncovered tags after m iterations. The algorithm terminates with a high probability greater or equal to $1/|T|^c$ for a constant

[4]The iterations of algorithm `SEQ` are called *loop iterations* to differentiate them from the iterations of the decentralized algorithms which consist of write/read rounds.

[5]Note that the result of Eq. 2.2 can be directly deduced using the law of binomial trials, since a tag t is covered by a reader v with probability p.

$c \geq 1$ if:

$$\begin{aligned} |\mathcal{T}|(1-p)^m &\leq \frac{1}{|\mathcal{T}|^c} \\ (1-p)^m &\leq \frac{1}{|\mathcal{T}|^{c+1}} \\ e^{pm} &\geq |\mathcal{T}|^{c+1} \\ m &\geq p^{-1}(c+1)\ln|\mathcal{T}| \end{aligned} \tag{2.4}$$

since $1+x \leq e^x$. The previous analysis suggests an upper bound of the expected number of loop iterations of `RANDOM` is $m = p^{-1}(c+1)\ln|\mathcal{T}|$, with probability $1/|\mathcal{T}|^c$ for $c \geq 1$. This expected value is referred to as the *random expectation of* $\mathcal{C}$, denoted $\mathbf{E}_{rnd}[|\mathcal{C}|]$.

Algorithm `RANDOM+`

The objective of some reader-tag RFC algorithms is to apply heuristics that reduce the random expectation of $\mathcal{C}$ (i.e., $\mathbf{E}_{rnd}[|\mathcal{C}|]$). Essentially, these heuristics apply different ranking functions $\pi(\mathcal{R})$ on `SEQ`. For instance, `RRE` orders the readers according to the number of their neighbor tags, whereas `DRRE` orders the readers according to the number of their **cov. neigh.** readers. Algorithm `RANDOM+` uses a different approach to reduce $\mathbf{E}_{rnd}[|\mathcal{C}|]$. This approach, described below, can be seen as a *local search heuristic*.

Algorithm `RANDOM+` runs in ψ iterations, for $\psi \geq 1$. In each iteration i, for $1 \leq i \leq \psi$, a new cover set $\mathcal{C}_i$ is generated. The cover set $\mathcal{C}_i$ is generated by running algorithm `RANDOM` over the set $\mathcal{C}_{i-1}$, where $\mathcal{C}_0 = \mathcal{R}$. As a result, $\mathcal{C}_1$ is equivalent to the set of non-redundant readers generated by `RANDOM`. An execution of a `RANDOM` in a `RANDOM+` iteration is called a *shuffling* of $\mathcal{R}$. This is because in each `RANDOM+` iteration the readers are rearranged in a different random order. The shuffling procedure decreases the number of non-redundant readers generated by `RANDOM`. This improvement can be significant in practice as shown in the simulation experiments results of Chapter 3.

The intuition behind the shuffling of the readers in `RANDOM+` is the following. Consider an execution of algorithm `RANDOM` that generates the set $\mathcal{C}$ with size m. According to the sequential version of `RANDOM` (line 5 in Algorithm 2.3), a reader is included in $\mathcal{C}$ only if it covers at least one tag that is not yet covered. This means that the last reader to

be included in $\mathcal{C}$, denoted $v_{(m')}$ and $m' \geq m$ [6], covers at least one tag, denoted $t_{(n)}$, that is not covered by any other reader in $\mathcal{C}$. On the other hand, it is possible that a reader $v' \in \{\mathcal{C} \setminus v_{(m')}\}$ have all its tags covered by other readers in $\{\mathcal{C} \setminus v'\}$. Therefore, v' may be considered redundant if the readers are shuffled in a new RANDOM$^+$ iteration. The shuffling procedure reduces the size of $\mathcal{C}$ by at least one if 1) there is a reader v' that have all its tag neighbors covered by other readers, and 2) v' is ranked by the ranking function as the last ranked reader after the shuffling procedure.

2.5.2 A Decentralized Implementation of RANDOM and RANDOM$^+$

The next step is to implement RANDOM and RANDOM$^+$ on an RFID reader network in a decentralized manner with minimum amount of write/read rounds. To emulate the random selections of readers in RANDOM, each reader v draws a random number $\alpha(v)$ from a uniform distribution. Each reader v writes $(\alpha(v), \mathbf{id}(v))$ in $\mathcal{M}(t)$ for each neighbor tag t. The unique identifier $\mathbf{id}(v)$ guarantees the comparability of the readers weights. Each tag is owned by the neighbor reader v with the maximum pair $(\alpha(v), \mathbf{id}(v))$. This gives an emulation of SEQ with the ranking function $\pi(\mathcal{R})$ of RANDOM. This is a decentralized implementation of RANDOM in RFID reader networks that is executed in a single write/read round.

To implement the i^{th} iteration of RANDOM$^+$ for $i \geq 1$, every reader v that finds itself non-redundant in the previous iteration $i-1$ draws a new random number $\alpha_i(v)$ from a uniform distribution and creates the pair $(\alpha_i(v), \mathbf{id}(v))$. A tag t is owned by the reader v with maximum $(\alpha_i(v), \mathbf{id}(v))$, where $v \in N_R(t)$. Note that a tag may have a new owner in each iteration of RANDOM$^+$. The last owner of a tag t is its actual owner. The pseudocode of RANDOM and RANDOM$^+$ are given in Algorithm 2.4 and Algorithm 2.5. An illustrative example of RANDOM is given in Figure 2.4.

2.6 MAX-MIN: a variation of RANDOM$^+$

A different version of RANDOM$^+$, called MAX-MIN, is introduced in this section. MAX-MIN consists of two iterations. In the first iteration, each tag is owned by the reader v with the *maximum* value $\mathcal{W}(v) = (\alpha(v), \mathbf{id}(v))$. The second iteration is executed by the non-redundant readers that survived the first iteration. In the second iteration, each tag is

[6] $m' \geq m$ because some readers in $(v_{(1)}, \ldots, v_{(m')})$ are not necessarily included in $\mathcal{C}$.

Algorithm 2.4 Algorithm `RANDOM` at reader v

1: $\mathcal{W}(v) \leftarrow (\alpha(v), \mathbf{id}(v))$;
2: **for** each $t \in N_T(v)$ **do**
3: `write`(v,t, $\mathcal{W}(v)$);
4: **for** each $t \in NT(v)$ **do**
5: `read`(v, t);
6: **if** $\mathcal{W}(v)$ is maximum in $\mathcal{M}(t)$ **then**
7: v owns t
8: **if** v does not own any tag **then**
9: v is redundant;

Algorithm 2.5 Algorithm `RANDOM`$^+$ (ψ) at reader v

1: **for** i : 1 to ψ **do**
2: Run algorithm `RANDOM` with $\mathcal{W}(v) = (\alpha_i(v), \mathbf{id}(v))$
3: If v is redundant, then v terminates the algorithm.
4: Reader v is non-redundant if it is non-redundant readers in all ψ iterations.

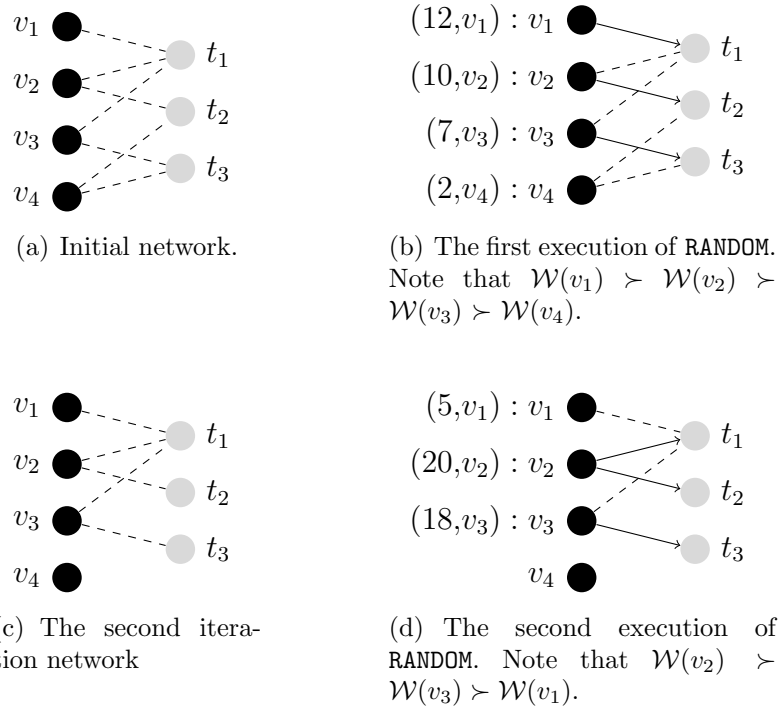
owned by the reader with the *minimum* value $\mathcal{W}(v) = (\alpha(v), \mathbf{id}(v))$. The value $\alpha(v)$ remains the same in both iterations.

The intuition behind `MAX-MIN` is the following. The expected number of uncovered tags in Eq. 2.3 follows a decreasing geometric function. Eq. 2.3 suggests that the probability of finding uncovered tags becomes smaller as m increases. The second round of algorithm `MAX-MIN` acts as an adverser to this geometric behavior as it attempts to visit the readers in an order opposite to the order used in the first round.

Example: an illustrative example of `MAX-MIN` is given in this section. `MAX-MIN` is executed over the network given in Figure 2.4(a). The output of the first iteration is the same as that in Figure 2.4(b). In the second iteration, the readers keep their weights of the first iteration. That is, the weights of the non-redundant readers in the second iteration are $(12, v_1)$, $(10, v_2)$ and $(7, v_3)$. Reader v_3 owns tags t_1 and t_3 since its weight, $(7, v_3)$, is the minimum weight written in these tags. Reader v_2 owns t_2.

2.7 `GDE`: Greedy Decentralized Elimination

The execution of multiple shuffling procedures by algorithm `RANDOM`$^+$ reduces the random expectation of $\mathcal{C}$ (i.e., $\mathbf{E}_{rnd}[|\mathcal{C}|]$) using randomization. On the other hand, `GREEDY` always



(a) Initial network.

(b) The first execution of RANDOM. Note that $\mathcal{W}(v_1) \succ \mathcal{W}(v_2) \succ \mathcal{W}(v_3) \succ \mathcal{W}(v_4)$.

(c) The second iteration network

(d) The second execution of RANDOM. Note that $\mathcal{W}(v_2) \succ \mathcal{W}(v_3) \succ \mathcal{W}(v_1)$.

Figure 2.4: Example of RANDOM$^+$ with $\psi = 2$. The pairs beside the readers are the weights of the readers. Assume that $\mathbf{id}(v_4) \succ \mathbf{id}(v_3) \succ \mathbf{id}(v_2) \succ \mathbf{id}(v_1)$. For simplicity, it is assumed that $\mathbf{id}(v) = v$. In the first iteration, the non-redundant readers are v_1, v_2 and v_3. These non-redundant readers execute the second iterations with new values. The readers that are non-redundant in the second iteration - and in overall - are v_2 and v_3.

generates non-redundant readers sets of expected size that is less or equal to $\mathbf{E}_{rnd}[|\mathcal{C}|]$. This is because GREEDY inserts into the cover $\mathcal{C}$ the reader that covers the maximum of not-yet covered tags in every iteration, and thus it outperforms RANDOM on average. Algorithm GREEDY orders the readers according to the number of their *active neighbor tags*, defined as the number of neighbor tags that are not owned yet. This order may change every time a new reader is included into the cover $\mathcal{C}$, or basically in every iteration of GREEDY. The continuous change of the readers order guarantees that the size of the cover set $\mathcal{C}$ generated by GREEDY is less or equal to $\mathbf{E}_{rnd}[|\mathcal{C}|]$ on average. A formal proof can be obtained using the conditional probabilities method (see [MR95, You95] for more details).

The main issue of GREEDY is that it requires a centralized node. Furthermore, the naive decentralized implementation of GREEDY suffers from a high communication cost, since it requires the election of a leader and it requires that a maximum reader is repeatedly

Table 2.2: Terminologies introduced in Section 2.7

Term	**Definition**
$\mathcal{T}^a$	The set of active tags
$\mathcal{T}^a(v)$	The set of active neighbor tags of reader v
$\mathcal{R}^a(v)$	The set of active neighbor readers of reader v

found until a cover is formed. The objective of algorithm `GDE` is to overcome this issue by introducing an efficient decentralized implementation of `GREEDY`.

The following terms are used in this section, and are summarized in Table 2.2.

Definition 2.7.1. *a reader* deactivates *itself if it terminates the execution of the algorithm. Readers that do not deactivate themselves are called* active readers.

Definition 2.7.2. *a tag is* deactivated *if it terminates the execution of the algorithm. The set of* active tags *is denoted $\mathcal{T}^a$. The set of active neighbor tags of a reader v is denoted $\mathcal{T}^a(v)$.*

Definition 2.7.3. *at an iteration i of `GDE`, a pair of active readers v and u are called* active neighbor readers *if they share the coverage of at least one active tag t at iteration i. The set of* active neighbor readers *of reader v is denoted $\mathcal{R}^a(v)$. That is, $\mathcal{R}^a(v) = \{v' \mid \{\mathcal{T}^a(v) \cap \mathcal{T}^a(v')\} \neq \emptyset\}$.*

Algorithm Description

Algorithm `GDE` runs in iterations. At least one reader and one tag are deactivated in each iteration. Each iteration is executed by the set of active readers and active tags at that iteration. A tag is deactivated only after being owned by a neighbor reader. The algorithm terminates when all readers and tags are deactivated.

In each iteration, every active reader v writes its weight $\mathcal{W}(v)$ in the memory $\mathcal{M}(t)$ for each active neighbor tag $t \in \mathcal{T}^a(v)$. The weight $\mathcal{W}(v)$ of a reader v is set to $(|\mathcal{T}^a(v)|, \mathbf{id}(v))$ in each iteration v is active in.

Definition 2.7.4 (Local maximum)**.** *A reader v is called a* local maximum *at iteration i if it is the reader with maximum $\mathcal{W}(v)$ among all its active neighbor readers at the same iteration. That is, v is a local maximum if $\mathcal{W}(v) \succ \mathcal{W}(v')$ for each $v' \in \mathcal{R}^a(v)$ in iteration i.*

A reader that recognizes that it is a local maximum reader owns all its active neighbor tags $\mathcal{T}^a(v)$. A tag is deactivated by its owner once it is owned. A reader v' that has no active tags deactivates itself. This may occur either because v' owned and deactivated all its active neighbor tags $\mathcal{T}^a(v')$, or because all the neighbor tags $N_T(v')$ of v' are owned by other readers. As a result, a tag is owned by exactly one reader. This is because the tags are owned by local maximum readers and are deactivated as soon as they are owned. Note that according to the definition of local maximum readers, it is not possible for a pair of local maximum readers to own the same active tag t at the same iteration.

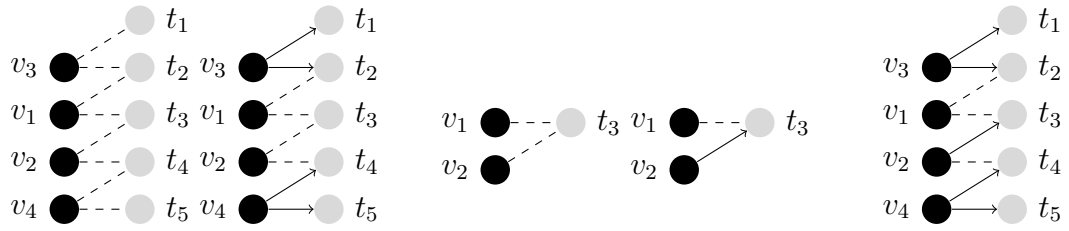
The implementation of the rules given above requires two write/read rounds per each iteration. Each active reader v writes its weight $\mathcal{W}(v)$ in $\mathcal{M}(t)$ for every active neighbor tag t in $\mathcal{T}^a(v)$. Then, an active reader v checks if it has the maximum weight $\mathcal{W}(v)$ at each tag $t \in \mathcal{T}^a(v)$. If this is the case, then v is a local maximum reader. A local maximum reader owns and deactivates its active neighbor tags by writing a special flag in their memories. The pseudocode of algorithm `GDE` executed at reader v is given in Algorithm 2.6.

Algorithm 2.6 Algorithm `GDE` at reader v

```
1: T^a(v) ← N_T(v);
2: while reader v is active do
3:    W(v) ← (|T^a(v)|, id(v))
4:    for each t ∈ T^a(v) do
5:       write(v,t,W(v))
6:    for each t ∈ T^a(v) do
7:       read(v, t)
8:    if v has the maximum W(v) in each t ∈ T^a(v) (i.e. local maximum) then
9:       v owns each t ∈ T^a(v)
10:      v deactivates each t ∈ T^a(v)
11:   update T^a(v) by eliminating all deactivated tags
12:   if T^a(v) = ∅ then
13:      v is deactivated
14: if v owns no tag then
15:   v is redundant and inactive
```

Theoretical Analysis

Theorem 2.7.5 and Theorem 2.7.6, given below, prove the correctness of `GDE`. Theorem 2.7.5 proves that the non-redundant readers generated by `GDE` are exactly the same



(a) First iteration (b) Second iteration (c) Final result

Figure 2.5: Example of GDE. Initially the network consists of four readers and four tags. The set $\mathcal{R} = \{v_4, v_3, v_2, v_1\}$ such that $\mathbf{id}(v_4) \succ \mathbf{id}(v_3) \succ \mathbf{id}(v_2) \succ \mathbf{id}(v_1)$. In the first iteration, the set of local maximum readers is $\{v_4, v_3\}$. Reader v_4 deactivates all its neighbor tags, which are t_4 and t_5. Reader v_3 deactivates also all its neighbor tags, which are t_1 and t_2. At this point, the readers v_4 and v_3 find that all their neighbor tags are deactivated. As a result, reader v_4 and v_3 terminate the execution of the algorithm. Reader v_1 and v_2 find that one of their neighbor tags, namely t_3, is not yet deactivated. Therefore, both readers moves to the second iteration of GDE. In the second iteration, the network consists of v_1 and v_2 and the only active tag is t_3 as shown in 2.5(b). Reader v_2 is the only local maximum reader since $\mathbf{id}(v_2) \succ \mathbf{id}(v_1)$ and both have only one active neighbor tag (that is, $\mathcal{T}^a(v_2) = \mathcal{T}^a(v_1) = \{t_3\}$). Therefore, v_2 owns and deactivates t_3. The algorithm terminates since all readers are deactivated. The set of non-redundant readers is $\{v_4, v_3, v_2\}$, as every one of these readers owned at least one neighbor tag.

as those generated by GREEDY. Theorem 2.7.6 proves that GDE terminates in at most $|\mathcal{R}|$ iterations, each of which consists of two write/read rounds.

Theorem 2.7.5. *Let $\mathcal{C}_s$, $\mathcal{C}_d$ be the set of non-redundant readers found by GREEDY and GDE respectively, then $\mathcal{C}_s = \mathcal{C}_d$.*

Proof. Let $\mathcal{C}_d(i)$ be the set non-redundant readers found by GDE in iteration i. That is, $\mathcal{C}_d = \bigcup_{1 \le i \le z} \mathcal{C}_d(i)$, where z is the number of iterations required by GDE. Let $v_s(j)$ be the non-redundant reader found by GREEDY in iteration j. That is, $\mathcal{C}_s = \bigcup_{1 \le j \le |\mathcal{C}_s|} v_s(j)$.

Note that in the set $\mathcal{C}_d(i)$ there is necessarily one reader that is larger than all the other readers in the set. Let $v_m(i)$ be the largest reader in $\mathcal{C}_d(i)$. Since each $v \in \mathcal{C}_d(i)$ is a local maximum reader, then no two readers in $\mathcal{C}_d(i)$ are neighbors to each other at iteration i. That is, for each u and u' in $\mathcal{C}_d(i)$, $\mathcal{T}^a(u) \cap \mathcal{T}^a(u') = \emptyset$. Therefore, $v_m(i)$ is non-redundant and deactivates all its active neighbor tags. This means that the set of active neighbors tags $\mathcal{T}^a(v')$ is not changed for all $v' \in \{\mathcal{C}_d(i) \setminus v_m(i)\}$. Also, $\mathcal{T}^a(v'')$ is either decreased or remains the same for each $v'' \in \{\mathcal{R} \setminus \mathcal{C}_d(i)\}$. This means that each reader $\mathcal{C}_d(i)$ will necessarily be a maximum reader at an iteration j in GREEDY. Therefore, each reader in $\mathcal{C}_d(i)$ belongs to $\mathcal{C}_s$, and thus, $\mathcal{C}_d \subseteq \mathcal{C}_s$. Since each maximum reader is necessarily a local maximum reader than $\mathcal{C}_s \subseteq \mathcal{C}_d$. Then the sets $\mathcal{C}_s$ and $\mathcal{C}_d$ are

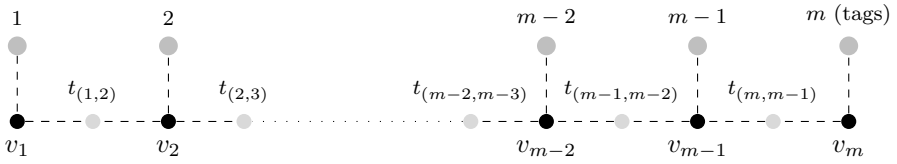


Figure 2.6: The line graph $\mathcal{L}(m)$. The readers are represented as black circles, the tags are represented as gray circles. The larger and darker gray circles are combination of tags located very closely to each other.

equivalent. □

Theorem 2.7.6. *Algorithm* GDE *terminates correctly in at most* $2|\mathcal{R}|$ *write/read rounds, under the* RT-BASIC *and* RT-DISK *model.*

Proof. There must be at least one local maximum reader in each iteration of GDE, which is essentially the maximum reader among all active readers. This is guaranteed by the uniqueness of the weights $\mathcal{W}(.)$. Therefore, GDE requires at most $|\mathcal{R}|$ iterations. Each deactivated reader v deactivates all its active neighbors tags. Each iteration consists of two write/read rounds as defined in the algorithm description.

The upper bound $|\mathcal{R}|$ is shown to be tight by building the following scenario. This scenario depends on the RT-DISK model. However, note that every network in the RT-DISK model is in the RT-BASIC model. For simplicity, let $\mathcal{R} = \{v_1, \ldots, v_m\}$ and let $\mathbf{id}(v_i) \succ \mathbf{id}(v_{(i-1)}$ for $1 < i \leq m$. Let define the following line graph $\mathcal{L}(m)$. Each reader v_i covers a set of tags T_i that are not covered by any other reader, and $|T_i| = i$. The readers are placed on a line, such that each pair of readers v_i and v_{i+1} share the coverage of a tag denoted $t_{(i,i+1)}$ for $1 < i < m$. This is depicted on Figure 2.6. Note that v_m and v_{m-1} covers the maximum number of tags, which is equal to m. Since $\mathbf{id}(v_m) \succ \mathbf{id}(v_{m-1})$, then v_m is the only local maximum at this configuration. The elimination of v_m generates the graph $\mathcal{L}(m-1)$. The same procedure is repeated $m = |\mathcal{R}|$ times. Every reader v_i is a local maximum in iteration i.

□

2.8 LIMITED-GDE: Limiting the number of iterations of GDE

Algorithm LIMITED-GDE reduces the number of iterations of GDE by limiting the number of its iterations to a constant integer $\psi \geq 1$. The constant ψ is a design parameter used to balance the number of iterations and the number of non-redundant readers.

The algorithm exploits similarities between GDE and RRE. Algorithm LIMITED-GDE executes GDE for $\psi - 1$ iterations at most. If algorithm GDE is found to terminate in these $\psi - 1$ iterations, then algorithm LIMITED-GDE terminates. Otherwise, every active tag t at iteration ψ is owned by its active neighbor reader v with the maximum weight $\mathcal{W}(v)$ among all active neighbors readers of t, where $\mathcal{W}(v) = (|\mathcal{T}^a(v)|, \mathbf{id}(v))$. Note that this is similar to algorithm RRE, since every tag is owned by the reader v that wrote in $\mathcal{M}(t)$ and has the maximum weight $\mathcal{W}(v)$.

Example: consider the reader network given in Figure 2.5. Let ψ be set to 2. GDE does not terminate in $\psi - 1$ iterations. At the ψ^{th} iteration, the set of active readers are $\{v_1, v_2\}$, while the only active tag is t_3. Both readers v_1 and v_2 write their weights in t_3. Reader v_2 owns tag t_3 since its weight is the largest according to tag t_3. The non-redundant readers are $\{v_4, v_3, v_2\}$, which is the same result found by executing GDE. However, only $2 \times (\psi - 1) + 1 = 3$ write/read rounds were executed, whereas in GDE executed 4 write/read rounds. The difference between the two algorithms is clearer in larger reader networks.

Theoretical Analysis

Theorem 2.8.1. *Algorithm LIMITED-GDE terminates correctly in at most* $2(\psi - 1) + 1$ *write/read rounds.*

Proof. Algorithm LIMITED-GDE terminates in ψ iterations. The last iteration consists of a single write/read round, whereas the other iterations consist of two write/read rounds. LIMITED-GDE executes GDE in the first $\psi - 1$ iterations. Thus, if the algorithm terminates in $\psi - 1$ iterations or less, then the correctness of LIMITED-GDE follows from the correctness of GDE. Otherwise, LIMITED-GDE executes RRE and each tag is owned by its maximum reader in the ψ^{th} iteration. There is necessarily one maximum reader of any tag t since the pairs $(|\mathcal{T}^a(v)|, \mathbf{id}(v))$ are unique and comparable. □

Theorem 2.8.2, below, shows the relationship between:

1. The set of non-redundant readers generated by `GDE` (denoted as $\mathcal{C}_g$),
2. The set of non-redundant readers generated by `LIMITED-GDE` after i iterations (denoted as $\mathcal{C}_l(i)$ for $1 \leq i \leq \psi$), and
3. The set of non-redundant readers generated by `RRE` (denoted as $\mathcal{C}_r$).

This is shown by proving that the number of non-redundant readers of `LIMITED-GDE` is decreased as the number of iterations increases. The minimum number of non-redundant readers achieved by `LIMITED-GDE` is the same as that of `GDE`, and the maximum number of non-redundant readers achieved by `LIMITED-GDE` is the same as that of `RRE` (that is, $\mathcal{C}_l(1) = \mathcal{C}_r$).

Theorem 2.8.2. *For an integer $\psi > 1$, $|\mathcal{C}_g| \leq |\mathcal{C}_l(\psi)| \leq |\mathcal{C}_l(\psi-1)| \leq \cdots \leq |\mathcal{C}_l(1)| = |\mathcal{C}_r|$.*

Proof. Let the set of active tags at iteration i be $\mathcal{T}_i^a$. Let $\max_t(\mathcal{R}, i)$ be the set of tags' maximum readers at iteration i of `LIMITED-GDE`. That is, a reader $v \in \max_{\mathcal{T}}(\mathcal{R}, i)$ if there exist a tag $t \in AT_i$ such that $v \succ v'$ for each $v' \in N_R(t)$. Let $\max_{loc}(\mathcal{R}, i)$ be the set of local maximum readers at iteration i. Then:

$$\begin{aligned} \mathcal{C}_l(1) &= \max_t(\mathcal{R}, 1) = \mathcal{C}_r \\ \mathcal{C}_l(i) &= \max_t(\mathcal{R}, i) \cup \{\max_{loc}(\mathcal{R}, i-1) \cup \cdots \cup \{\max_{loc}(\mathcal{R}, 0)\}, \text{ for each } i > 1 \end{aligned}$$

Note that $\mathcal{C}_g = \bigcup_{1 \leq i \leq z} \max_{loc}(\mathcal{R}, i)$, where z is the number of iterations required by `GDE` to terminate. Also note that $\max_{loc}(\mathcal{R}, i) = \max_t(\mathcal{R}, i) = \emptyset$ for all $i > z$ since all tags and readers are deactivated after z iterations. From definition of `LIMITED-GDE`, $\mathcal{C}_l(z+1) = \mathcal{C}_g$, and the same applies for all $\mathcal{C}_l(i)$ such that $i \geq z+1$.

Lastly, note that $\max_{loc}(\mathcal{R}, i) \subseteq \max_t(\mathcal{R}, i)$. This leads to $\max_t(\mathcal{R}, i) \subseteq \max_t(\mathcal{R}, i-1)$ since in each iteration i, the readers of $\max_{loc}(\mathcal{R}, i-1)$ are already in $\mathcal{C}_l(i)$ (see Eq. 2.5) and $\max_{loc}(\mathcal{R}, i-1) \subseteq \max_t(\mathcal{R}, i-1)$. Therefore:

$$\mathcal{C}_l(z+1) \subseteq \mathcal{C}_l(z) \subseteq \mathcal{C}_l(z-1) \cdots \subseteq \mathcal{C}_l(2) \subseteq \mathcal{C}_l(1)$$

which completes the proof. □

2.9 Conclusions

This chapter studied the problem of reader-tag RFC algorithms. A number of randomized algorithms were introduced; namely `RANDOM`, `RANDOM+`, and `MAX-MIN`. Algorithms `RANDOM` is a randomized algorithm inspired by a generalization of *single-round RFC algorithms* (that is, RFC algorithms that execute only a single write/read round). This generalization is represented by algorithm `SEQ` (given in Section 2.5.1). `RANDOM+` and `MAX-MIN` are extensions of `RANDOM`, as `RANDOM+` is defined abstractly as a multiple executions of `RANDOM`, whereas algorithm `MAX-MIN` is a special case of algorithm `RANDOM+`. Algorithms `RANDOM`, `RANDOM+`, and `MAX-MIN` are executed in $O(1)$ write/read rounds.

The chapter also introduced `GDE`, which is a deterministic algorithm that runs in $O(|\mathcal{R}|)$ write/read rounds at the worst case. `GDE` is an efficient decentralized implementation of algorithm `GREEDY`, which is known to be a powerful centralized RFC algorithm. To overcome the variable number of write/read rounds in `GDE`, the chapter introduced algorithm `LIMITED-GDE`. This algorithm runs in $O(1)$ write/read rounds. For instance, `LIMITED-GDE` (1) and `LIMITED-GDE` (2) (i.e., executing `LIMITED-GDE` with 1 or 2 iterations) limit the number of write/read rounds of `GDE` to 3 and 5 respectively. Lastly, it should be noted that most of the contributions of this chapter are directly related to algorithm `SEQ`. Using `SEQ` simplified the understanding of RFC algorithms and their properties. `SEQ` is used frequently in the next chapter to understand the simulation experiments results.

Future work on RFC algorithms shall consider additional constraints as these constraints add more practicality to RFC algorithms. An example of such constraints is the existence of faults in write/read rounds. Another constraint is the nonexistence of writable tags, or the existence of a mixture of writable tags and non-writable tags. Chapter 4 considers the case where the use of writable tags is not permitted. The mobility of the tags and load balancing between the readers are important constraints that shall be considered in future work. This chapter studied a version of the RFC problem that does not impose many constraints because this approach leads to a better understanding of the problem and for the design of more efficient algorithms. These objectives have been achieved by the results introduced in this chapter, and this is further confirmed in the next chapter.

Chapter 3

Comparative Study of Reader-Tag RFID Coverage Algorithms

This chapter studies the performance of the reader-tag RFC algorithms introduced in Chapter 2 using simulation experiments. The studied algorithms are `RRE` [CRK+09], `DRRE` [YYL08], `RANDOM`, `RANDOM+`, `MAX-MIN`, `GDE`, `LIMITED-GDE`, and `NTE` [AAH11b]. `LEO` [HCY07] is excluded from these comparisons since its description in [HCY07] lacks some important implementation details (see Section 2.4). `GREEDY` [CRK+09] is also excluded since it forms the exact same set of non-redundant readers formed by `GDE` (see Theorem 2.7.5).

3.1 Introduction

The performance of these algorithms is studied using the following performance metrics:

1. The number of non-redundant readers,
2. The convergence rate (i.e., the impact of the number of write/read rounds on the number of non-redundant readers), and
3. The number of write/read rounds.

The selection of the best combinations of the number of readers and tags is a difficult task since the applications that may use these algorithms vary from small indoor to large outdoor applications. The positioning of the readers and tags forms another challenging

task for the same reason. Thus, RFC algorithms are studied under different network topologies; each of which simulates certain scenarios where reader networks may be used. This would enrich the understanding of the studied algorithms behavior. These topologies are called:

1. *Uniform geometric topologies*,
2. *Region-based topologies*,
3. *Gaussian geometric topologies*,
4. *Arbitrary topologies* (or, non-geometric topologies).

The exact definitions of these topologies are given later in this chapter. The first three types of topologies are called *geometric topologies*. The results of this chapter are averaged over 100 experiments. Each experiment is ran over a different randomly constructed network. All algorithms are ran over the same set of networks.

The superiority of `GDE` is clear in all the experiments of this chapter. `RANDOM` outperforms `RRE` and `DRRE` in most experiments, whereas `MAX-MIN` has an even better performance as it outperforms `RANDOM` (2) and `RANDOM` (3) (i.e., `RANDOM`$^+$ with 2 and 3 iterations respectively) in most experiments. These results, however, occur in geometric topologies. The experiments of this chapter show that `RANDOM`, `MAX-MIN`, and `LIMITED-GDE` (1) (that is, `LIMITED-GDE` with $\psi = 1$) are shown to have a balance between the number of non-redundant readers, number of write/read rounds and the number of read and write operations. More details are given in the following.

3.2 The Number of Non-redundant Readers

3.2.1 Uniform Geometric Topologies

This section studies the performance of the reader-tag RFC algorithms under *uniform geometric topologies*. These topologies are constructed as follows; a set of readers and tags are spread uniformly randomly in a rectangular plane with an area of $200m \times 200m$. A reader covers a tag if the Euclidean distance between them is at most $5m$, which is the *interrogation range* of readers (i.e., r_i). The experiments consider only the networks that guarantee that every tag is covered by at least one reader.

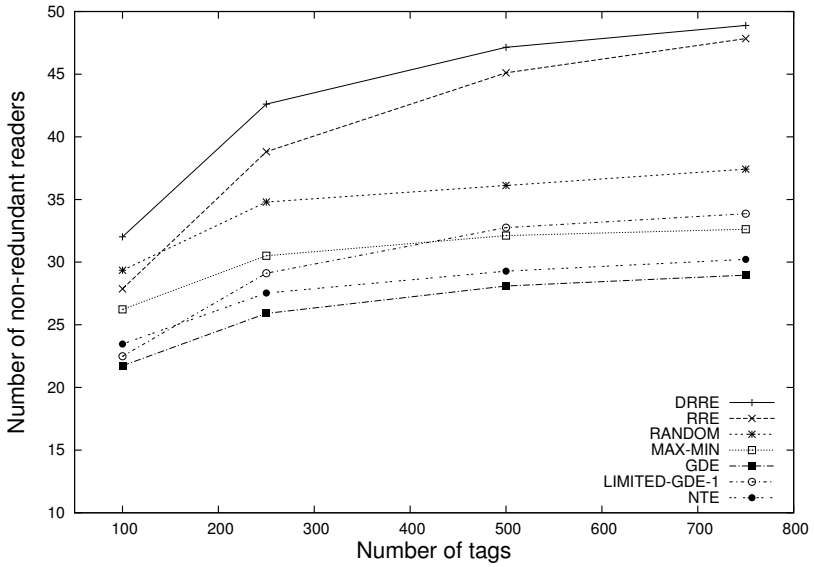


Figure 3.1: Impact of the number of tags on the number of non-redundant readers in uniform geometric topologies with 100 readers.

The number of tags impact: At first, the impact of the number of tags on the number of non-redundant readers is studied. The following experiments fix the number of readers to 100 readers, while the number of tags is in $\{100, 250, 500, 750\}$. The results of these experiments are shown in Figure 3.1. `GDE` outperforms all the other algorithms, even the centralized algorithm `NTE` which uses more complex rules. (The rules of `NTE` are given in Section 2.4). Nevertheless, `NTE` outperforms the algorithms except `GDE`. This is an expected superiority since all the other algorithms are decentralized, and hence, use less resources compared to centralized algorithms.

Figure 3.1 shows that `RRE` outperforms `DRRE` despite the fact that `DRRE` executes two write/read rounds whereas `RRE` executes only one write/read round. To elaborate more on this point, note that the weight $\mathcal{W}(v)$ of a reader v in `RRE` is $(|N_T(v)|, \mathbf{id}(v))$, where $N_T(v)$ is the number of neighbor tags of reader v, whereas the weight $\mathcal{W}(v)$ of a reader v in `DRRE` is $(|N_{R_s}(v)|, \mathbf{id}(v))$, where $N_{R_s}(v)$ is the number of **cov. neigh.** readers of v. To know their **cov. neigh.** readers, the readers must execute a write/read round in which each reader appends its identifier to all its neighbor tags memory contents. After that, each reader v reads the memory of all its neighbor tags and computes $N_{R_s}(v)$. As a result, a `DRRE` reader makes a decision of whether it is redundant or not at the end of the second write/read round. This issue, however, is not found in `RRE`.

Using SEQ to understand reader-tag RFC algorithms: Figure 3.1 shows that RANDOM forms less non-redundant readers compared to RRE in the uniform geometric topologies. The outperformance is more significant as the number of tags increases. For instance, RRE generates 25% more redundant readers than RANDOM in networks with 750 tags[1]. This is an improvement of about 10 readers on average. This result is related to the increase in the average tags counts as the number of tags increases (that is, as $|\mathcal{T}|$ increases, the average size of $N_T(v)$ for each $v \in \mathcal{R}$ increases as well). This result is explained informally in the following using the generalized sequential algorithm SEQ of Section 2.5.1.

Recall first that SEQ is a generalization of DRRE, RRE, and RANDOM. That it, the set of non-redundant readers $\mathcal{C}$ formed by the sequential versions of DRRE, RRE, and RANDOM are exactly the same as that of the decentralized versions. According to its description, SEQ sorts the readers in a descending order according to their values, where a value of a reader v is $\mathcal{W}(v)$[2]. The sorted list of readers is denoted $\{v_{(1)}, \ldots, v_{(m)}\}$, where $v_{(i)}$ indicates that $v_{(i)}$ is the i^{th} ranked reader by the algorithm (that is, $\mathcal{W}(v_{(i)}) \succ \mathcal{W}(v_{(j)})$ if $i < j$). SEQ passes by the readers in descending order. A reader $v_{(i)}$ is added to the solution $\mathcal{C}$ only if it covers at least one not-yet owned tag. The sorted set of readers that are included in the solution $\mathcal{C}$ is denoted $\{v_{c(1)}, \ldots, v_{c(k)}\}$, where k is size of $\mathcal{C}$ (that is, $v_{c(i)}$ is the i^{th} reader to be included in $\mathcal{C}$ by SEQ).

According to the definitions given above, it is necessary that $v_{(1)} = v_{c(1)}$. Thus, RRE must include $v_{(1)}$ in its solution $\mathcal{C}$, where reader $v_{(1)}$ is the reader that covers the maximum number of tags (since $|N_T(v_{(1)})|$ is the maximum among all other readers in $\mathcal{R}$). However, RRE main issue is caused by the fact that its sequential version sorts the readers only once. Thus, a reader $v_{c(i)}$, for $i > 1$, is not necessarily the reader that covers the maximum number of tags in iteration i. In fact, the inclusion of $v_{c(i)}$ in $\mathcal{C}$ at iteration i may lead to unnecessary increase in the size of $\mathcal{C}$. As an example, a possible scenario is when $v_{(2)}$ owns a small number of tags although $|N_T(v_{(2)})|$ is large. This happens if $|\{N_T(v_{(1)}) \cap N_T(v_{(2)})\}| = \beta$ where $0 < \beta \ll |N_T(v_{(2)})|$. Given that $\beta > 0$, $v_{(2)}$ must be included in $\mathcal{C}$ according to SEQ (that is, $v_{c(2)} = v_{(2)}$) although $v_{(2)}$ would own only a small subset of its neighbor tags. The tags that would be owned by $v_{(2)}$ can be owned by other readers given that they are few in quantity. If there are many readers having the

[1]This is calculated as $1 - |\mathcal{C}_{\texttt{RRE}}| \setminus |\mathcal{C}_{\texttt{RANDOM}}|$, where $|\mathcal{C}_{\texttt{RRE}}|$ is the number of non-redundant readers generated by RRE in this set of experiments. $|\mathcal{C}_{\texttt{RANDOM}}|$ is defined similarly

[2]Note that the definition of the reader values are different in each algorithm. For instance, in RRE the value $\mathcal{W}(v)$ is set to $(|N_T(v)|, \mathbf{id}(v))$, whereas it is set to $(\alpha(v), \mathbf{id}(v))$ in RANDOM.

same characteristic of $v_{(2)}$ (i.e., owns a small set of tags that could have been owned by other readers), then the number of non-redundant readers increases unnecessarily. In the studied geometric topologies, the readers are located in proximity of each others. Thus, the number of tags shared between the readers increases. This increases the number of readers which have the same characteristic of $v_{(2)}$ in the example above. This high level of intersection between readers becomes clearer as the average tag count increases in geometric topologies. This explains the weaknesses of `RRE`.

`GREEDY`, and hence `GDE`, solves the issue of `RRE` (described above) by resorting the set of readers every time a new reader is included in $\mathcal{C}$. That is, the reader that covers the maximum number of not-yet owned tags is included in $\mathcal{C}$ in each iteration. `RANDOM`, on the other hand, solves this issue by random selection of readers. This random strategy explains the superiority of `RANDOM` over `RRE` in this type of geometric topologies.

Figure 3.1 shows that algorithm `MAX-MIN` outperforms `LIMITED-GDE` (1). The outperformance can be explained using the same argument used to explain the superiority of `RANDOM` over `RRE`. This is because `MAX-MIN` can be considered as a two-iteration version of `RANDOM` (see Section 2.6). Similarly, `LIMITED-GDE` (1) can be considered as a two-iteration version of `RRE` (see Theorem 2.7.5).

The number of readers impact: Figure 3.2 and Figure 3.3 show the impact of the number of readers on the number of non-redundant readers in uniform geometric topologies. The number of tags is fixed to 250 tags and 1000 tags in the experiments of Figure 3.2 and Figure 3.3 respectively. `GDE` performance is slightly improved as the number of readers increases. `RRE` and `DRRE` generate a larger number of non-redundant readers as the number of overall readers increases, while the remaining algorithms maintain an approximatively a constant number of non-redundant readers as the number of readers increases. The impact of readers on the number of non-redundant readers is clearer in Figure 3.3. The same figure shows that `MAX-MIN` performance is improved, and it even outperforms `LIMITED-GDE` (1). This result suggests that `MAX-MIN` has a good performance in denser networks, and it shows the power of the randomized strategy that `MAX-MIN`, `RANDOM`$^+$, and `RANDOM` use to select the non-redundant readers - especially in dense networks. Lastly, note that `LIMITED-GDE` (1) performance degrades as the number of readers increases. For instance, the improvement of `GDE` over `LIMITED-GDE` (1) in Figure 3.2 is about 11% on average, while it is 21% on average in Figure 3.3. This is explained in more details when discussing the relationship between `GDE` and `LIMITED-GDE` in Section 3.3.

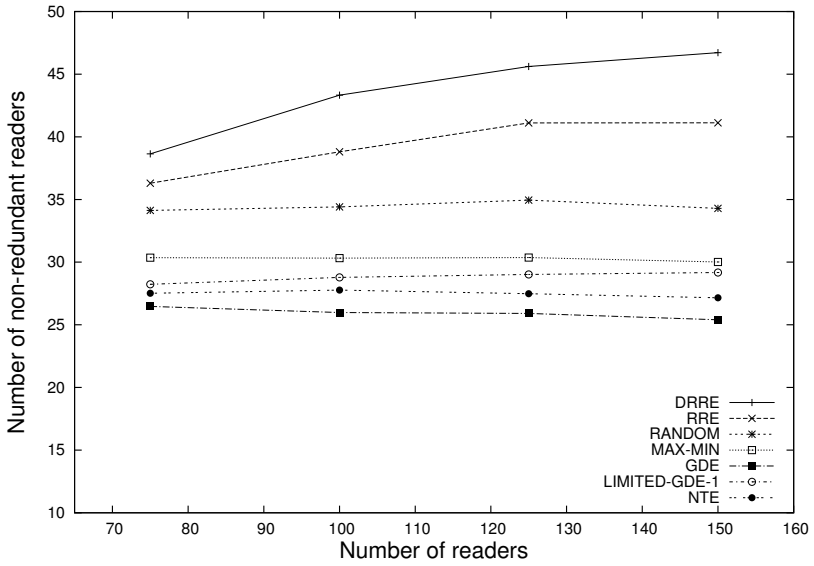


Figure 3.2: Impact of the number of readers on the number of non-redundant readers in uniform geometric topologies with 250 tags.

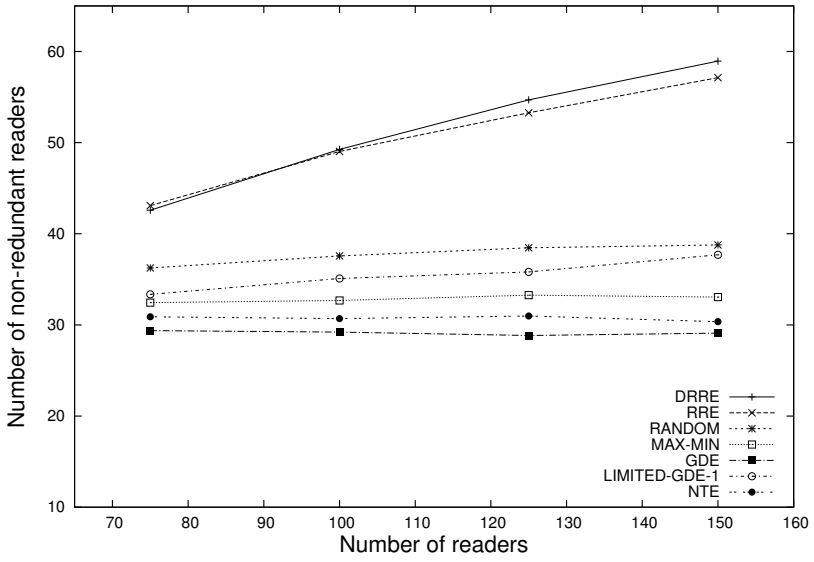


Figure 3.3: Impact of the number of readers on the number of non-redundant readers in uniform geometric topologies with 1000 tags.

3.2.2 Region-based Topologies

The following experiments emulate the execution of RFC algorithms in indoor scenarios. There is a lack of algorithms that construct network topologies used to simulate this type of scenarios. Algorithm `Build-Reg` is introduced in the following to solve this issue.

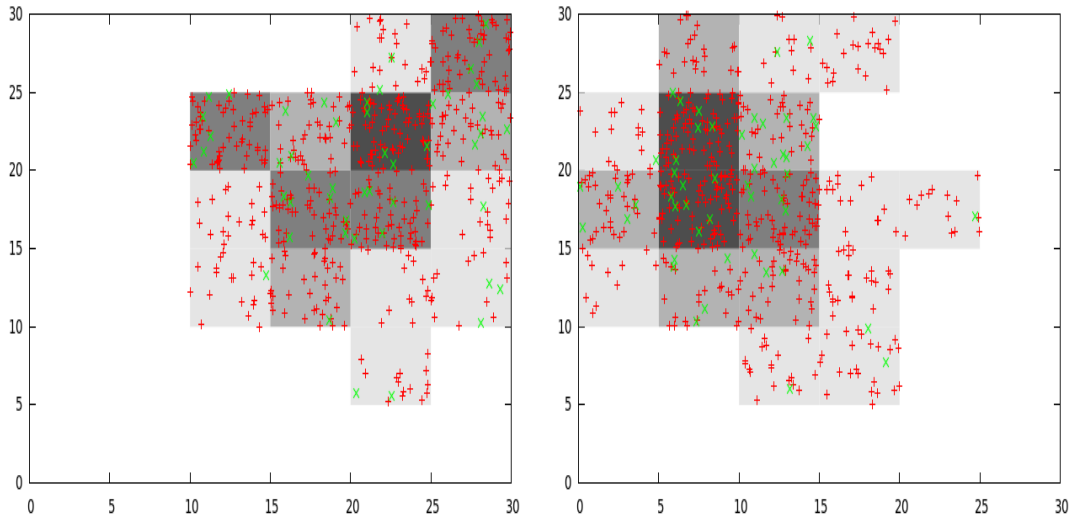


Figure 3.4: Samples of region-based topologies. Red points represent tags. Green points represent readers. For simplicity of demonstration, the regions are categorized into four attraction weight categories each of which has a different scale of gray color. White regions have attraction weight zero.

`Build-Reg` places the RFID readers and tags in a rectangular area that is divided into smaller disjoint rectangles, called *regions*. All regions have the same length and width. Two regions are called *neighbors* if they share one side (left, right, up or down side). Each region is given an *attraction weight* that is used to calculate the probability that a reader or a tag is located in it. Initially, all regions have attraction weight set to zero. A set denoted S is defined and it includes a copy of one the regions selected randomly with uniform distribution. `Build-Reg` runs in z iterations, where z is an input parameter. In iteration i, `Build-Reg` 1) selects a region s from S with a random uniform distribution, 2) finds a neighbor region of s denoted $n(s)$, 3) increases the attraction weight of $n(s)$, and 4) inserts a copy of $n(s)$ into S. Note that it is possible that S contains more than one copy of a reader. The probability that a reader or a tag is placed in a region is calculated as the attraction weight of the region divided by z. As a result, this algorithm constructs topologies in which readers and tags have different concentrations in the plane. These topologies will be called *region-based topologies*. Figure 3.4 shows two sample examples of region-based topologies.

The following experiments study the performance of reader-tag RFC algorithms in region-based topologies. The variable z of the construction algorithm is used to control

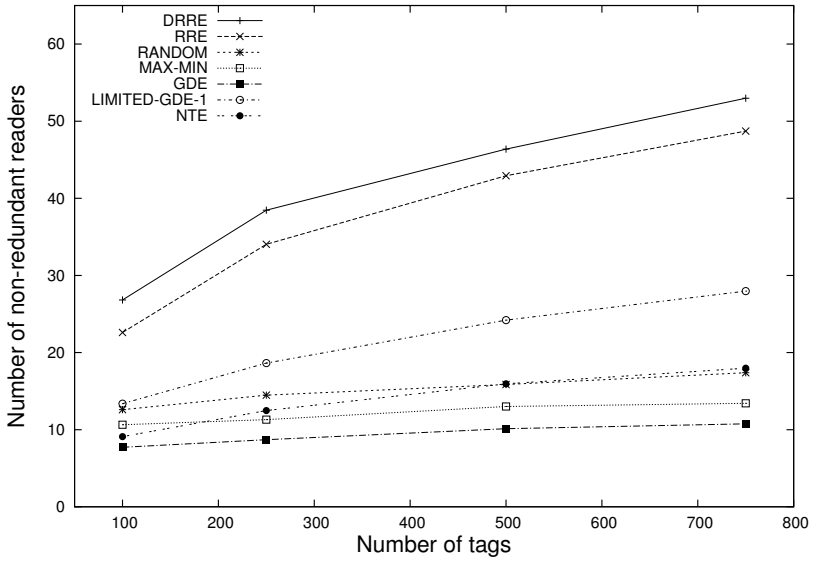


Figure 3.5: Impact of the number of tags on the number of non-redundant readers in region-based topologies with $z = 25$

the density of the reader network. This is because a larger value of z leads to more regions with attraction weight greater than zero, and hence the readers and tags can be scattered more widely in the plane. Two sets of experiments are done; the first fixes z to 25 while the second fixes it to 125. The number of readers in both sets of experiments is fixed to 100 readers. The number of tags is in $\{100, 250, 500, 750\}$. The area size is $100m \times 100m$. Each region is of size $5m \times 5m$ and the interrogation range r_i is set to $5m$. The results of the first set of experiments are shown in Figure 3.5. The results of the second set of experiments are shown in Figure 3.6.

The results of Figure 3.5 and Figure 3.6 confirm some of the previously obtained results. First, `GDE` outperforms all the others algorithms. Second, randomized algorithms such as `RANDOM` and `MAX-MIN` shows a significant superiority over other single write/read rounds (e.g., `RRE` and `DRRE`) as the network density increases. Interestingly, `MAX-MIN` may even outperform the centralized algorithm `NTE` in dense networks as shown in Figure 3.5.

3.2.3 Gaussian Geometric Topologies

The following experiments studies the performance of the studied RFC algorithms assuming that the readers are distributed according to a Gaussian distribution. Gaussian geometric topologies are therefore introduced. The readers and tags are placed in a

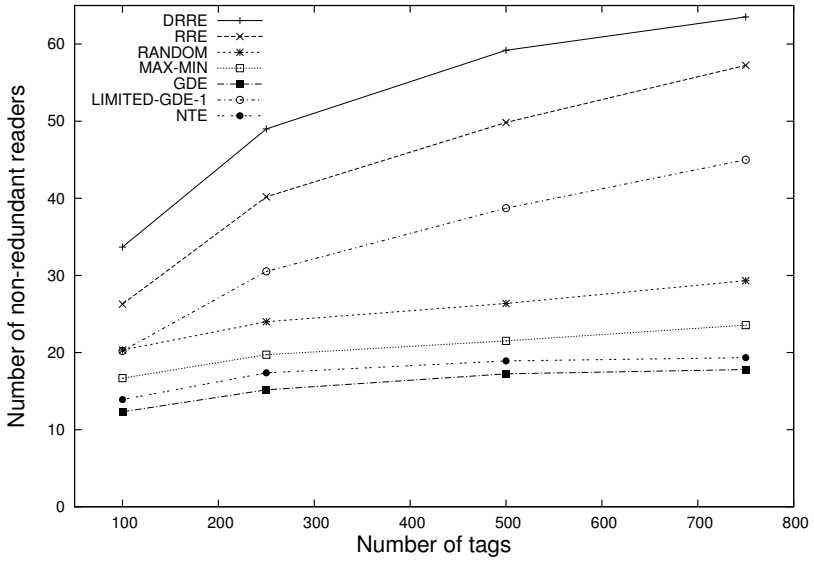


Figure 3.6: Impact of the number of tags on the number of non-redundant readers in region-based topologies with $z = 125$

rectangular area. The coordinates of a reader or tag, denoted (x_g, y_g), are drawn from a Gaussian distribution centered around the point (x_c, y_c). That is, $x_g \in N(x_c, \sigma^2)$ and $y_r \in N(y_c, \sigma^2)$, where $N(\mu, \sigma^2)$ is the Gaussian distribution with mean μ and standard deviation σ. The value of σ has a direct impact on the density of the network.

Each of the networks used in the following experiments consists of 100 readers. The area of the rectangular are where the readers and tags are placed is $200m \times 200m$. The coordinates of the center point (x_c, y_c) is set to $(100, 100)$. The standard deviation σ is set to $100/3$, as this value makes about 99% of the randomly drawn coordinates be placed within the plane. Any random point generated outside the area is eliminated and replaced by another random point. The number of tags is in $\{100, 250, 500, 750\}$. In these networks, the interrogation range r_i is set to be the minimum interrogation range that allows every tags to be covered by at least one reader.

The impact of the number of tags on the number of non-redundant readers in RFC algorithms is studied under Gaussian geometric networks. The results are shown in Figure 3.7. These results are similar to the previous results. The ranking of the algorithms according to their performance is approximately the same as in previous figures. However, `NTE` and `GDE` generate approximately the same number of non-redundant readers in these experiments. This result is explained in the following. Recall from Section 2.4 that `NTE` follows the same greedy approach used by `GREEDY`. Recall also from Theorem 2.7.5

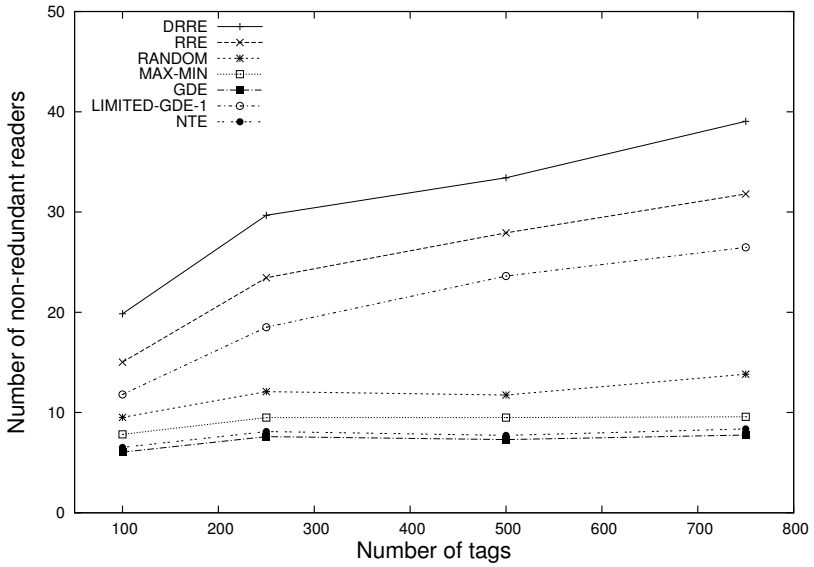


Figure 3.7: Impact of the number of tags on the number of non-redundant readers in Gaussian geometric topologies.

that GDE is a decentralized implementation of GREEDY. However, NTE orders the readers according to the ratio of the number of active neighbor tags to the number of active neighbor readers in each iteration, whereas GREEDY orders the readers according to the number of their active tags. In dense networks, all readers have a high number of neighbor readers which is approximately equivalent. Therefore, the measure that affects the ordering of the readers in these networks is mostly the number of neighbor tags, which is what GREEDY, and thus GDE, uses to order the readers. Figure 3.7 shows also that RANDOM significantly outperform RRE and DRRE. For instance, the number of non-redundant readers generated by RRE is about 2.32 of what RANDOM generates in reader networks with 750 tags. Moreover, note that MAX-MIN significantly outperforms LIMITED-GDE (1). For instance, the number of non-redundant readers generated by LIMITED-GDE (1) is about 2.74 of what MAX-MIN generates in reader networks with 750 tags. This outperformance is due mainly to the randomized strategy followed in RANDOM, RANDOM$^+$, and MAX-MIN.

3.2.4 Arbitrary Topologies

Most of the results observed above are related to the geometric nature of the studied topologies. To enforce the previous arguments, the following experiments study the performance of the algorithms in non-geometric topologies. These topologies are called

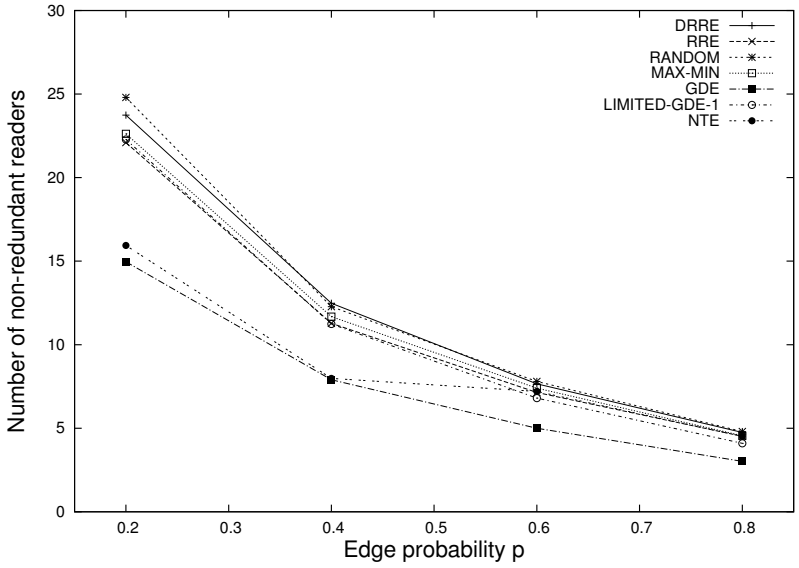


Figure 3.8: Impact of the edge probability on the number of non-redundant readers in arbitrary topologies.

arbitrary topologies. An arbitrary topology is a random bipartite graph $G = (\mathcal{R}, \mathcal{T}, E_s)$, where $\mathcal{R}$ is the set of readers and $\mathcal{T}$ is the set of tags. An edge $(v, t) \in ES$ represents a coverage relationship between a reader and a tag. The probability that an edge exists in E_s is fixed to a constant p, where $0 \leq p \leq 1$. Each of the topologies studied in the following experiments consists of 100 readers and 750 tags. The edge probability is in $\{0.2, 0.4, 0.6, 0.8\}$.

The impact of the edge probability on the number of non-redundant readers is shown in Figure 3.8. Most of the results in these experiments are repetitions of previous results. However, note that `RRE` outperforms `RANDOM` in these experiments. This is contrary to previous experiments. This confirms further the argument used previously to explain the superiority of `RANDOM` over `RRE` in geometric topologies.

Previous experiments show that the behavior of `RRE` and `RANDOM` depends on whether the network is geometric or arbitrary (i.e., non-geometric). Thus, it shall be natural to study the behavior of these algorithms in topologies that has characteristics of geometric and arbitrary topologies. These topologies are uniform geometric topologies in which an edge between a reader v and a tag t can be removed with a probability $1 - p$ for $0 \leq p \leq 1$. That is, each reader connects to a tag within its interrogation range with probability p. This construction method can be used to emulate the inaccuracy of the tag discovery procedure, and hence it adds more practicality to uniform geometric topologies

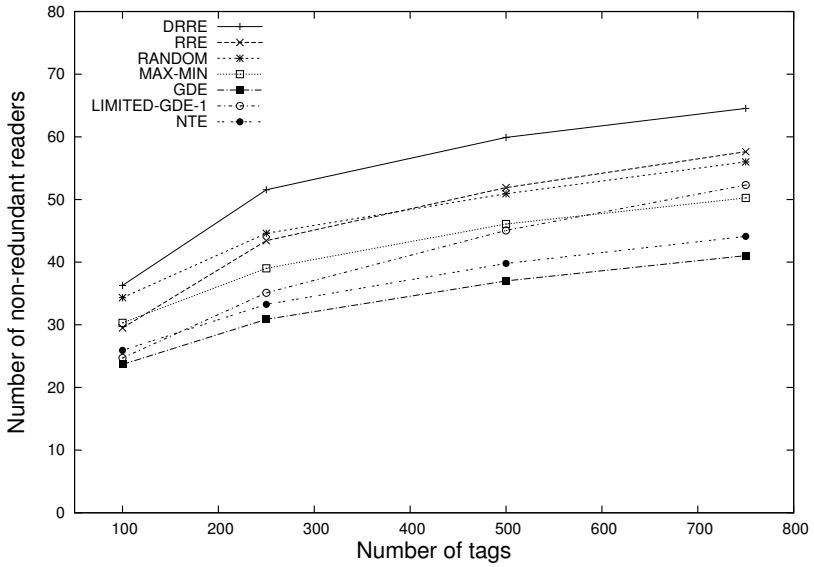


Figure 3.9: Impact of the number of tags on the number of non-redundant readers in uniform geometric topologies with probabilistic edges ($p = 0.9$, 100 readers))

and arbitrary topologies. The performance of RFC algorithms is studied under this type of topologies. The number of readers is set to 100 readers, while p is set to 0.9. The impact of the number of tags on the number of non-redundant readers is shown in Figure 3.9. All algorithms are observed to generates a larger number of non-redundant readers if compared against the case where uniform geometric topologies are used (see Figure 3.1). This is because the number of non-redundant readers in the networks of Figure 3.9 is less compared to the networks of Figure 3.1. The results show that the outperformance of `RANDOM` over `RRE` becomes clearer as the number of tags increases; whereas the performance of both algorithms is approximately the same in networks with smaller number of tags. This result suggests, as the density of the network increases, the effect of the uniform removing of edges becomes neglected. To confirm further this observation, the number of readers is increased to 150 in another set of experiments whose results are shown in Figure 3.10. Note that in Figure 3.10 the difference between `RANDOM` and `RRE` is more significant. This is similar to the behavior of the two algorithms in the experiments of Figure 3.1 and other figures.

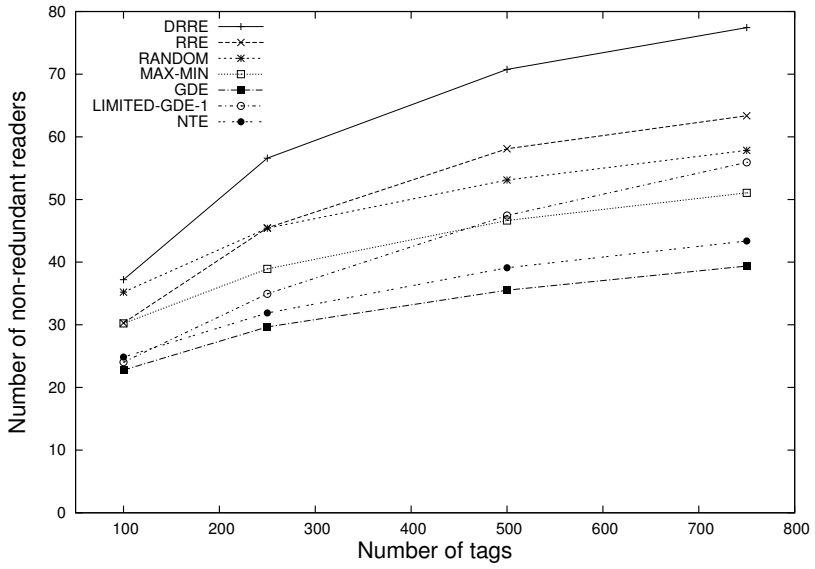


Figure 3.10: Impact of the number of tags on the number of non-redundant readers in uniform geometric topologies with probabilistic edges ($p = 0.9$, 150 readers)

3.3 The Convergence Rate

This section studies in more depth the performance of reader-tag RFID coverage algorithms that run in more than one write/read round. These algorithms are `GDE`, `LIMITED-GDE`, `RANDOM`$^+$, and `MAX-MIN`. Essentially, this section studies the impact of the number of write/read rounds on the number of non-redundant readers. This is called the *convergence rate*.

The convergence rate of `GDE` and `LIMITED-GDE` in uniform geometric topologies and region-based topologies is shown in Figure 3.11 and Figure 3.12 respectively. Region-based topologies are considered in order to study the impact of network density. Figure 3.11 shows a fast convergence rate of `LIMITED-GDE`. For instance, `GDE` and `LIMITED-GDE` (2) generate approximately the same number of non-redundant readers. Thus, the execution of `GDE` in this type of network would lead to a larger number of write/read rounds (and hence, longer execution time and higher energy consumption) but without significant improvements on the number of non-redundant readers, compared to the cheaper `LIMITED-GDE` (2). However, the convergence rate of `LIMITED-GDE` is slower in the denser topologies of Figure 3.12. This slow convergence is caused by the higher number of local maximum readers found in the early iterations of `GDE` and `LIMITED-GDE`,

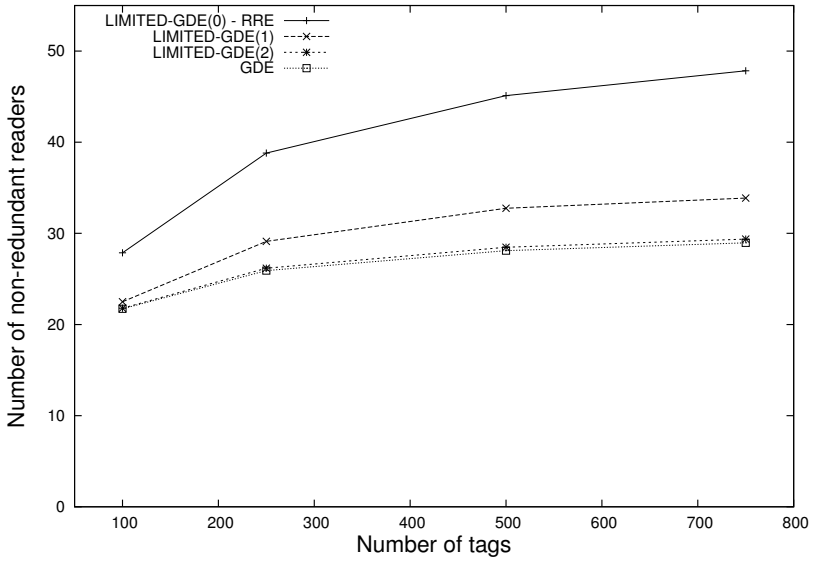


Figure 3.11: Convergence rate of `LIMITED-GDE` in uniform geometric topologies.

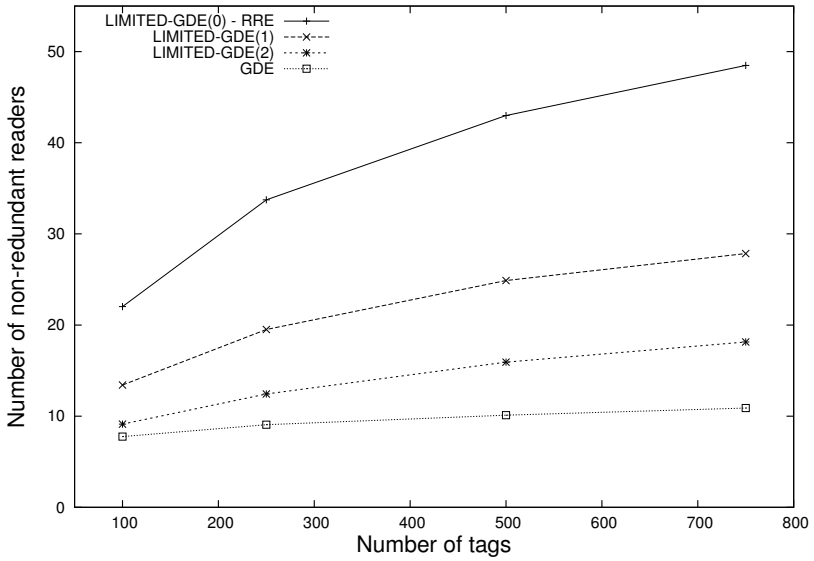


Figure 3.12: Convergence rate of `LIMITED-GDE` in region-based topologies.

which is itself caused by the higher density of the studied reader networks. An explanation of this result is given in the following. Recall that `GDE` and `LIMITED-GDE` run in iterations. In each iteration, a set of readers and tags are deactivated. There are two types of readers to be deactivated in any iteration; 1) the local maximum readers and 2) the readers that found that all their tags are deactivated. A reader of the second type is deactivated only because it is a neighbor of a local maximum reader. A local maxi-

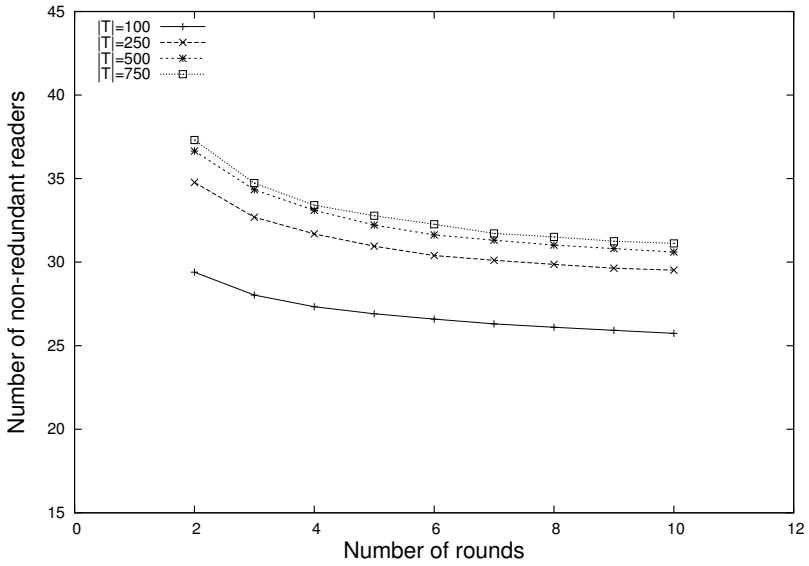


Figure 3.13: Convergence rate of RANDOM$^+$ in uniform geometric topologies.

mum reader must be larger than all its neighbor readers. The probability that a reader is local maximum reader is decreased as the number of its neighbor readers increased. For instance, if every reader is a neighbor to every other reader, then there is only local maximum reader and it is the maximum among all the readers. This means that there are fewer readers to be deactivated from the network if it is dense. This explains the slower convergence rate of GDE and LIMITED-GDE in the dense networks of Figure 3.12.

The convergence rate of RANDOM$^+$ is studied in the following experiments. Figure 3.13 shows the convergence rate of RANDOM$^+$ in uniform geometric topologies. The number of readers is set to 100 readers, while the number of tags is set to 100, 250, 500, and 750 tags. As expected, the number of non-redundant readers decreases as the number of RANDOM$^+$ iterations increases. This decrease becomes less significant as more iterations are executed, which suggests that the initial random solution (i.e., the solution $\mathcal{C}$ generated by the first RANDOM$^+$ iteration) cannot be further optimized. This does not mean, however, that the optimal solution is achieved. Thus, limiting the number of iterations in RANDOM$^+$ is sufficient to obtain a good balance between the number of non-redundant readers and the number of write/read rounds.

The fast convergence rate of RANDOM$^+$ in Figure 3.13 is due to the geometric nature of the studied topologies. More specifically, it is related to the increase in the number of shared tags between readers in proximity of each others. To elaborate further on

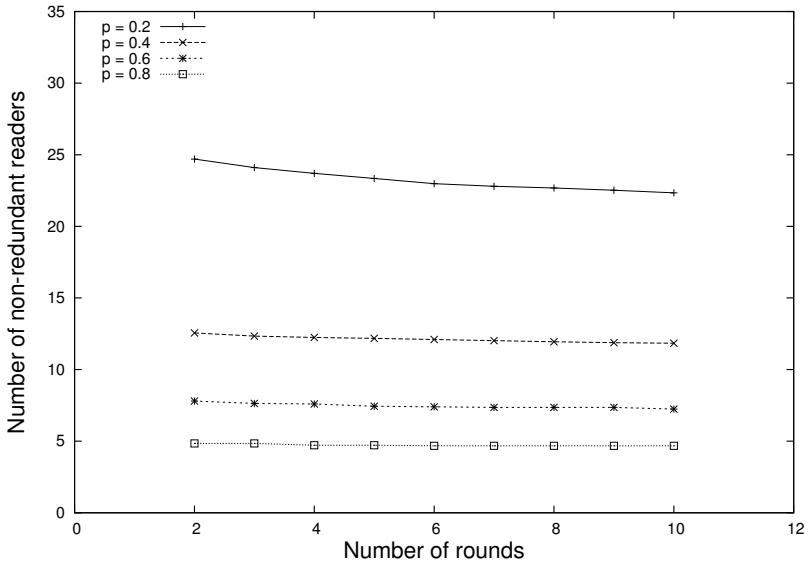


Figure 3.14: Convergence rate of RANDOM⁺ in arbitrary topologies.

this observation, the convergence rate of RANDOM⁺ is studied in arbitrary topologies. The number of readers is set to 100 readers, while the probability p is set to 0.2, 0.4, 0.6, and 0.8. The results are shown in Figure 3.14. Interestingly, the number of non-redundant readers is barely decreased as more RANDOM⁺ iterations are executed. This suggests hence that executing a large number of RANDOM⁺ iterations in arbitrary topologies has no significant impact. As a result, it is recommended to use RANDOM in arbitrary topologies or RRE, since it outperforms RANDOM in this type of topologies.

The following informal argument is used to explain the behavior of RANDOM⁺ in Figure 3.13 and Figure 3.14. Consider the sequential version of RANDOM (that is, algorithm SEQ). Recall that the order of the readers after shuffling is $\{v_{(1)}, \ldots, v_{(k)}\}$, whereas $\{v_{c(1)}, \ldots, v_{c(k')}\}$ is the sorted list of readers included by SEQ in the final solution $\mathcal{C}$. Note that $v_{c(i)}$ owns at least one tag that is not covered by any reader $v_{c(j)}$ where $j < i$. As an example, let the final solution of SEQ be $\{v_{c(1)}, v_{c(2)}, v_{c(3)}\}$ taken from the list of readers $\{v_{(1)}, \ldots, v_{(k)}\}$, where $k > 3$. Let $v_{c(1)}$ covers the tags t_1 and t_2, $v_{c(2)}$ covers the tags t_1, t_2 and t_3 and $v_{c(3)}$ covers t_2, t_3 and t_4. Note herein that $v_{c(3)}$ covers the tag t_4 that is not covered by any other reader. Thus, any improved solution must include $v_{c(3)}$. As a result, the final solution $\mathcal{C}$ could be improved if the readers are visited by SEQ in the order $v_{c(2)}$, $v_{c(3)}$, $v_{c(1)}$ for example. Note, however, that there is high level of intersection between the three reader $v_{c(1)}$, $v_{c(2)}$ and $v_{c(3)}$. This high level of intersection is the main

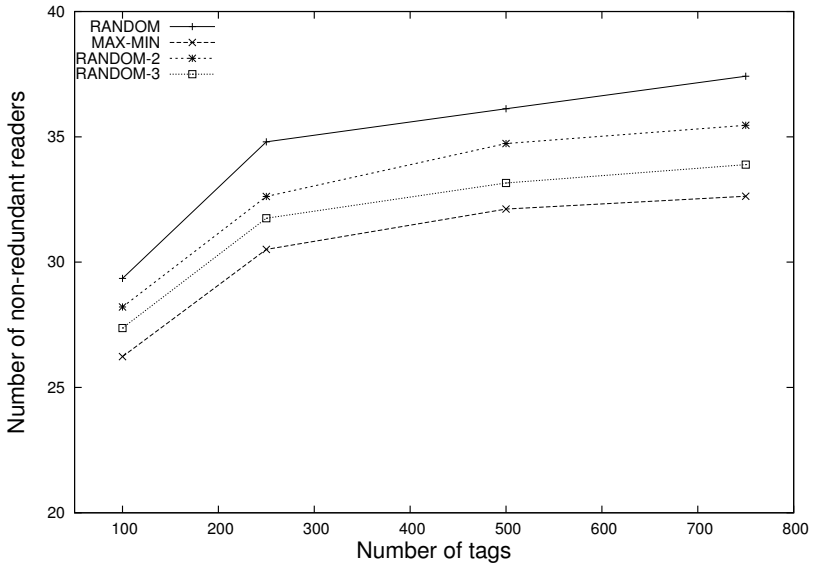


Figure 3.15: Comparison between `MAX-MIN` and `RANDOM`$^+$ in uniform geometric topologies.

reason behind the improvement achieved by the shuffling of the readers at the second iterations. This high level of intersection is more frequent in geometric topologies than it is in arbitrary topologies, and this explains the faster convergence rate of `RANDOM`$^+$ in uniform geometric topologies compared to arbitrary topologies.

A comparison is given between `MAX-MIN` and `RANDOM`$^+$ in the following experiments whose results are shown in Figure 3.15. The experiments are run over uniform geometric topologies. As expected, `MAX-MIN` outperforms `RANDOM` (1) (or simply, `RANDOM`). Interestingly however, `MAX-MIN` outperforms not only `RANDOM` (2) - which execute the same number of write/read rounds, but also it outperforms `RANDOM` (3) - which executes one write/read round more. These results confirm the effectiveness of the intuition behind designing `MAX-MIN`, which was discussed in Section 2.6.

3.4 The Number of Reads and Writes

This section compares the RFC algorithms with respect to the number of reads and write operations they execute. The memory requirements of the tags by each algorithm is also discussed in this section. Two important issues regarding the write operations must be elaborated before analyzing the results of this section:

1. For a reader v to write a sequence of bits B in the memory of a neighbor tag t, the

reader 1) reads the memory content $\mathcal{M}(t)$ of t and locally stores it in M_b, where M_b is in the memory of v, 2) appends to B to to M_b (that is, create M_a where $M_a = \{M_b \cup B\}$), and 3) *overwrites* M_a in the memory of tag t. Therefore, a write operation is also called an *overwrite* operation in this section.

2. Most of the studied algorithms do not necessitate that a reader writes a value in each write/read round it is active in. For instance, the readers in `RRE` and `RANDOM` are interested only in the maximum value already written in the memories of their neighbor tags. Therefore, a reader v overwrites its value $\mathcal{W}(v)$ in the memory $M(t)$ of a neighbor tag t only if it finds that $\mathcal{W}(v)$ is larger than what is already written in t. The same concept applies to `GDE` and `LIMITED-GDE`. Write operations are slightly different in `DRRE`. The first iteration of `DRRE` requires that a reader overwrites all its active neighbor tags in each write round. This is because each reader must write its identifier in the memory of all its neighbor tags. These written identifiers are then used by each reader in order to find its neighbor readers. This causes an increase in the number of write operations in `DRRE`.

The total number of read and overwrite operations executed by the studied RFC algorithms are shown in Figure 3.16 and Figure 3.17. The total number of read operations is computed as the sum of read operations executed by all the readers over the execution of the algorithm. The total number of overwrite operations is computed similarly. Note that the number of read and overwrite operations depends mainly on the number of the reader-tag cover relationships in the studied networks, and the number of write/read rounds executed by the algorithm.

The number of read operations is largest in `GDE` and `LIMITED-GDE`. However, the difference between `LIMITED-GDE` (2) and `GDE` is smaller than the difference between `LIMITED-GDE` (1) and `LIMITED-GDE` (2). This is related to the convergence rate of `GDE` and `LIMITED-GDE`. Note that, in every iteration of `GDE` or `LIMITED-GDE`, a set of tags and readers are deactivated. This is not the case in `RANDOM`$^+$, since all tags in `RANDOM`$^+$ are active in every iteration. Nevertheless, `RANDOM` (2) and `RANDOM` (3) outperforms `LIMITED-GDE` (1) and `LIMITED-GDE` (2) with respect to the number of read operations. This is because `LIMITED-GDE` readers execute two write/read rounds per iteration, whereas `RANDOM`$^+$ readers execute a single write/read round per iteration. Figure 3.17 studies the number of overwrites of the algorithms in hand. Note that `MAX-MIN` executes significantly more overwrites compared to `RANDOM` (2). This is despite that `MAX-MIN` and `RANDOM` (2) exe-

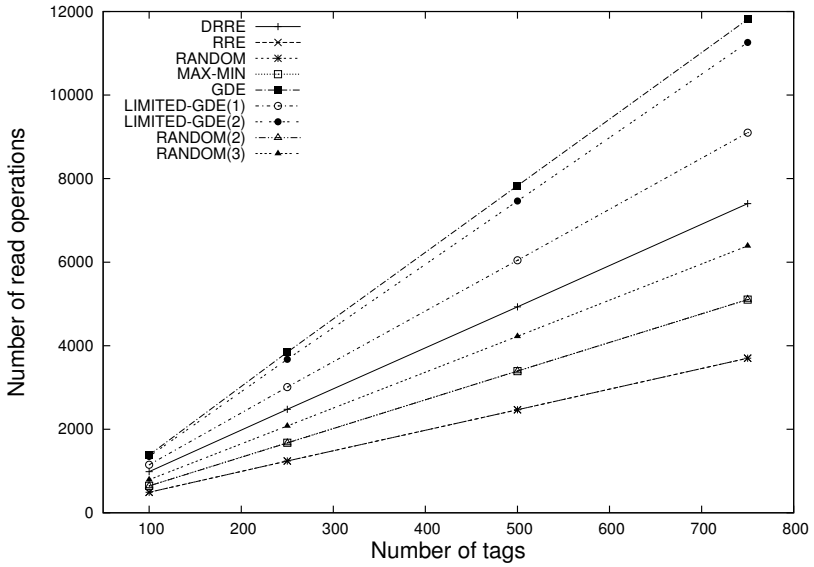


Figure 3.16: Impact of the number of tags on the total number of read operations in uniform geometric topologies.

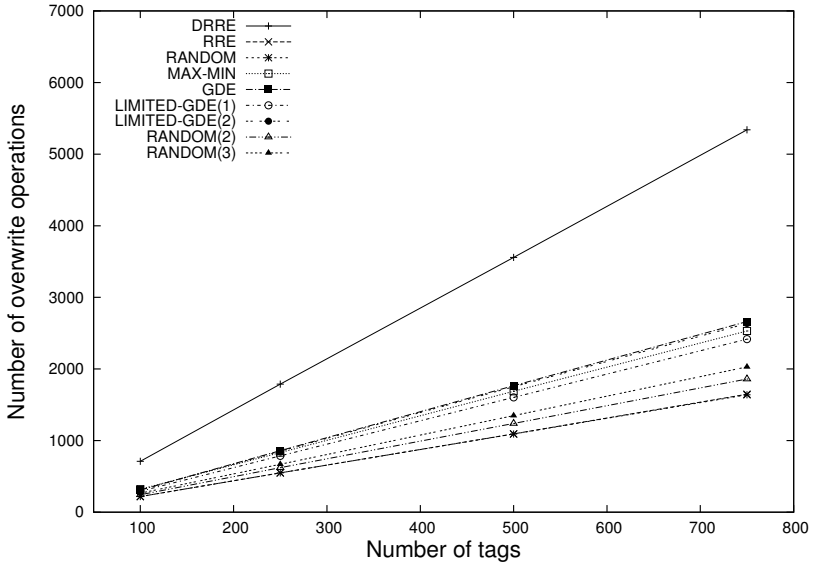


Figure 3.17: Impact of the number of tags on the total number of overwrite operations in uniform geometric topologies.

cute the same number of read operations. This a direct result of using the inverse order of readers when executing the second iteration of `MAX-MIN`. This also explains the smaller number of non-redundant readers generated by `MAX-MIN` compared to `RANDOM` (2).

3.5 Conclusions

This chapter studied the performance of the algorithms introduced in Chapter 2 (namely, `RANDOM`, `RANDOM`$^+$, `MAX-MIN`, `GDE`, and `LIMITED-GDE`) using simulation experiments. The algorithms were compared against well-known similar algorithms from the literature. The experiments considered different types of topologies; each of which emulates a different scenario where an RFID reader network may be used. Some of these topologies are first introduced in this chapter. Thus, future work shall concentrate on designing similar topologies that will be used as benchmarks to study the performance of RFID reader networks.

The simulation results, as theoretical analysis, show that the number of non-redundant readers is decreased as the number of write/read rounds executed by the algorithm is increased. `GDE`, for example, outperforms all other algorithms as it executes more write/read rounds than all other algorithms, whereas `RRE`, `DRRE` and `RANDOM` generate the largest number of non-redundant readers since they execute only 1 or 2 write/read rounds. The algorithms that gives the best balance between the number of non-redundant readers and the number of write/read rounds are found to be `MAX-MIN`, `RANDOM` (2), `LIMITED-GDE` (1) and `LIMITED-GDE` (2). `MAX-MIN` requires less write/read rounds than `LIMITED-GDE` (1) and `LIMITED-GDE` (2), but at the same time, `MAX-MIN` executes more overwrite operations compared to the same algorithms. This is an important weakness that shall be fixed in future work.

The other factor that affects the number of non-redundant readers in RFC algorithms is the readers weights values. Interestingly, it is found that algorithms such as `RANDOM` and `MAX-MIN` leads to smaller number of non-redundant readers - compared to other algorithms with the same number of non-redundant readers - despite the simplicity of the reader weight value. The same result is observed in `GDE` and `NTE`; where `GDE` outperforms `NTE` although `GDE` uses a weight value that is less complex. This suggests that simplicity leads not only to faster algorithms, but also to more efficient algorithms with respect to the number of non-redundant readers generated. Therefore, RFC algorithms shall avoid using complex rules to solve the RFC problem.

Chapter 4

RFID Coverage Using Reader-Reader Communication and Applications

This chapter studies the reader-reader RFID Coverage problem. At first, a local distributed algorithm, called `OB-COVERAGE` (Orientation-Based Coverage), is introduced for the problem. The algorithm is based on the practical assumption that the communication range r_c of readers is at least twice their interrogation range r_i. Then, algorithm `IOB-COVERAGE` (Iterated Orientation-Based Coverage) is introduced. `IOB-COVERAGE` reduces the number of non-redundant readers generated by `OB-COVERAGE` while requiring more communication rounds. The chapter studies the important problem of *Reader Collisions Avoidance (RCA)* and shows how `IOB-COVERAGE` may be used to solve it. Briefly, the RCA problem objective is to avoid all collisions caused by readers signals in a reader network. This increases, as a consequence, the efficiency of reader networks and RFC algorithms.

4.1 Introduction

Recall that existing RFC algorithms are either *centralized* or *reader-tag decentralized* algorithms. These algorithms suffer from major weaknesses consisting of the necessity of a centralized node or writable tags. These issues are solved by *reader-reader RFC algorithms* (i.e., RFC decentralized algorithms that use reader-reader communications).

Moreover, reader-reader RFC algorithms are easier to implement compared to reader-tag RFC algorithms due to the difficulty of implementing *write/read rounds* in reader-tag RFC algorithms. For example, there is only one known implementation of write/read rounds introduced in [CRK+09]. This does not mean that reader-tag RFC algorithms are not beneficial as there are significant advantages of using reader-tag communications, such as the efficient use of resources (if writable tags are among these resources).

Equipping RFID readers with wireless communication interfaces is common and inexpensive in modern days. For example, RFID readers with Bluetooth communication interfaces can be easily found in the market (see [mob07] for example). Furthermore, several companies manufactures RFID readers that can be integrated with smart phones (see for example readers manufactured by UGrokit[1] and Grabba[2]). Thus, studying the reader-reader RFC problem lies under one of the main objectives of this thesis, which is the efficient use of resources (that is, the wireless communication interfaces of RFID readers if available).

At first, this chapter shows that there are no local distributed reader-reader RFC algorithms under two reader network models defined in Section 4.2. These models, called RR-BASIC and RR-DISK, are similar to the RT-BASIC and RT-DISK model given in Chapter 2. Recall from Chapter 1 that a local distributed algorithm is a decentralized algorithm that runs in $O(1)$ communication rounds. It is possible to overcome this negative result by adding an additional, yet practical and widely used, assumption to the RR-DISK model. Briefly, this assumption, called later as Assumption **A.I.**, states that the communication range of readers is at least twice the interrogation range. The chapter introduces algorithm `OB-COVERAGE`, which is reader-reader RFC algorithm, is a local distributed algorithm that runs in one communication round, and uses Assumption **A.I**. The algorithm is based on orienting the communication links of the reader networks in a way that allows each reader to decide which tags it owns, and hence whether it is redundant or not. The readers orient their links based on a unique weight assigned to each of them. Algorithm `IOB-COVERAGE` improves `OB-COVERAGE` by reducing the number of its non-redundant readers. `IOB-COVERAGE` uses a principle similar to that used in `OB-COVERAGE`; however, it runs in $O(|\mathcal{R}|)$ communication rounds. `IOB-COVERAGE` significantly improves the results of `OB-COVERAGE` as shown by the experiments results in Section 4.8.

[1]UGrokit: http://www.ugrokit.com (last fetched 29-Oct-2013).
[2]Grabba: http://grabba.com/products (last fetched 29-Oct-2013).

This chapter also studies the *Readers Collisions Avoidance* (RCA) problem whose main objective is to avoid certain types of collisions found in RFID reader networks. There are special collisions types in RFID reader networks that are not found in regular wireless networks. This is due to the existence of two different entities in these networks; readers and tags. The collisions in reader networks can be categorized into: 1) *tags collisions*, and 2) *readers collisions*. Tags collisions occur when multiple tags in the proximity of each other transmit signals simultaneously. Readers collisions occur when multiple readers in the proximity of each other transmit signals simultaneously. There are two types of readers collisions; collisions occurring from interrogation signals, and collisions occurring from communication signals. The chapter focuses on avoiding the both types of readers collisions. The chapter shows how algorithm `IOB-COVERAGE` is used to solve the RCA problem while requiring only minor modifications. The modified algorithm is called `IOB-(RCA+COV)`.

The RFC problem and RCA problem are strongly related. First, avoiding collisions in a reader network improves its performance and the performance of the distributed algorithms that run on it. Thus, avoiding reader collisions improves the performance of reader-reader RFC algorithms. Fortunately, reader-tag RFC algorithms overcome this issue by depending on the implementation of write/read rounds which explicitly solves this issue. Second, minimizing the number of readers in a reader network, which is the one of the objectives of RFC algorithms, reduces the traffic in the network. This leads as a result to a reduction in the the amount of collisions in the network.

Contributions

The main contributions of this chapter are:

1. Algorithm `OB-COVERAGE`: which is a local distributed reader-reader RFC algorithm. More details in Section 4.5.

2. Algorithm `IOB-COVERAGE`: which is reader-reader RFC algorithm that reduces the number of non-redundant readers of `OB-COVERAGE`, but runs in $O(|\mathcal{R}|)$. More details in Section 4.6.

3. Algorithm `IOB-(RCA+COV)`: which is a decentralized RCA algorithm. More details in Section 4.7.

Chapter Organization

Section 4.3 studies the locality property in RFC algorithms. Section 4.5 and Section 4.6 introduce algorithms `OB-COVERAGE` and `IOB-COVERAGE`. Section 4.7 studies the RCA problem and introduces algorithm `IOB-(RCA+COV)`. The empirical performance of the algorithms introduced in this chapter is studied in Section 4.8 using simulation experiments. Section 4.9 concludes the chapter.

4.2 Problem Formulation

There are two types of relationships in the RFID reader networks used in reader-reader RFC algorithms. A *coverage relationship* indicates that a reader may read the memory of a tag. A *communication relationship* (simply, communication links) between a pair of readers v and u indicates that v and u are able to communicate with each other using reader-reader communications (i.e., by exchanging direct messages). Thus, v and u are called *communication neighbor readers* (for short, **comm. neigh.** readers). Communication links are modeled by the graph $\mathcal{G} = (\mathcal{R}, E_c)$, where $\mathcal{R}$ is the set of readers and E_c is the set of communication links. The coverage relationships in the reader network are modeled by the bipartite graph $\mathcal{G}_B = (\mathcal{R}, \mathcal{T}, E_s)$, where E_s is a set of *sensing edges* (or, *coverage relationships*).

The objective of the reader-reader RCA problem is to solve the RFID coverage (RFC) using reader-reader communications. Recall from Section 2.3 that the RFC problem consists of two subproblems, which are the tag reporting problem (i.e., assigning an owner reader to each tag) and the redundant RFID readers elimination problem (i.e., minimizing the number of readers in the network such that each tag is covered by at least one reader). The set of non-redundant readers is called a *cover* and denoted $\mathcal{C}$. More details can be found in Section 2.3.

Assumptions: this chapter considers two models for reader networks that use reader-reader communications; RR-BASIC and RR-DISK. In the RR-BASIC model, the edges of E_s and E_c are set arbitrarily. That is, a communication link may exist between any pair of arbitrary readers and a coverage relationship may exist between any arbitrary reader and tag. The RR-DISK model sets an edge (v, t) in E_s if the Euclidean distance between v and t, denoted $d(v, t)$, is within a constant distance r_i (called the *interrogation*

range). Furthermore, the RR-DISK model sets an edge (v, u) in E_c if $d(v, u)$, is within a constant distance r_c (called the *communication range*). The RR-DISK model, in its basic form, does not specify any relationship between r_i and r_c. Furthermore, it should be noted that r_i and r_c is common for all readers and tags. Both models share the following assumptions:

1. The tags are not *writable*. The tags are *passive* (that is, a tag cannot transmit any message to a certain reader v without first being interrogated by v).
2. The readers have no previous knowledge of the reader network topology, and have no knowledge of their positions or their neighbor tags positions.
3. Each reader v is assumed to have a unique comparable identifier, denoted $\mathbf{id}(v)$. Each tag t is assumed to have a unique identifier, denoted $\mathbf{id}(t)$.
4. There are no centralized nodes.

4.3 Locality of Decentralized RFC Algorithms

Local distributed algorithms are of significant practical importance since they are scalable and fault-tolerant. This is because the final decision taken by any node in a local distributed algorithm depends only on its κ-neighborhood[3] for a constant κ. This section studies whether it is possible to solve the reader-reader RFC problem using local distributed algorithms.

Local distributed algorithms are defined in many references as decentralized algorithms that run in a constant number of communication rounds [Suo13]. Since reader-tag RFC algorithms depend on write/read rounds instead of communication rounds, a slight modification shall be made on this definition such that a reader-tag RFC algorithm is a local distributed algorithm if it runs in a constant number of write/read rounds. This definition keeps the most important property of local distributed algorithms, in this context, which is the property that the decision taken by any of its readers depends only on its κ-neighborhood for a constant κ. Accordingly, `RRE`, `DRRE`, `LEO`, `RANDOM`, `RANDOM`$^+$, `MAX-MIN`, and `LIMITED-GDE` are all local distributed algorithms. Thus, the answer to this

[3]A node u is said to be in the *κ-neighborhood of v* if the shortest path between v and u consists of at most κ hops.

section's question is positive in case of reader-tag RFC algorithms. This is true for the RT-BASIC and RT-DISK model.

Despite being possible in the case of reader-tag RFC algorithms, solving the reader-reader RFC problem using local distributed algorithms is not possible in the RR-BASIC and RR-DISK models. This is proven by showing a similarity between the election problem, defined below, and the tag reporting problem, which is a subproblem of the RFC problem.

4.3.1 The Tag Reporting and the Leader Election Problems

Let us first study whether the reader-reader RFC problem can be solved using a local distributed under the RR-BASIC model. Studying the problem under this basic set of assumptions helps in understanding the problem in more depth. The restriction of minimizing the number of non-redundant readers in the RFID coverage problem is relaxed in the following analysis, as it makes the problem **NP Hard**. Therefore, the focus of the analysis of the section is on the tag reporting problem (i.e., the problem of assigning one and only one owner for each tag). The following important and practical prerequisite for solving the tag reporting problem is listed.

Theorem 4.3.1. *Let the input reader network be modeled as $\mathcal{G} = (\mathcal{R}, E_c)$, where E_c is the set of communication links between the readers of $\mathcal{R}$. Then, no deterministic decentralized algorithm can always correctly solve the tag reporting problem if $\mathcal{G}$ is not connected.*

Proof. Assume that $\mathcal{G}$ is not connected and that it consists of two connected components G_{c_1} and G_{c_2}, as shown in Figure 4.1. A tag t may be covered by two readers v_1 and v_2, where v_1 belongs to G_{c_1} and v_2 belongs to G_{c_2}. Both readers may claim the ownership of t, but none of them would know that t is owned by the other (given that there are no mean of communication between v_1 and v_2). Therefore, the tag reporting solution is not correct as there is a tag owned by more than one reader. □

Remark 4.3.2. *The connectivity of $\mathcal{G}$ is an important prerequisite that shall be considered when solving the tag reporting problem. It may also cause an additional cost of the network in some cases. This requirement does not exist in the case of reader-tag RFC algorithms.*

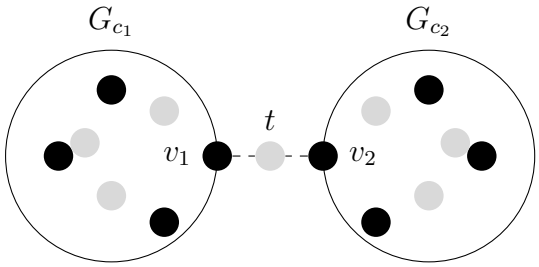


Figure 4.1: The network used in the proof of Theorem 4.3.1. Two readers v_1 and v_2 (black filled circles) covering a tag t (gray filled circles), each reader belongs to a different connected component G_{c_1} or G_{c_2}. (Note: the communication links and coverage relationships are not included to simplify the demonstration of the figure).

The *leader election problem* is a well-known problem in the field of distributed algorithms, and it is defined below:

Definition 4.3.3 (Leader election)**.** *Given a network $G = (V, E)$, where V is a set of nodes and E is the set of edges, each node must decide at the end of the leader election algorithm whether it is a* leader *or not, such that exactly one node decides to be a leader.*

Lemma 4.3.4. *Every tag reporting algorithm can be configured to solve the leader election problem in the RR-BASIC model.*

Proof. Given a reader network $\mathcal{G} = (\mathcal{R}, E_c)$ such that each reader v in $\mathcal{R}$ covers a special tag t_1 (that is, $t_1 \in N_T(v)$ for each $v \in \mathcal{R}$). If a tag reporting algorithm is correct, then there must be only one reader v_l that owns tag t_1. Given that v_l is the only owner of t_1 and that every other reader does not own t_1, then v_l announces itself as a leader. Thus, a tag reporting algorithm can be used to solve the leader election problem. □

Theorem 4.3.5. *For a reader network $\mathcal{G} = (\mathcal{R}, E_c)$, the message complexity lower bound for the* tag reporting problem *is $\Omega(|E_c| + |\mathcal{R}| \log |\mathcal{R}|)$, and its time complexity lower bound is $\Omega(\mathcal{D})$, where $\mathcal{D}$ is the* diameter *of the network*[4].

The correctness of Theorem 4.3.5 follows from Lemma 4.3.4. This is because the lower bounds given in Theorem 4.3.5 are the same lower bounds of the leader election problem [Pel90]. These lower bounds leads to the following theorem[5].

[4]The dimension $\mathcal{D}$ of a network is the longest shortest path in term of number of hops between any pair of nodes in the network.

[5]The tag reporting problem can be considered as $|\mathcal{T}|$ leader elections as shown in the following. Given a reader network $\mathcal{G} = (\mathcal{R}, \mathcal{E})$, let build $|\mathcal{T}|$ virtual networks as follows. The neighbor readers $N_R(t)$ for

Theorem 4.3.6. *There is no deterministic local distributed algorithm that always and correctly solves the tag reporting problem and the reader-reader RFID coverage problem under the RR-BASIC model.*

The result of Theorem 4.3.6 remains correct even if the RR-DISK model is used. The following proves this result.

Theorem 4.3.7. *There is no deterministic local distributed algorithm that always and correctly solves the tag reporting problem and the reader-reader RFID coverage problem under the RR-DISK model.*

Proof. Let us consider a reader network with a set of readers $\mathcal{R}$. Let $r_c = r_i \times \sqrt{2(1-\cos(\frac{360}{|\mathcal{R}|})}$. Let $r_i \gg r_c$ by letting $|\mathcal{R}| \gg 5$. The set of tags $\mathcal{T}$ contains a special tag t_1. All tags in $\{\mathcal{T} \setminus t_1\}$ are placed randomly around the readers, whereas t_1 is placed such that the distance from t_1 to every reader $v \in \mathcal{R}$ is r_i. The distance between any pair of readers is set to r_c. The value of r_c implies that this configuration forms a ring network of readers as shown in Figure 4.2. The leader election problem is solved if the *owner* of t_1 becomes a leader, since every other reader knows that it does not own t_1. □

---

each tag t builds a virtual *induced network* $\mathcal{G}[N_R(t)]$, which is defined as follows. Given a graph (or network) $G = (V, E)$, and a set $V' \subseteq V$, an induced subgraph (or network) $G[V']$ is a subgraph of G such that: 1) the vertices of $G[V']$ are V', and 2) an edge (v, u) is an edge in $G[V']$ if and only if $v, u \in V'$ and $(v, u) \in E$. Since there is only one owner for each tag t if the tag reporting algorithm is correct, then the owner of t can be considered as the leader of the network $\mathcal{G}[N_R(t)]$. Similarly, any correct solution of the leader election in each network $\mathcal{G}[N_R(t)]$ leads to one and only one owner for each tag t, and this leads to a correct solution of the tag reporting problem.

There are two main issues in the reduction above. The first issue is that $\mathcal{G}[N_R(t)]$ is not necessarily connected. In this case, neither the tag reporting nor the leader election can be solved (see Theorem 4.3.1). Therefore, additional readers (and their communication links) must be inserted into $\mathcal{G}[N_R(t)]$ to guarantee its connectivity. In the worst case, $\Theta(|\mathcal{R}|)$ readers and $\Theta(|E_c|)$ communication links of $\mathcal{G}$ would be inserted into $\mathcal{G}[N_R(t)]$. The second issue is that a reader v has no knowledge of the readers of $N_R(t)$ since any reader may cover t (that is, v does not know which readers cover t). Therefore, assume that a reader v decides to own any tag $t \in \mathcal{T}$. Given that t may be covered by any reader in the network and that a reader v does not know which are the neighbor readers of t (i.e., $N_R(t)$), then reader v decision of owning tag t may depend on every other reader in the network in the worst case. This type of algorithms, called *total algorithms*, is formally studied in [Tel88]. Briefly, a total algorithm is a distributed algorithm in which the participation of all nodes in the network is required before a final decision is taken [Tel88]. Leader elections algorithms, maxima-finding algorithms, and other well-known algorithms are shown in [Tel88] to be total algorithms. As a result of these two issues in the reduction, the tag reporting problem has the same complexity of solving $|\mathcal{T}|$ leader elections in $\mathcal{G}$.

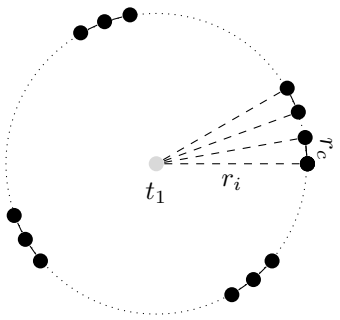


Figure 4.2: Reader network used in the proof of Theorem 4.3.7. Readers are represented in black circles, whereas tag t_1 is represented as a gray circle. The distance between each neighbor readers is $r_c = r_i \times \sqrt{2(1-\cos(\frac{360}{|\mathcal{R}|})}$.

4.4 Overcoming The Negative Results

Section 4.5 introduces a deterministic local distributed algorithm, called `OB-COVERAGE`, that solves the reader-reader RFC algorithm. The locality of `OB-COVERAGE`is achieved by assuming that every pair of coverage neighbors are also communication neighbors. More generally, a local distributed algorithm for the reader-reader RFC problem can be achieved if it is assumed that every pair of coverage neighbors can exchange messages via a path of constant number of edges[6].

`OB-COVERAGE` overcame the negative results of Section 4.3.1 by assuming that every *coverage neighbors* are also *communication neighbors*. This can be achieved by adding a practical assumption, called Assumption **A.I.** to the RR-DISK model. This assumption is defined below:

Definition 4.4.1 (Assumption **A.I**). *Assumption **A.I.** states that $r_c \geq 2r_i$ (that is, the communication range of readers r_c is at least twice their interrogation ranges r_i). All the readers have equal communication and interrogation ranges. A distance $d(v,u) \leq r_c$ between a pair of readers v and u implies that v and u are* communication neighbors *(i.e., $u \in N_{R_c}(v)$ and $v \in N_{R_c}(u)$). A distance $d(v,t) \leq r_i$ between a reader v and a tag t implies that v and t are neighbors (i.e., $t \in N_T(v)$ and $v \in N_R(t)$).*

[6]Another way to adapt to the negative results given above is by introducing an RFC algorithm that is based on 1) electing a leader reader, and then 2) executing a token-based traversal algorithm initiated by the leader (see [San06] for more details about leader election algorithms and token-based algorithms). This approach has major weaknesses - especially from a practical point of view. For instance, if the readers were assumed to be capable of performing heavier local computations, then it would be more practical to run a centralized algorithm at a leader reader after collecting the neighbor tags of all the readers. This is because an efficient centralized algorithm lead to less non-redundant readers usually. Moreover, leader election and token-based traversals algorithms are costly and non-scalable.

Assumption **A.I.** is commonly used and practical since the communication range can be increased by an increase in the transmission power at the reader. Also, the interrogation range of RFID readers and tags is usually small compared to the communication range. Varying the communication and interrogation range is a technique used in RFID reader networks, as in [KLK+07] [KLY+05] for example. Tang et al. in [TYL+09] used this assumption implicitly to simplify their algorithms. Moreover, the simulation experiments of [ZGDZ07] and [TYL+09] sets r_c to a value higher than the double of r_i. A similar assumption is commonly used in the field of wireless sensor networks, where the communication range of a sensor node is assumed to be at least twice its *sensing range* (see [TG05] for example).

Remark 4.4.2. *Assumption **A.I** is used by all the algorithms introduced in this chapter.*

Theorem 4.4.3. *According to Assumption **A.I**, if a pair of readers v and u share the coverage of the same tag t (that is, v and u are* coverage neighbor *readers), then v and u are* communication neighbor *readers (that is, they share a communication link). This implies that $N_{R_s}(v) \subseteq N_{R_c}(v)$*

Proof. Let v and u be a pair of readers covering a tag t. Let the Euclidean distances $d(v,t)$ and $d(u,t)$ be set to r_i. The readers v and u and tag t form a triangle with edges (v,u), (u,t) and (v,t), as illustrated in Figure 4.3. According to the law of cosines, the length of the edge (v,u) is equal to:

$$\begin{aligned} d(v,u) &= \sqrt{d(u,t)^2 + d(v,t)^2 - 2d(u,t)d(v,t)\cos\theta} \\ &= \sqrt{2r_i^2(1-\cos\theta)} \end{aligned}$$

The angle θ is the angle constructed from the edges (v,t) and (u,t). The distance $d(v,u)$ is maximized if θ is equal to π, which leads to $d(v,u) = 2r_i$. This proves that any two readers sharing the same tag are able to communicate with each other if $r_c \geq 2r_i$. □

4.5 `OB-COVERAGE`: Orientation-Based Coverage

This section introduces an RFID coverage algorithm called `OB-COVERAGE`. The algorithm is a local distributed algorithm that runs in a single communication round and uses

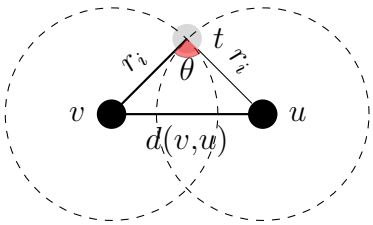


Figure 4.3: Reader network used in the proof of Theorem 4.4.3. The dashed circles represent the interrogation ranges of the readers.

Assumption **A.I.** of Section 4.4. The basic idea of `OB-COVERAGE` is to give a comparable and unique *weight* (or value), denoted $\mathcal{W}(v)$, to each reader v in the network. The readers weights are used to orient the edges of the network. Based on this orientation, a reader knows which of its neighbor tags (if any) shall be owned.

There are different methods to generate the reader weights called *weight generation functions* (more details in Section 4.5.2). The number of non-redundant readers generated by the algorithm is affected by the choice of the weight generation function. However, the algorithm is always correct as long as the reader weights are comparable and unique.

Each reader v exchanges its weight $\mathcal{W}(v)$ and its neighbor tags list $N_T(v)$ with all its communication neighbor readers in $N_{R_c}(v)$. Each edge (v, u) in E_c is virtually oriented towards the reader with the lower weight (that is, (v, u) is oriented from v to u if and only if $\mathcal{W}(v) \succ \mathcal{W}(u)$). For a reader v, a neighbor reader u is said to be an *inner* neighbor of v if the edge (v, u) is oriented from u to v. Respectively, u is said to be an *outer* neighbor of v if the edge (v, u) is oriented from v to u. The set of inner neighbors of v is denoted $N_i(v)$ and the outer neighbors of v as $N_o(v)$. Given these definitions, the statements $\{N_i(v) \cup N_o(v)\} = N_{R_c}(v)$ and $\{N_i(v) \cap N_o(v)\} = \emptyset$ are always true after this orientation procedure.

Each reader knows which of its neighbor tags shall be owned after locally executing a set of *ownership rules*. These rules are described in the following. First, let define the following terms. The set of neighbor tags of v is divided into *inner neighbor tags*, *outer neighbor tags* and *isolated neighbor tags*:

1. *An inner neighbor tag* of a reader v: is a tag t which is covered by v and an inner neighbor reader of v (that is, t is an inner neighbor tag of v if $t \in \{N_T(v) \cap N_T(u)\}$, and $u \in N_i(v)$).

2. *An outer neighbor tag* of a reader v: is a tag t which is covered by v and an outer

neighbor reader of v (that is, t is an outer neighbor tag of v if $t \in \{N_T(v) \cap N_T(u)\}$, and $u \in N_o(v)$).

3. *An isolated neighbor tag* of a reader v: is a tag that is covered by v only.

Therefore, the set of ownership is defined as follows:

Definition 4.5.1. *The ownership rules of* `OB-COVERAGE` *state that:*

Rule-1a: *a reader owns all its isolated tags, and*

Rule-2a: *a reader owns all its outer neighbor tags.*

Theorem 4.5.2. *The ownership rules of* `OB-COVERAGE` *give a correct solution of the RFC problem.*

Proof. Each tag is proven to be owned by exactly one reader. First, an isolated neighbor tag t of a reader v is covered by v only. Therefore, tag t must be owned by v. Otherwise, no reader owns t, and the algorithm solution is not correct. This proves the correctness of **Rule-1a**.

If a reader v owns an outer neighbor tag t, then there is no other reader u that would also owns t. The correctness of **Rule-2b** follows from the correctness of the previous statement. Thus, let $N_R(t) = \{v \cup \{u_1, \ldots, u_z\}\} = \{v \cup U'\}$ be the set of readers that cover tag t. According to Assumption **A.I**, all these readers must be communication neighbors (i.e., share communication links). Therefore, the readers in $N_R(t)$ form a clique of $z + 1$ readers. Given the uniqueness of the reader weights, there must be one reader that has a weight larger than all the other readers in this clique. Without loss of generality, let $\mathcal{W}(v) \succ \mathcal{W}(u_i)$ for every $u_i \in U'$. Therefore, every edge (v, u_i) for each $u_i \in U'$ is oriented from v to u_i. Thus, each reader u_i is an outer neighbor of v and t is an outer neighbor tag of v. As a result, v owns t. Moreover, tag t is an inner neighbor tag for every reader $u_i \in U'$ and thus none of them owns t. This means that tag t is owned by only one reader $v \in N_R(t)$. This completes the proof. □

4.5.1 Implementation details

The pseudocode of `OB-COVERAGE` is given in Algorithm 4.1, which is locally executed at each reader v. In Algorithm 4.1, $A(v)$ denotes a local array stored at each reader v and

Table 4.1: Terminologies used in Section 4.5

Term	**Definition**
$\mathcal{W}(v)$	The weight of reader v
$N_i(v)$	The set of inner neighbors of reader v
$N_o(v)$	The set of outer neighbors of reader v

Algorithm 4.1 Algorithm `OB-COVERAGE` at reader v

1: **for** each $(t, f(t)) \in A(v)$ **do**
2: $(t, f(t)) \leftarrow (t, \texttt{true})$ {Initialization}
3: Exchange message $M = (\mathcal{W}(v), N_T(v))$ with each neighbor reader $u \in N_R(v)$.
4: **if** received message $M = (\mathcal{W}(u), N_T(u))$ from a neighbor $u \in N_R(v)$ **then**
5: **if** $\mathcal{W}(u) \succ \mathcal{W}(v)$ **then**
6: **for** each $t' \in \{N_T(v) \cap N_T(u)\}$ **do**
7: $(t', f(t')) \leftarrow (t', \texttt{false})$
8: **for** each $(t, f(t)) \in A(v)$ **do**
9: **if** $f(t) = \texttt{true}$ **then**
10: $S(v) \leftarrow \{S(v) \cup t\}$ {i.e., own tag t}
11: **if** $|S(v)| > 0$ **then**
12: v is non-redundant (i.e., $\mathcal{C} \leftarrow \{\mathcal{C} \cup v\}$)

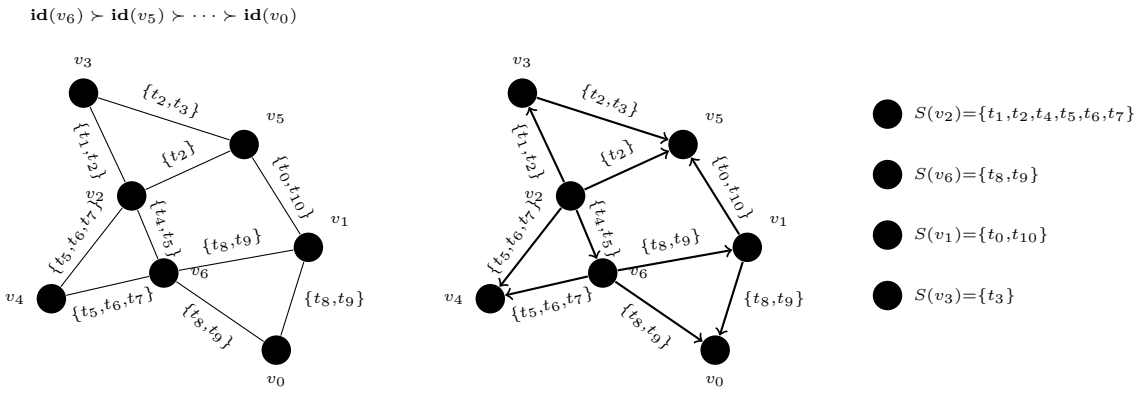
consists of pairs of the form $(t, f(t))$ for each $t \in N_T(v)$, where $f(t)$ is a boolean flag that denotes that a neighbor tag t is owned by v if and only if it is set to true.

The correctness of Algorithm 4.1 follows from observing that: 1) if a tag t is an isolated tag of v, then its flag $f(t)$ in $A(v)$ is always true during the execution of Algorithm 4.1, and 2) the value of $f(t)$ at a given reader v is changed only by a reader u which covers t and has a weight $\mathcal{W}(u) \succ \mathcal{W}(v)$. Also, if v covers t and it has the maximum weight among all the readers that cover t (i.e., $N_R(t)$), then all readers in $N_R(t)$ are neighbor readers and v causes $f(t)$ at every reader in $\{N_R(t) \setminus v\}$ to become false. Therefore, there is only one owner of every tag t according to the rules of Algorithm 4.1.

4.5.2 Weight Generation Functions

This section introduces three weight generation functions for `OB-COVERAGE`, defined below:

1. $f^s[wg(nt)]$: sets $\mathcal{W}(v)$ for each reader $v \in \mathcal{R}$ as $(|N_T(v)|, \mathbf{id}(v))$, where $|N_T(v)|$ is the number of its neighbor tags.



(a) Initial network

(b) The oriented network

(c) The owned tag by the non-redundant readers

Figure 4.4: Example of `OB-COVERAGE` using $f^s[wg(nt)]$ as a weight generation function. Readers are represented as black circles. Each edge represents a communication link between two readers. Each edge is labeled with the tags that are covered by the readers at both endpoints of the edge. For example, reader v_3 shares tags $\{t_2, t_3\}$ with reader v_5. However, reader v_3 covers $\{t_1, t_2, t_3\}$ in total. The set of isolated neighbor tags in this example are not listed for simplicity. It is assumed that $\mathbf{id}(v_i) = i$, and thus $\mathbf{id}(v_i) \succ \mathbf{id}(v_j)$ if $i > j$. In the oriented network (b), an edge is directed from reader v_i to reader v_j if $\mathcal{W}(v_i) \succ \mathcal{W}(v_j)$. The reader weights are assigned by function $f^s[wg(nt)]$. For example, $W(v_2)$ is set to $(6, 2)$ since it covers the tags $\{t_1, t_2, t_4, t_5, t_6, t_7\}$. In (c), the non-redundant readers are listed with all the tags they own. For example, reader v_2 owns all its neighbor tags whereas reader v_6 owns the tags $\{t_8, t_9\}$

2. $f^s[wg(nr)]$: sets $\mathcal{W}(v)$ for each reader $v \in \mathcal{R}$ as $(|N_{R_c}(v)|, \mathbf{id}(v))$, where $|N_{R_c}(v)|$ is the number of its neighbors readers.

3. $f^s[wg(r)]$: sets $\mathcal{W}(v)$ for each reader $v \in \mathcal{R}$ as $(\alpha(v), \mathbf{id}(v))$, where $\alpha(v)$ is a random uniform random value from $[0, 1]$ independently selected by v.

Example: Figure 4.4 gives as illustrative example of `OB-COVERAGE` using the weight generation function $f^s[wg(nt)]$ (for short, `OB-COVERAGE`(nt)). Note that this chapter will refer to `OB-COVERAGE`(nr) (respectively, `OB-COVERAGE`(r)) as the use of `OB-COVERAGE` with weight generation function $f^s[wg(nr)]$ (respectively, $f^s[wg(r)]$).

4.5.3 Theoretical Analysis

From the ownership rules and the implementation details of `OB-COVERAGE`, it is possible to deduce the following:

Theorem 4.5.3. *Algorithm `OB-COVERAGE` runs in a single communication round.*

Corollary 4.5.4. *Algorithm OB-COVERAGE is a local distributed algorithm.*

There is a strong relationship between algorithm OB-COVERAGE and single-round reader-tag RFID coverage algorithms such as RRE, LEO, DRRE, and RANDOM. For instance, OB-COVERAGE(nt) (that is, OB-COVERAGE with weight generation function $f^s[wg(nt)]$) generates the same set of non-redundant readers $\mathcal{C}$ generated by RRE. This is proven in Theorem 4.5.5. The same applies to OB-COVERAGE(r) and RANDOM (see Theorem 4.5.6). As a result, any single-round reader-tag RFID coverage algorithm that use the same concept as RRE, LEO, DRRE and RANDOM can be implemented using OB-COVERAGE if the weight generation function is well defined and if Assumption **A.I.** is used.

Theorem 4.5.5. *OB-COVERAGE(nt) generates the same set of non-redundant readers $\mathcal{C}$ generated by RRE.*

Proof. The properties of the sets of non-redundant readers of RRE (respectively, OB-COVERAGE(nt)) are given. Then, the equivalence between the two sets is shown. Note first that a reader v in RRE or OB-COVERAGE(nt) has the same weight $\mathcal{W}(v) = (|N_T(v)|, \mathbf{id}(v))$.

According to RRE. a tag t is owned by a reader v if v has the largest weight $\mathcal{W}(v)$ among all the readers that wrote in the memory $\mathcal{M}(t)$ of t (that is, $\mathcal{W}(v) \succ \mathcal{W}(u)$ for each reader $uinN_R(t)$ such that $u \neq v$). According to OB-COVERAGE(nt), a reader v owns all its isolated tags and all its outer neighbor tags. First, if a tag t is an isolated neighbor tag of reader v, then the set of neighbors readers $N_R(t)$ of t consists of v only. Then, v is necessary the reader with the largest value $\mathcal{W}(v)$ in $N_R(t)$. Let consider the second part of OB-COVERAGE(nt) ownership rules. If a tag t is an outer neighbor tag of v, then t is owned by v. According to Assumption **A.I.**, there must be an outer neighbor reader u of v such that $u \in N_R(t)$ (otherwise, t is isolated neighbor tag of v). Given that u is an outer neighbor reader of v, then the edge (v, u) is oriented from v to u. Thus, $\mathcal{W}(v) \succ \mathcal{W}(u)$. This applies for every reader in $N_R(t)$ other than v (that is, $v \succ u$ for each reader $u \in N_R(t)$ such that $u \neq v$). As a result, both algorithms generate the same set of non-redundant readers. □

Theorem 4.5.6. *OB-COVERAGE(r) generates the same non-redundant readers generated by RANDOM given that each reader v selects the same random value α in OB-COVERAGE(r) or in RANDOM.*

Table 4.2: Terminologies used in Section 4.6

Term	**Definition**
$\mathcal{R}_i^a$	The set of *active readers* at any given iteration i.
$N_T^i(v)$	The set of neighbor tags of reader v which are not owned yet.
$N_{R_c}^i(v)$	The set of active communication neighbor readers of reader v at iteration i
$N_{R_s}^i(v)$	The set of active coverage neighbor readers of reader v at iteration i

Proof. Note that a reader v in `OB-COVERAGE`(r) and `RANDOM` has the same weight $\mathcal{W}(v)$. The same procedures of the proof of Theorem 4.5.5 is followed afterwards, given that these procedures do not depend on the actual values of the reader weights. □

4.6 `IOB-COVERAGE`: Iterated Orientation Based Coverage

This section introduces algorithm `IOB-COVERAGE`, which improves the performance of `OB-COVERAGE` with respect to the number of non-redundant readers it generates. `IOB-COVERAGE` runs in iterations, each of which consists of two communication rounds. Similar to `OB-COVERAGE`, the readers of `IOB-COVERAGE` are assigned unique and comparable weights. The readers weights in `IOB-COVERAGE` may be different in each iteration. In every iteration of `IOB-COVERAGE`, at least one reader in $\mathcal{R}$ becomes *inactive*. Moreover, at least one tag is owned at each iteration. This is decided by letting each active reader execute a set of *ownership rules*.

The set of *active readers* at any iteration i is denoted $\mathcal{R}_i^a$. Each iteration i of `IOB-COVERAGE` is run over the induced network $\mathcal{G}[\mathcal{R}_i^a]$, since all inactive readers terminate the execution of the algorithm. At any given iteration i, the set of neighbor tags of a reader v which are not owned yet is denoted $N_T^i(v)$. Similarly, the set of *active communication (respectively, coverage) neighbor readers* of a reader v at iteration i is denoted $N_{R_c}^i(v)$ (respectively, $N_{R_s}^i(v)$).

The communication links of the induced network $\mathcal{G}[\mathcal{R}_i^a]$, at iteration i, are oriented using the same method used in `OB-COVERAGE`. That is, for any communication link (v, u) in $\mathcal{G}[\mathcal{R}_i^a]$, the link is oriented from v to u if and only if $\mathcal{W}(v) \succ \mathcal{W}(u)$. The active neighbor readers $N_R^i(v)$ of a reader v are then divided into *inner neighbor readers* and *outer neighbor readers*. If a communication link (v, u) is oriented from u to v then u

is an inner neighbor reader of v, otherwise it is an outer neighbor reader. Given the uniqueness of the readers weights, there must be some readers in $\mathcal{G}[\mathcal{R}_i^a]$ which have no inner neighbors. These readers are called *local maximum readers*. As an example, reader v_2 in Figure 4.4(b) is a local maximum reader.

Definition 4.6.1 (Local maximum reader)**.** *A reader v is a* local maximum reader *at iteration i if $\mathcal{W}(v) \succ \mathcal{W}(u)$ for each $u \in N_{R_c}^i(v)$. That is, the weight of v is larger than all its active neighbor readers.*

Definition 4.6.2. *The set of* ownership rules *of* `IOB-COVERAGE` *state that:*

Rule-1b: *At an iteration i, a local maximum reader v owns all its neighbor tags that are not owned yet (i.e., each tag t in $N_T^i(v)$).*

Rule-2b: *A reader v that finds that all its neighbor tags are owned in iteration i becomes inactive and terminates the execution of the algorithm (that is, v is not in $\mathcal{R}_{i+1}^a$). As a result, a local maximum reader at any given iteration becomes inactive since all its neighbor tags are owned by itself.*

Rule-3b: *The algorithm terminates at iteration i if $\mathcal{R}_i^a$ is empty (that is, if all readers are inactive).*

Theorem 4.6.3. *The ownership rules of* `IOB-COVERAGE` *give a correct solution of the RFC problem.*

Proof. The proof is based on showing that the rules terminate and that each tag is owned exactly by one owner reader.

First, there is at least one local maximum reader among the set of active readers due to the uniqueness of the readers weights. Note also that, from the definition of local maximum readers, an active reader that has neighbor tags that are not owned yet and has no communication neighbors is a local maximum. Therefore, **Rule-1b** and **Rule-2b** lead to **Rule-3b** (that is, the termination of the algorithm).

Each tag is owned exactly by one owner. This is due to the following. At each iteration i a local maximum reader v_m owns all its neighbor tags that are not owned yet (i.e., $N_T^i(v_m)$). The tags of $N_T^i(v_m)$ cannot be owned by any other reader v' at the same iteration because if any tag $t \in N_T^i(v_m)$ is covered by v', then v_m and v' are neighbors. Readers v_m and v' cannot both be local maximum readers at the same iteration since either $\mathcal{W}(v_m) \succ \mathcal{W}(v')$ or the opposite is true but not both. Therefore, every tag is owned by exactly one reader. □

Algorithm 4.2 Algorithm `IOB-COVERAGE` at reader v

1: $S(v) \leftarrow \emptyset$, $N_R^0 \leftarrow N_R(v)$, $i \leftarrow 0$
2: $N_T^0(v) \leftarrow N_T(v)$
3: **while** `isActive`() **do**
4: `isLocalMaximum` $\leftarrow$ `false`
5: {First iteration round}
6: exchange message $M = \mathcal{W}(v)$ with each neighbor reader $u \in N_R^i(v)$
7: **if** $\mathcal{W}(v) \succ \mathcal{W}(u)$ for each $u \in N_R^i(v)$ **then**
8: `isLocalMaximum` $\leftarrow$ `true`
9: **for** each $t \in N_T^i(v)$ **do**
10: $S(v) = \{S(v) \cup t\}$ {i.e., own all neighbor tags not owned yet}
11: {Second iteration round}
12: exchange message $M =$ (`isLocalMaximum`, $S(v)$) with each neighbor reader $u \in N_R^i(v)$
13: update $N_T^{(i+1)}(v)$ (by eliminating $S(u)$ for each neighbor local maximum u).
14: update $N_R^{(i+1)}(v)$ (by eliminating u for each neighbor local maximum u).
15: $i \leftarrow i + 1$
16: **if** $|S(v)| > 0$ **then**
17: v is non-redundant (i.e., $\mathcal{C} \leftarrow \{\mathcal{C} \cup v\}$)

Algorithm 4.3 `isActive`() at reader v
(used for `IOB-COVERAGE`)

1: **return** $(|N_T^i(v)| > 0)$

4.6.1 Implementation details

The pseudocode of `IOB-COVERAGE` is given in Algorithm 4.2 and Algorithm 4.3.

4.6.2 Weight Generation Functions

The weight of a reader is updated in each iteration of `IOB-COVERAGE`. Two weight generation functions are introduced. These functions are modifications of the weight generation functions of `OB-COVERAGE`. These functions are defined as follows:

1. $f^d[wg(nt)]$: sets $\mathcal{W}(v)$ for each active reader $v \in \mathcal{R}_{a_i}$, at iteration i, as $\mathcal{W}(v) = (|N_T^i(v)|, \mathbf{id}(v))$, where $|N_T^i(v)|$ is the number of its neighbor tags that are not owned yet.

2. $f^d[wg(nr)]$: sets $\mathcal{W}(v)$ for each active reader $v \in \mathcal{R}_{a_i}$, at iteration i, as $\mathcal{W}(v) =$

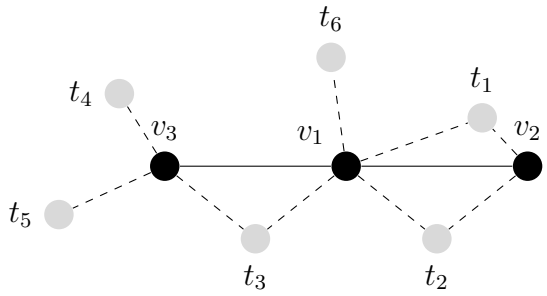


Figure 4.5: Example of `IOB-COVERAGE` is given in the following. The algorithm is executed over the network of Figure 4.5. The weight generation function $f^d[wg(nt)]$ (for short, `IOB-COVERAGE`(nt)). Assume that $\mathbf{id}(v_i) = i$ and $\mathbf{id}(v_i) \succ \mathbf{id}(v_j)$ if $i > j$. Therefore, the readers weights are $\mathcal{W}(v_1) = (4, 1)$, $\mathcal{W}(v_2) = (2, 2)$ and $\mathcal{W}(v_3) = (3, 3)$. Initially, all readers are active and all tags are not owned yet. Reader v_1 is a local maximum reader since its weight is larger than all its active neighbor readers. Thus, v_1 owns all its neighbor tags (i.e., t_1, t_2, t_3 and t_6). As a result, reader v_1 and v_2 deactivate themselves since all their neighbor tags are owned. Reader v_3 updates its weight to $\mathcal{W}(v_3) = (2, 3)$ (for t_4 and t_5 since they are not owned yet). Reader v_3 is a local maximum reader. Thus, it owns all its neighbor tags that are not owned yet (i.e., t_4 and t_5). The non-redundant readers are v_1 and v_3.

$(|N^i_{R_c}(v)|, \mathbf{id}(v))$, where $|N^i_{R_c}(v)|$ is the number of its neighbor active readers.

4.6.3 Theoretical Analysis

Theorem 4.6.4 states that `GDE` and `IOB-COVERAGE`(nt) generates the same set of non-redundant readers $\mathcal{C}$. This is proven by showing that the set of local maximum readers of `GDE` and `IOB-COVERAGE`(nt) are equivalent despite the slight difference in the definition of local maximum readers in `GDE` and `IOB-COVERAGE`.

Theorem 4.6.4. *`IOB-COVERAGE`(nt) generates the same non-redundant readers $\mathcal{C}$ generated by `GDE` and `GREEDY`.*

The proof of Theorem 4.6.4 follows techniques similar to that in the proofs of Theorem 4.5.5 and Theorem 2.7.5. The proof is thus omitted to avoid repetitions.

Remark 4.6.5. *`GDE` and `IOB-COVERAGE`(nt) sets $\mathcal{W}(v) = (|N_T(v)|, \mathbf{id}(v))$ for each $v \in \mathcal{R}$. However, a local maximum reader in `GDE` at a given iteration is a reader that has a weight larger than the weights of all its* coverage neighbors *(i.e., $N_{R_s}(v)$), while a local maximum reader in `IOB-COVERAGE`(nt) is a reader that has a weight larger than the weights of all its* communication neighbors *(i.e., $N_{R_c}(v)$). Despite the differences in the definitions, the sets of non-redundant readers generated by both algorithms remain*

the same. The definition of local maximum readers in IOB-COVERAGE(nt) *makes the algorithm suitable for* avoiding readers collisions *in reader networks.*

Corollary 4.6.6. IOB-COVERAGE(nt), GDE *and* GREEDY *have the same approximation ratio which is* $O(\log|\mathcal{T}|)$

Theorem 4.6.7. *Algorithms* IOB-COVERAGE(nt) *and* IOB-COVERAGE(nr) *executes* $O(|\mathcal{R}|)$ *communication rounds, under the* RR-BASIC *and* RR-DISK *model.*

Proof. There is at least one local maximum reader in every iteration. Every iteration consists of two communication rounds. Thus, the number of communication rounds is at most $2|\mathcal{R}|$. This bound is tight as shown in the following network, which is based on the RR-DISK model. Let $|\mathcal{R}| = m$, and the network name be $\mathcal{J}(m)$. The readers are placed on the perimeter of a circle with radius equals to $2r_c$ and centered at the coordinates $(0, 0)$. This creates a complete graph of readers (i.e., every reader is connected to every other reader). The readers shall be placed as far as possible from each other - as shown in Figure 4.6. For each reader v_i, place a tag t_i on the line joining the center point $(0, 0)$ and v_i, such that $d(v_i, t_i) = r_i$ and $d((0, 0), t_i) = r_i + \frac{r_c}{2}$. This guarantees that each reader covers only one tag.

There is only one local maximum reader in this configuration which is the reader with maximum identifier. This is because all readers have the same number of tags and the same number of readers. The elimination of the local maximum reader generates the network $\mathcal{J}(m-1)$. Thus, IOB-COVERAGE(nt) and IOB-COVERAGE(nr) executes $m = |\mathcal{R}|$ iterations, each of which executes communication rounds in the worst case. □

4.7 Reader Collision Avoidance Using RFID Coverage Algorithms

Algorithm IOB-COVERAGE may be used to solve an important problem in RFID reader networks, called the *reader collision avoidance* (RCA) problem. Minor modifications are needed to transform IOB-COVERAGE into an RCA algorithm. These modifications are called the *RCA rules* of IOB-COVERAGE. The RCA rules of IOB-COVERAGE suffer from major weaknesses. These weaknesses are solved by algorithm IOB-(RCA+COV), which is introduced in Section 4.7.2. The RCA algorithms introduced in this section are the only known decentralized RCA algorithms that do not use the locations of the readers.

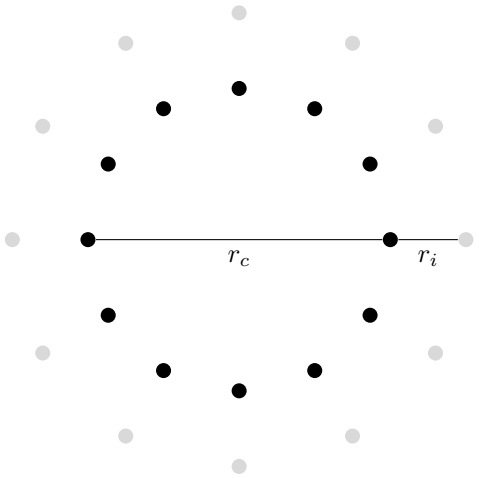


Figure 4.6: Reader network $\mathcal{J}(m)$ which is used in the proof of Theorem 4.6.7

There are two types of signals in a typical reader network: *communication signals* and *interrogation signals*. Communication signals are used in wireless communications between the readers, whereas interrogation signals are used by readers to interrogate (cover) neighbor tags. *Collisions* between communication and interrogation signals constitute a major problem in reader network. As the number of collisions increases, the performance of the network degrades. The negative impact of the problem is clearer as the size of the network increases. Collisions in RFID networks is of special interest as it is caused by two types of signals. This is contrary to collisions in traditional wireless networks, where collisions are mainly caused by communication signals.

Problem Formulation: There are three types of collisions in an RFID reader network:

1. *Tags collisions*, which occur as a result of multiple neighbor tags transmitting signals at the same time.

2. *Reader-tag collisions*, which occur as a result of multiple neighbor readers simultaneously interrogating a tag or more such that the interrogation signals collide with each other, and

3. *Reader-reader collisions*, which occur as a result of multiple neighbor readers transmitting communication signals simultaneously.

Reader collisions may be caused by the collision of an interrogation signal with a communication signal. For example, a reader v_1 interrogates tag t_1 while v_2 communicate

with reader v_3 while all these entities are in-proximity of each other. One of the methods to handle this type of collisions is to assign different frequencies for communication and interrogation signals, as suggested in the EPCglobal Gen2 standard [cla05]. Other methods are discussed in Section 4.7.1.

This section focuses on avoiding both types of reader collisions. This is called the *Reader Collisions Avoidance* (RCA) problem. Tag collisions are out of the scope of this thesis. Surveys of tag collisions avoidance algorithms can be found in [SSYH06,BSRS10].

Before getting into the details of the RCA algorithms introduced by this section, a brief survey of related work is given below. This survey helps in understanding the RCA problem in more depth.

4.7.1 RCA Algorithms: Literature Survey

Several RCA algorithms follow the Spatial Time Division Multiple Access (`STDMA`) scheme to solve the RCA problem. This scheme assumes that the readers are synchronized such that the time is divided into *time slots*. Each reader is aware of the number of the current time slot, denoted `time`, where `time` is calculated with respect to a reference known to all readers. The `STDMA` scheme schedule the readers such that each reader transmits its signals in specific time slots. The transmissions of the readers shall not interfere with each other.

`Colorwave`-like Algorithms: `Colorwave` [WES03] is a randomized implementation of the `STDMA` scheme. Each reader selects a random color $\delta(v)$ from $\{0, \ldots \chi - 1\}$ for a constant input parameter χ. A reader v is allowed to transmit in a time slot `time` if `time` **mod** $\chi = \delta(v)$. To avoid readers collisions, a reader v must have a color $\delta(v)$ different than all the colors of its communication neighbors (i.e., $\delta(v) \neq \delta(u)$ for all $u \in N_{R_c}(v)$). Given that the selection of $\delta(v)$ is random, a node v may select a color already selected by a communication neighbor u. Thus, collisions may occur between v and u. If such collisions are detected, v shall select a new random color $\delta(v)$.

`Colorwave` main issue is in the constant input parameter χ. If χ is large, then the *throughput of the reader network* is low (that is, the average number of tags read at a time slot). If χ is small then the number of collisions increases. To solve these issues, the authors of [WES03] introduced a method to dynamically find χ using estimations of the number of collisions detected by a reader. The algorithm is called Variable-Maximum

Distributed Color Selection (`VDCS`). Algorithm `HiQ`, introduced in [HES06], uses a similar approach to `VDCS`. However, `HiQ` uses a more sophisticated estimation (learning) method that requires more resources. Algorithm `PDCS` (Probabilistic Distributed Color Selection) [GFMR11] is a probabilistic version of `Colorwave` that uses multiple channels instead of a single channel. The main issue in `Colorwave`-like algorithms is their non-deterministic nature.

Algorithms based on the Centralized `GA-1`: Zhou et al. introduced in [ZGDZ07] a deterministic centralized greedy algorithm called `GA-1`. The algorithm runs in iterations. Let the reader network be denoted $\mathcal{G} = (\mathcal{R}, E_c)$. Initially, let $\overline{\mathcal{R}}$ be an empty set, and $\mathcal{R}'$ be a copy of the set $\mathcal{R}$. At each iteration i, a maximal weighted independent set, denoted R^i_{mis}, is generated from $\mathcal{G}[\mathcal{R}']$. The readers of R^i_{mis} are then removed from $\mathcal{R}'$ and inserted into $\overline{\mathcal{R}}$. This procedure is repeated until every tag in the network is covered by at least one reader in $\overline{\mathcal{R}}$. Note that $\overline{\mathcal{R}}$ becomes a cover set $\mathcal{C}$. Each reader v in R^i_{mis} is assigned a color $\delta(v) = i$. Assume that the algorithm terminates at iteration χ_G, then let each reader v transmits its signals if `time` **mod** $\chi_G = \delta(v)$. This solves the RCA problem deterministically.

The authors of [ZGDZ07] introduced a similar algorithm to `GA-1`, called `GA-M`. Algorithm `GA-M` assumes that multiple frequency channels are available. Similar centralized and decentralized algorithms of those of [ZGDZ07] are introduced in [TYL+09]. The main issue in the decentralized algorithm of [TYL+09], called `RASPberry`, is that it assumes that the readers know their physical location. This requires that the readers are equipped with additional hardware.

Variable Interrogation Ranges: In [KLY+05] and [KLK+07], reader collisions are minimized by minimizing the overlapping interrogation regions between readers. This is done by varying the interrogation range at each reader. The value of the interrogation ranges are assigned by a centralized node. There are no decentralized versions of the algorithms of [KLY+05] and [KLK+07].

Tag Reporting for RCA: Any tag reporting algorithm can be used to avoid reader collisions using the `STDMA` scheme. Nevertheless, the tags $\mathcal{T}$ must be ranked from $\{0, \ldots, |\mathcal{T}| - 1\}$ and this rank must be known to all the readers of $\mathcal{G}$ (that is, there is a one-to-one mapping function, $\mathbf{rank}(t) : \mathbf{id}(t) \rightarrow \{0, \ldots, |\mathcal{T}| - 1\}$). Let $S(v)$ be the

set of tags *owned* by a reader v. Then, no reader collisions occur if a reader v transmit its signals in a time slot `time` such that `time` **mod** $|\mathcal{T}| = \mathbf{rank}(t')$ for any $t' \in S(v)$. This solution is correct since it guarantees that only one reader transmit a signal at any given time slot. This is true because 1) the ranking function $\mathbf{rank}(t)$ is one-to-one (that is, if a tag is ranked i, then no other tag is also ranked i), and 2) a correct tag reporting solution guarantees that every tag is owned by exactly one reader. There are two major weaknesses of this solution. First, it is practically time-consuming to obtain the tags rankings. Second, the throughput of the reader network is low since only one reader transmit at a time.

`IOB-COVERAGE` for RCA: `IOB-COVERAGE`, which is implicitly a tag reporting algorithm, can be used as an efficient `STDMA` RCA algorithm using minor modifications. These modifications, called the *RCA rules* of `IOB-COVERAGE`, are different than the general conversion technique given above. The RCA rules are discussed in the following.

Recall that `IOB-COVERAGE` runs in iterations. The local maximum readers at iteration i own all their not-yet owned neighbor tags and deactivate themselves. The *RCA rules of `IOB-COVERAGE`* state that each non-redundant reader v, that is deactivated at iteration i, is assigned $\delta(v) = i$. Let χ_{iob} be the maximum value of $\delta(v)$ for each non-redundant reader v (that is, χ_{iob} is the iteration in which `IOB-COVERAGE` terminates). As a result, a non-redundant reader v transmits its signals (communication or interrogation signals) at time slot `time` if `time` **mod** $\chi_{iob} = \delta(v)$.

Theorem 4.7.1. *The RCA rules of `IOB-COVERAGE` correctly and deterministically solves the RCA problem.*

Proof. Reader-reader collisions are avoided since the set of local maximum readers deactivated at iteration i are not neighbors. This is because if a pair of readers v and u are communication neighbor readers and local maximum readers at the same iteration, then $\mathcal{W}(v) \succ \mathcal{W}(u)$ and $\mathcal{W}(u) \succ \mathcal{W}(v)$. This is not possible given the uniqueness of the reader weights. Note that a pair of communication neighbor readers v and u may be local maximum readers at different iterations (i.e., iteration i and iteration j where $i \neq j$). A reader-reader collision between v and u does not occur since $\delta(v) \neq \delta(u)$, and thus both readers transmit their signals at different time slots. This solves reader-reader collisions.

Reader-tag collisions are avoided because the communication signals of non-redundant

readers deactivated at iteration i do not collide with each other, and $r_c \geq 2r_i$ according to Assumption **A.I.**. □

4.7.2 Algorithm `IOB-(RCA+COV)`

This section introduces algorithm `IOB-(RCA+COV)`. The algorithm solves weaknesses found in some of the RCA algorithms discussed in Section 4.7.1. These algorithms are `GA-1` [ZGDZ07], `RASPberry` [TYL+09], and the *RCA rules* of `IOB-COVERAGE`. These issues are basically:

1. The *RCA rules* of `IOB-COVERAGE` may lead to situations where some readers never transmit any signal. This is because these readers are considered redundant by `IOB-COVERAGE`. The same issue is found in `GA-1` [ZGDZ07]. There is no solution provided for this issue in [ZGDZ07].

2. `GA-1` main issue is that it is centralized, whereas `RASPberry` [TYL+09] main issue is its dependence on the assumption that every reader know its location. These assumptions are strong and they are preferred to be avoided in practical scenarios.

Note that `Colorwave` [WES03] does not suffer from these problems. Instead, `Colorwave` suffers from other weaknesses that are mainly related to its indeterministic nature (see Section 4.7.1 for more details).

Algorithm `IOB-(RCA+COV)` solves the issues mentioned above. The algorithm assumes that the readers do not know their neighbor tags in advance. Algorithm `IOB-(RCA+COV)` is essentially an execution of `IOB-COVERAGE` with the following modifications:

1. Each local maximum reader at iteration i is given a color $\delta(v) = i$,

2. A reader v becomes *inactive* if all its communication neighbor readers are inactive, instead of becoming inactive if all its active neighbor tags are inactive. That is, function `isActive()` of Algorithm 4.3 is changed such that it returns true only if $|N^i_{R_c}(v)| > 0$ (instead of returning true only if $|N^i_T(v)| > 0$). However, as in `IOB-COVERAGE`, a local maximum reader at iteration i deactivates itself at the same iteration. This rule guarantees that every reader becomes a local maximum reader exactly once. The new pseudocode of `isActive()` is given in Algorithm 4.4.

Algorithm 4.4 `isActive`() at reader v
(used for `IOB-(RCA+COV)`)

1: **return** $(|N^i_{R_c}(v)| > 0)$

These modifications lead to assigning a color $\delta(v)$ to each reader in $\mathcal{R}$ such that $\delta(v) \neq \delta(u)$ for any pair of communication neighbor readers. This is because no pair of communication neighbor readers can be local maximum readers at the same iteration. Also, given the new definition of function `isActive()` in Algorithm 4.4, every reader must be a local maximum reader exactly once.

A practical `IOB-(RCA+COV)`: `IOB-COVERAGE` assumes that the readers know all their neighbor tags before executing the algorithm. To obtain this knowledge efficiently, the readers must run a *tag discovery algorithm*, which requires itself an RCA algorithm to be efficient. This is considered as a weakness in `IOB-COVERAGE`. Algorithm `GA-1` [ZGDZ07] provided a solution to this issue. Algorithm `IOB-(RCA+COV)` may solve this issue with few modifications. In fact, `IOB-(RCA+COV)` may solve the RFC problem even if the neighbor tags are not known in advance. This is done by letting each local maximum reader v_m at iteration i, once v_m knows it is a local maximum, to:

1. run a discovery procedure to discover all its neighbor tags (i.e., $N_T(v_m)$),
2. own all the tag that are not owned yet, and
3. inform all its communication neighbors readers about the tags that were recently owned by sending $S(v_m)$ to each of them.

These additional rules solve the RFID coverage problem and the RCA problem[7]. Thus, the weaknesses listed in this section are all solved. An example is given in Figure 4.7.

4.8 Simulation Experiments

This section studies the performance of the algorithms introduced in this chapter using simulation experiments. These algorithms are `IOB-COVERAGE` and `IOB-(RCA+COV)` un-

[7] $N^i_T(v)$ is not initialized or updated in every iteration since it is assumed that $N_T(v)$ is not known in advance (that is, line 2 and line 13 of Algorithm 4.2 are not executed).

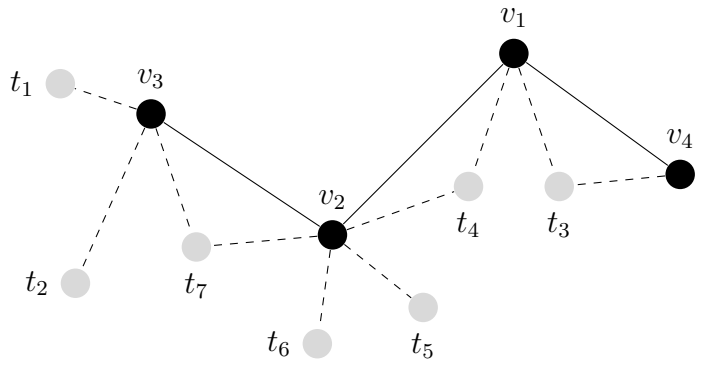


Figure 4.7: Example of `IOB-(RCA+COV)`. The weight generation function used in this example is $f^d[(nr)]$. Black circles represent readers and gray circles represent tags. Continuous lines represents communication links. Dashed lines represent coverage links. Recall that `IOB-(RCA+COV)` assumes that the readers do not know their neighbor tags in advance. Assume that $\mathbf{id}(v_i) = i$ and $\mathbf{id}(v_i) \succ \mathbf{id}(v_j)$ if $i > j$. The readers weights are $\mathcal{W}(v_1) = (2, 1)$ (since v_i has two communication neighbors, v_2 and v_4), $\mathcal{W}(v_2) = (2, 2)$, $\mathcal{W}(v_3) = (1, 3)$ and $\mathcal{W}(v_4) = (1, 4)$. Initially, v_2 is a local maximum. Reader v_2 discovers all its neighbor tags and own them. Then, v_2 informs its communication neighbor readers (i.e., v_3 and v_1) that it has owned the tags t_4, t_5, t_6 and t_7. At this point, v_2 deactivates itself and sets $c(v_2)$ to 0, since it is the first iteration. In the second iteration, v_3 and v_4 are local maximum readers. Both readers discover their neighbor tags and own them. Therefore, $c(v_3)$ and $c(v_4)$ are set to 1. In the third iteration, v_1 is the only active reader. Thus, v_1 is a local maximum reader. Reader v_1 discovers all its neighbor tags which are t_3 and t_4. However, v_1 realizes that these tags are already owned. This is because the owners of these tags informed v_1 after owning these tags. Given that all its neighbor tags are owned, v_1 considers itself redundant. Nevertheless, $c(v_1)$ is set to 2.

der the weight generation functions $f^d[(nt)]$ and $f^d[(nr)]$. For simplicity, these algorithms are denoted `IOB-COVERAGE`(nt), `IOB-COVERAGE`(nr), `IOB-(RCA+COV)` (nt), and `IOB-(RCA+COV)` (nr). The algorithms are compared against the centralized `GA-1` algorithm of [ZGDZ07]. The analysis focuses on the number of iterations executed by the algorithm. This metric gives an indication of the execution time of the algorithms, and is also used to measure the maximum number of time slots needed to read all tags in RCA algorithms. This section does not study the performance of `OB-COVERAGE` due to its similarity to single-round reader-tag RFC algorithm as shown in Theorem 4.5.5 and Theorem 4.5.6.

Figure 4.8 and Figure 4.9 depicts the average number of iterations as a function of the number of tags and the number of readers in algorithms `IOB-COVERAGE`, `IOB-(RCA+COV)`, and `GA-1`[8]. Algorithm `GA-1` outperforms all the other algorithms, whereas

[8]These experiments assumed that the readers weights in algorithm `GA-1` are the neighbor tags instead of the number of *covered subelements*, since using the latter weight function would generate a higher number of iterations. Algorithm `GA-1` uses an input parameter denoted γ, which is set to 2. See [ZGDZ07] for more details.

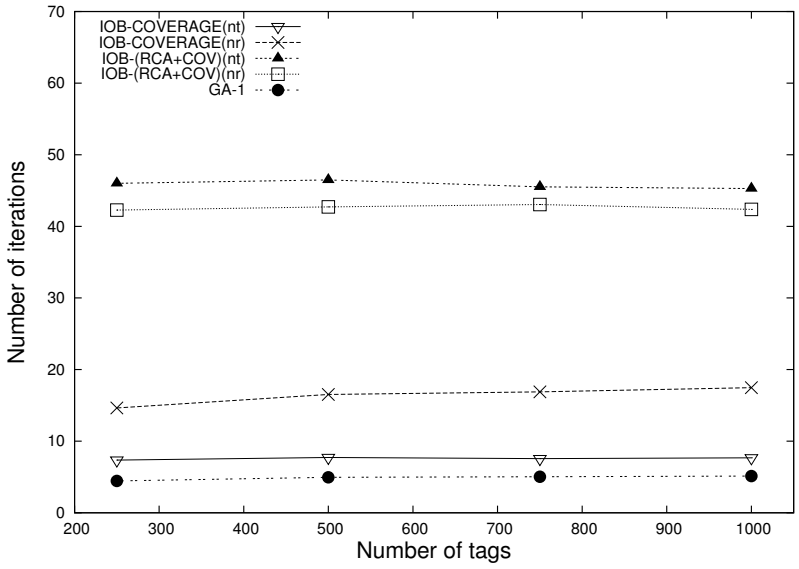


Figure 4.8: Impact of the number of tags on the average number of iterations (100 readers)

`IOB-COVERAGE`(nt) is second in performance. This is because `GA-1` eliminates a larger number of readers in each iteration compared to `IOB-COVERAGE`(nt), as it eliminates a weighted maximal independent set in each iteration whereas `IOB-COVERAGE`(nt) eliminates only the local maximum readers in each iteration. Recall that `GA-1` is centralized while `IOB-COVERAGE`(nt) is decentralized. Thus, the superiority of `GA-1` is somehow expected. Algorithm `IOB-(RCA+COV)`, in both of its versions, requires a large number of iterations compared to the other algorithms. This is because every reader v in `IOB-(RCA+COV)` is assigned a value in $\delta(v)$, whereas only the non-redundant readers are assigned such value in the other algorithms. `IOB-(RCA+COV)` (nr) requires less iterations compared to `IOB-(RCA+COV)` (nt) because a reader in `IOB-(RCA+COV)` is deactivated if all its communication neighbor readers are also inactive. This makes `IOB-(RCA+COV)` (nr) more practical than all the other algorithms, especially as `IOB-(RCA+COV)` (nr) can be easily modified such that it does not require previous knowledge of the neighbor tags for each reader.

4.9 Conclusions

This chapter studied the reader-reader RFID coverage problem. Section 4.3.1 showed that there is no local distributed algorithm for the RFC problem in general. To over-

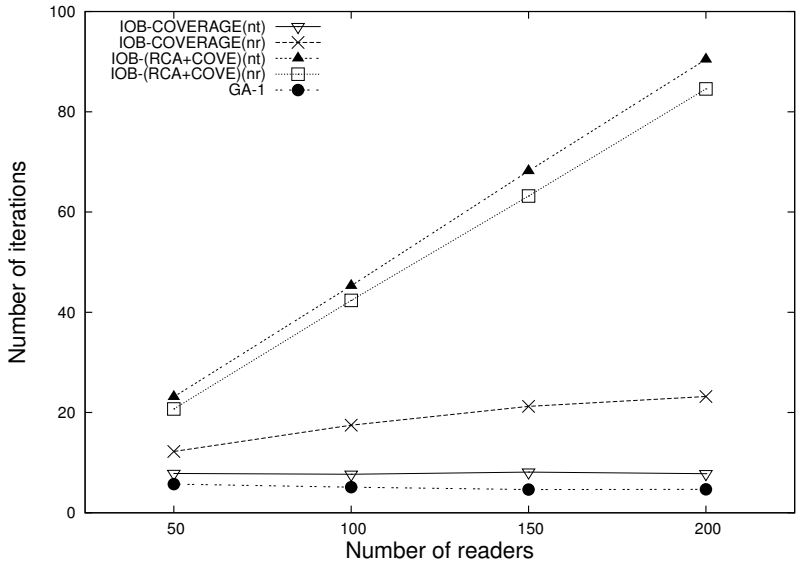


Figure 4.9: Impact of the number of readers on the average number of iterations (1000 tags)

come this negative result, Assumption **A.I.** is introduced in Section 4.4. The assumption states that the communication range of the readers shall be at least twice the interrogation range. Using this assumption, a local distributed algorithm for the reader-reader RFC problem, called `OB-COVERAGE`, is introduced in Section 4.5 whereas Section 4.6 introduced a reader-reader RFC algorithm, called `IOB-COVERAGE`, that runs in $O(|\mathcal{R}|)$ communication rounds.

This chapter studied as well the reader collision avoidance problem in Section 4.7. Section 4.7.1 introduced a brief literature survey of RCA algorithms, and showed how to convert any tag reporting algorithm into an RCA algorithm. Section 4.7.1 showed how to use algorithm `IOB-COVERAGE` as an RCA algorithm by introducing the *RCA rules* of `IOB-COVERAGE`. Section 4.7.2 discusses main issues in `IOB-COVERAGE` as an RCA algorithm and in other major RCA algorithms. These issues are solved using algorithm `IOB-(RCA+COV)`, which is a modification of `IOB-COVERAGE`. Algorithm `IOB-(RCA+COV)` is the first decentralized deterministic RCA algorithm that uses no location information.

About Assumption A.I.: Most of the advantages of the algorithms of this chapter are results of Assumption **A.I.**. This assumption, or similar ones, should be investigated more in future works. Finding the advantages of this assumption is basically a direct consequence of investigating the problem under a basic set of assumptions as done in Section 4.3.1. This approach is encouraged in studying other RFID reader networks

problems as a step towards designing more efficient algorithms.

The main advantage of Assumption **A.I.** is that it allows every pair of coverage neighbor readers to be communication neighbor readers as well. As a result, any reader-tag RFC algorithm can be simulated by a reader-reader RFC algorithm. The opposite is not true, since two readers may be communication neighbors but not coverage neighbors. Therefore, Assumption **A.I.** makes the RT-DISK model stronger than the RR-BASIC, RT-BASIC, and RT-DISK models. Interestingly, the RT-DISK model without Assumption **A.I.** can be considered as a weak model. This is because the RFC problem cannot be solved using local distributed algorithms under this model, while this is possible under the RT-BASIC and the RT-DISK models.

In practice, there are more issues to be considered when deciding which model to use. For example, reader-reader RFC algorithms require that the readers form a connected network and each tag is covered by at least one reader. On the other hand, the reader-tag RFC algorithms only require that each tag is covered. Achieving connectivity between the readers may be costly in practice. Furthermore, the use of reader-tag communication can be considered as a way of using the available resources in a more efficient manner; a general approach to improve the scalability of the `IoT`. As a result, combining reader-tag and reader-reader communications may be an interesting future research direction. In fact, combining reader-tag and reader-reader communications can improve the performance of the network, including its scalability and fault-tolerance.

Part III

Data Processing and Transportation: Readers Inter-Communication

Chapter 5

The Bluetooth Scatternet Formation (BSF) Problem

This part of the thesis studies the problem of interconnecting readers in a reader communication network. The technology used to achieve this objective is Bluetooth. This technology is especially considered herein due to the important role it plays in the Internet of Things. A special interest is given to the problem of forming efficient Bluetooth multihop networks, which is commonly known as the *Bluetooth Scatternet Formation* (BSF) problem. This chapter introduces the BSF problem, followed by a literature survey. Algorithm `Eliminate` is then introduced, which is an improvement of a well-known decentralized BSF algorithm called `BlueMIS`. Chapter 6 introduces algorithm `BSF-UED`. The objective of `BSF-UED` is to overcome main weaknesses in a category of BSF algorithms, called *mesh-based BSF algorithms*. Algorithms `Eliminate` and `BSF-UED` focus on forming scatternets in a time-efficient manner.

5.1 Introduction

The efficiency of an RFID reader network is increased if its readers are able to communicate with each other. For many reasons, Bluetooth [blu10], which is a wireless communication technology, is perfectly suitable for readers inter-communication. This is mainly due to the major advantages of Bluetooth, such as the low-cost of communications, low energy consumption, and the high tolerance to signals noise and interference caused by wireless technologies working in proximity. Another advantage of Bluetooth,

which is the most important according to this thesis, is its wide availability in modern mobile phones, laptops, tablets and many other consumer electronic devices. About 95 % of today's mobile phones are Bluetooth-enabled according to statistics in [Nok11]. Other statistics in [abi13b] estimated that about 3.5 billions Bluetooth-enabled devices have been installed in 2012. This figure is estimated to triple by 2018.

There is a wide interest in equipping RFID readers with Bluetooth in the academia [RMI+10] [SF03] and the industry (see for example Bluetooth-enabled RFID readers manufactured by IDBlue[1] and Technology Solutions UK Ltd[2]). For example, attaching RFID readers to Bluetooth-enabled smart phones became a common method to enable RFID readings at low costs (see for example readers manufactured by UGrokit[3] and Grabba[4]). Bluetooth is not only seen as an enabler for RFID reader inter-communications, but it is also considered a main enabling technology of the Internet of Things [abi13a] [abi13b]. Note that Bluetooth and RFID have been used together in applications other than intercommunication between readers (see for example [Smi07], [SHR06], and [KHR+12]).

Point-to-point communication between a pair of Bluetooth devices is relatively straightforward, but the construction of efficient *multi-hop* topologies is a challenging problem. This problem, which is called the *Bluetooth Scatternet Formation (BSF)* problem, attracted much attention a decade ago. At some point, the problem became less attractive as the question of whether it can be used for more than just a cable-replacement technology started to be debated. There is, however, a recent revival of interest in using Bluetooth in ad-hoc networking context. This revival is due to the current use of Bluetooth in home automation, heart-rate monitors, smart phones, blood-glucose meters, smart watches, window and door security sensors, car key fobs, blood-pressure cuffs and other consumer electronic devices. The Bluetooth specifications kept evolving to overcome issues found in previous version, while new marketing trademarks were recently pushed forward by the Bluetooth Special Interest Group (SIG) in anticipation of a new networking trend around Bluetooth, including *Bluetooth Smart* and *Bluetooth Smart Ready*[5].

[1]IDBlue: http://idblue.com/rfid-readers/uhf-rfid-reader (last fetched 29-Oct-2013).
[2]Technology Solutions UK Ltd: http://www.tsl.uk.com/bluetooth/ (last fetched 29-Oct-2013).
[3]UGrokit: http://www.ugrokit.com (last fetched 29-Oct-2013).
[4]Grabba: http://grabba.com/products (last fetched 29-Oct-2013).
[5]Bluetooth Smart Ready: http://www.bluetooth.com/pages/Bluetooth-Smart-Devices.aspx. Fetched on Dec. 20, 2012

The Bluetooth Scatternet Formation Problem

According to the specifications [blu10], a Bluetooth device can be either *master* or *slave* when it communicates. A master along with its slaves is called a *piconet*. All communications in a piconet are controlled by the master (that is, any message in the piconet must pass by the master). The master shall use a *intra-piconet scheduling algorithm* to control the flow of messages in a piconet. As a result, master nodes consume more energy compared to slaves. The number of *active* slaves (that is, slaves that can participate in the piconet's communication) is limited to seven. More slaves are possible in a piconet if some of them are inactive (or *parked*)[6]. In such cases, the master dynamically parks and un-parks its slaves to regulate communications within the piconet. This procedure imposes a penalty on the piconet throughput, since the the slaves parked by a master m cannot directly collaborate in any of the activities of the piconet of m. Thus, maintaining the number of slaves per piconet below seven is highly desirable. A piconet that has at most seven slaves is called *outdegree limited*. In the following, a piconet with a master and slaves is modeled as a star graph. A master-slave relationship is modeled as a directed edge from the master to the slave. Thus, the number of slaves of a master is its outdegree.

A *scatternet* is an interconnection of several piconets. Scatternets are the solution for multi-hop communications in Bluetooth. A scatternet interconnects several piconets by having some nodes playing a dual role in a pair of piconets, namely, master in one piconet and slave in the other (*M/S bridge*) or slave in both (*S/S bridge*). *M/M bridges* are not allowed. That is, a node cannot be master to more than one piconet at the same time. A bridge node must schedule its time between the different piconets it belongs to, using a so-called *interpiconet scheduling algorithms*. As a result, a large number of bridges in a scatternet imposes a penalty on the throughput of the scatternet. Among those, M/S bridges impose a higher penalty than S/S bridges because an M/S bridge causes all its slaves to be inactive when it is itself inactive. It is also preferable to keep the number of piconets a node belongs to – its number of *roles* – to a minimum. A classical metric in this regard is the *average* number of roles per node, which should be minimized as

[6]Restricting the number of slaves inside a piconet to seven is related to the fact that the master and the slaves of a piconet have unique identifiers represented by three bits. The specification does not clarify why the unique identifiers of the piconet members shall be represented with three bits only. The author of this thesis believes that studying the outdegree limitation for cases other than seven slaves per piconet may lead to more understanding of the Bluetooth Scatternet Formation problem and may help in improving future versions of the Bluetooth specifications.

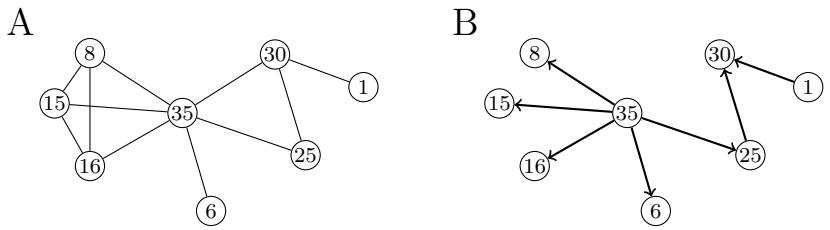


Figure 5.1: Example of a Bluetooth scatternet. The underlying network is given in (A). The nodes are represented by circles. The index inside a node is its identifier. An edge between a pair of nodes models a communication link between them (i.e., the nodes may communicate with each other). A scatternet of the network of (A) is given in (B). An arrow directed from a node v to a node u indicates that v is master of u. In (B), there are 3 piconets which are those of $35, 25$ and 1. The slaves of piconet 35 are $\{25, 6, 16, 15, 8\}$. Node 30 is an S/S bride interconnecting the piconets 1 and 35. Node 25 is an M/S bridge whose master is 35 and the set of its slaves is $\{30\}$.

well. A scatternet must guarantee that there is a path between any pair of nodes in it (i.e., it must be *connected*). Note that communication is bidirectional between each given master and its slave. A scatternet such that all its piconets are outdegree limited is called an *outdegree limited scatternet*. Limiting the outdegree of a scatternet, which is a main objective in many BSF algorithms, improves the performance of the scatternet significantly. Figure 5.1 gives an example of a scatternet.

An algorithm that forms scatternets is called a BSF algorithm. The way piconets shall be interconnected to form scatternets is not specified in Bluetooth specifications and left open to the research community. The many possible approaches as well as the number of quality metrics to be assessed on the formed scatternets make this problem challenging. One difficulty is that some of the metrics are conflicting (i.e., improving one may deteriorate another). BSF algorithms shall aim at achieving a balance between all these metrics under reasonable assumptions. These quality metrics are discussed in details in Section 5.2.1.

Contributions

This chapter reviews the BSF algorithms in the literature. The survey, given in Section 5.3, is considered as an update of the surveys found in [SZ06] and [WHC05]. The chapter's survey focuses on the following criteria when comparing BSF algorithms:

1. *The execution time of the algorithms*, from an empirical and theoretical point of view.

2. *The formation of outdegree-limited scatternets.*

3. *Achieving a balance between the scatternets quality metrics*; a BSF algorithm shall form scatternets that are efficient with respect to quality metrics other than execution time. This is a challenging problem since the main quality metrics of BSF algorithms are conflicting. That is, the improvement of one metric may lead to the degradation of another. and,

4. *Other criteria that increase the practicality of the algorithms.* For example, a BSF algorithm shall not run only in single-hop networks, shall not depend on any knowledge of nodes positions, shall not assume any knowledge of the distances between the nodes, and shall not assume the existence of a centralized node, such a server or a fixed distinct node.

Section 5.3 gives a special focus to *mesh-based BSF algorithms* since this category of BSF algorithms achieve the three main criteria given above.

As a continuation of the survey, this chapter introduces a new BSF algorithm called `Eliminate`. The algorithm is simple, fast and forms deterministically connected and out-degree limited scatternets. The algorithm improves the empirical execution time of `BlueMISI`. Algorithm `Eliminate` is a local distributed algorithm (that is, it runs in $O(1)$ communication rounds). The chapter also introduces a general approach to improve the empirical execution time of BSF algorithms.

Chapter Organization

Section 5.2 defines the BSF problem formally. A literature survey is given in Section 5.3. Section 5.4 introduces algorithm `Eliminate`. Further improvements on `Eliminate` are given in Section 5.5. Simulation experiments are studied in Section 5.6. Section 5.7 concludes the chapter.

5.2 Problem Formulation

The scatternet formation problem can be defined as follows. The input underlying network is modeled as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of n Bluetooth nodes and $\mathcal{E}$ is the set of edges. An edge $(v, u) \in \mathcal{E}$ implies that v and u can communicate

with each other by exchanging messages. This also implies that the Euclidean distance $d(v,u)$ is at most r_c, and that v and u have already *discovered* each other. More about the device discovery procedures in Section 5.5.1. The objective is to form a *scatternet* $\mathcal{S} = (\mathcal{V}, \mathcal{E}')$ such that $\mathcal{S}$ is a directed subgraph of $\mathcal{G}$ with $\mathcal{V}$ as set of nodes and $\mathcal{E}'$ as set of edges, whereas if an edge (v,u) is in $\mathcal{E}'$, then $(v,u) \in \mathcal{E}$ and $(u,v) \notin \mathcal{E}'$. The set $\mathcal{E}'$ represents the *master-slave relationships* between neighbor nodes. That is, an edge (v,u) in $\mathcal{E}'$ implies that v is the *master* of u, and u is the *slave* of v. The set of all masters (respectively, slaves) of a node v is denoted $M(v)$ (respectively, $S(v)$). The *outdegree* of a node v is the size of the set $S(v)$ (that is, $|S(v)|$). A scatternet is preferred to be outdegree-limited. That is, the outdegree of all nodes in the scatternet shall be limited to at most seven.

A *piconet* of a master v is the set $\rho(v)$ such that $\rho(v) = \{v \cup S(v)\}$. Each piconet has exactly one master. Thus, the number of piconets is equal to the number of masters in the scatternet. Each node v has a unique identifier $\mathbf{id}(v)$. A node v is said to be *larger* than node u, denoted as $v \succ u$, if the identifier $\mathbf{id}(v)$ of v is larger than $\mathbf{id}(u)$. Node u, in this case, is said to be a smaller neighbor of v. The set of larger neighbors of a node v is denoted $N_l(v)$. The set of smaller neighbors is defined similarly and denoted $N_s(v)$. See Table 5.1 for a summary of terminologies used in this chapter.

Connectivity: The formed scatternet must be connected. That is, there is a path between any two nodes in the scatternet. More accurately, the scatternet must be *weakly connected*[7] given that the scatternet $\mathcal{S} = (\mathcal{V}, \mathcal{E}')$ is modeled as a directed graph of master-slave relationships. It is often considered that a BSF algorithm does not achieve its objective if it does not form connected scatternet [SZ06] [BBMP04].

Definition 5.2.1. *An algorithm* **A** *correctly solves the BSF problem if, given a connected graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$*, it forms a connected scatternet* $\mathcal{S} = (\mathcal{V}, \mathcal{E}')$*.*

5.2.1 BSF Algorithms Performance Metrics

The quality of a BSF algorithm is measured by 1) the efficiency of the algorithm, and 2) the quality of the formed scatternets. A summary of the main quality metrics for a BSF algorithm are given in the following.

[7] A directed graph $\overline{G}$ is *weakly connected* if its undirected version G is connected. An undirected graph is *connected* if there is a path between any pair of nodes in G.

Table 5.1: Terminologies used in Chapter 5

Term	**Definition**
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	The input underlying network
$\mathcal{V}$	The set of Bluetooth nodes
n	The number of nodes (i.e., $\|V\|$)
$\mathcal{E}$	The set of edges in $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
$\mathcal{S} = (\mathcal{V}, \mathcal{E}')$	The formed scatternet
$\mathcal{E}'$	The set of edges (or, master-slave relationships) in $\mathcal{S}$ (if an edge (v, u) is in $\mathcal{E}'$, then $(v, u) \in \mathcal{E}$ and $(u, v) \notin \mathcal{E}'$)
$N(v)$	The set of neighbors of a node v
$S(v)$	The set of slaves of v
$M(v)$	The set of masters of a node v
$\mathbf{id}(v)$	The unique identifier of node v
$\rho(v)$	The piconet of a master v where $\rho(v) = \{v \cup S(v)\}$
$v \succ u$	The identifier of v is *larger* than that of u (i.e., $\mathbf{id}(v) > \mathbf{id}(u)$)
$v \prec u$	The identifier of v is *smaller* than that of u (i.e., $\mathbf{id}(v) < \mathbf{id}(u)$)
$N_s(v)$	The set of smaller neighbors of a node v (i.e., have smaller identifiers)
$N_l(v)$	The set of larger neighbors of a node v (i.e., have larger identifiers)

1. **Performance metrics to measure the efficiency of the BSF algorithm:**

   (a) *Execution time*: a short execution time of a BSF algorithm allows it to be re-executed frequently as changes occur on the underlying network, due to the mobility of devices for example. Execution time is measured theoretically and empirically. (*note:*(this chapter gives special attention to this metric.)

   (b) *The number of messages exchanged*: this is an indication of the energy consumed by the algorithm, and may be measured theoretically or empirically[8].

   (c) *The resources used by the algorithm*: shall be minimized. For instance, a decentralized algorithm is preferred over a centralized algorithm, as the latter requires the existence of centralized node and thus requires more resources. Furthermore, a BSF algorithm shall assume no knowledge of nodes positions, since this requires equipping the nodes with extra hardware. As less resources are used, the problem becomes more challenging, but at the same time, the applicability of the algorithms is increased since it can be used in more sce-

[8]Note that there are no agreed-on measure for the measurement of energy consumption in the literature of BSF algorithms.

narios.

2. **Performance metrics to measure the quality of the formed scatternets:** These metrics are considered standard metrics for BSF scatternets (see [SZ06] and [WHC05]), and were used in major BSF algorithms such as those of [PBC03] [LSW04] [WTH09] [PBC04] [VGSR05] [BBMP04] and [SWRW09].

   (a) *Outdegree limitation:* Each piconet in the formed scatternets shall not have more than seven slaves. A piconet master can keep at most seven of its slaves active at a time while all other slaves are parked. This procedure imposes a penalty on the piconet throughput and on the scatternet throughput in general. A scatternet shall eliminate all outdegree unlimited piconets, or at least avoid them as much as possible.

   Note: Forming an outdegree limited scatternets may lead to the formation of disconnected scatternets. `Bluenet` [WTH09] and `BluePleidas` [DHM+07], for example, suffer from this issue. Furthermore, a BSF algorithm that deterministically forms connected scatternets may be forced to form outdegree unlimited scatternets - as it is case with algorithm `BlueStars` [PBC03]. Thus, connectivity and outdegree limitation can be seen as conflicting quality metrics.

   (b) *The number of masters:* (i.e., piconets) shall be minimized. A large number of masters causes a scatternet to consume more energy. This is because the master execute more tasks compared to the slaves. Thus, masters consume more energy in comparison to slaves.

   (c) *The number of M/S and S/S bridges*: shall be minimized. A bridge is critical for the piconets it interconnects. For example, a bridge can be active in only one piconet at a time. As a result, a bridge must schedule itself between the piconets it interconnects to allow communications between these piconets. Inter-piconet scheduling introduces a penalty on the overall performance of the scatternet. M/S bridges have a higher penalty on the performance of scatternets because all the slaves of an M/S bridge are inactive if that bridge is inactive in its piconet. Note as well that the extra tasks a bridge performs to interconnect piconets and the extra traffic that pass via a bridge lead to a higher energy consumption at the bridges.

(d) *The average role per node*: shall be minimized. The number of roles per node is the number of piconets it belongs to. The amount of traffic passing via a bridge increases as the number of piconets that bridge belongs to. The negative impact of the intra-piconet scheduling at a bridge node becomes clearer at the number of roles of the bridge increases.

Dealing with the conflicting BSF algorithms quality metrics: The main issue of the quality metrics given above is that some of them are conflicting. That is, the improvement of one metric may lead to the degradation of another. For instance, `BlueStars` [PBC03] is BSF algorithm that forms scatternets with good qualities in a very short time, but at the same time forms scatternets with very large number of slaves (significantly more than seven slaves). In fact, the good qualities of `BlueStars` are basically caused by relaxing the outdegree limitation requirement. A serie of studies suggested solutions to fix the outdegree limitation issue of `BlueStars`, such as `BlueTree` [ZBC01], Li et al. algorithm [LSW04], `Bluenet` [WTH09], `BluePleidas` [DHM+07], and finally `BlueMesh` [PBC04] and `BlueMIS` [ZDS08]. All these algorithms, however, suffer from at least one major weakness. More details about these algorithms are given in Section 5.3.4. Achieving a balance between the scatternets quality metrics is considered hence as a major challenge in the BSF problem.

5.3 Related Work

This section introduces a brief survey of BSF algorithms. The existing BSF algorithms are categorized into four categories: 1) centralized, 2) single-hop, 3) tree-based, and 4) mesh-based BSF algorithms. Each of these categories is discussed in the following. The survey focuses on the execution time of the studied algorithms and on BSF algorithms that use as few resources as possible.

5.3.1 Centralized BSF Algorithms

Centralized BSF algorithms assume that the topology of the network are known in advance. These algorithms assume that a centralized node, which can be a special server or an elected leader node, collects information about all the nodes in the network in order to obtain such knowledge. The centralized node executes an algorithm that assigns to each

node a role (that is, master or slave) and to which piconet each node belongs to. This category includes one of the first centralized BSF algorithms to appear in the literature, called algorithm `BTCP` (Bluetooth Topology Construction Protocol) [SBTL01]. `BTCP` is a simple BSF algorithm whose centralized node is an elected leader. `BTCP` assumes that every node is in the communication range of all other nodes and it assumes that the number of nodes in the network is at most 36. The centralized node assigns to each node in the network a role such that: 1) each master has at most seven slaves, 2) each piconet is connected to every other piconet via an S/S bridge, and 3) a bridge node connects only two piconets.

Centralized algorithms may use more sophisticated role assignment algorithms due to the availability of a centralized node (e.g., evolutionary algorithms [SA04] [HWH06], linear programming optimization or others [HCS+05] [KCJ+07]). These role assignment algorithms aim at optimizing a larger number of quality metrics simultaneously. The main weakness of centralized BSF algorithms is their dependence on a centralized node, which as a result affects negatively the scalability of the algorithms. Centralized BSF algorithms, in general, do not perform well with respect to execution time.

5.3.2 Single-hop BSF Algorithms

Single hop BSF algorithms assume that all nodes in the network are in the communication range of each others. The input network thus is modeled as a complete graph (i.e., there is an edge between every pair of nodes). Many single hop BSF algorithms form scatternets structured as known graph topologies. For instance, Daptardar introduced in [Dap04] a BSF algorithm that constructs a regular mesh-like or regular cube-like scatternets. Song et al. introduced in [SLWW05] algorithm `dBBlue`, which is an algorithm that forms scatternets having the structure of the well-known de-Bruijn graph [dBE46]. Algorithm `dBBlue` uses an interesting property of de-Bruijn graph, which is its $O(\log n)$ hops diameter. Zhang et al. in [ZHS03] introduced an algorithm that forms ring-like scatternets. The nodes of the ring are piconets. A slave in a given piconet has either one role or serves as an S/S bridge to connect a pair of piconets. The scatternets formed by this algorithm have no M/S bridges.

Wang et al. in [WSL04] introduced a single-hop BSF algorithm that uses *virtual coordinates*. Each node selects a random virtual location (position) and shares it with all the other nodes. Afterwards, the nodes build a geometric structure to build the scatternet

such as a delaunay triangulation [DCvKO08], Gabriel graph [GS69], relative neighborhood graph [Tou80], Yao graph [Yao82] or a minimum spanning tree [LRSC01]. The nodes executes a distributed algorithm to be assigned a role. Barriere et al. introduced in [BFNO03] a sophisticated theoretical algorithm based on projective geometry. The main issue of the algorithm of [BFNO03] is the high level of abstraction of its scatternet formation procedure, which led to ignoring some important system specifications and high execution time in practical Bluetooth networks.

The main weakness of single-hop BSF is the necessity that every node is in the radio range of all others. This assumption reduces significantly the applicability of these algorithms.

5.3.3 Tree-based BSF Algorithms

A tree-based BSF algorithms constructs a spanning tree of the network to be used as a backbone of the scatternet. The spanning tree guarantees the scatternet connectivity. Additional links (i.e., master-slave relationships) may be added to the scatternet to improve its routing performance, as suggested in [SZ06], or to improve its fault tolerance, as suggested in [YL12] and [CMA04].

Law et al. introduced in [LMS03] a tree-based single-hop BSF algorithm. The algorithm is based on the idea of merging trees until one spanning tree is constructed, where every node is initially considered as a tree. The technique of merging trees is inspired by the work of Gallager et al. in [GHS83]. Law algorithm [LMS03] assumes that the nodes have no initial knowledge of their neighbors and that the device discovery procedure is run simultaneously with the scatternet formation procedure. That is, each node, during the execution of the algorithm, searches for undiscovered neighbors and makes itself discoverable by its neighbors. Upon the discovery of a previously undiscovered neighbor, a node executes some rules to form the scatternet, called *scatternet formation rules*. The objective of this rules is to form trees and then merge them into a single tree. The details of these rules can be found in [LMS03]. A similar single-hop tree-based BSF algorithm is introduced by Tan et al. in [TMGB02]. The main difference between the algorithms of [LMS03] and [TMGB02] is in the details of the scatternet formation rules.

Another set of tree-based BSF algorithms, such as [WSL04], [HYHB06] and [ZBC01], assume that the device discovery and scatternet formation procedures are separate. It is assumed in these algorithms that the BSF algorithm consists of at least two phases.

A device discovery procedure is executed in the first phase. After the execution of the device discovery phase, every node becomes aware of all its neighbors (or at least a subset of its neighbors). A spanning tree is then built in the next phases.

Algorithm `Bluetree` [ZBC01] uses the approach of separating the device discovery and scatternet formation procedures. The algorithm is multi-hop and has two versions. The first version of `Bluetree` assumes that there is an *initiator node* (that is, a unique node that initiates the execution of the algorithm). `Bluetree` does not introduce any method to find an initiator node. Initially, the initiator node slaves all its neighbors. The initiator node orders its slaves to become masters (i.e., become M/S bridges). These bridges slave all their neighbors that are not already slaves to other masters. This procedure continues until every node belongs to the scatternet. This procedure builds a spanning tree rooted at initiator node.

The assumption of the existence of an initiator node is considered as the major weakness of this version of `Bluetree`. The second version of `Bluetree` overcomes this issue by assigning the initiator role to multiple nodes. Essentially, a node that has an identifier larger than all its neighbors identifiers becomes initiator. There must be at least one node with this property given the uniqueness of the identifiers. Each initiator acts as defined in the first version of `Bluetree`, and builds a tree rooted at itself. This procedure shall form a forest of trees that span over all the nodes (that is, every node belongs to exactly one tree in the forest). The forest trees are then interconnected, in a second phase, into a single spanning tree. The algorithms of [DW03] [PRT04] and [YL12] introduce minor modifications on `Bluetree`.

The unit disk graph model for outdegree-limitation: `Bluetree` - in both of its version - applies simple rules to limit the outdegree of the formed scatternets. These rules assumes that the underlying network can be modeled using the unit disk graph model of wireless networks; defined below:

Definition 5.3.1. *The* unit disk graph *model of wireless networks assume that any pair of nodes in the network are neighbors if the Euclidean distance between them is at most r_c, for a constant r_c.*

A property of unit disk graphs is that, any node v has at most five neighbors that are not neighbors to each others (i.e., *independent*). The property is proven in Figure 5.2. Using this property, a `Bluetree` is able to slave at most five of its neighbors without

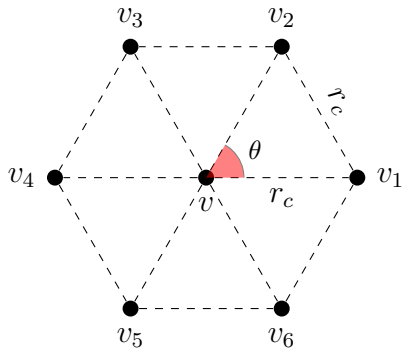


Figure 5.2: Demonstration of the independence property in unit disk graphs. The unit disk graph shown in the figure is used to prove this property using contradiction. Assume first that it is possible to place six neighbors of a node v such that all of none of them is neighbor to the others. The graph places a node v in the center surrounded by 6 neighbors, denoted $v_1, \ldots, v_6$. The distance between v and each of its neighbors is set to r_c. The idea of the proof is to place the nodes $v_1, \ldots, v_6$ as far as possible from each other so that none of them is neighbor to the other. This configuration gives a graph as shown in the figure. Take a pair of neighbors of v (e.g., v_1 and v_2) and denote the distance between them as $d(v_1, v_2)$. If the neighbors of v are meant to be independent, then $d(v_1, v_2)$ (i.e., the Euclidean distance between v_1 and v_2) is supposed to be greater than r_c. Note, however, that the angle θ formed by the edges (v, v_1), (v, v_2) and (v_1, v_2) is equal to $360/6 = 60$. As a result, $d(v_1, v_2) = r_c$ following the law of cosines. Thus v_1 and v_2 are neighbors. This gives a contradiction. Thus, at most 5 neighbors of v are not neighbors to each other (i.e., independent) in a unit disk graph.

negatively affecting the connectivity of the formed scatternet. In fact, the independence property of unit disk graphs is used by all the existing BSF algorithms that deterministically form outdegree-limited scatternets.

SHAPER: Cuomo et al. introduced in [CMA04] a multihop tree-based BSF algorithm, called `SHAPER`. The algorithm is multi-hop and mixes the device discovery and scatternet formation procedures together. `SHAPER` builds, in a first phase, a rooted spanning tree using a technique similar to the minimum spanning tree construction of Gallager et al. [GHS83]. The formed tree is a scatternet. The second phase of `SHAPER`, which is optional, adds a set of edges (i.e., master-slave relationships) to the formed scatternet to improve its performance using an optimization procedure called `DSOA` (Distributed Scatternet Optimization Algorithm). `DSOA` is a sequential algorithm that consists of heavy calculations. `DSOA` can be implemented in a distributed manner by using a token-based traversal algorithm as shown in [CMA04]. This leads, however, to an increase in the execution time of the BSF algorithm. Methfessel et al. introduced in [MPL11] a modification of `SHAPER` that overcomes some practical issues in its implementation.

Tree-based BSF algorithms main advantage is the design simplicity. A main issue in

tree-based BSF algorithms, however, is that implementing them requires sophisticated procedures. These procedures, which are necessitated by the complexity of the Bluetooth specifications, increases the execution time and communication overhead of tree-based BSF algorithms [SZ06]. Moreover, tree-like scatternets suffer from weaknesses in fault-tolerance and from nodes that are subject to larger communication overhead compared to other nodes.

5.3.4 Mesh-based BSF Algorithms

Mesh-based BSF algorithms overcome the main issues of tree-based BSF algorithms by forming mesh-like scatternets. A mesh, in this context, is defined as a structure that is not a tree. Mesh-based BSF algorithms are usually easier to implement. The majority of mesh-based BSF algorithms run on top of a device discovery algorithm. Major BSF algorithms, such as `BlueStars` [PBC03], `BlueMesh` [PBC04], `BlueMIS` [ZDS08] and `Bluenet` [WTH09], are mesh-based BSF algorithms.

BlueStars: Petrioli et al. in [PBC03] introduced a BSF algorithm called `BlueStars`. The algorithm runs in three phases. The first phase executes a device discovery algorithm. The second phase builds a set of disjoint piconets that span over all the nodes of the network, whereas the third phase interconnects the disjoint piconets into a single scatternet. The construction of the disjoint piconets in the second phase is is achieved by building a *maximal independent set* V_{mis} of the underlying network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. That is, V_{mis} is a subset of $\mathcal{V}$ such that none of the nodes in V_{mis} are neighbors to each other (independent) and there is no V_{mis} is not a subset of any independent set in $\mathcal{V}$ (maximal). Every node in V_{mis} is assigned a master role, whereas a node that is not in V_{mis} is assigned a slave role. The details of this procedure are as follow. First, every node waits to be contacted by all its larger neighbors before executing the algorithm. Therefore, the nodes that are largest in their neighborhoods initiate the algorithm. A node that has not been slaved by any of its larger neighbors attempts to slave to all its smaller neighbors. However, a `BlueStars` node can be slaved by only one master during this phase. If a node v is slaved already by a master u, then v informs all its neighbors about its new role (i.e., slave) and the identity of its master (i.e., u).

The details of the third phase of `BlueStars` are given next. In this phase, *neighbor piconets* are interconnected via bridge nodes called *gateways*. A pair of disjoint piconets

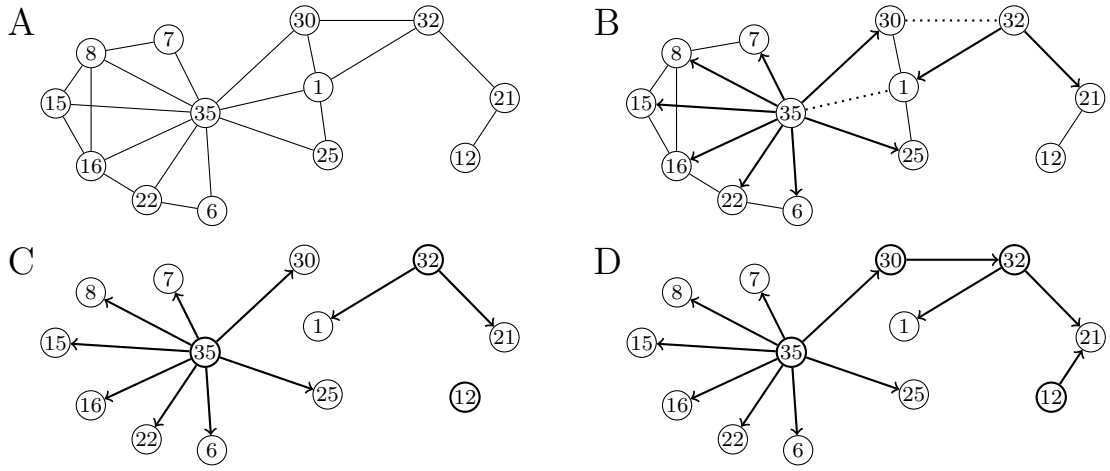


Figure 5.3: Example of `BlueStars`. The nodes that have no larger neighbors (i.e., 35 and 32) initiate the algorithm. Nodes 35 and 32 slaves all their smaller neighbors that are not slaved yet. A master-slave relationship is represented by a directed arrow. A dashed line between nodes 35 and 1 indicates that 35 attempted to slave 1 but node 1 is found to be slaved by another node, namely 32. Afterward, node 12 finds that all its larger neighbors are slaves, while it has no smaller neighbors. Therefore, node 12 becomes a master with no slaves. This completes the procedures of phase 1 with a the set of masters $\{35, 32, 12\}$. An example of a 2-hop path is $35, 30, 32$. An example of a 3-hop path is $35, 30, 1, 32$. The final scatternet is given in (D).

$\rho(m_1)$ and $\rho(m_2)$ are neighbor piconets if their masters m_1 and m_2 are separated by either 1) one slave s_x that is neighbor to both m_1 and m_2 but belongs to either $\rho(m_1)$ or $\rho(m_2)$ (2-hop path), or 2) two slaves s_1 and s_2, such that: s_1 and s_2 are neighbors, s_1 belongs to $\rho(m_1)$ and s_2 belongs to $\rho(m_2)$ (3-hop path). The masters of each pair of neighbor piconets select, using unambiguous local rules, a unique gateway or a unique pair of gateways to interconnect the piconets. Figure 5.3 gives an illustrative example of algorithm `BlueStars`.

Theorem 5.3.2. *The time complexity of* `BlueStars` *is $O(n)$, where n is the number of nodes in the network.*

Proof. The most costly procedure in `BlueStars` - time-wise - is the procedure of forming disjoint piconets (i.e., the first phase). Node v_i *waits*, in the first phase, for a neighbor v_j if v_j is a larger neighbor of v_i. Assume that v_j itself waits for its larger neighbor v_k. Thus, v_i also waits for v_k. This forms a waiting chain $v_i \succ_w v_j \succ_w v_k$, where $v \succ_w u$ indicates that v *waits* for u. A waiting chain that consists of n nodes is easily formed such that $v_1 \succ_w \ldots, \succ_w v_n$. This occurs, for example, if the input graph is a line graph such that v_i is connected to v_{i+1} for $1 \leq i < n$, and $v_j \succ v_i$, for $1 \leq i < j \leq n$. In this

case, the messages flow must start from the largest nodes to the smallest ones. Each message transmission takes one time unit. This leads in consequence to at most n time units executed at the first phase.

In the second phase, there is a message exchange 1) between the slaves and masters (in order for the masters to identify the neighbor piconets), 2) between the masters and gateways (to inform the gateways which piconets they must interconnect), and 3) between the gateways and neighbor gateways (to interconnect the neighbor piconets). This can be done in at most 3 communication rounds. Therefore the time complexity of `BlueStars` is $O(n)$. □

The weakness of `BlueStars` (and how to solve it): The simplicity of `BlueStars` and its short execution time are its main advantages. The main issue of `BlueStars` is that it does not limit the outdegree of the scatternets. (See the example of Figure 5.3). Many approaches were proposed to solve the outdegree problem of `BlueStars`. For instance, Li et al. in [LSW04] assumed that the nodes are aware of their geographic locations and that the underlying network can be modeled as a unit disk graph. Given these assumptions, the nodes can form a geometric degree-limited structure such as Yao graph [Yao82]. The algorithm of [LSW04] suggests to run `BlueStars` over the formed degree-limited structure. This guarantees in return that the scatternet is outdegree-limited. Another approach to solve the outdegree limitation problem of `BlueStars` is the use of randomization. For instance, Dubhashi et al. suggests in [DHM$^+$07] that a degree-limited overlay is formed such that each node connects to at most c of its neighbors, where $c = 5, 6$ or 7. It is shown in [DHM$^+$07] that this procedure forms a network that is degree limited to at most c and is connected with a high probability under certain conditions. `BlueStars` is then executed over the degree-limited formed network. `Bluenet` [WTH09] is another algorithm that follows the randomization approach to solve the outdegree limitation problem of `BlueStars`. `Bluenet` [WTH09] and the algorithm of [WTH09] trade off connectivity with outdegree limitation. The numerous algorithms introduced to solve the outdegree limitation problem suggest that it is challenging to achieve connectivity and outdegree limitation when forming Bluetooth scatternets without additional simplifying assumptions or without suffering from negative side effects. This is true for the algorithms introduced in the following, which all deterministically solve the outdegree limitation problem but suffer at the same time from drawbacks.

BlueMesh: Basagni et al. introduced in [PBC04] an algorithm, called `BlueMesh`, that deterministically solves the outdegree limitation issue of `BlueStars`. The algorithm guarantees connectivity and outdegree limitation given that the underlying network is modeled as a unit disk graph (see Definition 5.3.1). `BlueMesh` also balances between many conflicting scatternet quality metrics. Nevertheless, theoretical analysis and simulation results show that the execution time of `BlueMesh` is significantly higher than that of `BlueStars`. Algorithm `BlueMesh` is described in the following.

First, each node discovers all its neighbors. The algorithm then runs in iterations. Each iteration i is executed over a network denoted G_i, where G_i is an induced network of G_{i-1}, and G_0 is the input network[9]. In each iteration, a maximal independent set of the underlying network V_{mis} is built. The set V_{mis} is constructed using a technique similar to that introduced in `BlueStars` [PBC03]. Each node v in V_{mis} slaves a subset $S(v)$ of its *promising slaves* set $S_p(v)$. The set $S_p(v)$ consists of the smaller neighbors of v and the set of the larger neighbors of v that were slaved by other nodes. The set $S(v)$ has the following properties: 1) its size is at most seven, 2) every node in $S_p(v)$ is either in $S(v)$ or a neighbor to a node in $S(v)$ (that is, $S(v)$ is an independent set of $S_p(v)$), and 3) $S(v)$ is not a a subset of any subset of $S_p(v)$ that has the first two properties. Contrary to `BlueStars`, a `BlueMesh` node may slave a neighbor that has been already slaved by another node.

A pair of piconets $\rho(m_1)$ and $\rho(m_2)$ are neighbors at iteration i if: 1) $\rho(m_1)$ and $\rho(m_2)$ share one slave or more (thus, $\rho(m_1)$ and $\rho(m_2)$ are called *connected neighbor piconets*), 2) if there is a pair of slaves s_1 in piconet $\rho(m_1)$ and s_2 in piconet $\rho(m_2)$, called *gateways*, such that s_1 and s_2 are neighbors in the graph G_i (thus, $\rho(m_1)$ and $\rho(m_2)$ are called *disconnected neighbor piconets*). At iteration i, the masters of every pair of neighbor piconets that are strictly disconnected select a unique pair of gateways using a unambiguous rules. These gateways enters iteration $i+1$. The network induced by all gateways entering iteration $i+1$ is denoted G_{i+1}. The algorithm terminates if the network G_{i+1} contains no nodes. An illustrative example of `BlueMesh` is given in Figure 5.4.

The time complexity of a `BlueMesh` iteration is $O(n)$. This can be derived following the techniques of Theorem 5.3.2. The number of iterations of `BlueMesh` shall be derived in Theorem 5.3.3. A theoretical analysis of this aspect of `BlueMesh` is first studied here.

[9]A subgraph G' of G is *induced* if every edge (v,u) is in G' if and only if v and u are in the vertex set of G' and (v,u) is in G.

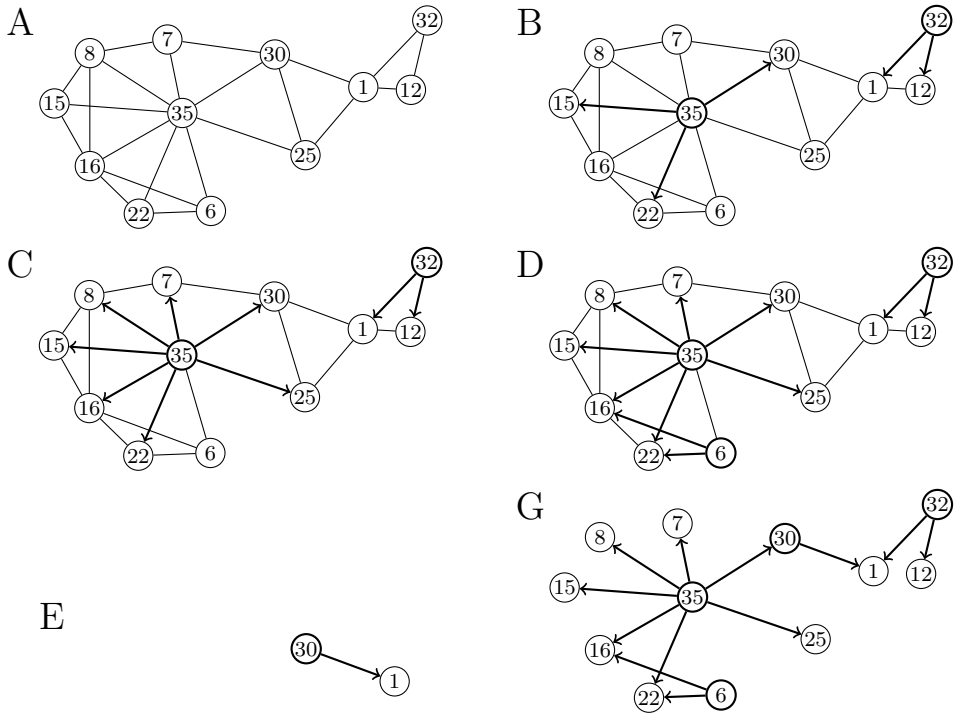


Figure 5.4: Example of `BlueMesh`. The algorithm is initiated by the nodes that have no larger neighbors (i.e., 35 and 32). The set of promising slaves of of 32 (i.e., $S_p(32)$) is of size less than seven. Thus, 32 slaves all of them (master-slave relationships are shown in directed thick arrows). Node 35 has more than seven promising neighbors (i.e., $S_p(35)$). The set $S(35)$ is selected to be $\{30, 22, 15\}$. The set $S(35)$ is a maximal independent set of $S_p(35)$ since each node in $S(35)$ is not a neighbor to any other in the same set. Node 35 may add more nodes to $S(35)$ as long as its size does not exceed seven. Let say therefore that $S(35) = \{30, 22, 15, 25, 16, 8, 7\}$. Node 6 finds that it is not slaved by any of its larger neighbors. This occurs after node 6 is contacted by all its larger neighbors, which are $35, 22$ and 16. The promising slaves of node 6 are 16 and 22. Both of these nodes are slaved by 6. Therefore, there are three piconets with the masters $35, 32$ and 6. Piconets 35 and 6 are connected neighbor piconets since they share the slaves 22 and 16. The piconets 35 and 32 are interconnected via 30 and 1 - and hence they are disconnected neighbor piconets. Nodes 30 and 1 therefore move to the next iteration and form the induced graph G_1, shown in (E). Node 30 starts the next iteration of `BlueMesh` and slaves its only promising neighbor, node 1. The algorithm terminates, and the final scatternet is shown in (G). Since piconets 35 and 6 share two slaves, it is possible that one of the masters free one of their shared slaves and hence share only one slave. This, however, is an optional procedure.

Theorem 5.3.3. *The number of iterations executed by* `BlueMesh`*, in the worst case, is* $O(\log n)$ *in arbitrary graphs and* $O(1)$ *in unit disk graphs, where* n *is the number of nodes in the network.*

Proof. The worst case scenario is built as follows. Start with the simple network of two nodes v and u linked by the edge (v, u). Assume that nodes v and u are the only surviving nodes in iteration k, where k is the index of the last phase of the algorithm.

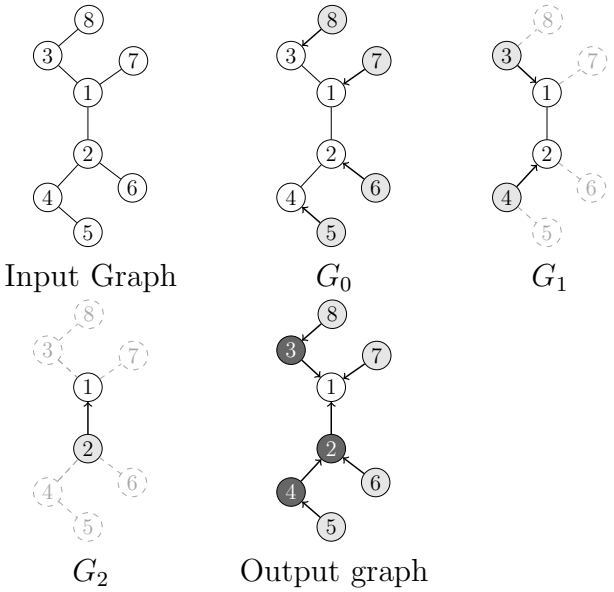


Figure 5.5: Worst case example of `BlueMesh`.

If a node v survived iteration $k-1$ and moved to iteration k, then it must had a larger neighbor v'. This means that there are at least 4 nodes v, v', u and u' in iteration $k-1$. As a result, the maximum number of nodes that move to iteration i is $|V_{i-1}|/2$, where $|V_i|$ is the number of nodes in iteration i. Therefore, the maximum number of phases is at $O(\log |V_0|) = O(\log n)$. The worst case scenario of `BlueMesh` is shown in Figure 5.5.

Note that if a node has more than 5 neighbors in a unit disk graph then at least two of them are also neighbors. Following the previous argument, if a node survived k iterations then it must have at least k largest neighbors that are not neighbors to each other. This means that k is at most 5 in unit disk graphs. Therefore, the maximum number of `BlueMesh` iterations if run over unit disk graphs is $O(1)$. □

Although the number of `BlueMesh` iterations in arbitrary graphs is $O(\log n)$, the time complexity of `BlueMesh` is $O(n)$. This is because the number of nodes executing any given iteration i is at most $n_i = n/2^i$. Since each iteration with n_i nodes is executed in at most n_i communication rounds, then the total number of communication rounds is at most:

$$\sum_{i=0}^{\log n} n_i = n \sum_{i=0}^{\log n} (\frac{1}{2})^i \in O(n)$$

`BlueMesh` suffers from a long execution time compared to other major mesh-based BSF algorithms. This is shown by simulation experiments in Chapter 6. The cause of this result is that the empirical execution time of `BlueMesh` (and most other BSF algorithms) is affected mainly by the number of communication rounds executed. Thus, an algorithm that executes a large number of communication, even if it is constant, has in consequence a long execution time. `BlueMesh` is among the BSF algorithms that runs a large number of communication rounds. The long execution time of `BlueMesh` is the cost of achieving deterministic connectivity and outdegree-limitation.

M-dBBlue: Song et al. introduced in [SWRW09] algorithm `M-dBBlue`. The algorithm idea is to build a connected dominating set V_{cds} from the input network[10]. The algorithm does not guarantee outdegree limitation, but the authors gave theoretical upper bounds for the formed scatternets maximum outdegree if the input graph is a unit disk graph. The algorithm is based on a tree-based procedure to construct a dominating set. This procedure suffers from a high communication overhead.

BlueMISI [ZDS08]: is another mesh-based BSF algorithm that guarantees connectivity and outdegree limitation. `BlueMIS` assumes that the underlying network is modeled as a unit disk graph. `BlueMIS` runs in two phases, called `BlueMISI` and `BlueMISII`. The first phase forms in a time-efficient manner a scatternet that is not necessarily efficient with respect to all quality metrics. `BlueMISII` improves the scatternet formed in the first phase by applying simple heuristics. A description of `BlueMISI` is given next. `BlueMISI` is similar to a well-known topology control algorithm for wireless ad-hoc networks, called `XTC` [WZ04]. Each node v in `BlueMISI` constructs a subset $S(v)$ of its neighbors $N(v)$ such that $S(v)$ is the set of slaves of v and $S(v)$ is maximal independent set of $N(v)$. That is, $S(v)$ is an independent subset of $N(v)$ (i.e., no pair of nodes in $S(v)$ are neighbors to each other) and $S(v)$ is not a subset of any independent subset of $N(v)$. The set $S(v)$ is constructed as follows. Node v contacts its neighbors in order from the smallest neighbor to the largest one, whereas the nodes are totally ordered according to their unique identifiers or any other total order. Node v adds a neighbor u into $S(v)$ if u is not a neighbor to any node already in $S(v)$. Since $S(v)$ is the set of slaves of v, a symmetry case may occur if a pair of neighbors v and u selected themselves

[10] A connected dominating set V_{cds} of a graph $G = (V, E)$ is a subset of V such that any node is either in V_{cds} or a neighbor to a node in V_{cds}, and such that the induced graph of V_{cds} is connected.

Algorithm 5.1 `BlueMISI` at node v

1: $D \leftarrow \emptyset$, $U \leftarrow N(v)$, $S(v) \leftarrow \emptyset$, $M(v) \leftarrow \emptyset$, where $S(v)$ and $M(v)$ are the sets of slaves and masters of v respectively.
2: **while** $U \neq \emptyset$ **do**
3: $u \leftarrow \min(U)$
4: send a message to u, call the message *attack*
5: $S(v) \leftarrow \{S(v) \cup u\}$
6: $U \leftarrow \{U \setminus \{u \cup N(u)\}\}$
7: $D \leftarrow \{D \cup \{N(v) \cap N(u)\}\}$
8: **for** each $u \in D$ **do**
9: send a message to u, call the message *dummy*
10: **if** received an *attack* message from u **then**
11: $M(v) \leftarrow \{M(v) \cup u\}$
12: **if** received messages (*attack* or *dummy*) from all neighbors **then**
13: locally break symmetry in $M(v)$ and $S(v)$
14: terminate

as masters and slaves to each other at the same time (that is, $u \in S(v)$ and $v \in S(u)$). These symmetries can be broken using an unambiguous local rule (e.g., if $v \succ u$, then v becomes the master of u and u becomes the slave of v). The pseudocode of `BlueMISI` is given in Algorithm 5.1. An illustrative example of `BlueMISI` is given in Figure 5.6.

Remark 5.3.4 (`BlueMISI` correctness)**.** *The construction of $S(v)$ at any* ***BlueMISI*** *node v shall follows an order common to all other nodes in the network. The unique identifiers of nodes is an example of such common order. That is, a node v inserts into $S(v)$ the neighbor u that is the* minimum *in $\{N(v) \setminus S(v)\}$ such that u is not neighbor to any node in $S(v)$. This is repeated until there is no neighbor with such properties. Let us assume that v defined the* minimum *neighbor as the neighbor with smallest identifier. Let us assume that another node w gave a different definition of the* minimum *neighbor (e.g., the neighbor with largest identifier). Then, this scenario may lead to a disconnectivity in the formed scatternet. The necessity of having a common order among all nodes is found to be an issue that degrades the execution time of* ***BlueMISI****. This is one of the factors that explain why* ***BlueStars*** *execution time is shorter than* ***BlueMISI****'s. More details are given in the simulation experiments in Section 5.6.*

Theorem 5.3.5. ***BlueMISI*** *runs in $O(1)$ communication rounds. Thus,* ***BlueMISI*** *is a local distributed algorithm.*

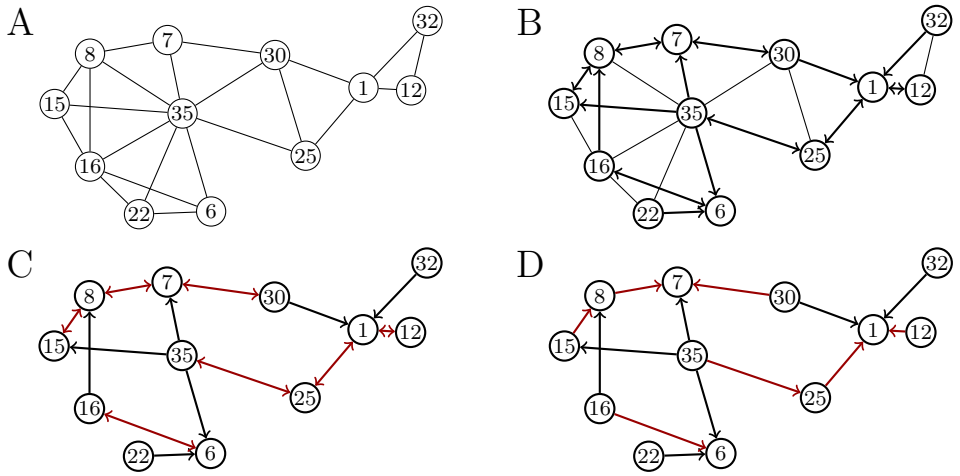


Figure 5.6: Example of `BlueMISI`. The input network is shown in (A). In (B), a directed arrow is drawn from each node v to each $u \in S(v)$. For instance, the set $S(35)$ is built as follows. Node 35 selects its smallest neighbor, node 6, and inserts it into $S(35)$. With this insertion, the neighbors 22 and 16 cannot be inserted into $S(35)$ any more since they are neighbors to both 6 and 35. Afterward, the smallest neighbor in the set of promising slaves is 7, and hence it is inserted into $S(35)$. With the insertion of 7, the nodes 8 and 30 cannot be inserted into $S(35)$ in the future. Node 15 and 25 are the only remaining promising slave of 35. Both are inserted into $S(35)$ as they are not neighbors to each other. Thus, $S(35) = \{6, 7, 15, 25\}$. Note that $25 \in S(35)$ and $35 \in S(25)$. This is a symmetry that shall be removed. (C) shows all symmetries in red arrows. To break this symmetry, the node with larger identifier becomes the master while the node with the smaller identifier becomes the slave. This is shown in (D). The network given in (D) is the formed scatternet. A directed arrow represent a master-slave relationship.

The locality of `BlueMISI`: according to the knowledge of the thesis author, `BlueMISI` is the only existing local distributed BSF algorithm[11]. This is the main advantage of `BlueMISI`. Its disadvantage is the large number of piconets in the formed scatternets.

`BlueMISII`: is an optional phase of `BlueMIS` that aims to improve the quality of the scatternets formed by `BlueMISI` using simple rules. A main objective of these rules is to reduce the number of piconets in `BlueMISI` scatternets. These rules preserve the connectivity and outdegree limitation of the formed scatternets as proven in [ZDS08]. Some implementation details, however, are not included in the description of these rules in [ZDS08] which causes as a result different possible interpretations. For instance, there could be cases where a node cannot execute its rules until some other nodes, and possibly all other nodes, execute their rules in order to achieve an outdegree limited scatternets. A token-based algorithm that forces each node to execute its rules at a time is a possible

[11] `Eliminate`, introduced in Section 5.4, is another local distributed BSF algorithm.

Algorithm 5.2 compute $MIS_s(v)$ at node v

$U \leftarrow N_s(v)$, $MIS_s(v) \leftarrow \emptyset$
while $U \neq \emptyset$ **do**
  $u \leftarrow \max(U)$
  $MIS_s(v) \leftarrow \{MIS_s(v) \cup u\}$
  $U \leftarrow \{U \setminus \{u \cup N(u)\}\}$
return $MIS_s(v)$

- but not time-efficient - solution to this problem. Another solution is to let each node execute its rules independent of all other nodes in the network. This solution has short execution time, but at the same time causes the formed scatternets to have outdegree unlimited piconets. The simulation experiments in Chapter 6 prefer the second solution given that the execution time metric, according to this thesis, is more prior compared to the outdegree limitation metric.

5.4 Algorithm `Eliminate`

This section introduces a new BSF algorithm, called `Eliminate`. The algorithm is a modification of `BlueMISI` that improves its average execution time. Simulation experiments in Section 5.6 show that `Eliminate` form scatternets with better qualities compared to those formed by `BlueMISI`.

The idea of `Eliminate` is that each node v constructs a maximal independent set of its smaller neighbor $N_s(v)$, denoted $MIS_s(v)$. Each node v slaves all its neighbors in $MIS_s(v)$. This is contrary to `BlueMISI` where each node v slaves neighbors found in a maximal independent set from all the neighbors of v. The set $MIS_s(v)$ is constructed as shown in Algorithm 5.2. Let δ be the maximum degree in the network. Let δ_s be the maximum number of smaller neighbors for all nodes in the network. Since $\delta_s \leq \delta$ on average, then the average time complexity of `Eliminate` is less than that of `BlueMISI`. Both algorithms, however, have the same worst-case time complexity.

The rules of `Eliminate` guarantee that there are no symmetries occurring from a pair of neighbor nodes v and u being masters and slaves to each other simultaneously. This is because a node v can be master only to a smaller neighbor u found in $MIS_s(v)$. Therefore, u cannot be the master of v given that u is smaller than v, and no node is allowed to slave a larger neighbor. This means that a node v may know all its slaves

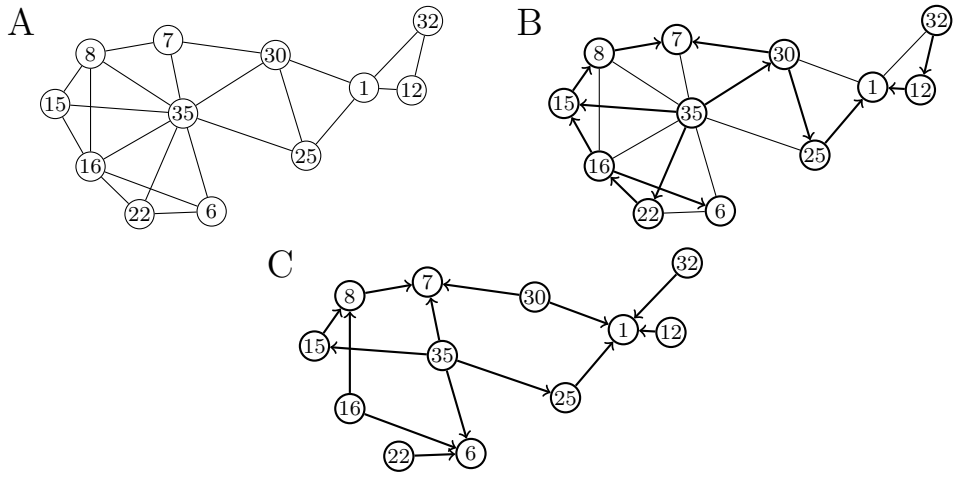


Figure 5.7: Example of `Eliminate`. The input network is given in (A). In (B), each node v constructs a maximal independent set of all its smaller neighbors, denoted $MIS_s(v)$. A directed arrow from a node v to a neighbor u indicates that $u \in MIS_s(v)$. For example, $MIS_s(30) = \{25, 7\}$ and $MIS_s(16) = \{15, 6\}$ and $MIS_s(6) = MIS_s(1) = \emptyset$. Each node in $MIS_s(v)$ is slaved by v. The formed scatternet is shown in (C).

$S(v)$ if it has its 2-hop neighborhood list. This 2-hop neighborhood list may be obtained using a single communication round. However, a node v may not know all its masters $M(v)$ using only its 2-hop neighborhood list. Thus, the construction and slaving of the nodes of $MIS_s(v)$ shall be done on-the-fly, such that: 1) the smaller neighbors of v are contacted in order, and 2) a smaller neighbor u in $N_s(v)$ is inserted into $MIS_s(v)$ and slaved if u is not already in $MIS_s(v)$.

The rules of `Eliminate` guarantee that the formed scatternet is connected if the underlying network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is connected. Also, the formed scatternets are guaranteed to be outdegree limited assuming that the underlying network is modeled as a unit disk graph. This is formally proven in Theorem 5.4.1 and Theorem 5.4.2.

Theorem 5.4.1. *`Eliminate` forms a connected scatternet if the underlying network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is connected.*

Proof. The proof is based on showing that a scatternet formed by `Eliminate` contains a minimum spanning tree if the edges are ordered such that $(x_1, y_1) \succ (x_2, y_2)$ if $y_2 \succ y_1$ or if $y_2 = y_1$ and $x_2 \succ x_1$.

An edge (v, w) is deleted from a scatternet formed by `Eliminate` only if (v, w) belongs to a triangle of edges (v, u), (u, w), and (v, w) in $\mathcal{G}$ such that: $v \succ u \succ w$ and hence $(v, u) \succ (u, w) \succ (v, w)$. The edge (v, w) is never included in a minimum spanning tree

of $\mathcal{G}$ since it the largest edge in a circle of 3 edges in $\mathcal{G}$. As a result, `Eliminate` scatternet contains a minimum spanning tree of $\mathcal{G}$.

□

Theorem 5.4.2. *`Eliminate` forms an outdegree limited scatternet with maximum outdegree of* 5 *if the underlying network* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *is a unit disk graph.*

Proof. Observe that each node v creates a maximal independent set of its smaller neighbors $N_s(v) \subseteq N(v)$. If $\mathcal{G}$ is a unit disk graph, then the size of the maximal independent set of $N(v)$, and hence $N_s(v)$ does not exceed 5. □

5.5 Further Improvements on `Eliminate` Execution Time

Many distributed algorithms, including `BlueMISI` and `Eliminate`, consist of a sequence communications rounds [Lyn96], where in a communication round each reader is able to:

1. receive a message from all its neighbors,
2. perform local computation, and
3. send a message to all its neighbors.

To improve the execution time of such algorithms, either 1) the number of communication rounds is decreased or 2) the execution time of a communication round is decreased (i.e., by introducing improved implementations of communication rounds). This section follows the second approach to improve the execution time of `Eliminate` and `BlueMISI`.

This section studies two implementations of communication rounds in Bluetooth networks, called `OrderedExchange` and `RandomExchange`. These implementations exist already in the literature but were never studied in depth. Simulation experiments show that `OrderedExchange` is faster than `RandomExchange` in Bluetooth networks; although `RandomExchange` is theoretically better. The main disadvantage of using `OrderedExchange` with a given BSF algorithm is that some changes on the algorithm may be necessary. Note that these modifications do not change the final output of the algorithm. For example, using `OrderedExchange` with `BlueMISI` necessitates modifications on the algorithm; however, using `RandomExchange` to implement `BlueMISI` communication rounds would lead to no modifications on the algorithm since it implicitly

uses `RandomExchange`. A case study in Appendix A shows how there are at least two implementations of `BlueMISI` using `OrderedExchange`. These implementations, called `ComputeMISI` and `ComputeMISII`, significantly improve the execution time of `BlueMISI` as shown in Section 5.6.

A brief background knowledge on the device discovery and link establishment procedures of Bluetooth is given in the following section. The details of `OrderedExchange` and `RandomExchange` are discussed afterward.

5.5.1 Bluetooth: Basics of Link Establishment

Bluetooth is a wireless technology that uses the 2400-2480 MHz Industrial Scientific and Medical (ISM) radio bands. These bands are divided into 79 channels (1 MHz each). Bluetooth devices use the Frequency Hopping Spread Spectrum (FHSS) technique for communication. A pair of nodes communicating with each other alternates between a sequence of pseudo-random frequency channels known to both nodes. The nodes exchange their messages during this alternation. The pseudo-random sequence of frequency channels that a node follows is generated by its clock index.

Device Discovery Procedures: for a node to discover a neighbor, it must switch to a state called `INQUIRY`. The inquirer node broadcasts small packets, called the `ID` packets, in different channels to announce its existence. A node that wants to be discovered, called the scanner, switches to a state called `INQUIRY SCAN`. The scanner alternates pseudo-randomly between a set of channels. If a scanner receives one of the packets of an inquirer, it sends back a packet to the same inquirer. The two devices exchange some packets then. Among the data exchanged between the inquirer and the scanner is the clock indexes of both devices. A node uses the clock index of its neighbor node in order to generate the pseudo-random sequence of the frequency channels that it follows. This additional knowledge simplifies future discovery of neighbor nodes (if needed), and the link establishment procedure. The procedure of discovery is terminated thereafter.

Link Establishment Procedures: Bluetooth is a *connection-oriented* communication standard. That is, any pair of neighbor nodes must build a link (or, a connection) before exchanging messages. For a node v to establish a link with a neighbor u, node v switches to a state called `PAGE` while node u switches to a state called `PAGE SCAN`. Node

v sends packets specifically designated to u in different channels. Node u on the other hand alternates between a sequence of frequency channels generated from its clock index. Node v sends packets specifically to u in the frequency channels sequence of u, since the clock index of u is already known to v. This is done to increases the speed of finding node u. If u received a packet from v, then both nodes exchange some packets in order to configure the link. The link establishment procedure is terminated at this point. Note that a link between a pair of nodes v and u represents a piconet of one master and one slave. The master in this example is v while the slave is u.

According to the Bluetooth specifications, for any pair of Bluetooth devices to communicate with each other, they both need to be in the same scatternet or the same piconet. Given the unavailability of scatternet before a BSF algorithm is executed, most BSF algorithms uses the following technique to exchange messages in order to build the scatternets. If a node v needs to send a message to a neighbor u, then v build a temporary piconet with u (using the link establishment procedures given above). Both nodes exchange messages as needed before destroying the piconet. Any standard forwarding technique may be used to send a messages from a node v to one if its non-neighbor nodes in the network.

5.5.2 `OrderedExchange` and `RandomExchange`

The main difficulties in the implementation of a communication round in a Bluetooth network are the following:

1. A Bluetooth node cannot broadcast a message to all its neighbors simultaneously. This is because Bluetooth is a connection-oriented communication technology, and

2. A node v must be in the `PAGE` state while a node u must be in the `PAGE SCAN` state if v needs to establish a link with u.

Note that the main objective of a communication round is to allow each node to send a message to all its neighbors and receive a message from all its neighbors (if needed) during the round. Therefore, a communication round may be defined as an algorithm that guarantees that, given a network $G = (V, E)$, every edge $(v, u) \in E$ is visited by a message exactly once during the execution of the algorithm. The message that visits an edge (v, u) is not necessarily the same as the message that visits (u, v), and it may be even an empty message.

`RandomExchange` and `OrderedExchange` are two implementations (or, algorithms) that takes into consideration the difficulties given above, while using the definition of communication rounds given above.

A description of `RandomExchange` is given in the following. Each node v alternates randomly between the `PAGE` and `PAGE SCAN` states. The time during which a node resides in each of the states is uniformly random. When a node v is in the `PAGE` state, it attempts to send a message to a neighbor u that is not yet contacted. If the attempt of v is successful, then the edge (v, u) is said to be *visited* and u is contacted by v. The attempt of v may fail if node u is in the `PAGE` state while v is attempting to contact it. The attempt may also fail if u is being contacted by another node w while v is trying to contact u. When v is in the `PAGE SCAN` state, it waits to receive messages from neighbors that are not yet visited.

A pair of neighbor nodes v and u may have, in the worst case, the same alternation sequence between the `PAGE` and `PAGE SCAN` states. As a result, the edges (v, u) and (u, v) are never visited, and hence `RandomExchange` never terminates. This *deadlock* scenario occurs with a low probability in practical Bluetooth networks. Under the assumption of no deadlocks occurring, `RandomExchange` time complexity is $O(\Delta)$ delay units, where Δ is the maximum degree of the network G.

`OrderedExchange` does not suffer from deadlocks and terminates deterministically in a finite time. Every `OrderedExchange` node v waits to receive a message from all its larger neighbors $N_l(v)$ before sending a message to all its smaller neighbors $N_s(v)$. The nodes that have no larger neighbors initiate the communication round execution. As a result, each edge $(v, u) \in E$ such that $v \succ u$ is visited exactly once. If (v, u) is assumed to be *not* equivalent to (u, v), then another communication round is required. The second communication round, however, is initiated by the nodes that have no smaller neighbors. Every node, in the second round, waits for all its smaller neighbors and sends a message to all its larger neighbors. The first communication round of is called a *descending* communication round since messages are sent from the nodes with larger identifiers to those with smaller identifiers. The second communication round is called an *ascending* communication round. The pseudocode of `OrderedExchange` is given in Algorithm 5.3. An illustrative example of `OrderedExchange` is given in Figure 5.8

Algorithm 5.3 `OrderedExchange` at node v (descending)

1: $U \leftarrow N_l(v)$
2: **while** $U \neq \emptyset$ **do**
3: upon reception of a message from neighbor $u \in U$, $U \leftarrow \{U \setminus u\}$
4: **for** each $u \in N_s(v)$ **do**
5: send a message to u

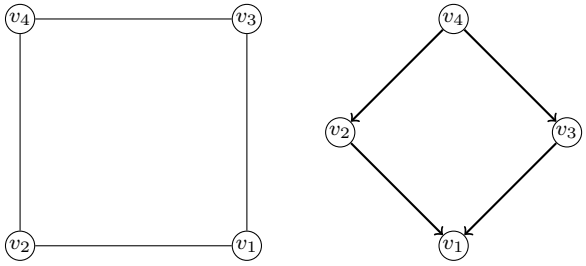


Figure 5.8: Example of `OrderedExchange`. Consider the graph $G = (V, E)$, where $V = \{v_4, v_3, v_2, v_1\}$ and $E = \{(v_4, v_3), (v_4, v_2), (v_3, v_1), (v_2, v_1)\}$. Assume that the order of the nodes is $v_4 \succ v_3 \succ v_2 \succ v_1$. The only node in G that has no larger neighbors is v_4. The nodes v_3, v_2 and v_1 wait for their larger neighbors to send them messages, while node v_4 starts sending messages to all its smaller neighbors (that is, v_2 and v_3). Therefore, edges (v_4, v_3) and (v_4, v_2) are visited. Upon the reception of v_4 messages, v_3 and v_2 send a message to v_1 which is the only smaller neighbor to both nodes. That is, the edges (v_2, v_1), (v_3, v_1) are visited. Thus, every edge in E is contacted exactly once. The execution of one descending round of `OrderedExchange` is thus terminated.

5.5.3 `RandomExchange` vs. `OrderedExchange`: Simulation Experiments

The average execution time of `RandomExchange` and `OrderedExchange` is studied in this section. Both algorithms are fine-tuned to achieve the optimal performance. For instance, `RandomExchange` is affected by the length of the period of time a node resides in either `PAGE` or `PAGE SCAN`; and hence several simulation experiments were performed to achieve the best value of this parameter. On the other hand, `OrderedExchange` is affected by the order a node sends message to its neighbors. For instance, assume the the smaller neighbors of a node v are $u_1, u_2 \ldots, u_k$, where $u_i \succ u_j$ if $i > j$. Node v may contact its smaller neighbors following different orders such as: 1) an ascending order (that is, the sequence $u_1, u_2, \ldots, u_k$), 2) a descending order (that is, the sequence $u_k, \ldots, u_2, u_1$), or 3) any other order (random order for example). The execution time of a descending `OrderedExchange` depends on the order followed to contact the smaller neighbors. If a descending order is followed, then the average time complexity is $\Theta(n)$. The average

Table 5.2: Percentage of improvement of `OrderedExchange` over `RandomExchange`.

number of nodes / degree	5	10	15	20	25
30	91.9	74.6	69.7	72.3	77.2
50	89.0	63.7	60.5	62.7	65.2
70	89.4	60.7	62.9	56.3	57.3
90	88.2	63.0	55.1	48.4	50.6
110	84.6	66.9	55.0	52.1	45.0

time complexity may reach $\Theta(n^2)$ if an ascending or a random order are followed. To achieve these time complexities, it is assumed that a node can send only one message at a time slot, but may receive n messages simultaneously. It is found that this model is the best to explain the simulation results. More details can be found in [JJZ12].

`RandomExchange` execution time depends on the average nodal degree whereas `OrderedExchange` depends on the number of nodes[12]. The networks used for simulation are modeled as connected as follows. The nodes are modeled as points uniformly and randomly placed in a rectangular plane of length and width of 30 units each. An edge between any pair of points, or nodes, is set if the Euclidean distance between them is at most 10 units. The simulation experiments are executed using the `UCBT` [Wan].

Table 5.2 computes the ratio $((T_{rnd} - T_{ord})/T_{ord}) \times 100$, where T_{rnd} and T_{ord} are the averages of the execution time of `RandomExchange` and `OrderedExchange` respectively. The results of Table 5.2 shows that `OrderedExchange` outperforms `RandomExchange` in all the networks considered. The results suggest that if the nodal degree is fixed and the number of nodes is extrapolated to very large value then `RandomExchange` will outperform `OrderedExchange` at some point. This makes the simulation results match the theoretical analysis, since theoretical analysis (that is, complexity analysis) assumes that the input size (i.e., the number of nodes) is large. Bluetooth networks, in practice, are rarely found in very large number of nodes. Thus, it is more practical to use `OrderedExchange` to implement communication rounds in Bluetooth networks.

5.5.4 `Eliminate` and `BlueMISI` using `OrderedExchange`

The simulation experiments of Section 5.5.3 shows that the use of `OrderedExchange` improves the execution time of a communication round compared to `RandomExchange`.

[12]For a better accuracy, `OrderedExchange` time complexity depends on the longest path $\pi = \{v_1, \ldots, v_k\}$ in the network such that $v_i \succ v_j$ for every $1 \leq j < i \leq k$. More details can be found in [JJZ12].

Therefore, `OrderedExchange` is used to implement the communication rounds of `Eliminate`. This is done by letting every node v builds the maximal independent set of its smaller neighbors (i.e., $MIS_s(v)$) only after being contacted by all its larger neighbors. Building $MIS_s(v)$ is done on-the-fly. That is, it is not necessary to gather the 2-hop neighborhood before building $MIS_s(v)$. Appendix A shows how `OrderedExchange` is used to improve the execution time of `BlueMISI` by introducing two implementations called `ComputeMISI` and `ComputeMISII`.

5.6 Simulation Experiments

This section studies the performance of `Eliminate` and compares it against similar BSF algorithms; namely, `BlueMISI` and `BlueStars`. The experiments consider three versions of `BlueMISI`; two of which are implemented using `OrderedExchange` and called `ComputeMISI` and `ComputeMISII` (see Appendix A for details), whereas the third is implemented using `RandomExchange` as specified in [ZDS08]. The simulation is performed using the `UCBT` [Wan] simulator. As in Section 5.5.3, the networks used for the simulation are built by placing uniformly randomly nodes in a rectangular area of length and width equal to 30 units each. The communication range of the nodes is 10 units. The number of nodes in the experiments is in $\{30, 50, 70, 90, 110\}$.

Execution Time: Figure 5.9 compares the average execution time of `BlueMISI` (using `RandomExchange`), `ComputeMISI`, `ComputeMISII`, `Eliminate` and `BlueStars`. Each value in Figure 5.9 is averaged over 100 experiments. Note the superiority of `ComputeMISI` and `ComputeMISII` over `BlueMISI`. The averages of the execution time of `ComputeMISI` and `ComputeMISII` are about 53% of the average execution time of `BlueMISI` in networks with 70 nodes and less. This shows the impact of using `OrderedExchange` in Bluetooth networks. The figure shows that `ComputeMISI` and `ComputeMISII` are not significantly different from each other.

`Eliminate` outperforms all other algorithms. Its execution time is about 60% the execution time of `ComputeMISI` and `ComputeMISII`. The reason behind this superiority is that `Eliminate` runs in one communication round of `OrderedExchange`, whereas `ComputeMISI` and `ComputeMISII` run in two `OrderedExchange` communication rounds.

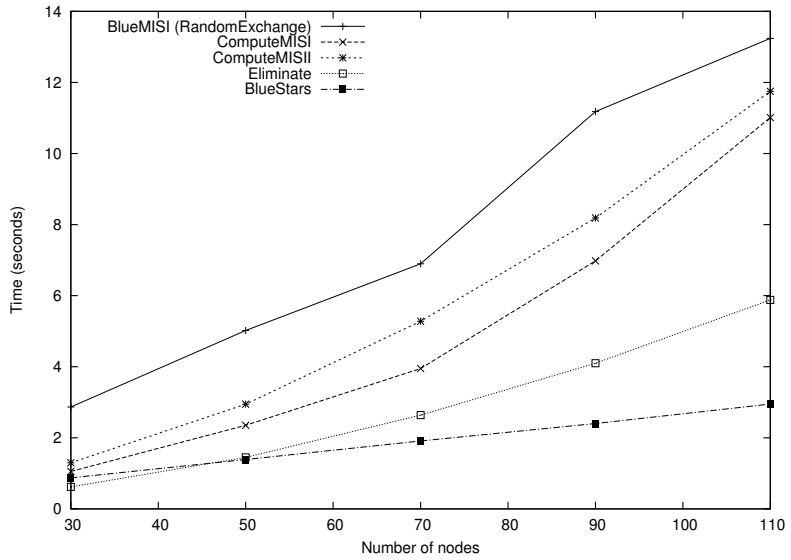


Figure 5.9: Average execution time of `Eliminate`, `BlueMISI` and `BlueStars`.

`Eliminate` and `BlueStars` use `OrderedExchange` to implement their communication rounds. `Eliminate`, however, requires only one communication round whereas `BlueStars` require more than that. Yet, Figure 5.9 shows that `BlueStars` outperforms `Eliminate` in terms of the execution time. This outperformance is because the nodes in `BlueStars` do not follow a specific order to contact the neighbors, while such an order must be followed in `Eliminate` (see Algorithm 5.2 and Remark 5.3.4). This order must be common to all nodes, since otherwise, the formed scatternet is not guaranteed to be connected. Recall that the proof of the connectivity of `Eliminate` scatternets depends on such order (see Theorem 5.4.1). This point is elaborated in the following.

To construct the set $MIS_s(v)$, an `Eliminate` node v must contact its smaller neighbors in order. Assume that v wants to contact node u and inserts it in $MIS_s(v)$, whereas u is busy in a connection with another node. This case forces v to repetitively attempt to reestablish the connection until u is free, since v has no choice but to contact u in this case. In `BlueStars`, if v wants to contact a neighbor u while u is busy, then v simply tries to contact any other neighbor with the same characteristics of u. These unsuccessful connections explain the outperformance of `BlueStars`.

`Eliminate` is still one of the strongest competitors of `BlueStars` in term of execution time, given that `Eliminate` forms connected and outdegree limited scatternets deterministically while `BlueStars` does not guarantee outdegree limitation. According to the thesis author's knowledge, the only algorithm that matches the execution time of

`BlueStars` is `BluePleidas` [DHM+07]. However, `BluePleidas` does not guarantee the formation of connected scatternets. It remains an *open question* whether it is possible to form outdegree-limited scatternets using one communication round and without the need of geometric knowledge or the necessity of contacting the neighbors in a common order - as it is the case in `Eliminate` and `BlueMISI`.

`Eliminate` can be implemented using a single `RandomExchange` communication round. Therefore, `Eliminate` is a local distributed algorithm. The time complexity of `BlueStars`, on the other hand, is always $O(n)$, even if `RandomExchange` is used to implement its communication rounds. This is because `BlueStars` builds a *maximal independent set* (MIS) of the input network (to build the disjoint piconets) using an MIS algorithm that has $O(n)$ time complexity. Linial et al. showed in [Lin92] that the lower bound time complexity of building an MIS is $\Omega(\log^* n)$. That is, there is no local distributed algorithm for building an MIS. Therefore, `BlueStars` is not a local distributed algorithm no matter what implementation is used.

There are other MIS algorithms in the literature that achieve better time complexities than the MIS algorithm used in `BlueStars`. For example, an MIS may be built in $O(\log n)$ communication rounds using the randomized algorithm of [MRSDZ11], and it may be built in $O(\log^* n)$ communication rounds if the input network is assumed to be a unit disk graph, as shown in [SW08]. These algorithms require a relatively large number of communication rounds on average compared to `BlueStars`. Thus, the use of algorithms such as those of [MRSDZ11] or [SW08] may not necessarily lead to the expected improvement in the empirical execution time of `BlueStars` due to 1) the characteristics of Bluetooth networks, and 2) the relatively small number of nodes in practical Bluetooth networks. Assume, however, that the number of nodes in a Bluetooth networks is set to be very large. Then, the MIS algorithms of [MRSDZ11] and [SW08] would outperform the MIS algorithm used in `BlueStars`. Consequently, `Eliminate` would outperform `BlueStars` since `Eliminate` requires only a single communication round.

`Eliminate` vs. `BlueMISI`: Table 5.3 and Table 5.4 study the properties of the scatternets formed by `Eliminate` and `BlueMISI`. The scatternets are compared in terms of the number of masters, number of M/S bridges, number of S/S bridges, average role per node, average piconet size, maximum piconet size, and average shortest path. There is no significant difference between the two algorithms, except with respect to the number of M/S bridges and the average role per node metrics. `Eliminate` outperforms

`BlueMISI` with respect to the number of M/S bridges. This is because the set of slaves of an `Eliminate` node is a subset of its smaller neighbors, whereas the set of slaves of a `BlueMISI` node is a subset of all its neighbors. Thus, the probability that a master is slaved is higher in `BlueMISI`. Note that the number of masters is high in both algorithms. The differences between the two algorithms with respect to the average role per node and the number of S/S bridges is due to the differences in the number of M/S bridges. `BlueMISI` slightly outperforms `Eliminate` in term of the average shortest path metric, because the scatternets of `BlueMISI` contains a higher number of edges.

Table 5.3: Comparison between the scatternets of `Eliminate` and `BlueMISI`. The average with the standard deviation of different performance metrics are given below. `Eliminate` results are given in the gray cells, whereas `BlueMISI` results are in the white cells.

Number of nodes	**Avg. piconet size**		**Max. piconet size**		**Number of masters**	
30	1.61 (0.12)	1.706 (0.12)	3.01 (0.39)	3.02 (0.31)	25.67 (0.95)	26.32 (1.07)
50	1.81 (0.09)	1.93 (0.13)	3.32 (0.46)	3.486 (0.49)	45.7 (1.05)	46.58 (1.11)
70	1.95 (0.08)	2.12 (0.12)	3.65 (0.47)	3.79 (0.40)	65.91 (0.96)	66.88 (1.30)
90	2.05 (0.07)	2.21 (0.12)	3.88 (0.32)	3.94 (0.23)	85.75 (1.05)	87.04 (1.22)
110	2.12 (0.06)	2.31 (0.10)	3.99 (0.08)	3.98 (0.18)	105.87 (0.90)	107.16 (1.33)
	Number of M/S bridges		**Number of S/S bridges**		**Average role/node**	
30	5.03 (1.88)	12.93 (2.31)	2.39 (1.12)	2.20 (0.99)	1.31 (0.01)	2.37 (0.01)
50	8.16 (2.16)	20.86 (2.91)	2.76 (1.09)	2.16 (1.11)	1.29 (0.01)	2.73 (0.01)
70	10.34 (2.27)	26.53 (4.08)	2.73 (1.02)	2.30 (1.27)	1.25 (0.01)	2.98 (0.01)
90	11.35 (2.66)	31.92 (3.99)	3.05 (1.13)	2.22 (1.06)	1.22 (0.01)	3.11 (0.02)
110	12.68 (2.94)	36.55 (4.52)	3.05 (1.06)	2.1 (1.17)	1.19 (0.01)	3.21 (0.02)

Table 5.4: Comparison between the average shortest path of the scatternets of `Eliminate` and `BlueMIS`

Number of nodes	**Avg. shortest path**	
30	3.44 (0.37)	3.04 (0.30)
50	3.46 (0.22)	2.98 (0.16)
70	3.53 (0.14)	2.95 (0.12)
90	3.61 (0.12)	2.96 (0.10)
110	3.68 (0.09)	2.95 (0.08)

5.7 Conclusions

This chapter introduced the Bluetooth Scatternet Formation problem. Studying this problem is motivated by the benefits of using Bluetooth in RFID reader networks. The chapter introduced an up-to-date literature survey of BSF algorithms. A special interest was given to mesh-based BSF algorithms and the outdegree limitation problem. A summary of major mesh-based BSF algorithms and how they handled the outdegree limitation problem is given in Table 5.5.

Table 5.5: Summary of major mesh-based BSF algorithms

Algorithm	**Outdegree Limitation**	**Drawbacks**
`BlueStars` [PBC03]	No	Very large piconets
Li et al. [LSW04]	Yes	Assumes nodes locations are known
`BluePleidas` [DHM+07]	Yes	Non-deterministic connectivity
`Bluenet` [WTH09]	Yes	Non-deterministic connectivity
`BlueMesh` [PBC04]	Yes	Long execution time
`M-dBBlue` [SWRW09]	No	Long execution time
`BlueMISI` [ZDS08]	Yes	Large number of masters
`BlueMISII` [ZDS08]	Yes/No	Outdegree limitation depends on implementation but may affect negatively execution time

The chapter introduced as well algorithm `Eliminate`, which is basically a modification of `BlueMISI`. `Eliminate` improves the average theoretical execution time of `BlueMISI`. The simulation experiments of Section 5.6 show that `Eliminate` significantly outperforms `BlueMISI` with respect to the empirical execution time. `BlueStars` outperformed `BlueMISI` with respect to the same metric. Note, however, that `BlueStars` forms scatternet with unlimited outdegrees. This is a major weakness in `BlueStars`. Lastly, the simulation experiments of Section 5.6 show that `Eliminate` outperforms `BlueMISI` in several other performance metrics.

Chapter 6

BSF-UED: A Time-efficient Bluetooth Scatternet Formation Algorithm based on Unnecessary-Edges Deletion

This chapter introduces a time-efficient mesh-based BSF algorithm, called `BSF-UED` (BSF based on Unnecessary-Edges Deletion). The algorithm uses techniques introduced in Chapter 5 to improve the execution time. `BSF-UED` aims at forming scatternets that achieve a balance between the conflicting quality metrics of Bluetooth scatternets. This chapter compares the performance `BSF-UED` against major BSF algorithms using detailed simulation experiments.

6.1 Introduction

The literature survey of Chapter 5 shows that there is little attention given to measuring the performance of BSF algorithms using the algorithms execution time. For example, few BSF algorithms are presented as time-efficient. These algorithms were not evaluated under the complex baseband and link layers of Bluetooth, despite the high specificity of these layers and their impact on the execution time. The corresponding studies either do not mention what simulator was used, or present simulations that relied on naive

simulators such as `simjava` [HM98] or `bluehoc`[1], which makes it hard to assess the real efficiency of the proposed algorithms.

To the author's knowledge, `BlueMISI` and `Eliminate` are the fastest existing BSF algorithms that deterministically form outdegree-limited scatternets. The main issue of these algorithms is that they form scatternets with very large number of masters (i.e., piconets). This chapter aims at overcoming this issue by introducing algorithm `BSF-UED`. The introduced algorithm focuses on three performance metrics, 1) *connectivity*, 2) *execution time*, and 3) *outdegree limitation*. `BSF-UED` aims also at achieving a balance between the conflicting quality metrics of scatternets. That is, `BSF-UED` aims at achieving connectivity, fast execution time, and outdegree limitation without causing any of the other performance metrics to be negatively and substantially affected. See Section 6.5 for more discussion about the balanced performance of Bluetooth scatternets. `BSF-UED` uses concepts found in `BlueMIS`, `BlueStars` and `BlueMIS` in order to meet these requirements and to preserve the advantages of these algorithms. Moreover, `BSF-UED` uses `OrderedExchange` to achieve time-efficiency.

6.1.1 Contributions

The idea of `BSF-UED` is to delete edges that are unnecessary for the connectivity of the formed scatternets using simple local rules. `BSF-UED` is time-efficient, deterministically forms connected scatternets in multi-hop networks and heuristically forms outdegree limited to seven. `BSF-UED` may form scatternets that are not outdegree limited. However, simulation experiments show that using `BSF-UED` with a heuristic, called `H1`, generates scatternets that are outdegree limited in virtually all the cases (e.g., only one outdegree non-limited scatternet was formed over 5000 experiments performed). `BSF-UED` forms scatternets with a low average role per node and low average piconet size. `BSF-UED` is efficient in terms of the number of messages transmitted. Simulation experiments also show that `BSF-UED` achieves a balance between the scatternets quality metrics in comparison to major BSF algorithms, namely; `BlueStars`, `BlueMesh`, `BlueMISI`, and `BlueMISII`. Of particular significance, `BSF-UED` is found to be about 3 times faster in execution time than `BlueMesh` while forming scatternets with similar properties[2]. Moreover, the exe-

[1] Bluehoc: http://bluehoc.sourceforge.net/. (last fetched 20-12-2012).

[2] A special importance is given to `BlueMesh` because it is one of the few known algorithm to achieve a balance between most scatternets quality metrics. `BlueMesh` is considered in [SZ06] as one of the most successful algorithms.

cution time of `BSF-UED` and `BlueMISI` are found to be approximately the same, while `BSF-UED` scatternets do not suffer from the weaknesses found in `BlueMISI` scatternets.

6.1.2 Chapter Organization

Section 6.2 introduces the terms and assumptions used in this chapter. Section 6.3 introduces `BSF-UED`. Section 6.4 gives a comparative study of `BSF-UED` against major mesh-based BSF algorithms. Section 6.5 concludes the chapter.

6.2 Problem Formulation

The algorithms introduced in this chapter use the following terms and assumptions. The input underlying network is denoted $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, while the output scatternet is denoted $\mathcal{S} = (\mathcal{V}, \mathcal{E}')$. The set $\mathcal{E}$ denotes the set of edges of the input network, and $\mathcal{E}'$ for the set of edges of the output scatternet.

This chapter denotes the set of all piconets in a scatternet $\mathcal{S} = (\mathcal{V}, \mathcal{E}')$ as $\mathcal{P}_{\mathcal{S}}$. Note that it is not necessary that every node v in the scatternet $\mathcal{S} = (\mathcal{V}, \mathcal{E}')$ is a master, and hence it is not necessary that there is a piconet $\rho(v) \in \mathcal{P}_{\mathcal{S}}$ for each $v \in V$. A summary of the terms used in this chapter is given in Table 6.1. More details are found in Section 5.2.

Assumptions: the input graph G is assumed to be a unit disk graph (UDG). `BSF-UED` assumes that a node has no knowledge of its location or the distance to its neighbors. As all mesh-based BSF algorithms, the nodes are assumed to be static. Thus, the procedures of nodes joining or leaving the scatternet are not discussed in this section.

6.3 Algorithm Description

This section describes the details of `BSF-UED`. The idea of the algorithm is to delete edges that are unnecessary for the connectivity of the scatternets using simple local rules. `BSF-UED` runs in two phases. The first phase forms isolated outdegree limited scatternets, which are then interconnected during the second phase. Each pair of nodes sharing an edge e gives a distinct color to e using local rules. This coloring is used to categorize the edges into three categories c_1, c_2 and c_3:

Table 6.1: Terminologies used in Chapter 6.

Term	**Definition**
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	The input underlying network
V	The set of Bluetooth nodes
n	The number of nodes (i.e., $\|V\|$)
$\mathcal{E}$	The set of edges in $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
$\mathcal{S} = (\mathcal{V}, \mathcal{E}')$	The formed scatternet
$\mathcal{E}'$	The set of edges (or, master-slave relationships) in $\mathcal{S}$ (if an edge (v, u) is in $\mathcal{E}'$, then $(v, u) \in \mathcal{E}$ and $(u, v) \notin \mathcal{E}'$)
$N(v)$	The set of neighbors of node v
$N_s(v)$	The set of smaller neighbors of node v (i.e., have smaller identifiers)
$N_l(v)$	The set of larger neighbors of node v (i.e., have larger identifiers)
$S(v)$	The set of slaves of node v
$M(v)$	The set of masters of node v
$\mathbf{id}(v)$	The unique identifier of node v
$\rho(v)$	The piconet of a master v where $\rho(v) = \{v \cup S(v)\}$
$\mathcal{P}_\mathcal{S}$	The set of all piconets in a scatternet $\mathcal{S}$

1. The edges of category c_1 may be necessary for the scatternet connectivity, but they do not cause the scatternet maximum outdegree to exceed seven.

2. The edges of category c_2 are not necessary for the scatternet connectivity and may cause the scatternet maximum outdegree to exceed seven.

3. The edges of category c_3 may be necessary for the scatternet connectivity and may cause the scatternet maximum outdegree to exceed seven.

Given this categorization, a priority is given to the use of the edges of category c_1 to form the scatternet. The use of the edges of category c_2 is avoided, and the edges of c_3 are used only if needed to avoid the scatternet disconnectivity. The procedures of phase 1 and phase 2 are given in the following.

6.3.1 Phase 1: piconet construction

The first phase of `BSF-UED` generates a forest of disjoint outdegree-limited piconets such that every node is either a master or a slave in exactly one piconet. The basic idea of this phase is the following. Each node v has a subset of its smaller neighbors, called *preys* and

denoted $preys(v)$, that initially contains all the smaller neighbors of v (that is, $N_s(v)$). Node v slaves all its preys except those that are already slaved by another node. Thus, a node v *attempts to slave* all its preys. This procedure forms disjoint piconets. This phase is implemented using a single `OrderedExchange` communication round in order to improve its execution time (see Section 5.5.2 for details about `OrderedExchange`).

Avoiding outdegree-limitation through nodes delegation: Given that the set $preys(v)$ is initially set to $N_s(v)$, the size of $preys(v)$ may exceed seven. Thus, the procedure described above may form piconets with more than seven slaves. Restricting v from slaving more than seven neighbors in phase 1 is done through the *nodes delegation* procedure, which is explained in the following. Each node v stores a local variable called the *piconet capacity*, denoted $\varphi(v)$, and is initially set to 7. Node v uses $\varphi(v)$ to limit the number of slaves in its piconet, if it becomes a master. Assume without loss of generality that the size of $preys(v)$ is larger than 7. If v is already contacted by all its larger neighbors, or if it has no larger neighbors then v executes the following. First, v selects the node with maximum identifier in $preys(v)$, denoted u. Node u is deleted from $preys(v)$. Node v finds a subset of *common neighbors*, denoted $CN(v, u)$, that are neighbors of both v and u, and have identifiers smaller than both $\mathbf{id}(v)$ and $\mathbf{id}(u)$. Two cases may occur:

1. If $CN(v, u) = \emptyset$, then v attempts to slave u.
2. Otherwise,, v *delegates* u to capture the common neighbors in $CN(v, u)$ and does not attempt to slave u or any neighbor in $CN(v, u)$. In such case, v deletes the delegated neighbors from its preys set $preys(v)$. Note that the delegation means that v released the responsibility of slaving the nodes of $CN(v, u)$, but it does not mean that u must slave, or must attempt to slave, the delegated nodes.

The above process repeats until v has enough capacity to slave the remaining preys (that is, $|preys(v)| \leq \varphi(v)$). Once this condition is true, node v attempts to slave all its neighbors in $preys(v)$.

Detailed Strategy

For ease of analysis, a distinct *color* is given to each edge $(v, u) \in \mathcal{E}$. The color of an edge (v, u) is denoted by $c(v, u) \in \{white, black, silver, green, red, blue\}$. Initially,

Algorithm 6.1 Procedure `construct()` at node v

```
1: while ∃u ∈ N_l(v) : c(u, v) = white do
2:    wait
3: if state(v) = slave then
4:    for all u ∈ N_s(v) do
5:       c(v, u) ← green.
6: else
7:    state(v) ← master
8:    capture()
```

$c(e) = white$ for all $e \in \mathcal{E}$. Given an edge $(v, u) \in \mathcal{E}$, such that $v \succ u$), the meaning of each color is as follows:

- *black*: v captured u.
- *silver*: v attempted to slave u, but u is already slave to another master w (that is, $\exists w \in N_l(v) : c((w, u)) = black$).
- *green*: v is slaved by a node $w \neq u$, and thus gave up on capturing u (that is, $\exists w \in N_l(v)$ and $c((w, v)) = black$, thus $c((v, u)) = green$ for each $u \in N_s(v)$).
- *red*: v delegated the capture of u to another neighbor w such that $v \succ w \succ u$.
- *blue*: v delegated to u the capture of a common neighbor w such that $v \succ u \succ w$.

The details of these procedures are illustrated in Algorithms 6.1, 6.2 and 6.3. A flow diagram of these procedures is given in Figure 6.1. An illustrative example is then given in Figure 6.2.

Remark 6.3.1. *Procedure* `contact(u)` *of Algorithm 6.3 is considered to be atomic: if nodes v and v' attempts to contact a node u, then only one of them can enter procedure* `contact(u)` *at a time. This is guaranteed in Bluetooth specifications, since a node can communicate only with one node at a time.*

The Piconet Capacity

A node controls its piconet capacity to limit its piconet in phase 1. Whenever a node v attempts to slave a prey u, the piconet capacity of v (i.e., $\varphi(v)$) is decreased by one (see line 7 of Algorithm 6.3). This occurs whether the attempt is successful (i.e., (v, u) is *black*) or not (i.e., (u, v) is silver).

Algorithm 6.2 Procedure `capture()` at node v

```
1: preys ← N_s(v)
2: while (|preys| > φ(v)) do
3:    u ← max(preys)
4:    preys ← {preys \ {u}}
5:    CN(v,u) ←FindCommonNeighbors(u)
6:    if CN(v,u) = ∅ then
7:       contact(u).
8:    else
9:       c(v,u) ← blue
10:      preys ← {preys \ {CN(v,u)}}
11:      for all w ∈ CN(v,u) do
12:         c(v,w) ← red
13:         φ(w) ← φ(w) − 1                    {to be explained in phase 2}
14: for all u ∈ preys do
15:    contact(u).
```

Algorithm 6.3 Procedure `contact(`u`)` at node v

```
1: if state(u) ≠ slave then
2:    state(u) ← slave
3:    c(v,u) ← black.
4:    ρ(v) ← ρ(v) ∪ u
5: else
6:    c(v,u) ← silver.
7: φ(v) ← φ(v) − 1.
```

Whenever an edge (v, u) is colored *red*, node u decreases its piconet capacity $\varphi(u)$ by one (see line 13 of Algorithm 6.2). The intuition behind this is that there is a chance that u will be a master of v in phase 2. Note that if (v, u) is colored *red*, then there must be a node w that is neighbor to both v and u, such that $v \succ w \succ u$, and (v, w) is colored *blue*. That is, u is delegated to w. Note that w may slave u in phase 1. Thus, one way to interconnect the piconets w and v would be through u being a master of v.

How to Find $CN(v, u)$?

This section describes the procedure of selecting a set of common smaller neighbors $CN(v, u)$ by v. Define $T = \{preys(v) \cap N_s(u)\}$. The nodes of the set $CN(v, u)$ are the λ largest nodes in T. The value of λ is based on the following rules. The set T contains the nodes that can be potentially part of $CN(v, u)$. This means that $CN(v, u) \subseteq T$, and

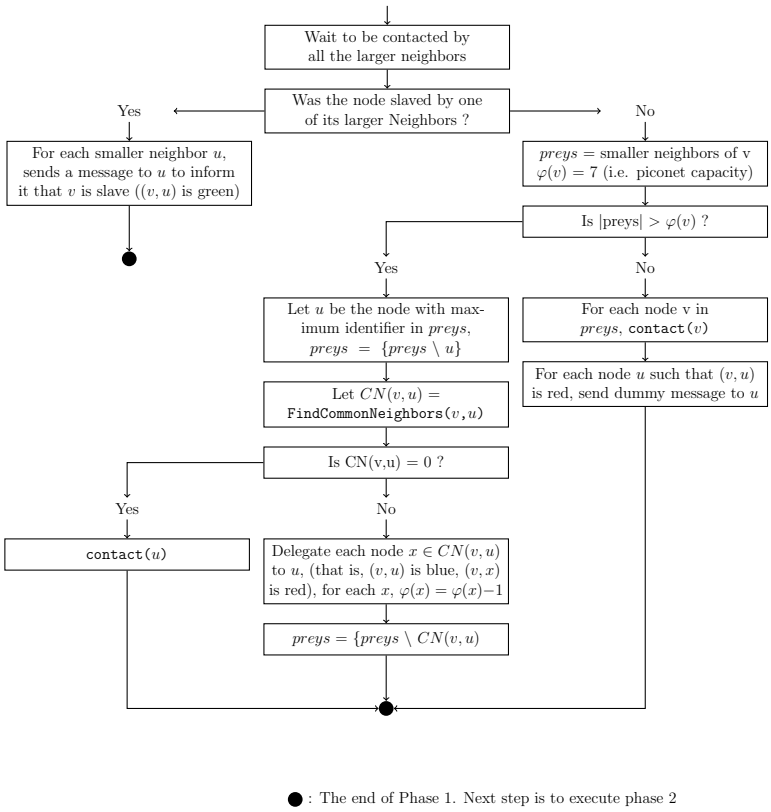


Figure 6.1: Flow diagram of phase 1 of `BSF-UED` at node v.

hence:

$$|CN(v,u)| \leq |T| \tag{6.1}$$

Node v can delegate at most seven nodes to u[3]. Therefore:

$$|CN(v,u)| \leq 7 \tag{6.2}$$

The set of nodes $\{preys(v) \setminus CN(v,u)\}$ should be of size at least $\varphi(v)$, since these nodes can be slaved by v without causing v to become outdegree unlimited. Thus:

$$\begin{aligned} |preys(v)| - |CN(v,u)| &\geq \varphi(v) \\ |CN(v,u)| &\leq |preys(v)| - \varphi(v) \end{aligned} \tag{6.3}$$

Therefore, $\lambda = |CN(v,u)| = \min(|preys(v)| - \varphi(v), 7, |T|)$.

[3]This rule is added to avoid overflowing u with delegated nodes.

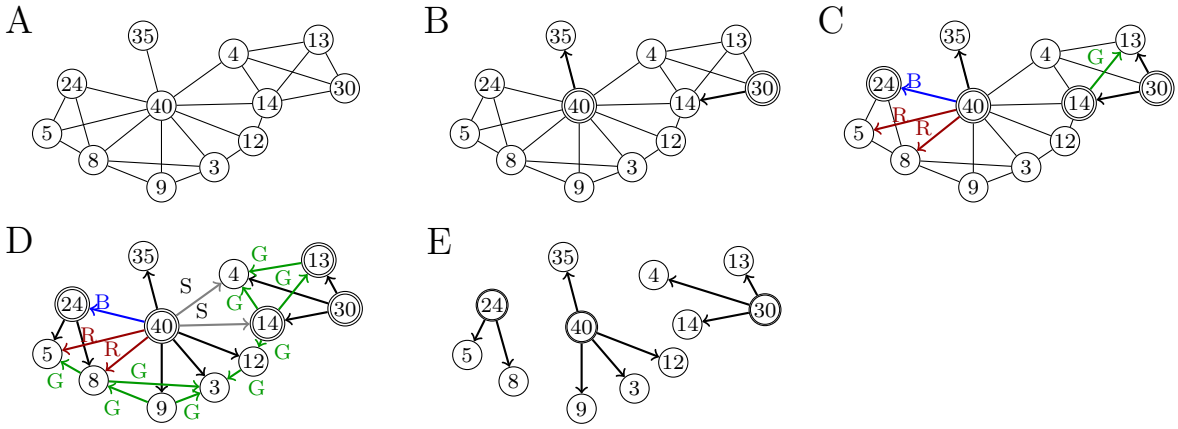


Figure 6.2: Example illustrating the procedures of phase 1 of `BSF-UED`. The input network is shown in (A). Initially, nodes 40 and 30 start the execution of the algorithm since they are the largest in their neighborhoods. Node 40 captures node 35 because $|preys(40)| > \varphi(40)$ *and* the set of common neighbors $CN(40, 35)$ is an empty set. As a result of this capture, node 40 decreases its piconet capacity by 1 (i.e., $\varphi(40) = \varphi(4) - 1 = 7 - 1 = 6$). Node 30 captures node 14 and all its neighbors in $preys(30)$ because $|preys(30)| \leq \varphi(35)$. The next target prey of node 40 is 24 (C). The size of the set $preys(40)$ is still greater than $\varphi(40)$ which is equal to 6 now. Thus, the edge (40,24) is colored blue, and the common neighbors between 40 and 24, which are 5 and 8, are delegated from 40 to 24. The edges (40,8) and (40,5) are colored red because of the delegation. Now, node 40 may attempt to capture all its preys since $|preys(40)| \leq \varphi(40)$. Node 40 was successful to capture nodes 3,9,12 as slaves but not nodes 4 and 14 as they are slaved already by node 30. Thus, the edges $(40, 4)$ and $(40, 14)$ are colored silver. Node 24 has been contacted by all its neighbors without being slaved by any of them. Thus, it acts as a master. Node 24 captures nodes 5 and 8. Lastly, note that a node that has been slaved by a larger neighbor shall inform all its smaller neighbors with this action. Thus, it colors the edges shared with all its smaller neighbors green. For example, node 8 has been slaved by its larger neighbor 24. Thus, the edge (8,5) is colored green. The procedure creates disjoint piconets shown in (E).

Theoretical Analysis

The procedures of phase 1 always terminates and forms a set of disjoint piconets that are outdegree-limited to 7 given that the input network is modeled as a unit disk graph. Phase 1 guarantees also that every node is either master or slave in exactly one piconet. For clarity, let $\mathcal{E}_{color}$ denotes the subset of edges that are colored *color* (e.g., $\mathcal{E}_{black} = \{(v, u) \mid (v, u) \in \mathcal{E} : c(v, u) = black, v \succ u\}$).

Lemma 6.3.2 (Termination)**.** *Phase 1 of `BSF-UED` terminates in a finite time.*

Proof. Procedure `construct()` is implemented in one `OrderedExchange` communication round. This guarantees that every edge $(v, u) \in \mathcal{E}$ such that $v \succ u$ is contacted exactly once. An `OrderedExchange` communication round is initiated by the nodes that are largest in their neighborhood. Additionally, every node v colors every edge (v, u) where

$u \in N_s(v)$ with a color other than white. Therefore, no edge (v, u) where $v \succ u$ remains white, and this guarantees that `construct()` terminates. □

A direct result of Lemma 6.3.2 is that the time complexity of phase 1 is the same as that of `OrderedExchange`. That is, the time complexity of phase 1 is $O(n)$.

Lemma 6.3.3. *After the execution of phase 1, each node is either a* master *or* slave.

Proof. During the execution of procedure `construct()`, if a node is not yet made slave during its waiting period, then it turns itself into a master. Otherwise, the node is a slave. □

Lemma 6.3.4 (Disjoint piconets)**.** *Let $G_b = (V, \mathcal{E}_{black})$ be the spanning subgraph of the input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the edges $\mathcal{E}_{black} \subseteq \mathcal{E}$. After the execution of phase 1, G_b is a forest of disjoint piconets.*

Proof. The proof follows from Lemma 6.3.3 and the condition that is not possible for a node to be captured twice (see procedure `contact()`). □

Lemma 6.3.5. *Let $G_{bs} = (V, \{\mathcal{E}_{black} \cup \mathcal{E}_{silver}\})$ be the spanner subgraph of the input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the set of edges $\{\mathcal{E}_{black} \cup \mathcal{E}_{silver}\} \in \mathcal{E}$. After the execution of phase 1, G_{bs} is outdegree-limited to 7, given that $\varphi(v)$ is initially set to 7 for each $v \in V$.*

Proof. The proof is based on counting the number of times a node v calls procedure `contact()`, since this is the only procedure that colors an edge with black or silver. The code of `capture()` is the most interesting for this proof given that it is the only procedure that calls `contact()`. The code of `capture(`v`)` is divided into a while-loop (starting from line 2 to line 13), which is used to limit the outdegree of the master, and a for-loop (starting from line 14 to line 15) which is used to capture all neighbors in case $|preys(v)| \leq \varphi(v)$.

Without loss of generality, assume that $|preys(v)| > 7$. At the while-loop of `capture()`, v `contact` a neighbor u if and only if v and u has no common neighbors that are smaller than both of them. There is at most 5 neighbors of v having the property of u, assuming that the input network is a unit disk graph. Therefore, node v calls procedure `contact()` from the while-loop at most 5 times. In this case, v would have only 5 neighbors such that none of them is neighbor to the other. Node v `contact` a neighbor u from the for-loop of `capture()` only if $|preys(v)| \leq \varphi(v)$, and if it was not already `contacted`

in the while-loop. As a result, given that the execution of the while-loop stops only if $|preys(v)| \leq \varphi(v) = 7$, and given that each time a node v executes `contact()` it decreases $\varphi(v)$ by 1, then v does not execute `contact()` in the while-loop and the for-loop, combined, more than 7 times at most. □

Theorem 6.3.6. *Phase 1 forms a set of disjoint piconets that are outdegree limited to 7 such that every node is either master or slave in exactly one piconet.*

Lemma 6.3.7, introduced in the following, is not strictly relevant to the objectives of phase 1, but it is helpful to prove the correctness of phase 2. It establishes that the set of blue edges $\mathcal{E}_{blue}$ are not necessary for the connectivity of the input network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

Lemma 6.3.7. *Let $G_{nb} = (V, \{\mathcal{E} \setminus \mathcal{E}_{blue}\})$ be the spanner subgraph of the input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the set of edges $\{\mathcal{E} \setminus \mathcal{E}_{blue}\}$. After the execution of phase 1, G_{nb} remains connected.*

Proof. This is proven by showing that, following a specific order of edges, a minimum spanning tree (MST) of $\mathcal{G}$ does not include blue edges. Note that if an edge $(v, u) \in \mathcal{E}_{blue}$, then u is the largest node among $preys(v)$ at that while-loop of `capture()` (see line 2 to 13 of `capture()`). If there is an edge (v, u) that is colored blue, then there must be two edges (v, w) and (u, w) in $\mathcal{E}$ such that $v \succ u \succ w$, and edge (v, w) is colored red (see line 12 of `capture()`). Therefore, v, u and w forms a circle of three edges.

To build the MST, the edges are lexicographically ordered such that $(v_1, u_1) \succ (v_2, u_2)$ if $v_1 \succ v_2$ or if $v_1 = v_2$ and $u_1 \succ u_2$). For any triangle v,u and w such that (v, u) is blue and (v, w) is red and $v \succ u \succ w$, the blue edge (v, u) is surely not included in the MST of $\mathcal{G}$ since it is the largest edge in the corresponding 3-circle. □

6.3.2 Phase 2: Piconets Interconnection

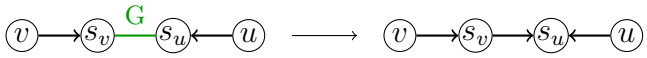
The second phase of `BSF-UED` interconnects the disjoint piconets formed in phase 1 into a connected scatternet, while maintaining its maximum outdegree to a reasonable value. Abstractly, phase 2 creates a meta-graph $G_m = (V_m, E_m)$ in which every piconet formed in phase 1 is a vertex in V_m. There are two general steps performed in phase 2. First, the set of edges E_m is defined such that the graph G_m is connected. An edge $(v, u) \in E_m$ denotes that the piconets $\rho(v)$ and $\rho(u)$ are *neighbor piconets*. Each edge in E_m corresponds to a path of 1, 2, or 3 hops in the underlying network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

connecting the masters of $\rho(v)$ and $\rho(u)$. The set of all neighbor piconets of a piconet v is denoted $\mathcal{N}_{\mathcal{P}}(\rho(v))$ (or $\mathcal{N}_{\mathcal{P}}(v)$ for short). After the construction of E_m, a procedure, called the *elimination procedure* and inspired from algorithm `Eliminate`, is executed in order to delete some edges of E_m while keeping the graph G_m connected. That is, each vertex $v \in V_m$ creates the set $MIS_l(v)$, where $MIS_l(v)$ is a maximal independent set of the larger neighbor piconets of v, denoted $\mathcal{N}_{\mathcal{P}l}(v)$. The main issue, however, is that the meta-graph G_m is not a unit disk graph. Thus, the size of $MIS_l(v)$ is not necessarily bounded to five as it is the case in `Eliminate`. This is the main reason why `BSF-UED` does not form outdegree limited scatternets deterministically. The elimination procedure, however, helps in heuristically minimizing the maximum and average outdegree of the scatternet. The construction of the set E_m is explained in the next section. Afterward, the elimination procedure is explained in detail.

Detailed Strategy

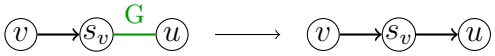
The construction of the set E_m: The set E_m is constructed as follows. A pair of piconets $\rho(v)$ and $\rho(u)$ are *neighbor piconets* if their masters v and u can be interconnected through one of the following types of paths, or *interconnection rules*. A node that interconnects a pair of neighbor piconets is called a *gateway*. The interconnection rules of `BSF-UED` are given in the following. Without loss of generality, it is assumed below that $v \succ u$:

***I*-rule 1** (*Three-hop interconnection*): through the edge (s_v, s_u); where $s_v \in \rho(v)$ (i.e., slave in $\rho(v)$), and $s_u \in \rho(u)$, and $c(s_v, s_u) = green$. (*interconnection operation*: s_v slaves s_u.)

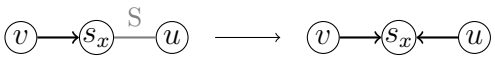


***I*-rule 2** (*Two-hop interconnection*):

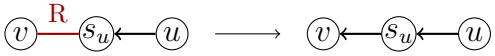
***I*-rule 2a**: through the edge (s_v, u); where $s_v \in \rho(v)$, and $c(s_v, u) = green$. (*interconnection operation*: s_v slaves u.)



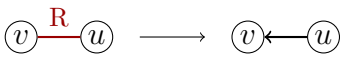
***I*-rule 2b**: through the edge (v, s_x) or (u, s_x); where $s_x \in \rho(v)$ or $s_x \in \rho(u)$, and $c(v, s_x) = silver$ or $c(u, s_x) = silver$. That is, s_x is smaller than both v and u, and it belongs to either $\rho(v)$ or $\rho(u)$ but not both. Both of the piconets masters v and u attempted to slave s_x but only one of the attempts was successful. (*interconnection operation*: u slaves s_x or v slaves s_x. (that is, the silver edge becomes black.)



***I*-rule 2c**: through the edge (v, s_u); where $s_u \in \rho(u)$ and $c(v, s_u) = red$. That is, s_u is smaller than both v and u. s_u is slaved by u, and s_u was delegated by v. (*interconnection operation*: s_u slaves v.)



***I*-Rule 3** (*One-hop interconnection*): through the edge $(v, u) \in \mathcal{E}$ where $c(v, u) = red$. Both v and u are masters of different piconets. (Operation: u captures v as $u \prec v$.) In this rule, the fact that u captures v (and not the opposite) is important; it follows the anticipated decrease of $\varphi(u)$ in phase 1 when the edge (v, u) was colored red. In a sense, u is *more prepared* than v to handle new slaves. The impact of this operation is seen while starting the elimination procedure[4].



Interconnection operations and gateways: a pair of neighbor piconets are interconnected using one of the interconnection rules given above, using the *interconnection operation* given with each interconnection rule. The *interconnection operation* shows the gateways used to interconnect the neighbor piconets[5]. For instance, two neighbor piconets $\rho(v)$ and $\rho(u)$ are interconnected via *I*-rule 1 if s_v slaves s_u, given that s_v and s_u are slaves to $\rho(v)$ and $\rho(u)$ respectively, and $c(s_v, s_u) = green$ as described above. In this example, s_v and s_u are the gateways. In Rule 2a, the nodes s_v and u are the gateways.

[4]This case occurs if v delegated to a neighbor w the responsibility of u, and hence (v, u) is red. Then, u becomes master as w does not slave u. Recall that if v delegated u to w, then it is not necessary that w slaves u.

[5]Recall that the nodes that are used to interconnect a pair of neighbor piconets are called *gateways*.

Interconnection rules priorities: the interconnection rules are listed in order from the highest priority to the lowest (i.e., I-rule 1 has the highest priority, then I-rule 2a, I-rule 2b, I-rule 2c, and I-rule 3). The rule with the highest priority is used in case a pair of neighbor piconets can be interconnected via more than one interconnection rule.

Remark 6.3.8. *To implement this part of phase 2, each slave s_v informs its master v about each neighbor w_i in $N(s_v)$ after the termination of phase 1. For each $w_i \in N(s_v)$, s_v sends to v: 1) the master of w_i (that is, $M(w_i)$ - since there is at most a master to w_i in phase 1), 2) the piconet capacity value of w_i (that is, $\varphi(w_i)$), and 3) the color of the edge (s_v, w_i). The master v, therefore, knows all its neighbor piconets, all the possible interconnection rules, and all the gateways to interconnect to all its neighbor piconets.*

A master v should assign to a gateway s_v in a piconet $\rho(v)$ at most seven neighbors, if possible. It is possible to fix the number of assigned slaves to s_v to seven at most, but this may lead to the scatternet disconnectivity. This is achieved by the elimination procedure discussed below.

The elimination procedure: The main issue in the construction of E_m is that it may lead to gateways slaving more than seven slaves (i.e., forming outdegree unlimited piconets). The elimination procedure is thus introduced to minimize the number of outdegree unlimited piconets formed in phase 2 and to minimize the average outdegree of the formed piconets. Note, however, that this procedure does not guarantee that all formed piconets are outdegree limited, but simulation results show that most of the piconets formed in phase 2 are outdegree limited. The elimination procedure is a simple and time-efficient heuristic that achieves a trade-off between the execution time and outdegree limitation. Note that it is possible to guarantee the outdegree limitation of formed piconets with more sophisticated procedures. This approach, however, would lead to longer execution time which would contradict with the priorities of `BSF-UED` requirements, as time-efficiency is given a higher priority than outdegree limitations in the objectives of `BSF-UED`. The basic idea of the elimination procedure is to let each piconet $\rho(v)$ build a maximal independent set $MIS_l(\rho(v))$ of its larger piconet neighbors $\mathcal{N}_{\mathcal{P}l}(v)$. The piconet $\rho(v)$ interconnects only with the neighbor piconets in $MIS_l(\rho(v))$. To interconnect $\rho(v)$ to a neighbor piconet $\rho(u)$, the master v selects the appropriate interconnection rule, and asks the corresponding gateway to perform the interconnection opration. A master v should assign to a gateway s_v in a piconet $\rho(v)$ at most seven

neighbors, if possible. It is possible to fix the number of assigned slaves to s_v to seven at most, but this may lead to the scatternet disconnectivity. The details of this procedure is given below.

Each master v of a piconet $\rho(v)$ constructs a *gateway table* (denoted as $\mathcal{T}(v)$) using the information received by its slaves (see Remark 6.3.8). The entries of a gateway table represent all the rules that can be applied to interconnect to neighbor piconets. Each entry consists of the following elements:

- u: the master of the neighbor piconet $\rho(u)$.
- s_u: the gateway of piconet $\rho(u)$ (note that s_u may be equivalent to u).
- s_v: the gateway of piconet $\rho(v)$ (note that s_v may be equivalent to v).
- $\varphi(s_v)$: the piconet capacity of s_v.
- I: the interconnection rule of the tuple (that is, I-rule 1, ..., I-rule 3).
- `role`: the role to be played in the new relation (either *master* or *slave*). If `role` is *master*, s_u becomes master to s_v according to interconnection rule I. If `role` is *slave*, s_u becomes slave to s_v.

As already mentioned, a given neighbor piconet $\rho(v)$ can be interconnected to $\rho(u)$ by multiple rules. Therefore, the gateway table $\mathcal{T}(v)$ may contain several entries for a same neighbor piconet $\rho(u)$, with different gateways s_v and s_u.

Each piconet $\rho(v)$ constructs a maximal independent set of its *larger piconet neighbors*, denoted $MIS_l(\rho(v))$, where the set of larger piconet neighbors of $\rho(v)$ are the neighbor piconets of $\rho(v)$ whose masters identifiers are larger than the identifier of v. The construction is done on-the-fly; each time a piconet $\rho(v)$ interconnects to a larger neighbor piconet $\rho(u)$, it does not interconnect with any neighbor piconet $\rho(w)$ that is common neighbor to both $\rho(v)$ and $\rho(u)$ and has a larger identifier than both. The details of the procedure are given in `interconnect()` in Algorithm 6.4.

Remark 6.3.9. *Any master-slave relationship between gateways s_v and s_u added by procedure* `interconnect()` *is represented as an edge (s_v, s_u), where s_v is the master of s_u. The set of all edges interconnected by* `interconnect()` *is denoted $\mathcal{E}_{black'}$. The output of* `interconnect()` *is the graph $G = (V, \{\mathcal{E}_{black} \cup \mathcal{E}_{black'}\})$, since $\mathcal{E}_{black}$ is the set of master-slave relationships formed in phase 1.*

Algorithm 6.4 Procedure `interconnect()` at piconet v

1: $U_l \leftarrow \mathcal{N}_{\mathcal{P}l}(v)$, $U_s \leftarrow \mathcal{N}_{\mathcal{P}s}(v)$, where $\mathcal{N}_{\mathcal{P}s}(v)$ is the set of smaller piconets of v.
2: $B \leftarrow \emptyset$
3: **while** $(U_s \neq \emptyset)$ **do**
4: wait for gateways of smaller piconets to contact.
5: upon receipt of a message from a gateway s_u from piconet u; $U_s \leftarrow \{U_s \setminus \{u\}\}$.
6: **while** $(U_l \neq \emptyset)$ **do**
7: $u \leftarrow \min(U_s)$
8: select best row $(u, s_u, s_v, \varphi(s_v), I, \texttt{role}) \in \mathcal{T}(v)$
9: **if** $(\texttt{role} = mas)$ **then**
10: v orders s_v to be master of s_u
11: $\varphi(s_v) \leftarrow \varphi(s_v) - 1$
12: update $\mathcal{T}(v)$
13: **else**
14: v orders s_v to be the slave of s_u
15: $\varphi(s_u) \leftarrow \varphi(s_u) - 1$
16: $B \leftarrow \{B \cup \{\mathcal{N}_{\mathcal{P}l}(v) \cap \mathcal{N}_{\mathcal{P}l}(u)\}\}$
17: $U_l \leftarrow \{U_l \setminus \{u \cup \mathcal{N}_{\mathcal{P}l}(u)\}\}$
18: **for all** (piconet $u \in B$) **do**
19: select any tuple $(u, s_u, s_v, \varphi(s_v), I, \texttt{role}) \in \mathcal{T}(v)$
20: v asks s_v to contact s_u without either being master nor slave of s_u.

Selecting the best rows from $\mathcal{T}(v)$: an issue that needs more clarification in `interconnect()` is how to select the best pair of gateways to interconnect a pair of piconets $\rho(v)$ and $\rho(u)$ (see line 8 of `interconnect()`). In order to do this, a node v ascendingly sorts the rows of $\mathcal{T}(v)$ in a lexicographical order of $(\rho(u), d(I), -\varphi(s_v))$, where $d(I)$ is a number given to the interconnection rule I (that is, $d(I\text{-Rule 1}) = 1$, $d(I\text{-rule 2a}) = 2$, $d(I\text{-rule 2b}) = 3$ etc ..). Whenever master v attempts to select the best gateway to a neighboring piconet $\rho(u)$ with master u, it simply finds the first occurrence of u in $\mathcal{T}(v)$. That is, master v starts interconnecting with the smallest neighbor piconets to the larger ones. In case of multiple choices, master v prefers the lower rules (that is, I-rule 1 to I-rule 2a etc ..), and in case of multiple choices, master v selects the slave to interconnect with $\rho(u)$ via the slave s_v with the largest piconet capacity $\varphi(s_v)$.

The piconet capacity in phase 2 and updating $\mathcal{T}(v)$: each time a gateway s_v becomes master to a gateway s_u, the piconet capacity of s_v (i.e., $\varphi(s_v)$) is decreased by one (see line 11 of `interconnect()`). Thus, the gateways table $\mathcal{T}(v)$ should be updated

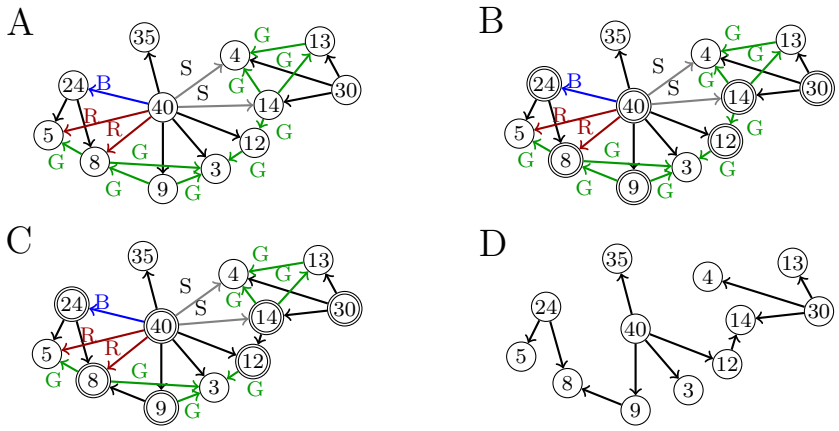


Figure 6.3: Example illustrating the procedures of phase 2 of `BSF-UED`, taking over the resulting network of the example in phase 1 (Figure 6.2). (*B: Blue, R: Red, G: Green, S: Silver*). Those piconets that have no larger piconets are the ones to start, while others wait. Thus, piconets 24 and 30 starts; both of which should interconnect with piconet 40. There are several interconnection rules between piconets 24 and 40. The interconnection rule (or, path) with highest priority is $24, 8, 9, 40$. In this rule, 8 should be the slave of 9. There are no other neighbor piconets that piconet 24 shall interconnect to. The interconnection rule (or, path) with highest priority to interconnect piconet 30 and 40 is $30, 14, 12, 40$. In this rule, 14 should be the slave of 12. The formed scatternet is shown in (D).

after each of such changes by simply sorting it again (see line 12 in `interconnect()`). Note that, using this method, a gateway s_v with higher capacity $\varphi(s_v)$ is always preferred. Also, this method preserves the priorities of the interconnection rules. The order of the rows of $\mathcal{T}(v)$ reduces outdegree of the gateways of v.

An example illustrating the procedures of phase 2 is given in Figure 6.3.

Theoretical Analysis

This section proves that the output of phase 2, which is $G = (V, E' = \{\mathcal{E}_{black} \cup \mathcal{E}_{black'}\})$, is a connected scatternet.

Theorem 6.3.10 (Connectivity)**.** *Let $G_m = (V_m, E_m)$ be such that (v, u) is in E_m if and only if v and u are two piconets that can be interconnected with any of the* interconnection rules*. Then, G_m is connected.*

Proof. The interconnection rules consider all cases of green, silver and red edges (i.e., all edges in the sets $\mathcal{E}_{green}$, $\mathcal{E}_{silver}$ and $\mathcal{E}_{red}$). By definition, the endpoints of any edge (v, u) with silver or red color must belong to two different piconets, while green edges may be between nodes belonging to the same or different piconets. Thus, any such edge

interconnects two different piconets. The interconnection rules does not consider blue edges, which are not needed for the connectivity of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ according to Lemma 6.3.7. □

Theorem 6.3.11. *If a graph $G = (V, E)$ is connected, then the graph $G' = (V, \{(v, u) \in E \mid u \in MIS_l(v), v \in V\})$ is also connected, where $MIS_l(v)$ is the maximal independent set of the larger neighbors of v.*

Proof. This follows from the correctness of algorithm `Eliminate`. See Theorem 5.4.1. □

Theorem 6.3.12. *The graph $G = (V, \{\mathcal{E}_{black} \cup \mathcal{E}_{black'}\})$ is connected.*

Proof. $G = (V, \{\mathcal{E}_{black} \cup \mathcal{E}_{black'}\})$ is the output of `interconnect()`. Note that `interconnect()` let each piconet $\rho(v)$ formed in phase 1 connects to all its neighbor piconets in $MIS_l(\rho(v))$ with an edge $(s_v, s_u) \in \mathcal{E}_{black'}$ (see Remark 6.3.9). Therefore, the connectivity follows from Theorem 6.3.10 and Theorem 6.3.11. □

Theorem 6.3.13. *The graph $G = (V, \{\mathcal{E}_{black} \cup \mathcal{E}_{black'}\})$ is a scatternet $\mathcal{S} = (\mathcal{V}, \mathcal{E}')$.*

Proof. It is sufficient to prove that no pair of nodes are slaves and masters to each other at the same time. $G = (V, \mathcal{E}_{black})$ is a set of disjoint piconets. Thus, only the added master-slave relationships at phase 2 are considered, which are in $\mathcal{E}_{black'}$. The proof follows from three arguments. First, if two piconets $\rho(v)$ and $\rho(u)$ are interconnected in procedure `interconnect()`, then v is larger than u (see line 7 in `interconnect()`). Second, note that $\rho(v)$ contacts a neighbor piconet $\rho(u)$ only once (see line 17 in `interconnect()`). Third, according to Lemma 6.3.4, a slave s_v belongs to only one piconet and each node is either a slave or a master according to Lemma 6.3.3. Therefore if a gateway s_v is assigned to be a master to another gateway s_u, then s_u cannot be assigned as the master of s_v in a later stage of `interconnect()`. Thus, the formed network is connected and does not contain a pair of nodes v and u that are masters and slaves to each other at the same time. □

6.3.3 Heuristic `H1`: Decreasing the Number of Outdegree Unlimited Piconets

This section introduces a simple local heuristic, called `H1`, that aims to decrease the number of outdegree unlimited piconets formed in phase 2. This heuristic does not cause

an increase in the execution time of the algorithm, whereas simulation experiments show that heuristic `H1` significantly improves the quality of the formed scatternet.

Heuristic `H1` eliminates virtually all outdegree-unlimited piconets, while not increasing the execution time of the algorithm. The heuristic is as follows. Assume that piconet v assigned gateway s_v to become master of gateway s_u. Assume s_v is already a master to seven slaves. Meanwhile, s_u may have slaved some nodes in the interconnection phase. Node s_v checks if s_u is a master to less than seven slaves. If this is the case, then s_u slaves s_v instead. Heuristic `H1` is not executed if s_v has less than seven slaves. Interestingly, such a simple rule can improve the properties of the scatternet significantly, as shown by the simulation results in Section 6.4.

There is another heuristic that has been used in `BSF-UED` that also helps in heuristically limiting the maximum outdegree of the scatternet. This heuristic is based on the priorities given to the interconnection rules. Note that interconnection rules that do not involve red edges are given the lowest priorities in the interconnection rules. Interestingly, if none of the interconnection rules involving red edges are used, then the formed scatternet is outdegree limited. To prove this statement, note the following observations. First, note that black and silver edges do not cause excess on the outdegree of the scatternet (see Lemma 6.3.5). Note that blue edges are not used in the scatternet formation and do not cause the scatternet disconnectivity (see Lemma 6.3.7). Therefore, it remains to show that green edges do not cause excess in the outdegree of the formed scatternet. Consider a scenario in which a node v is master to node s_v, such that node s_v shall use more than 7 green edges to interconnect with neighboring piconets. Assume that node s_v must be the master of all the nodes on the other end of its green edges (call them green neighbors of s_v). Let assume that no interconnection rules involving red edges are necessary for interconnection. Then, there is a maximal independent set of the green nodes of v that is of size at most 5. This is because edges between these green nodes are either silver or green. Therefore, this contradicts the argument that s_v shall connects to more than 7 green neighbors.

6.4 Simulation Experiments

This section studies the empirical performance of `BSF-UED` compared against `BlueStars`, `BlueMesh`, `BlueMISI` and `BlueMISII`. Algorithm `BlueMISI` is implemented using the improvements given in Appendix A, namely `ComputeMISI`. The main findings show that

`BSF-UED` is a time-efficient BSF algorithm that has a similar execution time to `BlueMISI` and about 1/3 the execution time of `BlueMesh`. `BlueStars` execution time is shorter than the execution time of `BSF-UED` and `BlueMISI`. Note, however, that `BlueStars` does not guarantee outdegree limitation, and hence, its execution time cannot be compared against the other studied algorithms. Furthermore and more importantly, relaxing the outdegree limitation requirement significantly simplifies the BSF algorithm design. Therefore, `BlueStars` is included in this comparison study only to be considered as a benchmark. The experiments results of `BSF-UED` shows that it is ranked among the best algorithms with respect to most performance metrics. The experiments analysis below assumes that any master node with no slaves is a non-master. This situation happens when a node locally declares itself as a master but does not succeed in slaving any neighbor. This assumption is applied to all studied algorithms. The execution time is represented in the following as a time-series for ease of demonstration. All other performance metrics are represented as bars plots. Deviations from mean values are shown with error bars that represents the standard deviation.

6.4.1 Execution Time

Figure 6.4 shows a comparison of the execution time of the studied algorithms. `BlueStars` outperforms the other algorithms because of its simplicity. This is achieved with the cost of forming very large piconets. The execution time of `BSF-UED` is about one third ($\frac{1}{3}$) the execution time of `BlueMesh`. Algorithms `BSF-UED` and `BlueMISI` have similar execution times.

All of the studied algorithms use `OrderedExchange` to implement the communication rounds. `BlueStars` and `BlueMesh` use a maximal independent set algorithm that, by its nature, uses `OrderedExchange`, whereas `BSF-UED` and `BlueMISI` (or basically, `ComputeMISI`) explicitly use `OrderedExchange` to implement the communication rounds. Counting the number of `OrderedExchange` rounds executed by a given BSF algorithm gives a good indication of its empirical execution time. `BlueStars` requires three communication rounds for 1) forming the piconets, 2) identifying the neighbor piconets, and 3) interconnecting the neighbor piconets. On the other hand, each iteration (or, phase) of `BlueMesh` requires four rounds: 1) a communication round to exchange the 1-hop neighborhood relationships between neighbor nodes in order to construct the 2-hop neighborhood relationships, 2) a communication round to form outdegree limited piconets, 3) a

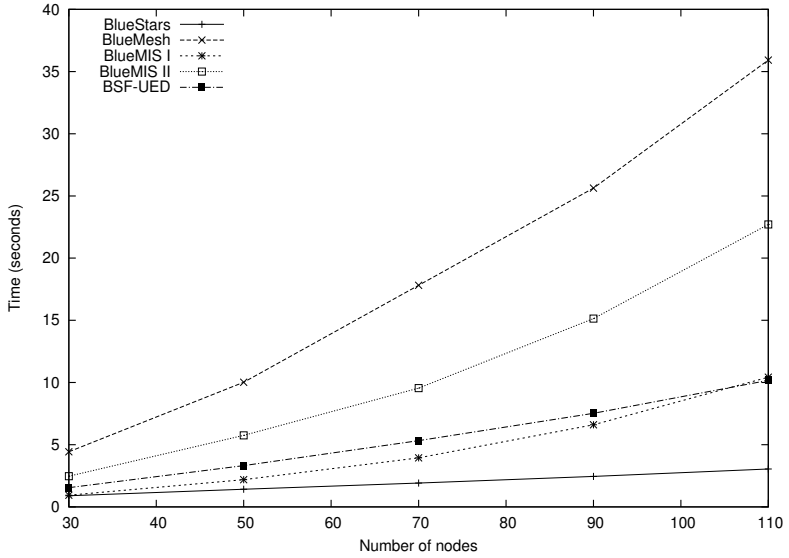


Figure 6.4: Comparison of the execution time

communication round for masters to discover their neighboring piconets (i.e., slaves sends information about their neighbors to their masters), and 4) a communication round for each node surviving the current iteration (or, phase) to find its neighbors in the next phase (i.e., its neighbors in G_i where i is the index of the next phase). `BSF-UED` requires three communication rounds; two of which are executed in phase 1. `BlueMISI` requires two communication rounds. The large number of communication rounds executed by `BlueMesh` explains its long execution time compared to all other algorithms. Interestingly however, `BlueStars` outperforms `BSF-UED` and `BlueMISI` with respect to execution time despite executing more communication rounds. This outperformance is caused by a restriction imposed on `BlueMISI` and `BSF-UED` that states that a node shall contact its neighbors in a specific order. In some parts in `BlueMISI` and `BSF-UED`, if a node v contacts a neighbor u while u is busy, then v shall wait until u responds and v is not allowed to contact any other neighbor until u responds. This restriction imposes a penalty on the execution time of `BSF-UED` and `BlueMISI`. This restriction does not exist in `BlueStars`, and this explains the outperformance of `BlueStars` compared to `BSF-UED` and `BlueMISI`. Note that a similar argument was used in Section 5.6 in order to explain the outperformance of `BlueStars` compared to `Eliminate`.

6.4.2 Number and size of piconets

The number of piconets in the formed scatternets is studied in this section. The results are shown in Figure 6.5. `BlueStars` outperforms all the other algorithms according to this metric. However, this is a trade-off with the number of outdegree unlimited piconets. The following experiment is performed in order to study this trade-off. The experiment states that each master node in `BlueStars` becomes a slave and each slave node becomes a master. This led to a significant increase in the number of piconets in `BlueStars` and to limiting the size of `BlueStars` piconets to about 3 slaves per piconet at the most. Thus, this modified (or, inverse) version of `BlueStars` performs similarly to `BlueMISI` since `BlueMISI` is known to form scatternets with a large number of piconets. Note, however, that `BlueMISI` guarantees outdegree limitation in unit disk graphs.

The experiments show that `BlueMISI` scatternets have the largest number of piconets. This is because all nodes initially declares themselves as masters in `BlueMISI`. A node v becomes slave only in one case; which occurs if each slave u of v has larger identifier and has also considered v as a slave. In such case, each slave u becomes the master of v and thus v is left with no slaves and v becomes a slave. The rules of `BlueMISII` significantly decreases the number of piconets of `BlueMISI`. This significant decrease, however, caused the piconets of `BlueMISII` to become outdegree unlimited. `BlueMesh` forms scatternets with a logical number of piconets, which increases as the number of nodes increases. `BSF-UED` and `BSF-UED H1` have approximately similar results compared to `BlueMesh`. Note that the heuristic `H1` causes an increase in the number of piconets as the number of nodes increases.

The maximum size of piconets is studied next. The average maximum outdegree of the formed scatternets is shown in Table 6.2. Note that `BlueMISII` and `BlueStars` introduce the largest piconets on average. This is a significant weakness of these algorithms since a piconet with more than seven slaves introduces a penalty in the piconet and scatternet throughput. This is because a master with more than seven slaves must keep only seven of them unparked (or, active). Parked nodes still belong to the piconet but do not collaborate in its activities, and hence cannot send or receive messages within the piconet. Outdegree limitation is one of the most important quality metrics in Bluetooth scatternet formation. This is why it has been the main concern of many BSF algorithms (see Section 5.3 or [SZ06]).

The piconets formed by `BlueMISI` have the minimum maximum size on average. This

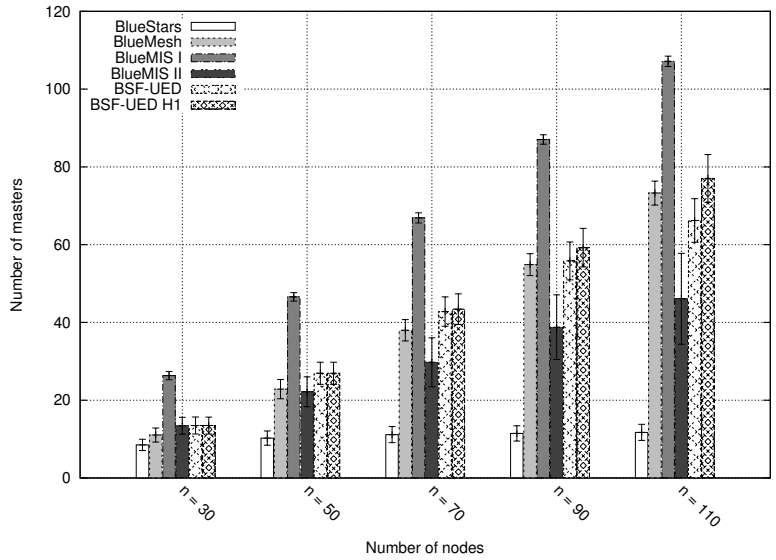


Figure 6.5: Comparison of the number of piconets

is because the set of slaves of a master v in `BlueMISI` is a maximal independent set of its neighbors, which is of size at most five in unit disk graph and on average is less. `BlueMesh` on average forms scatternets with at most seven slaves. For `BSF-UED`, the average maximum size of piconet can reach 14.37 slaves. Piconets with such large size are few. For instance, the simulation analysis shows that, on average, the number of piconets that have more than seven slaves are 6% in the 110-node networks (4 outdegree unlimited piconets from 66 piconets on average). When applying heuristic `H1`, the average maximum piconet size of `BSF-UED` drops to seven. Most of the formed scatternets are found to be outdegree limited. In fact, it is found that `BSF-UED` with `H1` formed only one outdegree unlimited scatternet (out of 5000 scatternets formed in all the experiments). The maximal outdegree of this outdegree unlimited scatternet was 8!. `BSF-UED` is therefore very close to the optimum, which gives the belief that the heuristics could be further improved to achieve this deterministically. Lastly, the average piconet size of `BSF-UED` is about 2.4 as shown in Table 6.3. `BSF-UED` is only outperformed by `BlueMISI`, however the outperformance is not significant. This shall be seen as an advantage for `BSF-UED`.

6.4.3 Average number of roles per node

The number of roles of a node is the number of piconets it belongs to. The average number of roles per node is the sum of number of roles among all nodes divided by the number of

Table 6.2: Comparison of the average maximum piconet size with standard deviation (in brackets)

Number of nodes	`BlueStars`	`BlueMesh`	`BlueMISI`	`BlueMISII`	`BSF-UED`	`BSF-UED H1`
30	8.64 (1.70)	6.90 (0.25)	3.02 (0.31)	8.21 (1.6)	5.99 (0.76)	5.99 (0.76)
50	13.28 (2.38)	7.00 (0)	3.48 (0.49)	12.86 (3.04)	6.39 (0.54)	6.38 (0.52)
70	18.42 (3.55)	7.00 (0)	3.79 (0.40)	18.06 (4.40)	6.94 (1.37)	6.51 (0.49)
90	23.37 (3.76)	7.00 (0)	3.94 (0.23)	22.06 (4.90)	10.16 (2.87)	6.91 (0.27)
110	27.87 (4.09)	7.00 (0)	3.98 (0.18)	27.62 (6.33)	14.56 (3.58)	7 (0.03)

Table 6.3: Comparison of the average piconet size with standard deviation (in brackets)

Number of nodes	`BlueStars`	`BlueMesh`	`BlueMISI`	`BlueMISII`	`BSF-UED`	`BSF-UED H1`
30	3.92 (0.57)	2.36 (0.48)	1.71 (0.12)	3.46 (0.54)	2.28 (0.32)	2.28 (0.320)
50	5.56 (0.97)	3.54 (0.29)	1.93 (0.13)	4.18 (0.75)	2.13 (0.18)	2.13 (0.18)
70	7.13 (1.27)	3.42 (0.52)	2.12 (0.12)	4.99 (1.03)	2.17 (0.18)	2.14 (0.16)
90	8.78 (1.40)	3.78 (0.27)	2.21 (0.12)	5.21 (1.17)	2.52 (0.22)	2.37 (0.15)
110	10.41 (1.73)	3.92 (0.21)	2.31 (0.11)	5.72 (1.40)	2.96 (0.30)	2.53 (0.19)

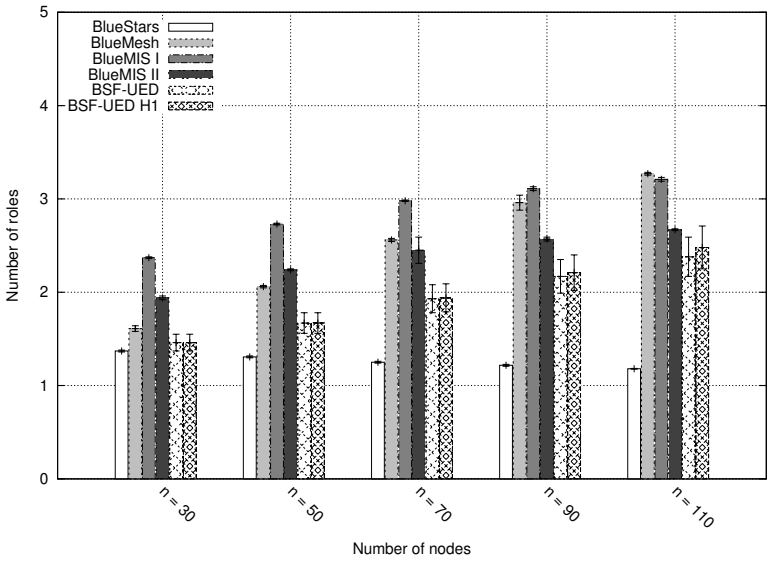


Figure 6.6: Comparison of the average role per node

nodes. The results are shown in Figure 6.6. First, note that heuristics `H1` of `BSF-UED` does not change the average number of role per node significantly. `BlueStars` outperforms all other algorithms according to this metric. The superiority of `BlueStars` is caused by the small number of piconets, the large size of piconets and the condition that its first phase (i.e., phase 1 in `BlueStars`) forms disjoint piconets. Note that the second best algorithm is `BSF-UED`.

6.4.4 Number of bridges

This section studies the number of bridges and M/S bridges in the formed scatternets. The results are given in Figure 6.7 and Figure 6.8. One of the main weaknesses of `BSF-UED` is in the number of M/S bridges in its scatternets. `BlueStars` and `BlueMesh` outperform `BSF-UED` with respect to this metric. This result is related to the *nodes delegation* procedure followed by `BSF-UED` (see Section 6.3.1). That is, 1) the procedure `FindCommonNeighbors()` and 2) the fact that the nodes cannot be slaved by more than one master in phase 1. Note that the node delegation procedure leads to the formation of large number of piconets, which shall be interconnected in phase 2. On the other hand, the same procedure has an important role in improving the average size of the piconets. This is just an example of how improving a scatternet quality metric may degrade another metric. Similarly, `BSF-UED` interconnection rules (see Section 6.3.2) give priorities to rules that form M/S bridges in phase 2. However, the priorities of these rules improve the execution time of `BSF-UED`. The high number of M/S bridges is the cost of achieving heuristically outdegree limited scatternets in short execution time which was one of the main objectives of `BSF-UED`. At the same time, this result is acceptable and its impact is reduced given that `BSF-UED` forms piconets having small number of slaves. Moreover, the nodes in `BSF-UED` scatternets have few roles on average. Therefore, many of the slaves that are mastered by an M/S bridge will not be significantly affected when their masters works as a slave. That is, let assume that v is an M/S bridge and it has u as its slave and w as its master. When v is active in the piconet of its master w, the slaves of piconet v (including u) are inactive in the piconet of v. This is because the master v controls the flow of packets in its piconet. However, because there is a large number of M/S bridges, then the chance that the slave u belongs to another piconet (i.e., other than v) is higher. Therefore, u is active in a piconet other that of v while its master v is not active in its own piconet. This reduces the negative impact of forming a large number of piconets.

`BlueMISI` does not suffer from scatternets with large number of M/S bridges despite the significantly large number of piconets in its scatternets. This is because there are some nodes in the scatternet that have many masters (i.e., very large indegree). Such nodes lead to a major disadvantage in `BlueMISI`, because they may cause bottleneck in the scatternet. That is, the involvement of these nodes as bridges to multiple piconets at the same time may delay the transmission of messages between these piconets, causing thus issues in the scatternet throughput.

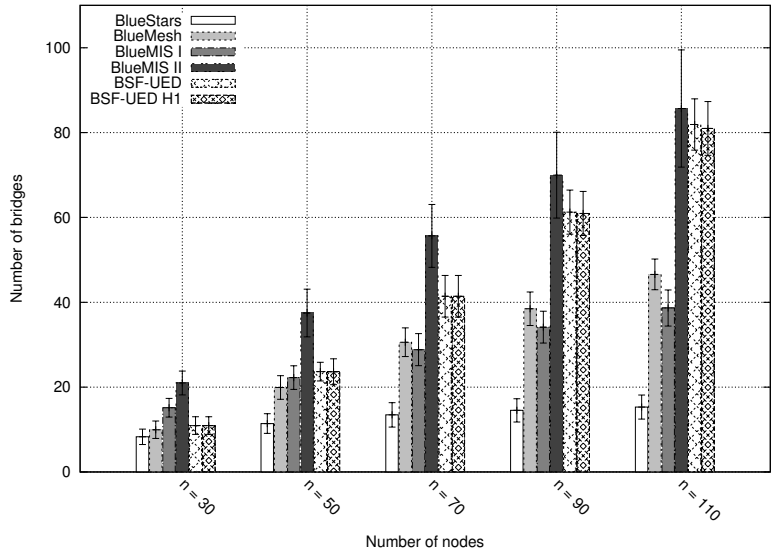


Figure 6.7: Comparison of the number bridges

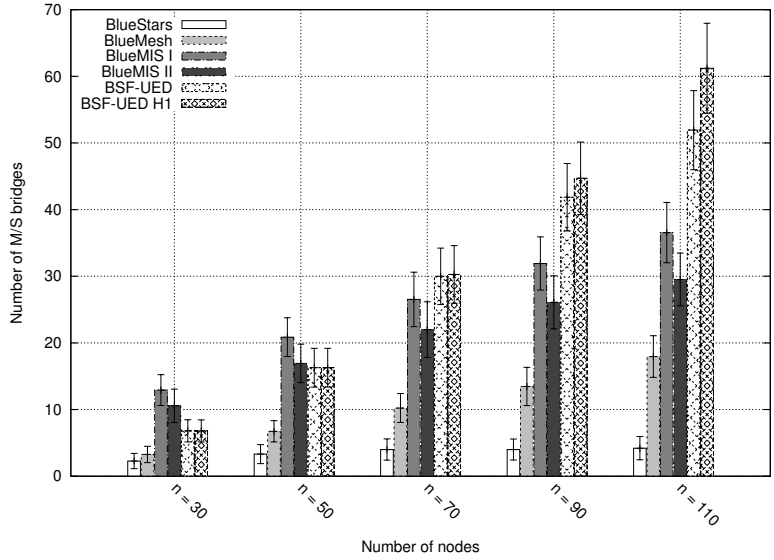


Figure 6.8: Comparison of the number of M/S bridges

6.4.5 Average shortest path

Figure 6.10 shows the average length of the shortest paths of the formed scatternets. The measure gives an indication of the routing cost in general. It is calculated by the minimum number of hops between every pair of nodes on average. Note that, in Bluetooth scatternets, routing between a pair of nodes is not affected only by the number of

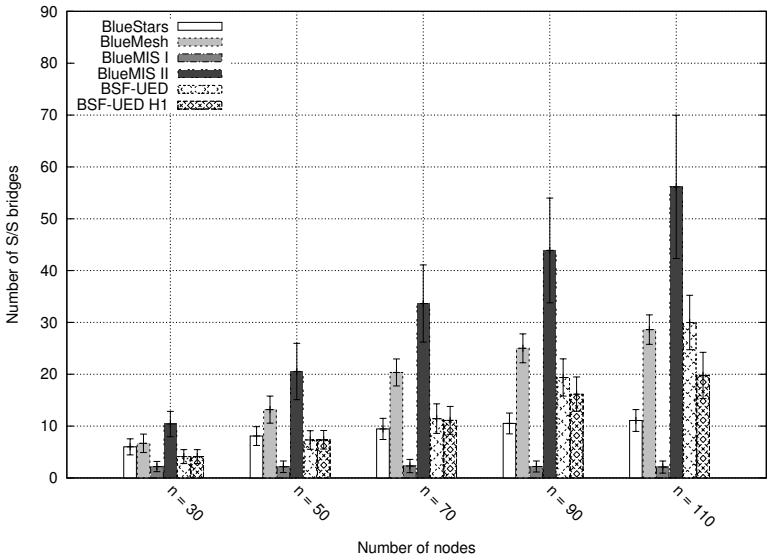


Figure 6.9: Comparison of the number of S/S bridges

hops in the path connecting them, but also by the properties of the nodes in the path (e.g., number of masters in the path, number of slaves, number of bridges, average role per node and number of slaves per master). Figure 6.10 shows that the shortest paths of `BlueMIS` have the lowest number of hops on average. `BSF-UED` and `BlueStars` have approximately similar measurements. For `BSF-UED`, this is a direct consequence of deleting the unnecessary edges, which is a general approach followed in `BSF-UED`. This measure may hence be improved by increasing the number of edges in the scatternet. For instance, masters that do not have their piconets filled with seven slaves may slave random neighbors. Note however that the difference between `BSF-UED` and the best algorithm with respect to this metric is not significant and it is about 1 to 1.5 hops on average.

6.4.6 Number of messages

This section studies the number of messages sent by all the network nodes. This is an approximative measure of the energy consumed by the network during the execution of the algorithm. Note that the study of the energy consumption of BSF algorithms may be more accurate if a detailed energy model of Bluetooth is used. The results are listed in Figure 6.11. A theoretical analysis shows that the number of messages in each of the algorithms is within a constant to the number of edges of the network. Algorithm `BSF-UED` significantly outperforms algorithms `BlueMesh`, `BlueMISI` and `BlueMISII`. This

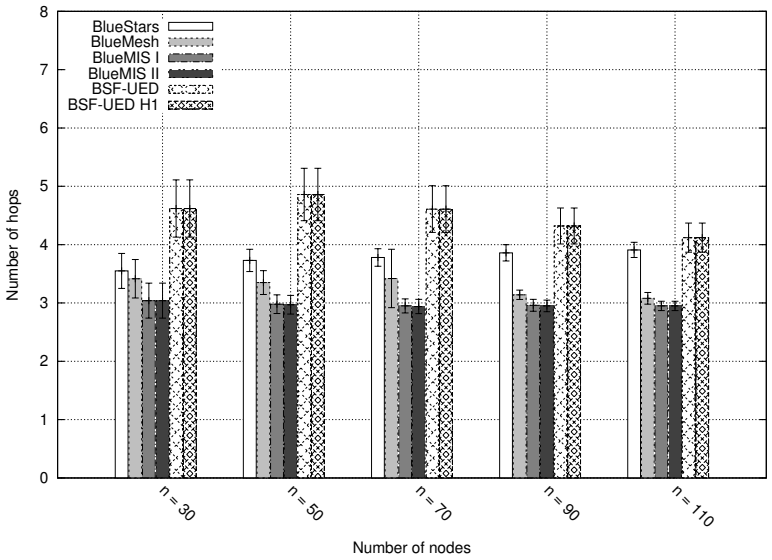


Figure 6.10: Comparison of the average shortest path

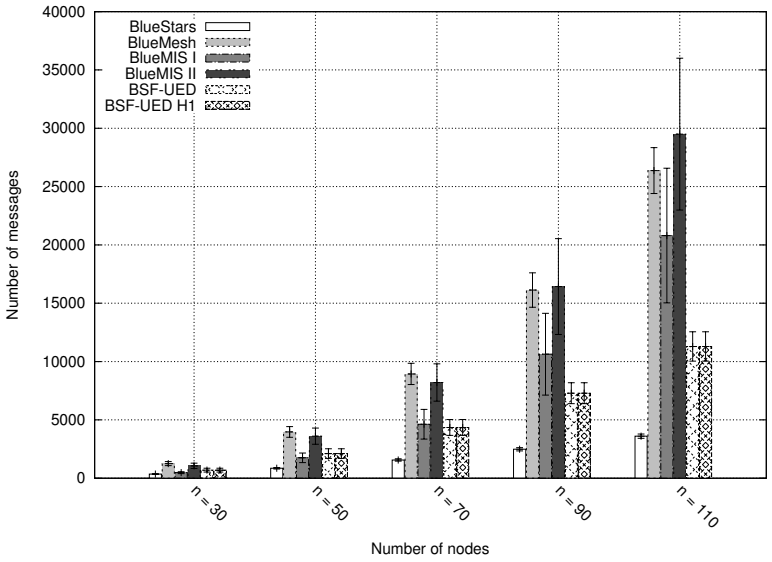


Figure 6.11: Comparison of the number of messages sent in total

is one of the advantages of `BSF-UED`.

6.5 Conclusions

This chapter introduced algorithm `BSF-UED`, which is a BSF algorithm that aimed to improve weaknesses found in major mesh-based BSF algorithms. `BSF-UED` is a time

efficient algorithm that deterministically forms connected scatternets and heuristically forms outdegree limited scatternets. A major objective of `BSF-UED` is to achieve a balance between the conflicting quality metrics of Bluetooth scatternets. Simulation experiments show that `BSF-UED` succeeds it in achieving this objective. `BSF-UED` was compared against major BSF algorithms; namely, `BlueStars`, `BlueMesh`, `BlueMISI` and `BlueMISII`. `BSF-UED` outperformed these algorithms with respect to a number of important quality metrics. Of particular significance is the fact that `BSF-UED` outperforms `BlueMesh` in the average piconet size, average role per node and number of messages; it is also about 3 times faster in execution time. The main weakness of `BSF-UED` is the high number of M/S bridges. This weakness is tolerable to some degree, as explained in Section 6.4.4, since the average number of slaves per piconet and the average roles per node are low. In term of other metrics, `BSF-UED` and `BlueMesh` generate similar results to a certain degree, whereas `BSF-UED` solves the main issues in `BlueMISI`. A future research direction is to solve the issues of `BSF-UED`, and to adapt it to mobile scenarios.

The comparative study performed in this chapter shows that there are many issues to be addressed in the field of BSF algorithms despite the numerous studies in the field. The comparative study shows that it is a challenging task to achieve a perfect balance between the several quality metrics of Bluetooth scatternets. Every BSF algorithm studied is found to suffer from at least one weakness with respect to a certain performance metric. Such weaknesses are usually caused by the improvement of some other quality metrics. The author suggets to build BSF algorithms over simple heuristics which takes into account the execution time as a main priority. After achieving such time-efficient algorithms, additional heuristics shall be considered one at a time. This procedure should be repeated until a good understanding of the BSF problem is obtained, and an algorithm that achieves balanaced performance is designed. This approach has been applied in `BlueStars`, `Eliminate`, `BlueMISI`, and `BSF-UED`, which are all considered as time-efficient algorithms. Among the benefits of a simple time-efficient BSF algorithm is that it can always be improved using additional simple heuristics. This approach has been already used, with not much success, in algorithm `BlueMISII`.

The Outdegree-Limitation Problem : Deterministic algorithms for the outdegree-limited BSF problem were introduced for the unit disk graph model only. Future work should relax this additional assumption. This relaxation will make the problem more challenging and make the introduced algorithms more practical. Moreover, there should

be more research on studying the outdegree limitation problem for any outdegree κ, and not only for $\kappa = 7$ as stated in the specifications. Such studies handles any unexpected changes on the specifications of Bluetooth.

About Bluetooth scatternets performance metrics: This thesis measures the balance between the performance metrics of Bluetooth scatternets based on judgments of the simulation experiments results for each performance metric separately. This approach is used in this thesis to make the results of the simulation experiments comparable to similar previous research. One way to measure performance metrics balance is to give a weight to each performance metric and then combine all the weighted performance metrics in a single new performance metric. This approach requires further investigations to decide how to assign weight to each performance metric. This may require also more experiments on how each performance metric affects the quality of the formed scatternet.

The nature of Bluetooth scatternets led to the adoption of quality metrics that are different than those used in non-Bluetooth networks. However, the large number of quality metrics of Bluetooth scatternets complicates the study of the scatternet performance. Therefore, the author sees that a set of new generalized quality metrics to study the performance of Bluetooth scatternets must be introduced in future work. These quality metrics shall be few in number, be easy to implement, and give a good indication of the scatternet performance. Such generalized quality metrics shall take into consideration the inter-piconet and intra-piconet scheduling in the scatternets. Solving this problem would simplify the design of new BSF algorithms and leads to the introduction of more efficient BSF algorithms.

Changes on the specifications: The author suggests a change in the specifications of Bluetooth in order to make it more suitable for ad-hoc wireless networking. For example, a change in the link establishment procedures is necessary in order to achieve faster BSF algorithms. The author suggests that the restriction of outdegree limited piconets shall be relaxed. Furthermore, more focus shall be given to inter-piconet and intra-piconet scheduling algorithms.

Part IV

Object Name Resolution

Chapter 7

Object Name Resolution

A major component of IoT is the *Object Name System (ONS)*, which is analogue to the Internet Domain Name System (DNS). ONS is a global distributed database whose main objective is to answer the following query: given an object identifier (such as a tag identifier), return the address of the *remote server* of the object (i.e., the server responsible for executing intelligent tasks on behalf of the object, which also includes storing data about the object). This chapter gives a literature survey of existing ONS architectures. The survey extracts the main requirements that shall be met by an ONS architecture. Among these requirements is geographic awareness. The chapter focuses on the geographic awareness requirement, and introduces a geographic aware P2P network, called `GCAN` (Geographic Content Addressable Network), over which the ONS can be built.

7.1 Introduction

Remote intelligence is an important enabling concept of the `IoT`. It states that the tasks that makes an IoT *thing* (or, object) intelligent are delegated to a specialized fixed server in the Internet, called the *remote server* of the object. Intelligence in this context varies from basic intelligence such as storing data about the object (e.g., name, location, or description), to more complex levels of intelligence such as making decisions (e.g., order missing items in a supermarket shelf). See [WMZA02] [MFH09] for more details about the `IoT` objects intelligence levels. The concept opposite to remote intelligence is *local intelligence*. An object is said to have local intelligence if the object intelligent tasks are

performed locally at the object (i.e., a local processor attached to the object). Remote intelligence is especially useful in scenarios where it is not feasible to equip an object with extra costly hardware devices, since remote intelligence can be enabled by attaching an RFID tag to the object.

Remote intelligence leads to an important problem in the `IoT`, called the *object name resolution* problem. Given an object t, the objective of the object name resolution problem is to find the address of the remote server $s(t)$ responsible for t (or simply, the unique identifier of $s(t)$, denoted $\mathbf{id}(s(t))$). There are essentially two general approaches to this problem:

Approach.1 The identifier $\mathbf{id}(s(t))$ is stored in t itself, assuming that t has an internal memory (e.g., t is attached to a barcode, RFID tag or an embedded processor). Despite simplifying the name resolution procedure, this approach complicates the procedure of assigning a different remote server to the object.

Approach.2 The *records* $(\mathbf{id}(t), \mathbf{id}(s(t))$ (or simply, $(t, s(t))$) for all the `IoT` objects, denoted T_{ons}, are stored in a specialized database, called the *Object Name System* (ONS) database. This approach simplifies the modification of any record $(t, s(t))$ for any object $t \in T_{ons}$, but it requires that the ONS database is carefully designed. This approach is followed by EPCglobal framework architecture [epc08] and the Ubiquitous ID architecture [uid06][1]. The EPCglobal framework architecture specifies how the ONS database shall be designed, whereas the Ubiquitous ID architecture keeps the problem open. This chapter focuses on problems caused by following **Approach.2**.

Problem Formulation

Given the large number of objects in the `IoT`, an ONS database must be scalable (that is, must be able to handle the huge amount of objects in the `IoT`). The ONS must also be global (that is, it may be queried from anywhere in the world). In order to achieve these goals, the ONS database must be distributed among several inter-connected database

[1]The Auto-ID center, which is the parent organization of EPCglobal, adopted originally this approach with RFID systems. This is because storing little data in an RFID tag simplifies the tag hardware design. It is believed that this approach led to a significant reduction in RFID tags manufacturing cost [Vio05]. The Ubiquitous ID architecture adopted this approach due to its simplicity and suitability for context-aware ubiquitous applications, which is considered as one of the main objectives of the Ubiquitous ID architecture.

servers, called *local ONS servers* and denoted V_{ons}. Each local ONS server stores records for a subset of all the objects T_{ons} and answers queries about these records. Formally, let $D = \{(t, s(t)) \mid t \in T_{ons}\}$ be the set of all records in the database. Then, every local ONS server v stores a subset $D(v) \subseteq D$, such that $D = \bigcup_{v \in V_{ons}} D(v)$ and $\bigcap_{v \in V_{ons}} D(v) = \emptyset$.

A local ONS server storing the record $(t, s(t))$ of an object $t \in T_{ons}$ is called the *owner* of t, denoted $w(t)$. A local ONS server v shall answer all *lookup* (or, search) queries about the objects it owns. To achieve the lookup function efficiently, the set of local ONS servers V_{ons} shall form a network $G_{ons} = (V_{ons}, E_{ons})$, where E_{ons} is the set of $\binom{|V_{ons}|}{2}$ possible edges between the nodes of V_{ons} (since V_{ons} are Internet nodes)[2]. Further improvement on lookup may be achieved by executing a *network formation algorithm* that optimizes G_{ons}. The network formation algorithm shall find a subset of the edges E_{ons}, denoted E'_{ons}, such that the lookup procedure performance is improved if executed over the network $G_{ons} = (V_{ons}, E'_{ons})$, with respect to:

1. *The routing cost*: measured by the number of messages required to complete the lookup procedure, and

2. *The lookup time*: which is the time required to complete the lookup procedure. This can be measured theoretically (by assuming that each message is sent from one node to its neighbor in 1 unit time) and/or empirically (by assuming that each message m is sent from one node to its neighbor in $t(m)$ seconds.)

The formed network $G_{ons} = (V_{ons}, E'_{ons})$ is called in this chapter as an *ONS architecture*.

Contributions

Section 7.2 extracts a set of essential general requirements for efficient ONS architectures. Later, existing ONS architectures are surveyed. It is found that ONS architectures can be categorized into *hierarchical ONS architectures* (discussed in Section 7.3) and *P2P ONS architectures* or DHT-based ONS architectures, (discusses in Section 7.4). The survey shows that each category handles only a portion of the general requirements but not all of them. However, the P2P ONS architectures may meet all the requirements by implementing techniques used in the field of P2P networks but never used in existing ONS architectures.

[2]Note that an Internet node v becomes a neighbor to another node u by simply knowing its IP address. Every other node in the path between v and u are usually ignored. In other words, v and u runs on top of an *overlay*.

Among the general requirements discussed in Section 7.2, there is the *geographic-awareness* requirement. Briefly, this requirement states that the database servers shall consider their geographic locations in order to improve the overall performance of the system, specifically the performance of the lookup procedure. Interestingly, hierarchical ONS architectures meet this requirement but not the P2P ONS architectures. This chapter suggests the use of geographic-aware P2P networks in order to meet this requirement. Existing geographic-aware P2P networks are surveyed in Section 7.4.2. A new geographic-aware P2P network, called `GCAN` (Geographic-aware Content Addressable Network), is introduced in Section 7.5. `GCAN` is easy to implement as it can be built over existing P2P networks (specifically, *chord-like networks*, which are P2P networks similar to `Chord` [SMK+01]). `GCAN` may be used for ONS or other applications that require geographic awareness, such as multimedia streaming for example.

Chapter Organization

Section 7.2 gives the general requirements of an ONS requirements. Section 7.3 and Section 7.4 surveys existing hierarchical ONS architectures and P2P ONS architectures, as any existing ONS architecture is categorized under one of these two types. Section 7.5 introduces `GCAN`; a Geographic aware Content Addressable Network that can be used to implement the ONS. Section 7.6 concludes the chapter.

7.2 General Requirements of an ONS Architecture

This section introduces the general requirements that an ONS architecture shall meet. These requirements are extracted from the existing ONS and DNS (Domain Name Service) architectures. Note that the DNS is analogue to ONS in the legacy Internet, where DNS is used to find the IP address of a given Internet host name (e.g., `www.uottawa.ca`). An ONS architecture shall meet the following requirements:

1. **Scalability**: the system performance shall not be significantly affected by a large number of inputs (i.e., number of objects and number of local ONS servers).

2. **Fault-tolerance**: the system shall tolerate failures in its nodes and links. This can be achieved, for example, by increasing the number of possible paths between any pair of nodes in G_{ons}.

3. **Avoiding single points of governance**: ONS is used mainly for business applications. Thus, it attracts business espionage, which is considered as a major problem in the business world [Nas04]. Assume for instance that there is a node v in V_{ons} through which all ONS queries pass. Node v may consequently eavesdrops on these queries, analyzes traffic generated by other parties (e.g., companies or governments), and use this analysis for criminal or unethical purposes. To avoid such behavior, nodes with similar characteristics to v shall be eliminated from G_{ons}.[3]

4. **Caching**: that is, if a node received an answer of a query about an object t, then the node shall store this answer for a specific period of time. The node uses the cached record until the record becomes outdated. The node, thus, may avoid re-executing the query unnecessarily.

5. **Replication (or, redundancy)**: that is, a record $(t, s(t))$ shall be stored in the local ONS server $v = w(t)$ and in other local ONS servers, denoted $\{v_1, \ldots, v_k\}$. Thus, the query for an object t, initiated by a node u, may be answered by any node $v' \in \{v, v_1, \ldots, v_k\}$. This approach reduces the execution time since v' may be closer to u, than is v to u. This approach also improves the fault-tolerance of the architecture. This is because queries for an object t are still answered even if $w(t)$ is faulty.

6. **Geographic Awareness**: the ONS is said to be *geographic aware* if its nodes are fully or partially aware of their geographic locations and use this knowledge to improve the overall performance of the system. Geographic awareness achieves the *locality principle*, which is found in many fields in computer science. The principle states that the transport path between a pair of endpoints of the same local region should remain within the region [Den05]. This is because arbitrary detours in the traffic path causes unnecessary overhead that affects the network performance.

   There are three main methods that Internet nodes may use to gain geographic awareness (note that the ONS nodes are all connected via the Internet):

   (a) *Static geographic location:* the exact geographic location of a node is statically stored in its memory.

[3] It is suspected that the PRISM espionage surveillance system was used for business espionage. See [Bar13] for more details.

(b) *Geolocation:* a node may use heuristics to get an approximation of its geographic location, and other nodes in the network. For example, the geographic location of a node may be deduced from the geographic location of its Internet Service Provider (ISP), given that an ISP usually publishes it geographic location publicly. See [MO09] for a survey of geolocation techniques.

(c) *Virtual coordinates:* a node shall not be aware of its exact geographic location as it can use the relative distance between itself and other nodes in the system. The relative distance between a pair of nodes may be measured, for example, by the round trip time of messages between these nodes. Relative distances allow a node to assign itself and other nodes in the networks coordinates in a virtual space, called virtual coordinates. The distance between a pair of nodes in the virtual space gives an indication of the actual distance between them. See [DGK10] for a survey of virtual coordinates techniques.

The *basic requirement* of an ONS architecture is that the network G_{ons} is connected, since the lookup algorithm is not correct if G_{ons} is not connected. The first three requirements in the list above are important, especially in the case of the `IoT`. The last three requirements give more practicality for the ONS.

7.3 Hierarchical ONS Architectures

Existing ONS architectures are categorized into *hierarchical* architectures and *P2P* architectures. This section gives a brief survey of existing hierarchical architectures, whereas P2P ONS architectures are surveyed in Section 7.4.

EPCglobal introduced and standardized an ONS architecture in [epc08], called `EPCglobal ONS` for simplicity. This architecture is part of the Internet DNS (Domain Name System). As a result, `EPCglobal ONS` inherited the main features and maturity of the DNS. In addition to local ONS servers, `EPCglobal ONS` uses a set of additional servers, called *root ONS servers* to inter-connect the local ONS servers. This is a main disadvantage of `EPCglobal ONS` since the problem can be solved without any additional servers.

To understand how `EPCglobal ONS` works, a brief idea of DNS is given in the following. *DNS nodes* are categorized into 1) local DNS servers and 2) root DNS servers. The root servers are used to interconnect the local DNS servers. Each DNS node is

responsible for answering queries for certain *hostnames*. Each hostname has the format $\mathtt{x_m}.\ \ldots\ .\mathtt{x_0}.\epsilon$ (where, $\mathtt{x_i}$ is a string with a certain format for each $0 \leq i \leq m$, and ϵ is the empty string, which is usually ignored). For example, `www.uottawa.ca`. is a hostname. There is a root server v_{rs} responsible for all hostnames with the format $\mathtt{x}.\epsilon$. Assume that the hostnames with this format are $\{\mathtt{v_1}.\epsilon,\ \ldots\ ,\mathtt{v_k}.\epsilon\}$. Then, v_{rs} stores the records $\{(\mathtt{v_1}.\epsilon, s(\mathtt{v_1}.\epsilon)),\ \ldots\ ,(\mathtt{v_k}.\epsilon, s(\mathtt{v_k}.\epsilon))\}$, where $s(\mathtt{v_i}.\epsilon)$ is the DNS node responsible for the hostname $\mathtt{v_i}.\epsilon$ (or simply, $\mathtt{v_i}.$). Therefore, the DNS nodes $\{s(\mathtt{v_1}.),\ \ldots\ ,s(\mathtt{v_k}.)\}$ are the *children* of v_{rs}. Well-known children for the actual root DNS server v_{rs} are $s(\mathtt{com}.)$, $s(\mathtt{net}.)$, and $s(\mathtt{org}.)$. Each DNS node $s(\mathtt{v_m}.\ \ldots\ .\mathtt{v_0}.)$ builds the set of its children similarly by storing records for all hostnames with the format $\mathtt{x}.\mathtt{v_m}.\ \ldots\ .\mathtt{v_0}.$. This architecture facilitates the lookup procedure. Every query for a hostname is forwarded first to the root v_{rs}, then to a child of v_{rs}, etc ..., until the server responsible for the queried hostname is found.

`EPCglobal ONS` proposes to convert the format of the identifier of each object t into a format understood by the DNS. Therefore, $\mathbf{id}(t)$ is converted into $\mathtt{t^0}.\ \ldots\ .\mathtt{t^k}.\mathtt{onspec.com}.$. The conversion function depends solely on $\mathbf{id}(t)$. See [epc08] for more details about the conversion function. The DNS is queried to find $s(t) = s(\mathtt{t^0}.\ \ldots\ .\mathtt{t^k}.\mathtt{onspec.com}.)$. Thus, `EPCglobal ONS` is a subtree of the DNS rooted at $s(\mathtt{onspec.com}.)$. The advantage of `EPCglobal ONS` is that it inherits the maturity and reliability of DNS. On the other hand, the existence of a root node in `EPCglobal ONS` (i.e., the *unipolarity* of ONS) leads to the following weaknesses:

1. *The root of `EPCglobal ONS` presents a single point of failure*: That is, the system fails if the root fails. The probability of failure is decreased by replicating the root server and its children. Partial failure of the Internet occurred in 2002 and 2007 due to Distributed Denial of Service (DDoS) attacks [dns07].

2. *The root of `EPCglobal ONS` presents a single point of governance*: This occurs because the root of `EPCglobal ONS` is managed by a company called `Verisign` [Vio04]. Single-point of governance is a major problem that hinders the fast adoption of the `IoT` and ONS, since both are used mostly for business applications. Single-point of governance of the DNS constituted always a debating issue. See [Kru13] for more details.

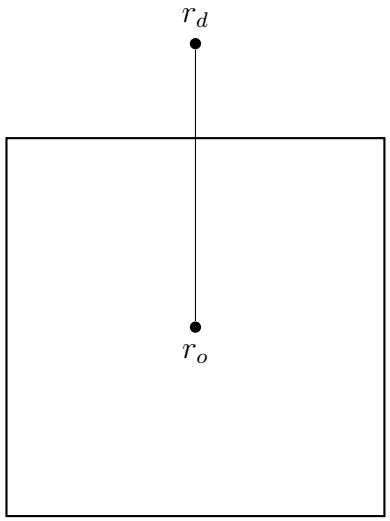
3. *The root of `EPCglobal ONS` presents a problem in the scalability of the ONS*: the

root servers in `EPCglobal ONS` may not handle the large number of ONS queries, which may be significantly larger than the number of DNS queries.

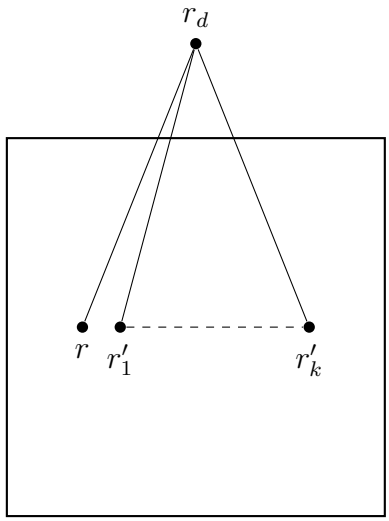
Evdokimov et al. proposed in [EFG08] that the root of `EPCglobal ONS` shall have many replicates, each of which is managed by a different organization. The architecture is called **Replicated Multipolar ONS**. For synchronization, the roots share a file called the *root file* that is managed by an independent organization. Unipolarity is not totally eliminated in this architecture due to the existence of the root file. Evdokimov et al. introduced also in [EFG08] an architecture called the **Regional Multipolar ONS** (`RMONS`). There are several roots in `RMONS` (called *regional roots*), each of which controls a disjoint *region*. A region can be a city, country, continent or others. Each regional root is connected directly to every other regional root (i.e., forms a complete network). Lookup procedure between local ONS servers that belong to different regions is as follows. Assume a local ONS server v located in the region $\texttt{R}(v)$ initiated a query for object t, which is owned by the local ONS server $u = w(t)$ located in region $\texttt{R}(u)$. First, v extracts the region where t exists from its unique identifier $\mathbf{id}(t)$. (This is possible in the EPCglobal standard since the EPC code contains a field for the country of the company that the object originated from [EFG08]). Let assume that $\texttt{R}(v) \neq \texttt{R}(u)$. Then, v forwards the query to the root of region $\texttt{R}(v)$, which itself forwards the query to the root of $\texttt{R}(u)$. The root of $\texttt{R}(u)$ lookup for $u = w(t)$ in its region using a lookup algorithm specific to region $\texttt{R}(u)$. `RMONS` does not totally eliminate the unipolarity of `EPCglobal ONS` due to the existence of regional roots.

A similar regional ONS architecture, called **Federated ONS** (`FONS`), is introduced in [BKFS11]. The newer version of `EPCglobal ONS` (see [epc12]) is based on `FONS`. This architecture uses a simple technique to inter-regional queries. Assume that v in $\texttt{R}(v)$ initiated a query for object t owned by $w(t)$. The query is first forwarded automatically to the root of $\texttt{R}(v)$. This is done by appending $\texttt{addr}(\texttt{R}(v))$ to the end of the object address (for example, $\texttt{addr}(\texttt{R}(v))$ is `onsepc.fr` if the region $\texttt{R}(v)$ is France, or `ons.epcglobalcanada.org` if the region $\texttt{R}(v)$ is Canada according to [epc12]). If the root of $\texttt{R}(v)$ finds that the query shall be forwarded to another region $\texttt{R}(w(t))$, then $\texttt{addr}(\texttt{R}(v))$ shall be replaced by $\texttt{addr}(\texttt{R}(w(t)))$. Otherwise, the root of $\texttt{R}(v)$ looks up for $w(t)$ in its region using the standard DNS lookup procedures. `FONS` eliminates the unipolarity of the `EPCglobal ONS` root server, but preserves the unipolarity of the DNS root server.

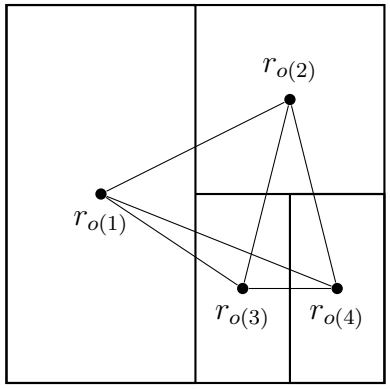
Lastly, Ning et al. suggested in [NLCQ12] that the regions shall be further divided



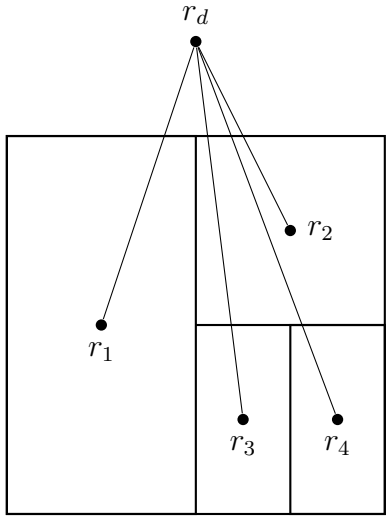
(a) `EPCglobal ONS`: A single ONS root r connected to the DNS root r_d.



(b) Replicated Multipolar ONS: An ONS root r with k replicated ONS roots $\{r_1, \ldots, r_k\}$, all connected to the DNS root r_d.



(c) Regional Multipolar ONS. Each region is rooted at r_i. Regional ONS roots connected to each other.



(d) Federated ONS. Each region is rooted at r_i. Regional ONS roots connected to each other by connecting to the DNS root r_d.

Figure 7.1: Major differences between hierarchical ONS architectures with respect to regions.

into **business areas** which consists of a set of companies. The division improves the efficiency of routing between companies that have high collaboration and may generate a high traffic between each others.

7.3.1 Caching and Replication in Hierarchical ONS Architectures

Hierarchical ONS architectures use the same caching and replication techniques used in DNS. **Replication** in DNS is implemented by copying the records of a DNS node v in a set of k other nodes $\{v_1, \ \ldots \ , v_k\}$. If a node u receives a query for a hostname owned by v, then u returns the union of v and its replicates (i.e., $\{v \cup \{v_1, \ldots, v_k\}\}$). For example, the root DNS server returns a result containing 13 addresses if queried for `com.`. A node receiving this result shall contact only one of these replicates. **Caching** in DNS is implemented by letting a server stores the result of its queries for a specific period of time, called *Time To Live* (`TTL`). The server uses the cached record instead of contacting the DNS repeatedly and unnecessarily. Assume, however, that the cached record led to a wrong server (e.g., the record has been assigned to a different server). Then, this causes a *cache miss*. Cohen and Kaplan proposed in [CK03] a set of heuristics to solve this problem. The proposed solutions depend on continuous unsolicited queries to update the cached values. This method decreases the number of cache misses, but lead to a higher number of transmitted messages.

7.3.2 Geographic-Awareness in Hierarchical ONS Architectures

`RMONS` [EFG08], `FONS` [BKFS11], the new standard of `EPCglobal ONS` [epc12] and the work of [NLCQ12] are all geographic aware, since they depend on regional roots each of which is responsible for a certain geographic region. These architectures assume that the unique identifier of each object holds information about the geographic location of its owner. Moreover, hierarchical ONS architectures use the DNS geographic-awareness technique. This technique is explained in the following. Recall that a node v_0 in DNS may be replicated into k replicates $\{v_1, \ldots, v_k\}$. Assume that u queried a node w for object t owned by v_0. Then, node w returns to u the addresses of v_0 and all its replicates. Node u selects the address of the *closest* node in $\{v_0, v_1, \ldots, v_k\}$ (i.e., the node that achieved historically the shorter lookup time). This problem is called the *DNS server selection* problem.

The most common solution of the DNS server selection problem is used in the `BIND` protocol [Alb01]. The closeness of nodes according to `BIND` is calculated by the exponential smoothed average response time. Essentially, a node u stores temporarily the addresses of $\{v_0, v_1, \ldots, v_k\}\}$ once it received them, and stores an estimation of the av-

erage response time for each of these nodes, denoted $\mathbf{resp}_{avg}(v_i)$ for $v_i \in \{v_0, \ldots, v_k\}$. The average response time $\mathbf{resp}_{avg}(v_i)$ is calculated using the last response time of v_i, denoted $\mathbf{resp}(v_i)$ using the following equation:

$$\begin{aligned} \mathbf{resp}_{avg}(v_i) &= \alpha\ \mathbf{resp}_{avg}(v_i) + (1-\alpha)\ \mathbf{resp}(v_i) \\ \mathbf{resp}_{avg}(v_j) &= \beta\ \mathbf{resp}_{avg}(v_j)\ \ \forall j \neq i, 0 \leq j \leq k \end{aligned}$$

where, α and β are hard coded variables ($\alpha = 0.7$, $\beta = 0.3$ - see [DSP08]). Deb et al. introduced in [DSP08] an improved solution for the DNS selection problem. The method uses a probabilistic auto-regression model to predict the value $\mathbf{resp}_{avg}(s_i)$. Simulation experiments in [DSP08] shows that this method achieves a better performance.

Section Conclusion

The existence of a root or several roots (i.e., the unipolarity) in hierarchical ONS architectures prevent them from achieving the scalability and fault-tolerance requirements. It also leads to the existence of a single-point of governance. On the other hand, these architectures meet the 1) *caching*, 2) *replication* and 3) *geographic awareness* requirements, since these architectures are based on the DNS. Hierarchical ONS architectures inherit the maturity and reliability of the DNS. Future work shall focus on improving the scalability and fault-tolerance of these architectures and remove any sources of unipolarity. Interestingly, P2P ONS architectures, surveyed in the next section, achieves these objectives and may achieve the other requirements using simple additions.

7.4 P2P ONS Architectures

This section surveys P2P ONS architectures, which are basically ONS architectures built on top of P2P Distributed Hash Tables (DHT). A definition of DHT is given first. Then, more details about `Chord` [SMK+01] and `Pastry` [RD01a] are given, which are the P2P DHT's over which existing ONS architectures are built. The section shows later how `Chord` and `Pastry` are used to implement ONS architectures. Details on how caching, replication and geographic awareness are handled by P2P ONS architectures are also

discussed in the section.

P2P Distributed Hash Tables

A *Distributed Hash Table* is a data structure that stores records on the form (**key**, **value**(**key**)), and accepts the operations:

1. `get`(**key**), which returns the value associated with **key** (that is, **value**(**key**)), and
2. `put`(**key**, **value**(**key**)), which stores the specified record.

The data structure is distributed among several nodes, denoted V_{dht}. The operations `get` and `put` shall be correctly executed at any node $v \in V_{dht}$. For example, if node $v \in V_{dht}$ executes `get`(k), while the record (k, **value**(k)) is stored at node $u \in V_{dht}$, then 1) v shall send a message to u, 2) u locally executes the operation `get`(k) (i.e., get **value**(k) from its internal memory), then 3) u returns the result of the operation to v. However, the transmission of these messages may follow a path of nodes starting by v and ending at u. All of these nodes are in V_{dht}. Thus, the nodes V_{dht} form a network $G_{dht} = (V_{dht}, E_{dht})$ such that no centralized node exists in G_{dht}. The network G_{dht} is called a *P2P network* since each node may communicate with others in a decentralized network.

Due to similarity with the terms introduced in Section 7.1, operation `get` is called the *lookup procedure* and its performance is also measured by: 1) the routing cost, and 2) the lookup time. (see Section 7.1 for more details). To improve the performance of the lookup procedure, the network $G_{dht} = (V_{dht}, E_{dht})$ shall be optimized by running a network formation algorithm that forms a subnetwork $G'_{dht} = (V_{dht}, E'_{dht})$, where $E'_{dht} \subseteq E_{dht}$. This optimization usually aims to reduce the number of neighbors of each node to make the network more scalable, while keeping the lookup procedure performance acceptable. Lastly, a DHT shall execute the following operations:

1. `join`(v): node v shall be able to join the nodes V_{dht} and collaborate with them in storing the data.
2. `leave`(v): node v shall be able to leave the nodes V_{dht}. The DHT is formed by $\{V_{dht} \setminus v\}$ after `leave`(v).

Efficient implementations of `join` and `leave` are essential for the scalability of DHT. All the properties of DHT, mentioned above, make it suitable for the implementation of a scalable ONS architecture.

Example: Chord-like networks (e.g., `Chord`, `Pastry`)

`Chord` [SMK+01] is one of the most common P2P networks and DHT's. Many P2P networks are basically modifications of `Chord`. Example of those are: `Pastry` [RD01a], `SkipNet` [HJS+03] and `Kademlia` [MM02]. These networks are called *Chord-like networks*. Existing P2P ONS architectures are built over `Chord` or `Pastry`. Thus, a brief idea of these networks and chord-like networks is given in the following.

`Chord` assigns the nodes V_{dht} unique identifiers from the set $\{0, \ldots, 2^\beta - 1\}$ for a constant integer β . The keys (or, identifiers) of the objects stored in `Chord` are assigned from the same set. Thus, there is at most $n_{\max} = 2^\beta$ nodes and objects records in `Chord`. The nodes are sorted with respect to their identifiers on the form $v_{(1)}, \ldots, v_{(n)}$ where $n \leq n_{\max}$ and $v_{(i)}$ is the i^{th} ranked node among the nodes currently available in V_{dht} for $1 \leq i \leq n$. Node $v_{(i+1)}$ is called the *successor* of $v_{(i)}$, denoted $\texttt{succ}(v_i)$, while $v_{(1)}$ is the sucessor of $v_{(n)}$. Each node connects to its successor. This guarantees the formation of a connected ring network. Lastly, each node v is responsible for (or, *owns*) all objects with identifiers in the range $[\mathbf{id}(v), \texttt{succ}(v))$. (that is, for each object t with identifier in $\mathbf{id}(t) \in [\mathbf{id}(v), \texttt{succ}(v))$, $v = w(t)$). `Pastry` follows the same principle for organizing the nodes.

Lookup in `Chord` and `Pastry` is performed in a greedy fashion. A node v executing $\texttt{get}(t)$ forwards the query to the neighbor $u \in N(v)$ with identifier $\mathbf{id}(u)$ closest to $\mathbf{id}(t)$. The forwarding procedure continues until the query reaches the owner of t' (i.e., $w(t)$). The routing cost of the lookup algorithm given above is $\Theta(n)$, while the size of the neighbors list $N(v)$ at any node v is 2 (i.e., the successor and the predecessor). To improve the routing cost, each `Chord` or `Pastry` node v creates a set of *long edges* (that is, add new neighbors, denoted $N^l(v)$, to its neighbors list $N(v)$). In both `Chord`, and `Pastry`, the set of long neighbors $N^l(v)$ of a node v is organized as $\{w(\mathbf{id}(t_{v_0})), \ldots, w(\mathbf{id}(t_{v_{\ell-1}}))\}$, where $w(\mathbf{id}(t))$ is the owner of the identifier of t. `Chord` specifies that $\mathbf{id}(t_{v_i}) = \mathbf{id}(v) + 2^i \ \mathbf{mod}\ n_{\max}$ for $1 \leq i < \log n_{\max}$. Note that it is possible that $w(\mathbf{id}(t_{v_i})) = w(\mathbf{id}(t_{v_j}))$ for $1 \leq i < j < \log n_{\max}$. `Pastry` specifies that t_{v_i} is an object whose identifier $\mathbf{id}(t_{v_i})$ shares a prefix of i bits with $\mathbf{id}(v)$, whereas the $(i+1)^{th}$ bit is different. Thus, $\mathbf{id}(t_{v_0})$ shares no prefix bits with $\mathbf{id}(v)$, while $\mathbf{id}(t_{v_\ell}) = \mathbf{id}(t_{v_{\log n_{\max}-1}})$ shares all bits with $\mathbf{id}(v)$ except the least significant bit. Note that this gives more freedom for a `Pastry` node to select its long neighbors. This is an important feature that is used by `Pastry` to implement a geographic awareness heuristic. The methods of long edges construction

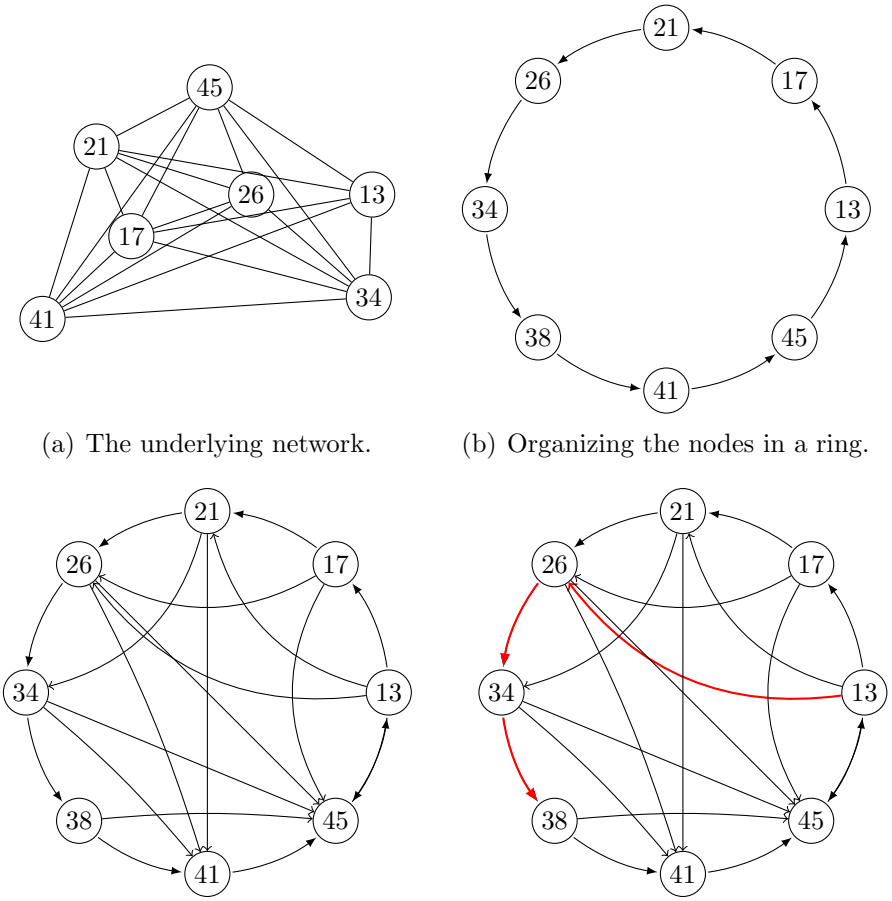
in `Chord` and `Pastry` guarantee that the routing cost is $O(\log n)$ assuming that the identifiers of the nodes are uniformly and randomly assigned. An illustrative example for the lookup procedure in `Chord` is given in Figure 7.2. Lastly, `Chord` and `Pastry` improve their networks *fault tolerance* by letting each node connects to $r \in \Theta(1)$ successors and predecessors.

The technique of constructing a ring structure with long edges is used in well-known P2P networks such as `Pastry` [RD01a], `Kademlia` [MM02], `SkipNet` [HJS+03], `Koorde` [KK03] and `Viceroy` [MNR02]. These P2P networks constructs the set of long edges differently and differ slightly in the lookup algorithm, which remains greedy in all these networks. P2P networks that follow this approach are called *chord-like networks*. Most existing chord-like network has a routing cost of $O(\log n)$ while requiring $O(\log n)$ long edges per node. There are few exceptions such as `Viceroy` [MNR02] for example, which preserves the $O(\log n)$ bound on routing while requiring only $O(1)$ long edges per node on average. `Koorde` achieves $O(\log n \setminus \log\log n)$ routing cost with $O(\log n)$ long edges per node, or $O(\log n)$ routing cost with $O(1)$ long edge per node.

How P2P ONS Architectures use P2P Networks?

Fabian and Gunther proposed in [FG07] to use P2P DHT's to implement the ONS, as P2P DHT's provide better load balancing between the ONS nodes, avoid single points of failure and governance, and are scalable. Moreover, P2P networks increase the security level of the system due to non-existence of a root. A P2P ONS architecture that is built over a version of `Pastry`is introduced in [Fab09]. The main objective of this architecture is to overcome security issues in the ONS. These issues are out of the scope of the thesis. Itsuki and Fujita in [IF09] and Xu et al. in [XWW11] introduced P2P ONS architectures that are based `Chord`. Ramasubramanian and Sirer introduced in [PPPW04] `CoDNS`, which is a P2P DNS architecture that can be integrated with the legacy DNS. `CoDNS` is built over `Pastry`. Similar P2P-based DNS architectures are `DDNS` [CMM02], built on top of `Chord`, and `Overlook` [TJ02], built on `Pastry` [RD01a].

There are essentially two methods to build an ONS system over a P2P network. The first method is to inter-connect the local ONS servers in a P2P network. ONS architectures that use this method are found in [Fab09] and [IF09]. The `CoDNS` architecture uses this method as well. This method eliminates the need of using root servers to interconnect local ONS servers as it is the case in hierarchical ONS architectures. The



(a) The underlying network.

(b) Organizing the nodes in a ring.

(c) Constructing the long edges of `Chord`.

(d) Path of a query initiated by node 13 for object 40.

Figure 7.2: Example of `Chord`. The nodes are shown according to their geographic locations in (a). The nodes of V are organized in a ring structure in (b). The example assumes that each node v identifier $\mathbf{id}(v)$ is shown inside the node. The identifier of nodes and objects is uniform in the interval $[0, 63]$ (i.e., $n_{\max} = 64$). Th ring structure of (b) constitutes the basic structure of `Chord`. Each node v is responsible for all objects in the range $[\mathbf{id}(v), \texttt{succ}(\mathbf{id}(v))$. For example, node 26 is responsible for all objects with identifier in the range $[26, 34)$. The locations of nodes in (b) are virtual for clearer demonstration. The procedure of building long edges is demonstrated in (c). For example, node 13 connects to the nodes responsible for the objects with identifiers in $\{(\mathbf{id}(13)+2^i)\ \mathbf{mod}\ 64 \mid 0 \leq i < \log 64\}$ $= \{13, 13, 17, 21, 26, 45\}$. Thus, node 13 connects to $17, 21, 26$ and 45. In (d), the path of a query for object 40 initiated by node 13 is demonstrated. The owner of 40 is 38. The query is forwarded from 13,26, 34 to 38.

second method is to inter-connect the root servers (e.g., the root servers in DNS, the regional root servers in `RMONS` or `FONS`) in a P2P network. This method is used by the ONS architecture introduced in [XWW11]. According to the author of this thesis, there is no need to use the second method since the number of root servers to be connected is

small.

7.4.1 Caching and Replication in P2P ONS Architectures

The current ONS P2P architectures do not explicitly state how caching and redundancy are achieved. This section, however, surveys some common caching and replication techniques used in P2P DNS architectures or other P2P networks used for similar applications. The first of these applications is `DHash` [DKK+01] which is a file storage management system built on top of `Chord`. `DHash` implements replication by copying the records of a node v into the memory of l successors and predecessors. Caching in `DHash` is done as follows. Assume that the path followed by a query's answer for object t is $\{v_0, \ldots, v_m\}$. Then, each node v_i in the path stores the address of the node from which it received the query's answer. A record in the cache is stored for a `TTL` seconds, where `TTL` is the Time To Live period of the cache. Thus, a query for t may be answered in the future by any node v_i found in the path if the record is still cached. `PAST` [RD01b], which is built over `Pastry`, uses a similar caching technique. This approach of caching is called *passive caching*, since it caches the queries answers as they pass via the lookup path without using any additional information.

Another approach for caching is called *proactive caching*. This approach can be seen as a mixture of caching an replication. An example of a proactive caching algorithm is `Beehive`, which is used by `CoDNS`. `Beehive` runs over `Chord` or any other P2P network that uses prefix routing [PRR99], which includes several chord-like P2P networks. The idea of `Beehive` is to replicate records in the system depending on their popularity. Each node divides the network nodes into layers from 0 to $\log n_{\max}$, where the identifiers of the nodes in layer i shares i bits in prefix. Therefore, every node is in layer 0 of a given node v, and only one node v exists in the layer $\log n_{\max}$ of v, where $n_{\max}$ is the maximum number of nodes that may join a chord-like network. Copying a record in each node in layer i allows it to be reached within i hops. A node v storing an original or a replicated record copies the record into lower layers depending on its *popularity*, where the definition of a record's popularity is given in [RS04]. `Beehive` is shown in [RS04] to reduce the routing cost to $O(1)$ on average. A similar method is used in `Overlook` [TJ02]. An `Overlook` node v replicates a record of an object t in a neighbor node $u \in N(v)$ if it is found that the number of queries for t is higher than a threshold. Node u is the neighbor of v that forwarded the highest number of queries of t to v.

7.4.2 Geographic Awareness in P2P ONS Architectures

The existing P2P ONS architectures do not consider implicitly the problem of geographic awareness. P2P ONS architectures built over `Chord` (e.g., [Fab09], [IF09] and [XWW11]) do not use any geographic awareness technique since `Chord` itself is not geographic aware. P2P ONS architectures built over `Pastry` explicitly use the geographic awareness technique of `Pastry`, which is discussed in the following section. This suggests that if the ONS is built over a geographic aware P2P network, then the P2P ONS architecture is also geographic aware. This section, thus, surveys existing geographic aware P2P networks. The techniques surveyed in this section may be build geographic aware P2P ONS architectures. These networks or techniques are categorized into:

1. *Geographic aware add-ons for chord-like networks*,
2. *Region-based geographic aware P2P networks*, and
3. *Tree-based geographic aware P2P networks*.

The details of these categories are discussed in the following.

Geographic Aware Add-ons for Chord-like Networks

Chord-like networks suffer from the lack of geographic awareness. Additional procedures shall run on top of these networks to make them geographic aware. The main advantage of these procedures is that they are built over the well-studied chord-like networks. An example of these procedures is `Vivaldi` [DCKM04] whose objective is to give the nodes *virtual coordinates* in a virtual Euclidean d-dimensional space S_{vir}. The Euclidean distance between a pair of nodes in S_{vir} gives an indication about the geographic distance between them. Given this knowledge, lookup in `Chord` is slightly modified as follows. A node v forwards a query, destined to node u, to the neighbor $w \in N(v)$ such that $\mathbf{id}(w) \in [\mathbf{id}(v), \mathbf{id}(u)]$ and w is closest to u in the virtual space S_{vir}. The main issue of in `Vivaldi` is that the virtual coordinates of the nodes may change as nodes join or leave the network. Thus, there may be a need for a centralized server. This server is continuously queried to obtain the up-to-date coordinates for each neighbor of a node each time this node wants to forward a query. Otherwise, the nodes may risk using records that are not up-to-date.

`Pastry` [RD01a] introduced a similar procedure. The set of long neighbors $N^l(v)$ of any `Pastry` node is divided into $N^l(v) = \{w(\mathbf{id}(t_1)), \ldots, w(\mathbf{id}(t_{\log n_{\max}-1})\}$, where $n_{\max}$ is the maximum number of nodes in `Pastry`. The identifier $\mathbf{id}(t_i)$ in $N^l(v)$ shares i prefix bits with $\mathbf{id}(v)$ while the $(i+1)^{th}$ bit is different. There may be a set objects $T_{p_i}(v)$ where every $t \in T_{p_i}(v)$ shares i prefix bits with v. Thus, v may chose to connect to any of the owners of the objects in $T_{p_i}(v)$ for $1 \leq i \leq \log n_{\max} - 1$. Using this property, `Pastry` introduces a heuristic that gives it geographic-awareness. The heuristic details are found in [RD01a].

Space filling curves are suggested in [KW06] and [SOTZ05] to add geographic awareness to chord-like networks. These curves are part of various *dimension reduction techniques* that has applications in different domains. Assume that the nodes are geographically located on a 2-dimensional space (e.g., a projection of the earth surface or part of it)[4]. Let the nodes be projected on a *line*, such that 1) the line is represented as the interval $[0, \ell - 1]$, 2) each node v is projected at coordinate $x_v \in [0, \ell - 1]$, and 3) the distance between a pair of nodes in line is approximately the same as the distance between them in the 2-dimensional space. The identifier of a node v, denoted $\mathbf{id}_p(v)$ is set to its projected coordinate x_v. The nodes build a chord-like network using the identifiers $\mathbf{id}_p(v)$ for each node v, whereas the projection line is called a *space filling curve*. The resulting chord-like network is geographic aware. *Hilbert curve* is an example of space filling curves (see Figure 7.3). The main issue of using space filling curves is that a deterministically correct solution of this problem is not always possible [GL96]. Thus, there may be cases where a pair of nodes are close to each in the 2-dimensional space but are significantly far from each other in the projected line (or, curve).

Region-based Geographic Aware P2P networks

Multidimensional P2P networks constitute an important category of P2P DHT networks that are used for handling multidimensional data. The nodes and objects identifiers in these networks are points in a d-dimensional hyperplane. Each node owns a set of points that form a disjoint *region* in the d-dimensional hyperplane. This property makes multidimensional P2P networks suitably used as geographic aware P2P networks, as the

[4]A geographic location on the surface of the earth is represented by its *longitude* and *latitude*. A perfect projection of the nodes on a 2-dimensional Euclidean space (e.g., a 2-dimensional map) is not always possible. An approximation *map projection* method is used for this purpose. Among the most used map projection methods is the *Azimuthal projection*. A survey of the most common map projection methods can be found in [Fur12]

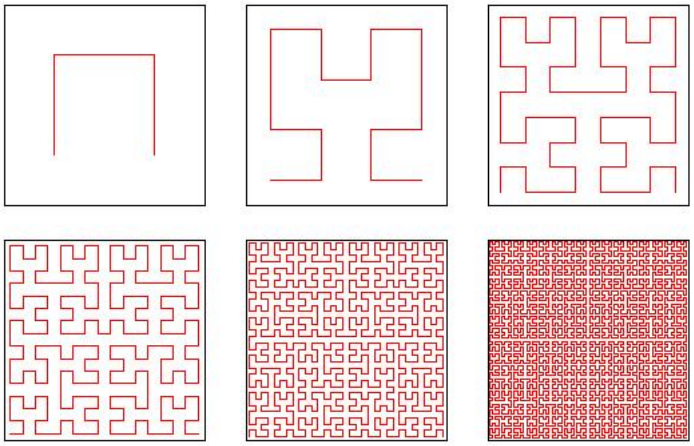


Figure 7.3: Hilbert space filling curve (source: wikimedia.org)

nodes and objects may be presented on a 2-dimensional space.

One of the first existing multidimensional P2P networks is `CAN` [RFH+01]. The d-dimensional space is divided into hyper-rectangular regions. Each region $\mathtt{R}(v)$ is controlled by exactly one node v. A pair of regions are called *neighbors* if they share a side (or part of it). That is, two regions are neighbors if it is possible to draw a line that passes by both of them without intersecting with any other region. The nodes V_p of the networks forms a graph $G_p = (V_p, E_p)$, where each node $v \in V_p$ connects to all the nodes controlling neighbor regions (called *neighbors of* v and denoted $N(v)$). This guarantees the connectivity of G_p. Routing in `CAN` is greedy. That is, each node v forwards the message to the neighbor $w \in N(v)$ which is closest to the destination u in the d-dimensional space, where the distance is Euclidean. The routing cost is $O(d\ n^{\frac{1}{d}})$ assuming that the nodes are uniformly distributed in the space.

A major issue in `CAN` is that a `CAN` node does not necessarily control the region it is located in. This is caused by the join procedure of `CAN` and explained in Figure 7.4. This issue hinders the use of `CAN` as a geographic aware P2P network. Another weakness in `CAN` is its routing cost, which is $O(dn^{\frac{1}{d}})$ assuming that the nodes are randomly uniformly distributed. This routing cost is considered high compared to the routing cost of chord-like networks. The main reason for this cost is the lack of *long edges* in `CAN`. Fortunately, long edges for `CAN` are introduced in `sCAN` [Sun07], `eCAN` [XZ02] and `c`^{2} [CZQ+05]. A similar network to `CAN` is `GeoPeer` [AR04]. The main difference is that the regions in `GeoPeer` are Voronoi diagram regions[5]. The routing cost in `GeoPeer` is $O(\sqrt{n})$ given that

[5]A Voronoi diagram region of a node v is the set of points in the space which are closest to v,

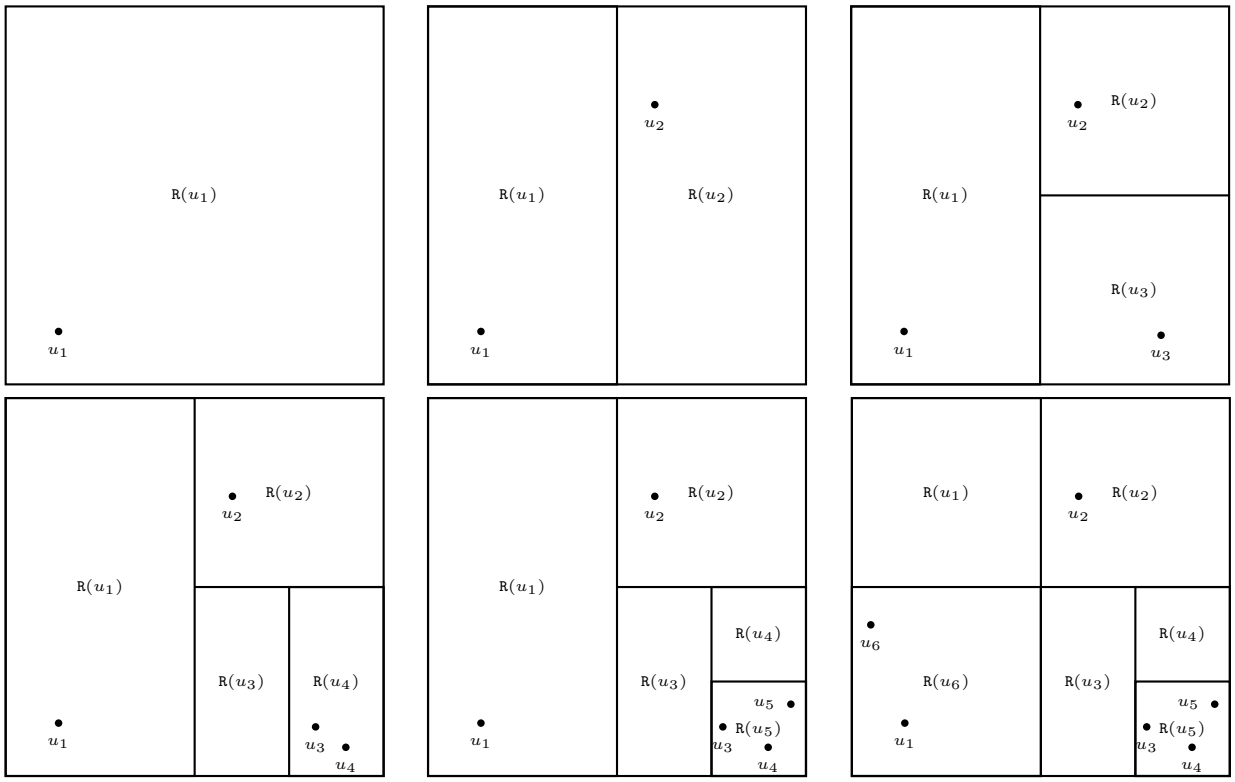


Figure 7.4: Example of the join procedure of `CAN`. The example shows that a node v may controls a region $\mathtt{R}(v)$ that v is not located in. Assume that the nodes and objects are points on a 2-dimensional space. Initially, the first node to join, denoted u_1, controls the whole space. Assume the next node to join is u_2 with identifier $\mathbf{id}(u_2)$. First, u_2 finds in which region is $\mathbf{id}(u_2)$ located. In this example, $\mathbf{id}(u_2)$ is controlled by the region of u_1, denoted $\mathtt{R}(u_2)$. Node u_1 divides its region into two equal regions. The first of which is controlled by u_1 whereas the second is controlled by u_2. The same procedure is followed whenever a node u_k joins the network. Note that u_4 is assigned a region $\mathtt{R}(u_4)$ where it is not located in. This is because $\mathbf{id}(u_4)$ is in $\mathtt{R}(u_3)$, before the joining. Any division of $\mathtt{R}(u_3)$ will lead to one of the nodes not controlling its region. This is one of main issues that hinders the use of `CAN` as a geographic aware P2P network.

1) the 2-dimensional space is rectangular with a diagonal length in $O(\sqrt{n})$, and 2) that the distribution of the nodes in the 2-dimensional space is uniform. Two methods are introduced in [AR04] to improve this routing cost.

`Geodemlia` [GSR+12] is a P2P network specifically designed for geographic awareness. It uses an approach inspired by [HP01] and YAO graph [Yao82]. The nodes and objects are represented as points in a 2-dimensional space. Each node controls the set of objects that are closest to it. A `Geodemlia` node v connects to all nodes controlling its neighbor regions. To improve routing cost, each node v with geographic location (x, y) creates α

compared to every other node [DCvKO08].

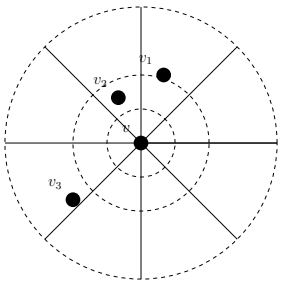


Figure 7.5: Sectors of a node v with $\alpha = 3$ and $\psi = 8$.

virtual circles. These circles are centered at (x, y). The i^{th} circle of a node v has radius 2^i, for $1 \leq i \leq \alpha$. A node v divides its circles into a ψ cones, each with an equal angle $360/\psi$. This divides the space around a node into $\alpha \times \psi$ *sectors*, as shown in Figure 7.5. A set of *long edges* of a node v is constructed by letting node v connects to the closest node in each of its sectors. Routing in `Geodemlia` is greedy. That is, each node v forwards a message to the neighbor with closest distance to the destination. The routing cost in `Geodemlia` is $O(\log n)$. This is due to the logarithmic evolution of the circles radii, and the structure of the long edges of `Geodemlia`. Kantere et al. introduced in [KSS09] a region-based P2P network, called `SpatialP2P`. The main difference between between `SpatialP2P` and `Geodemlia` is in the metric used to measure distance between the nodes.

Generally, the advantage of this category of P2P networks is the simplicity of projecting the actual geographic location into the virtual space of the network. The main disadvantage is the use of relatively more complex methods for lookup, node join and node leave, compared to chord-like networks. Furthermore, there is a lack of detailed studies of these networks contrary to the case of chord-like networks.

Tree-based geographic aware P2P networks

Tree-based geographic P2P networks organize their nodes in a tree-like structure. These networks are similar to hierarchical ONS architectures (see Section 7.3). There are few differences, however, some of which are found in the join and leave procedures.

`Globase.KOM` [KLS07] is a tree-based P2P network that relies on nodes and *supernodes* (that is, nodes with higher capacity and more robust connectivity). The space controlled by `Globase.KOM` is a map divided into regions. Each region is controlled by a supernode. A region may have nodes inside it connected somehow. The first node to join the network is a supernode that controls the whole space. As the number of nodes in a region

increases, its supernode v may divide it into two regions. The supernode v assigns one of the nodes in its region, say u, to become a supernode for the newly created region. Supernode v connects to supernode u via a *tree edge*. The structure is therefore a rooted tree of regions. The routing performance improves as the depth of the tree decreases. Note, however, that the load of the supernode increases as the the depth decrease. A similar architecture, called `RectNet`, was introduced by Heutelbeck and Hemmje in [HH06]. The main issue of tree-based geographic aware P2P networks is their inscalability. This is due to the tree hierarchy and the existence of supernodes.

7.5 GCAN: Geographic-Aware Content Addressable Network

`GCAN` *(Geographic-Aware Content Addressable Network)* is a yet-another geographic-aware P2P network and a distributed hash table. The most related P2P networks to `GCAN` are: `Chord` [SMK+01] (or, chord-like networks) and `CAN` [RFH+01]. `GCAN` is considered as an add-on that runs on top of any chord-like network in order to give it geographic awareness. `GCAN` modifies `CAN` such that it allows each node v controls points located in a region $\mathtt{R}(v)$ where v is also located. Recall that this is the main weakness that hinders the use of `CAN` or similar P2P structures as geographic-aware P2P networks (see Figure 7.4).

`GCAN` uses the methods of a chord-like network as *black-boxes*. That is, `GCAN` does not require any modification on the chord-like network on which it is built. Instead, only minor additions are required to add geographic awareness to chord-like networks. This is one of the advantages of `GCAN` since it is built on top of the well-studied and mature chord-like P2P networks.

This section gives the details of `GCAN`. The general definitions and assumptions are first given. A basic `GCAN` structure is then given in Section 7.5.2. This simple structure is used as an introduction to simplify the understanding of `GCAN`, and it is based over two assumptions, called **Assumption.1** and **Assumption.2**. The objective of the following sections is to relax these assumptions and introduce the join, leave, and routing procedures in `GCAN`.

7.5.1 General Definitions and Assumptions

This subsection gives the general definitions and assumptions used in this section. Two additional assumptions, called **Assumption.1** and **Assumption.2**, are given in Section 7.5.2 to simplify demonstration and will be relaxed later in the section.

The input network is denoted $G_{in} = (V_p, E_{in})$, where V_p is the set of nodes and E_{in} is the set $\binom{n}{2}$ edges between the nodes V_p (i.e., G_{in} is a complete network). Let $|V_p| = n$. Assume that the nodes are originally placed on a 2-dimensional Euclidean plane, denoted S. The plane S can be a map projection of the earth surface or part of it for example. Each node v is aware of its coordinates (x_v, y_v) in S. This knowledge may be obtained by any of the methods given in Section 7.2.

The nodes of V_p are assumed to be uniformly distributed in the plane S. Note that most (if not all) geographic aware P2P networks assume that the nodes are uniformly distributed in S.

The objective of `GCAN` is to form a P2P geographic aware network $G_p = (V_p, E_p)$. The size of the neighbors list $N(v)$ for each v in G_p is in $O(\log n)$. The routing cost is $O(\log n)$, and routing is geographic aware. The complexity of a node joining or leaving `GCAN` is in $O(\log n)$. Let $n_{\max}$ be the maximum number of nodes that can be in `GCAN`. Thus, $n_{\max} \geq n$.

7.5.2 A Basic `GCAN` Structure

This subsection introduces the basic structure of `GCAN` network is a torus of l *rows* and w *columns*, called *`GCAN` torus*. This structure is given to simplify the demonstration of `GCAN`. The rows of `GCAN` are numbered from 0 to $l-1$, and the columns are numbered from 0 to $w-1$. Assume without loss of generality that $l \leq w$. The end of each row or column is connected to its beginning. The intersection of a row and column is called a *`GCAN` cell*. The number of `GCAN` cells is denoted n_c, where $n_c = l \times w$.

Each node in S is projected into a `GCAN` cell such that the distance between a pair of nodes in S is approximately the same and within a scalar factor between them in the `GCAN` torus. This is called the *discretization procedure*. Each node u_{xy} with coordinates (x, y) is assigned to the `GCAN` cell $(\mathbf{v}(x), \mathbf{h}(y))$, where $\mathbf{v}(x)$ is the closest projected column to u_{xy}, and $\mathbf{h}(y)$ is the closest projected row to u_{xy}. Symmetries found in $\mathbf{v}(x)$ or $\mathbf{h}(y)$ are broken using any unambiguous rule. Note that $\mathbf{v}(x)$ is in $\{0, \ldots, l-1\}$, whereas $\mathbf{h}(y)$

Table 7.1: Terminologies used in Chapter 7

Term	**Definition**
V_p	The set of nodes
n	The number of nodes (i.e., $\|V_p\|$)
l	The number of rows in GCAN
w	The number of columns in GCAN
n_c	The number of cells in GCAN, where a cell is an intersection of a row and a column (i.e., $n_c = l \times w$)
$n_{\max}$	The maximum number of nodes that may be in GCAN
S	The 2-dimensional Euclidean plane where the nodes of V_p are originally located
$\mathbf{v}(x)$	The closest projected column to the point (x, y) in S
$\mathbf{h}(y)$	The closest projected row to the point (x, y) in S
$[\ (x, y)\ ,\ (x, y + c)\ ]$	A *range* (sequence) of cells in chord x starting from cell (x, y) to $(x, y + c)$
$(\ (x, y)\ ,\ (x, y + c))$	The range of cells $[(x, y), (x, y + c)]$ excluding (x, y) and $(x, y + c)$

is in $\{0, \ldots, w - 1\}$. The GCAN cell $(\mathbf{v}(x), \mathbf{h}(y))$ is called the GCAN *location* of u_{xy}.

Assume that the *discretization procedure* is run over a set of nodes V_p in the plane S such that it leads to the formation of a GCAN torus in which each cell is assigned exactly one node. Let G be the graph formed such that for each node $u_{xy} \in V_p$ with a GCAN location $(\mathbf{v}(x), \mathbf{h}(y))$, u_{xy} is connected to the nodes in the cells $(\mathbf{v}(x) + 1 \ \mathbf{mod}\ w, \mathbf{h}(y))$ and $(\mathbf{v}(x), \mathbf{h}(y) + 1 \ \mathbf{mod}\ l)$. Thus, the diameter $\mathcal{D}$ of G is $l + w$.[6]

Constructing long edges in the previous structure improves the diameter and hence the routing cost. Let each node u_{xy} creates *long edges* over each row and column it belongs to such that each node $u_{xy} \in V_p$ connects to the nodes in the cells $\{(\mathbf{v}(x) + 2^j \ \mathbf{mod}\ w, \mathbf{h}(y)) \mid 1 \leq j < \log w\}$ and in the cells $\{(\mathbf{v}(x), \mathbf{h}(y) + 2^i \ \mathbf{mod}\ l) \mid 1 \leq i < \log l\}$. An illustration is given in Figure 7.6. The diameter $\mathcal{D}$ of the resulting graph is $\log w + \log l = \log n_c$. As a result, the complexity of a greedy routing algorithm, in which the nodes of V_p know their GCAN locations, is $O(\log n_c)$.

Definitions and Assumptions (introduced in the Basic GCAN Structure)

Each row or column in GCAN can be seen as a ring with long edges. Thus, GCAN rows (respectively, columns) are called *horizontal chords* (respectively, *vertical chords*). These

[6]The diameter $\mathcal{D}$ of a graph or a network is the longest shortest path in term of number of hops.

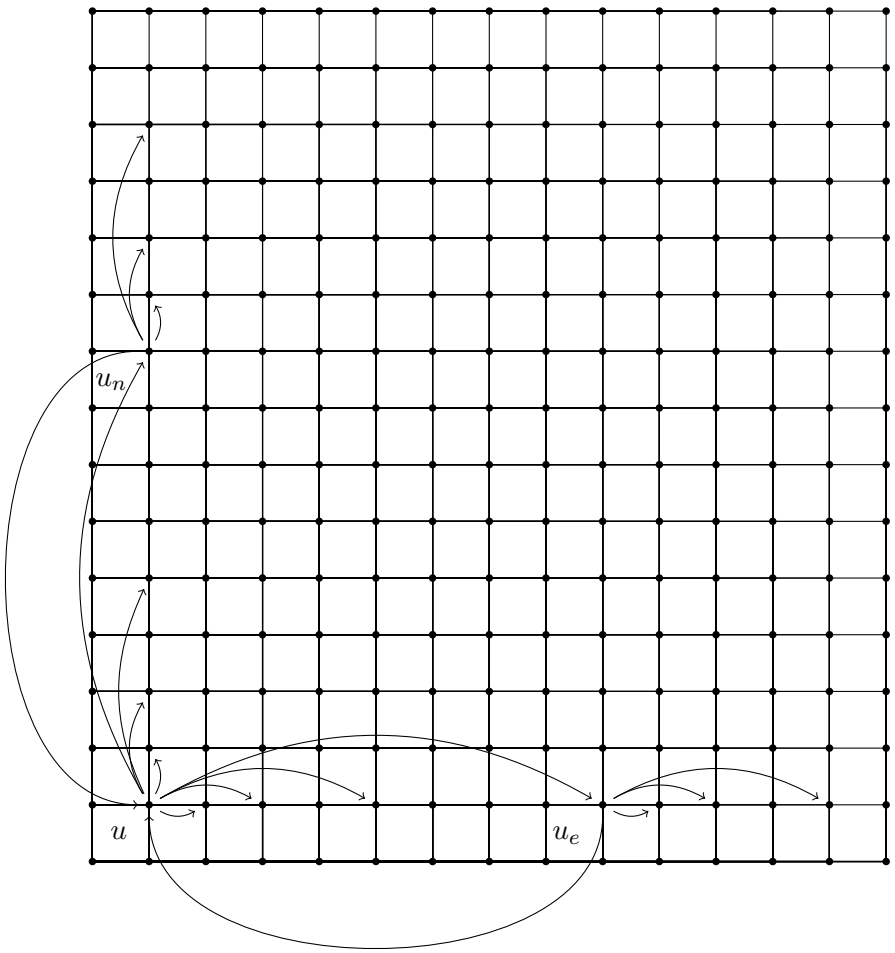


Figure 7.6: Basic structure of `GCAN` with long edges. The long edges are shown at node u. The vertical long edges of u_n and the horizontal long edges of u_e are also shown.

chords are implemented using any chord-like network implementation (e.g., `Chord` [SMK+01] or `SkipNet` [HJS+03]).

In any `GCAN` chord, a sequence of cells from cell (x, y) to $(x, y + c)$ is called a *range* and denoted [(x, y) , $(x, y + c)$]. Respectively, the cells starting from (x, y) to $(x + c, y)$ is a range denoted [(x, y) , $(x + c, y)$]. The endpoint of a range is inclusive by using the (or) symbols. That is, the range ((x, y) , $(x, y + c)$] includes the cells along the line (x, y) and $(x, y + c)$ including $(x, y + c)$ but not (x, y).

The basic structure of `GCAN` assumes that:

1. **Assumption.1**: the number of nodes in `GCAN` (i.e., $n = |V_p|$) is equal to the number of `GCAN` cells (i.e., n_c). It is not practically possible to achieve this assumption due

to the join and leave of the nodes in P2P networks. This assumption is relaxed in Section 7.5.3.

2. **Assumption.2**: each GCAN cell contains at most one node. This is not always possible to achieve because the distribution of the nodes V_p in the space S may require that more than one node resides in the same GCAN cell. This assumption is relaxed in Section 7.5.6.

In the following, u_{xy} is a node in V_p with coordinates (x, y) in the plane S and GCAN location $(\mathbf{v}(x), \mathbf{h}(y))$. The GCAN location is obtained using the *discretization procedure* previously defined.

7.5.3 A Simple Join Algorithm

This section discusses the join procedure of GCAN. The leave procedure is basically the opposite of the join, and hence its details are omitted. The join procedure relaxes Assumption **A.1**. Assumption **A.2** is not relaxed at this stage in order to simplify the analysis of GCAN. Assumption **A.2** is relaxed in Section 7.5.6 instead.

Nodes copies: GCAN generates virtual *copies* of each node in V_p to handle cases where the number of nodes n is less than the number of cells n_c. There are three types of nodes copies: *original*, *landmark* and *mirror* nodes copies. Each node v is responsible for all its copies. There is one *original* copy of each node in the network. There are at most l landmark copies of each node in the network. The nodes copies are assigned to distinct GCAN cells, such that any given time there is n_c nodes copies in the network or none (in case no node joined the network).

The basic idea: Let the node u_{xy} be the node with GCAN location $(\mathbf{v}(x), \mathbf{h}(y))$. Recall that $(\mathbf{v}(x), \mathbf{h}(y))$ is the intersection of the chords $\mathbf{v}(x)$ and $\mathbf{h}(y)$. For node u_{xy} to join GCAN, it must join a sequence of k horizontal chords $\{\mathbf{h}(y), \ldots, (\mathbf{h}(y) + k - 1) \textbf{ mod } l\}$, for $1 \leq k \leq l$. The identifier used to join these horizontal chords is $\mathbf{id}_h(u_{xy}) = \mathbf{v}(x)$. Moreover, node u_{xy} must join the vertical chord $\mathbf{v}(x)$ using $\mathbf{id}_v(u_{xy}) = \mathbf{h}(y)$ as identifier. The procedure of joining a horizontal chord is called a *horizontal join*, whereas the procedure of joining a vertical chord is called a *vertical chord*.

Joining any chord in GCAN is implemented using the join procedures of a chord-like network (e.g., Chord or SkipNet)[7].

Remark 7.5.1 (Joining a chord-like network)**.** *To join any chord c in GCAN or in any chord-like P2P network, a node u_{xy} must: 1) use a unique identifier that conforms to the specifications of c, 2) find its successor in c, and 3) connect to its successor and predecessor by changing corresponding pointers. This implies that the construction of long edges in chords are not part of the join procedure. This is true for all chord-like networks.*

Horizontal Joins: Let $\mathbf{h}^i(y) = \mathbf{h}(y) + i$ **mod** l. Thus, $\mathbf{h}^0(y) = \mathbf{h}(y)$. A joining node u_{xy} iteratively joins a sequence of k horizontal chords $\{\mathbf{h}^0, \ldots, \mathbf{h}^{k-1}\}$, for $1 \leq k \leq l$. That is, u_{xy} must join at least one horizontal chord. The identifier that u_{xy} used to join the horizontal chord $\mathbf{h}^i(y)$ is set to $\mathbf{v}(x)$. The successor of u_{xy} in a horizontal chord $\mathbf{h}^i(y)$ is denoted $\texttt{hSucc}(\mathbf{v}(x), \mathbf{h}^i(y))$.

A horizontal join is equivalent to occupying some GCAN cells with nodes copies according to the following rules:

Rule 1: The cells in the range $(\ (\mathbf{v}(x), \mathbf{h}^i(y))\ ,\ \texttt{hSucc}(\mathbf{v}(x), \mathbf{h}^i(y))\ )$ in any $\mathbf{h}^i(y)$ joined by u_{xy} must be occupied by *mirror* copies of u_{xy}.

Rule 2: The GCAN cell $(\mathbf{v}(x), \mathbf{h}^0(y))$ is occupied by the *original* copy of u_{xy}.

Rule 3: For every horizontal chord $\mathbf{h}^j(y)$, for $0 < j < k$, the cell $(\mathbf{v}(x), \mathbf{h}^j(y))$ is occupied by a *landmark* copy of u_{xy}.

Given the definitions of these rules, the following important rule is defined:

Rule 4: The horizontal successor of u_{xy} in any joined horizontal chord $\mathbf{h}^i(y)$ must be an *original* or a *landmark* copy of a node $v \in V_p$.

Vertical Joins: The question of how many horizontal joins shall a joining node u_{xy} executes is solved in the following. That is, the *stopping condition* of the horizontal joins is discussed.

[7] GCAN uses the join procedures of any chord-like network to implement its join procedures. However, if a certain chord-like network is used (e.g., Chord), then this network shall be used always. The use of different types of chord-like networks has not been investigated in details. This shall be considered as a future work.

Rule 5: After joining the horizontal chord $\mathbf{h}^0(y)$, node u_{xy} iteratively joins the chords above $\mathbf{h}^0(y)$ using $\mathbf{v}(x)$ as the identifier in each of these chords. This iterative join procedure terminates when u_{xy} reaches a chord $\mathbf{h}^i(y)$ such that the `GCAN` cell $(\mathbf{v}(x), \mathbf{h}^i(y))$ is occupied by an *original* copy of a node $v \in V_p$. Node v is called the *vertical successor* of u_{xy} and denoted $\texttt{vSucc}(\mathbf{v}(x), \mathbf{h}(y))$.

Therefore, the vertical successor of a node u_{xy} in `GCAN` must be an *original* copy of a node $v \in V_p$. Note that this is equivalent to joining the vertical chord $\mathbf{v}(x)$ using $\mathbf{h}(y)$ as the identifier. An example of the join procedure is given in Figure 7.7.

Theoretical Analysis

The performance of joining a P2P network is measured using message complexity. The complexity of joining a chord-like network with n nodes is $O(\log n)$. Therefore, given that `GCAN` join consists of k horizontal joins, each of which is in a horizontal chord with at most $\frac{n_c}{l}$ nodes, then `GCAN` join complexity is:

$$O(k \log \frac{n_c}{l}) \in O(l \log n_c) \in O(l \log n) \tag{7.1}$$

since $l \leq k$. The number of horizontal chords l is maximized if $l = w = \sqrt{n_c}$, since $l \leq w$. Therefore, `GCAN` join complexity may reach $O(\sqrt{n_c} \log n_c)$ in the worst case.

One way to improve this complexity is by limiting l to $\Theta(1)$. Thus, the join procedure complexity becomes $O(\log n_c) \in O(\log n)$. Section 7.5.4 introduces an improved version of the join procedure that has a $O(\log n)$ complexity on average without using any assumption on l.

7.5.4 An Improved Join Algorithm

This section introduces an improved join procedure that has a complexity of $O(\log n)$ on average. It is based on the following observations.

Proposition 7.5.2. *Let the set of horizontal chords joined by a node u_{xy} with `GCAN` location $(\mathbf{v}(x), \mathbf{h}(y))$ be $\{\mathbf{h}^0(y), \ldots, \mathbf{h}^{k-1}(y)\}$. Then, the set of ranges:*

$$\mathcal{L} = \{[(\mathbf{v}(x), \mathbf{h}^i(y)), \texttt{hSucc}(\mathbf{v}(x), \mathbf{h}^i(y))] \mid \forall\ 0 \leq i \leq k\}$$

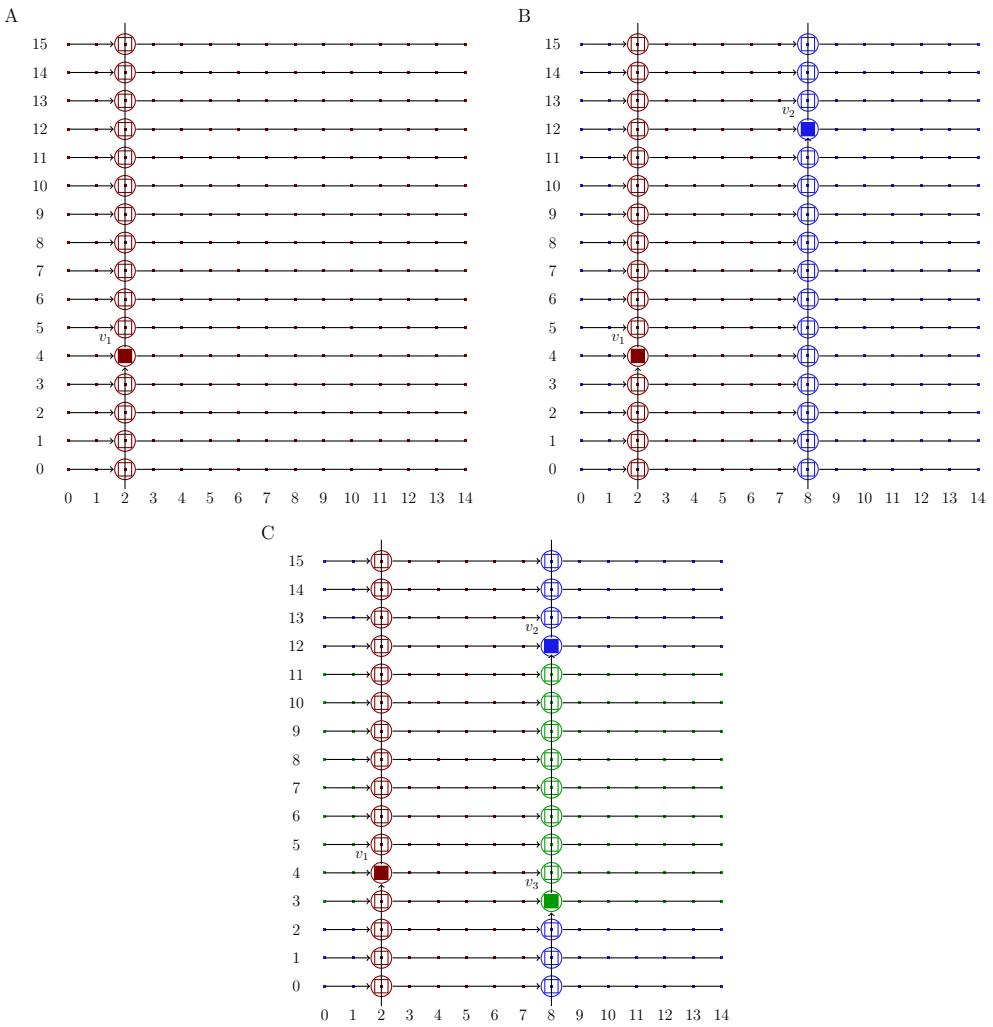


Figure 7.7: Example of the join procedure of Section 7.5.3. The *original* copy of a node is represented as filled square in a circle, the *landmark* copy as a square in a circle, and the *mirror* copy as a small square. Initially in (A), there is only node v_1, and it has joined the `GCAN` cell $(2, 4)$. The cell $(2, 4)$ is filled with an original copy. Node v_1 joins all the horizontal chords and terminates when it finds the original copy (in this case, the original copy belongs to v_1). The original copy of v_1 is connected to its horizontal and vertical successors via a directed arrows. The landmark copies of v_1 are connected to their horizontal successors via directed arrows in each joined chord. In (B), node v_2 joins in cell $(8, 12)$ and fill with an *original* copy of itself. Node v_2 joins the horizontal chord 12 until it finds an *original* or a *landmark* copy of a node. In this case, v_2 found a *landmark* copy of v_2. Node v_2 continues joining the horizontal chord until it finds an *original* copy of a node. In this case, v_2 found an *original* copy of itself. Note the changes in the mirror copies of v_1. In (C), node v_3 joins in cell $(8, 3)$ and fills it with an *original* copy of itself. It joins the horizontal chords until it finds an *original* copy of a node. In this case, v_3 found the *original* copy of v_2 in cell $(8, 12)$. Therefore, node v_3 joins the horizontal chords in the range $[3, 11]$.

forms a rectangular region *(that is, these ranges are all of equal sizes). The lower left side of this rectangular region is* $(\mathbf{v}(x), \mathbf{h}(y))$ *and occupied by an* original *copy of node* u_{xy}.

Proof. Note first the following properties of `GCAN`: 1) if a `GCAN` cell (v_x, h_y) is occupied by an *original* or a *landmark* copy of a node u, then each `GCAN` cell below and above (v_x, h_y) (i.e., $\{(v_x, h_y + i \bmod l) \mid \forall\ 0 \leq i < l\}$) is occupied by an *original* or a *landmark* copy. 2) Each successor of any node u in any horizontal chord must be the first *original* or *landmark* copy node encountered by u in that horizontal chord.

Given the previous properties, let the `GCAN` location of a node u_{xy} be $(\mathbf{v}(x), \mathbf{h}(y))$. Let $\mathbf{h}^a(y)$ and $\mathbf{h}^b(y)$ be any pair of horizontal chords that node u_{xy} joins, such that $\mathbf{h}^a(y) < \mathbf{h}^b(y)$. Let $s^a = \texttt{hSucc}(\mathbf{v}(x), \mathbf{h}^a(y))$, and $s^b = \texttt{hSucc}(\mathbf{v}(x), \mathbf{h}^b(y))$. Let the vertical chord where s^a (respectively, s^b) is located is $\mathbf{v}(x) + c_a$ (respectively, $\mathbf{v}(x) + c_b$). According to the proposition, s^a and s^b must be on the same vertical chord (that is, $c_a = c_b$). Assume without loss of generality that $c_b < c_a$, as shown in Figure 7.8 (A). This is a contradiction to the properties given at the beginning of this proof, because 1) s^a and s^b must be either *original* or *landmark* copy nodes, and 2) each `GCAN` cell below s^b must be *original* or *landmark*. Therefore, s^a cannot be the successor of u_{xy} in $\mathbf{h}^a(y)$. Figure 7.8-(B) and Figure 7.8-(C) give two alternative correct scenarios. □

Corollary 7.5.3 (`GCAN` as a DHT)**.** *The* ***GCAN*** *network is divided into* n rectangular regions*. The lower left side of each region is occupied by an* original *copy a node* $u \in V_p$*, called the region of* u*. Each node* u *controls its region. Thus,* ***GCAN*** *is used as a Distributed Hash Table (DHT).*

The properties of Proposition 7.5.2 and Corollary 7.5.3 are used to slightly modify the join algorithm of Section 7.5.3 such that its complexity becomes $O(\log n)$ on average without any restriction on l. Let us introduce first the following definitions and observations:

1. Let $\mathcal{H}(u_{xy})$ be the horizontal successors of u_{xy} in each of the k horizontal chord it joined (that is, $\mathcal{H}(u_{xy}) = \{\texttt{hSucc}(\mathbf{v}(x), \mathbf{h}^i(y)) \mid 0 \leq i \leq k\}$). $\mathcal{H}(u_{xy})$ is a set of *original* and *landmark* copies of a set of nodes denoted $\mathcal{S}_h(u_{xy}) = \{s^0, \ldots, s^{k'-1}\}$, where $k' \leq k$ (since, a node $v \in \mathcal{S}_h(u_{xy})$ may have multiple landmark copies in $\mathcal{H}(u_{xy})$).

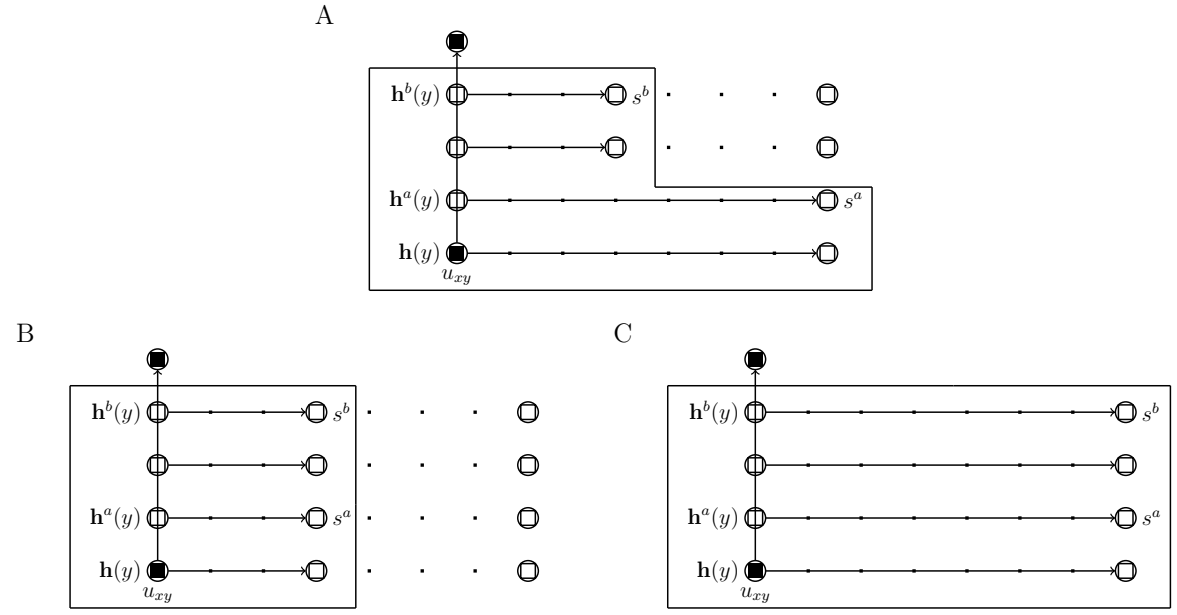


Figure 7.8: Network used in the proof of Proposition 7.5.2. Black filled squares in circles are original nodes. White squares in circles are landmark nodes, and small squares are copy nodes. The cells in the closed area are controlled by u_{xy} and filled by its copies. (A) illustrates an incorrect solution where the horizontal chords $\mathbf{h}^a(y)$ and $\mathbf{h}^b(y)$ have unequal ranges. (B) and (C) illustrate two alternative correct solutions.

2. According to Proposition 7.5.2, the *original* and *landmark* copies of the nodes in $\mathcal{S}_h(u_{xy})$ are all on the same vertical chord $v_x = \mathbf{v}(x) + c$. That is, s^i is connected to s^{i+1} in the vertical chord v_x, for each $s^i \in \mathcal{S}_h(u_{xy})$ and $0 \leq i < k'$.

Recall from Remark 7.5.1 that the most important tasks that a node u_{xy} should execute in order to join any chord are 1) selecting a unique identifier, and 2) searching for the successor of u_{xy} in the chord. Procedures executed other than these are of insignificant communication cost. Essentially, these procedures consist mainly from changes of pointers that require $O(1)$ messages. Therefore, a node u_{xy} joining `GCAN` shall find its successor in each `GCAN` chord it joins (i.e., its vertical successor (i.e., $\texttt{vSucc}(\mathbf{v}(x), \mathbf{h}(y))$) and its horizontal successors in $\mathcal{H}(u_{xy})$[8]. Essentially, node u_{xy} shall know the identifiers of the nodes in $\mathcal{S}_h(u_{xy})$. Thus, the steps executed by u_{xy} to join `GCAN` are:

1. u_{xy} finds its vertical successor $\texttt{vSucc}(\mathbf{v}(x), \mathbf{h}(y))$ which resides in the horizontal chord $\texttt{hchord}(\texttt{vSucc}(\mathbf{v}(x), \mathbf{h}(y))) = \mathbf{h}^k(y)$, whereas $\texttt{hchord}(i, j)$ is a function that

[8]The identifiers used in this join algorithm are the same as those used in the simple algorithm of Section 7.5.3.

returns the number of the horizontal chord where a GCAN cell (i, j) is located. The returned result of $\texttt{hchord}(i, j)$ is in $[0, l-1]$,

2. u_{xy} requests iteratively each horizontal successor s^i to provides u_{xy} with s^i vertical successor until the set $\mathcal{S}_h(u_{xy})$ is constructed. Note that s^0 is the horizontal successor of u_{xy} in the chord $\mathbf{h}^0(y)$.

The details of the improved join algorithm are given in Algorithm 7.1. In Algorithm 7.1, $\texttt{discrete-location}(u_{xy})$ is a function that returns the *GCAN location* of a node u_{xy}.

Algorithm 7.1 Algorithm $\texttt{improved-join}(u_{xy})$

$(\mathbf{v}(x), \mathbf{h}(y)) \leftarrow \texttt{discrete-location}(u_{xy})$
$k \leftarrow \texttt{hchord}(\texttt{vSucc}(\mathbf{v}(x), \mathbf{h}(y))$
$s^0 \leftarrow \texttt{hSucc}(\mathbf{v}(x), \mathbf{h}(y))$
$\mathcal{S}_h \leftarrow \{s^0\}$
$i \leftarrow 0$
while $k < \texttt{hchord}(\texttt{vSucc}(s^i))$ **do**
  $i \leftarrow i + 1$
  $s^i \leftarrow \texttt{vSucc}(s^{i-1})$
  $\mathcal{S}_h \leftarrow \{\mathcal{S}_h \cup s^i\}$

Theoretical Analysis

This section gives an analysis of the complexity of the improved join algorithm. First, the procedure of finding the vertical successor of a node u_{xy} in the vertical chord $\mathbf{v}(x)$ costs $O(\log l)$. Similarly, the procedure of finding the horizontal successor of u_{xy} at the horizontal chord $\mathbf{h}(y)$ is $O(\log w)$. This sums up to $O(\log l + \log w) \in O(\log n_c)$, since $n_c = l \times w$. It remains to count the number of horizontal successors of u_{xy} in $\mathcal{S}_h(u_{xy})$ (that is, k'). Given that the nodes are distributed randomly uniformly in GCAN (see Section 7.5.1), then $k' \in \Theta(1)$ on average. This is because the rectangular regions of GCAN have approximately the same lengths due to the uniform distribution of nodes. Therefore, the new join procedure complexity is $O(\log n_c) + \Theta(1) \in O(\log n_c)$ on average. This is equivalent to the join complexity of most chord-like networks.

7.5.5 Routing and Long Edges

Routing in GCAN is greedy. That is, each node u forwards a message m to the neighbor $v \in N(u)$ that is *geographically closest* to m destination, until the destination is reached. This requires that every node is also identified by its geographic coordinates or GCAN location. The routing cost of this greedy algorithm is $O(l + w)$, considering the join algorithms of Section 7.5.3 and Section 7.5.4. This is because the diameter of the formed structure is $l + w$. Note that $O(l + w) \in O(n_c)$, and may be improved to $O(\sqrt{n_c})$ if $l = w = \sqrt{n_c}$. The routing cost of GCAN can be further improved by creating *long edges* for each node. There are two main types of long edges in GCAN:

1. *Vertical long edges*: the original copy of u_{xy} connects to the nodes responsible for the cells $\{(\mathbf{v}(x), \mathbf{h}(y) + (2^i \ \mathbf{mod}\ l)) \mid 1 \leq i \leq \log l\}$.

2. *Horizontal long edges*: let the horizontal chords where s^i, for any $s^i \in \mathcal{S}_h(u_{xy})$, is successor of u_{xy} be denoted $\{h_1^i, \ldots, h_z^i\}$. Then, u_{xy} creates horizontal logarithmic edges in only one of these chords (e.g., h_1^i). These horizontal logarithmic edges connect to the nodes responsible for the cells $\{(\mathbf{v}(x) + (2^j \ \mathbf{mod}\ w), \mathbf{h}_1^i) \mid 1 \leq j \leq \log w\}$, for each $\mathbf{h}_1^i$.

Other type of long edges may be constructed with an insignificant additional communication cost (*backward long edges* and *diagonal long edges* for example).

Theorem 7.5.4. *The complexity of the greedy routing algorithm using the long edges of GCAN is* $O(\log n)$

Proof. We consider a pessimistic approach to prove the theorem. We use the following greedy routing algorithm. First, a node u_s finds the vertical chord $\mathbf{v}(d)$ where u_d is located. Then, the message is forwarded along $\mathbf{v}(d)$. Let the original copy of u_s (respectively, u_d) be located in the GCAN location $(\mathbf{v}(s), \mathbf{h}(s))$ (respectively, $(\mathbf{v}(d), \mathbf{h}(d))$). Consider the basic case where $\mathbf{h}(s) = \mathbf{h}(d)$ or $\mathbf{v}(s) = \mathbf{v}(d)$. In this case, the routing is $O(\log n)$ since both nodes are on the same chord. Assume that $\mathbf{h}(s) \neq \mathbf{h}(d)$ and $\mathbf{v}(s) \neq \mathbf{v}(d)$. Then, there is a *landmark* copy of u_d on $\mathbf{h}(s)$. Otherwise, there is an *original* or *landmark* copy of a node v on $\mathbf{h}(s)$ and $\mathbf{v}(d)$. Therefore, this copy of v can be reached by u_s using $O(\log n)$ hops since they are on the same chord $\mathbf{h}(s)$, and it can be reached by u_d using $O(\log n)$ hops since they are on the same chord $\mathbf{v}(d)$. Thus, the

total routing cost is $O(\log n)$. Note that the same routing cost is achieved if the u_s finds the horizontal chord where u_d is located and forwards the message along it.

Moreover, node u_d may be the successor (that is, in $\mathcal{S}_h(u_s)$). In such case, routing between u_d and u_s are done along the horizontal chord $\mathbf{h}(d)$ and costs $O(\log N)$. Otherwise, there must be a node v on $\mathbf{v}(s)$ and has an *original* or *landmark* copy on $\mathbf{h}(d)$. Therefore, this copy of v can be reached by u_s using $O(\log n)$ hops since they are on the same chord $\mathbf{v}(s)$, and it can be reached by u_d using $O(\log n)$ hops since they are on the same chord $\mathbf{h}(d)$. The total cost therefore is $O(\log n)$. This completes the proof. □

7.5.6 Last remarks, (Relaxing Assumption.2):

This section relaxes **Assumption.2**. That is, this section considers the case where multiple nodes may be assigned to the same `GCAN` location by the *discretization procedure* (see Section 7.5.2). The following solution is proposed. This solution is a slight modification of the improved join algorithm of Section 7.5.4. The idea is to divide the `GCAN` cell among the nodes that occupy it. *In the following, it is assumed that* $n_{\max}/n \in \Theta(1)$*, where* $n_{\max}$ *is the maximum number of nodes allowed in* ***GCAN****.*

Each node u_{xy} uses the string $\mathbf{id}_h(u_{xy}) = [x, y, \mathbf{id}(u_{xy})]$ as its identifier in horizontal chords (that is, the concatenation of x, y and $\mathbf{id}(u_{xy})$]. Similarly, a node u_{xy} uses $\mathbf{id}_v(u_{xy}) = [y, x, \mathbf{id}(u_{xy})]$ to join vertical successors. Each node u_{xy} joins k horizontal chords with the identifier $\mathbf{id}_h(u_{xy})$ as indicated by Algorithm 7.1 (i.e., Algorithm `improved-join`). The idea is that a multiple nodes copies may be found in the same `GCAN` cell. First, node u_{xy} always joins $\mathbf{h}(y) = \mathbf{h}^0(y)$. Then, node u_{xy} joins a horizontal chord $\mathbf{h}^i(y)$ for $i > 0$ if the `GCAN` cell $(\mathbf{v}(x), \mathbf{h}^i(y)$ does not contain any *original* node copy.

The problem, however, is in how to efficiently find which k horizontal chords to join using Algorithm `improved-join`. To solve this issue, each node u_{xy} in `GCAN` cell $(\mathbf{v}(x), \mathbf{h}(y))$ keeps track of two *vertical successor*: 1) $\texttt{vSucc}(u_{xy})$, which is the node v that has identifier $\mathbf{id}_v(v)$ that is the minimum greater identifier in comparison to $\mathbf{id}_v(u_{xy})$ in the vertical chord $\mathbf{v}(x)$, and 2) $\texttt{gvSucc}(u_{xy})$, which the node v that has identifier $\mathbf{id}_v(v)$ that is minimum greater identifier in comparison to $\mathbf{id}_v(u_{xy})$ in the vertical chord $\mathbf{v}(x)$ and is in a `GCAN` location $(\mathbf{v}(x), \mathbf{h}^i(y))$ for $i > 0$. This implies that if the vertical chord has only one node which is u_{xy} then $\texttt{vSucc}(u_{xy}) = \texttt{gvSucc}(u_{xy}) = u_{xy}$. Thus, Algorithm 7.1 (that is, $\texttt{join}(u_{xy})$) is modified such that node u_{xy} finds $\texttt{gvSucc}(u_{xy})$ and

`gvSucc`(s^i) for each horizontal successor $s^i \in \mathcal{S}_h(u_{xy})$, instead of finding the vertical successor `vSucc`(s^i). Note that the definition of `gvSucc`(u_{xy}) simplifies the details of keeping track of it by u_{xy}. These modifications do not introduce any changes to the routing cost, join procedure complexity and leave procedure complexity. Thus, the join procedure complexity is $O(\log n)$, whereas the routing cost remains $O(\log n)$.

7.6 Conclusions

This chapter formulated the ONS architecture problem and gave a set of general requirements for an ONS architecture. These requirements are extracted from the DNS requirements and from other systems used for lookup. A survey of ONS architecture were given. The survey categorized ONS architectures into hierarchical and P2P architectures. The main result of the survey is that P2P ONS architectures achieves better scalability and fault-tolerance with respect to Hierarchical ONS architectures. Moreover, P2P ONS architectures are free from single point of governance and do not need additional nodes to connect the local ONS servers, contrary to hierarchical ONS architectures. However, P2P ONS architectures are weak with respect to caching, replication and especially geographic awareness.

The survey showed that techniques from similar P2P systems may be employed to improve the performance of P2P ONS architectures. For instance, `CoDNS` [PPPW04] uses a technique called `Beehive` [RS04] to implement caching, which is a technique originally designed for applications other than DNS or ONS. Similarly, the P2P ONS of [Fab09] uses the geographic awareness techniques of `Pastry` as it is built over `Pastry`. This suggests that building the ONS over a geographic aware P2P network leads to a geographic aware ONS architecture. As a result, This chapter introduced `GCAN`, which is a geographic aware content-addressable P2P network. `GCAN` adds geographic awareness to chord-like networks. `GCAN` may be considered as a variation of `CAN` [RFH$^+$01] that solves some of its major issues (e.g., each node v in `GCAN` controls a region in which v is located; a property that is not found in `CAN`).

Despite the modifications of Section 7.5.6, the routing cost of `GCAN` is $O(\log n)$. Moreover, the complexity of join algorithm, and hence the leave algorithm, is $O(\log n)$. `GCAN` is scalable as each of its nodes keep track of $O(\log n)$ nodes in its neighbors list. This is similar to the performance of most chord-like networks. However and contrary to chord-like networks, `GCAN` is geographic aware and requires only few additions on top of chord-like

networks. As a result, the main objectives of `GCAN` were achieved in this chapter.

There are several directions to extend `GCAN`:

1. Simulation experiments are required. Such experiments shall be run on realistic environments (e.g., PlanetLab servers [CCR+03]).

2. Further theoretical analysis are necessary. For instance, the relationship between the shortest path in the `GCAN` structure and the Euclidean distance shall be analyzed. This would serve in better understanding of geographic-aware P2P networks.

3. A special attention shall be given to techniques to find the geographic location of a nodes in the Internet. Among these techniques are the *geolocation* and *virtual coordinates* techniques.

Part V

Conclusions

Chapter 8

Conclusions and Future Work

The chapter is divided into two sections. The first section gives the main conclusions of the thesis. The second sections proposes future related research directions based on the results of this thesis.

8.1 Conclusions

Each chapter in this thesis provided a list of conclusions. This section gives a summary of these conclusions. The thesis focused on the problem of scalability in the Internet of Things (`IoT`). More specifically, the thesis focused on the scalability issues found in the three central layers of the minimalist `IoT` architecture, which are objects coverage, data processing and transportation, and object name resolution (see Chapter 1 for more details). The thesis was divided into three parts, each of which studies one the central layers.

Part II studied the RFID Coverage (RFC) problem in both of its versions, which are the reader-tag RFC and the reader-reader RFC. Chapter 2 studied the reader-tag RFC problem. It was shown that major RFC algorithms may be outperformed with simpler randomized heuristics such as those used in `RANDOM`, `RANDOM`$^+$, and `MAX-MIN`. Chapter 2 also introduced algorithm `GDE`, which is a decentralized implementation of the greedy set cover algorithm that solely uses write/read rounds. The main weakness of `GDE` is that it requires $O(|\mathcal{R}|)$ write/read rounds in the worst-case, where $\mathcal{R}$ is the set of readers. To solve this issue, the number of write/read rounds is limited to a con-

stant in another algorithm introduced in the same chapter, called `LIMITED-GDE`. The empirical performance of reader-tag RFC algorithms was studied in Chapter 3. The performance metrics studied were 1) the number of non-redundant readers, 2) the number of write/read rounds and 3) the number of write/read operations. `GDE` was shown to outperform all other algorithms in term of the number of non-redundant readers. However, `MAX-MIN` and `LIMITED-GDE` were shown to provide the best balance between the different performance metrics. `RANDOM` outperformed similar single-round RFC algorithms (e.g., `RRE` and `DRRE`) in most studied scenarios despite its simpler rules. Note that `RANDOM` was the result of a generalization of reader-tag RFC algorithms, called `SEQ`. This generalization was an important step that led to the design of `RANDOM`, `RANDOM`$^+$, and `MAX-MIN`. Moreover, this generalization was shown to be important to understand the behavior of the RFC algorithms studied in Chapter 3.

Chapter 4 extended the work of Chapter 2 and Chapter 3 by studying the reader-reader RFC problem. At first, it was shown that it is impossible to solve the problem using a distributed local algorithm under certain sets of assumptions (see Section 4.3 for more details). An additional assumption, called **A.I**, was used therefore to overcome this issue. Using assumption **A.I**, a local distributed algorithm called `OB-COVERAGE` was introduced. An extension of `OB-COVERAGE`, called `IOB-COVERAGE`, was also introduced. `IOB-COVERAGE` runs in $O(|\mathcal{R}|)$ communication rounds instead. As an application of reader-reader RFC algorithms and `IOB-COVERAGE` more specifically, Chapter 4 studied the Readers Collision Avoidance (RCA) problem. The chapter showed that `IOB-COVERAGE` may be used to solve the RCA problem. The chapter also introduced a more efficient algorithm to solve the RCA problem, called `IOB-(RCA+COV)`, which avoids some of the main weaknesses in major RCA algorithms.

Part III studied the Bluetooth Scatternet Formation (BSF) problem. Chapter 5 gave a survey of existing BSF algorithms with a focus on certain performance criteria (see Section 5.1 for more details). Chapter 5 introduced a time-efficient algorithm, called `Eliminate`, that guarantees outdegree limitation. The approach followed to achieve `Eliminate` time-efficiency is based on studying the details of the link establishment procedures in Bluetooth and exploit this knowledge to employ techniques that help in achieving shorter execution time of distributed algorithms executed over Bluetooth networks. This approach is shown to lead to positive results. For example, simulation results showed that `Eliminate` is among the fastest existing BSF algorithms that always

guarantees outdegree-limitation in unit disk graphs [1]. The detailed study of the link establishment procedures showed that there must be changes on their standard specifications in future versions of Bluetooth.

Chapter 6 introduced algorithm `BSF-UED`. The algorithm achieves time-efficiency by using the same techniques used in `Eliminate`. `BSF-UED` aimed at achieving a balance between the conflicting performance metrics of Bluetooth scatternets while focusing on reducing the BSF algorithm execution time; a property that is shown to be very challenging to achieve. `BSF-UED` performance is compared against major BSF algorithms using simulation experiments. The experiments showed that `BSF-UED` is time-efficient and outperforms the studied BSF algorithms with respect to many performance metrics.

Part IV studied the object name resolution problem. Chapter 7 investigated existing Object Name Service (ONS) architectures and extracted major requirements that shall be met by any ONS architecture. It was concluded that P2P ONS architectures are more efficient in comparison with hierarchal ONS architectures with respect to scalability and fault-tolerance. However, P2P ONS architectures suffer from other weaknesses in caching, replication and especially geographic awareness. Chapter 7 suggested that caching, replication and geographic-awareness may be added to a P2P ONS architecture by using existing techniques used in other P2P architectures that are not necessarily used for ONS. Thus, a geographic-aware P2P architecture, called `GCAN`, was introduced. This architecture may be used as an ONS architecture and for other applications that require geographic awareness. Its main advantage is in its simplicity, since it is easily built over any chord-like P2P network. `GCAN` solves an important problem found in `Chord` [SMK+01] and similar chord-like networks, which is the lack of geographic awareness.

8.2 Future Work

The following suggested extensions of the contributions of this thesis are given in the following.

1. *Reader-tag RFC algorithms:* Despite the several reader-tag RFC algorithms in the literature, only one write/read round implementation currently exists. This implementation has been introduced in [CRK+09]. Thus, new implementations of

[1] To the knowledge of the thesis's author, only `BlueStars` [PBC03] is faster than `Eliminate`. However, `BlueStars` does not form outdegree-limited scatternets, while `Eliminate` does.

write/read rounds shall be introduced. The implementations of write/read rounds will redefine the mathematical models used to study the reader-tag RFC algorithms, and those used to study RFID reader networks that contain writable tags. This would lead perhaps to additional constraints that shall be taken into consideration when studying the reader-tag RFC problem. Even if no new write/read implementations appears, the reader-tag RFC problem shall be studied under more constraints, such as faulty links and load balancing between the readers. This may lead to redefinitions of the RFC problem.

2. *Reader-reader RFC algorithms:* this problem shall be studied under different mathematical models other than those used in Chapter 4). This would lead to better understanding of the problem and increases the applicability of its solutions.

   Reader-reader RFC algorithms lead to an important problem that shall be further investigated. Assume that the RFC algorithm formed a set of non-redundant readers, then how to guarantees that the induced graph of these non-redundant reader forms a multi-hop connected network. This problem is similar to a problem found in wireless sensor networks, called the *connected coverage problem* (see [Wan11] for more details). Most existing solutions of the connected coverage problem assume the availability of a centralized programmable node that is used to initiate the creation of such a multihop network. New decentralized algorithms shall be developed for this problem. Moreover, it is suggested that the performance of the existing algorithms of this problem be studied comparatively and in more depth.

3. *Bluetooth Scatternet Formation algorithms:* there shall be more work done on the BSF algorithms introduced in this thesis, namely `Eliminate` and `BSF-UED`. For instance, future work shall be directed towards reducing the number of piconets in `Eliminate` scatternets using local heuristic rules. This approach was followed by `BlueMISII` [ZDS08], but with not much success.

   There shall be more work on designing simple BSF algorithms that forms outdegree-limited scatternets using as few communication rounds as possible. These algorithms shall not use location information. Moreover, these algorithms shall not force its nodes to contact its neighbors in specific order, since a node contacting its neighbors in order is shown to have negative impact on the execution time of BSF algorithms, as it is case of `BlueMISI` and `Eliminate`.

The thesis suggests that the specifications of Bluetooth be revised, especially in terms of the link establishment procedures. It would be necessary as well to relax the restriction of outdegree limitation by modifying the *parking* procedures. Recall that a master keeps only seven of its slaves active, while every other slave is *parked*. Such a revision may lead to more efficient BSF algorithms, as it is found that most of the problems of BSF algorithms are due to outdegree limitation.

Lastly, the outdegree-limited BSF problem shall be studied without assuming that the network is modeled as a unit disk graph (e.g., the arbitrary graph model of wireless networks). Interestingly, this version of the problem has never been studied despite the rich literature of BSF algorithms. Studying this problem would lead to more practical BSF algorithms, since the unit disk graph model is not a very practical model for Bluetooth networks and wireless ad-hoc networks in general.

4. *ONS architectures:* it is suggested that the performance of `GCAN` be studied using simulation experiments and more theoretical analysis. These experiments shall be as realistic as possible. Furthermore, an API (Application Programming Interface) of `GCAN` shall be easily available.

   More research on geolocation shall be done, where geolocation is the problem of estimating the geographic location of an Internet node by knowing its IP address only. This problem is useful for geographic-aware ONS architectures. It is also useful for Internet marketing, legal control of the Internet, and multimedia streaming applications. Future geolocation techniques should aim at achieving street-level localization using more intelligent algorithms See [WBF+11] for an example heuristic that aims to achieve such a level of accuracy. A similar problem that shall be further investigated is the formation of virtual network coordinates, which is the problem of finding virtual coordinates of the nodes of a given network such that distance between a pair of nodes in the virtual space reflects the geographical distance between them.

   A question that requires further research is the following; some geolocation and virtual coordinates techniques are based on fixed Internet servers with known geographic locations. Given that, is is possible to make a collaborative geolocation or virtual coordinates algorithm? More specifically, is it possible to use a P2P network (e.g., `GCAN`) to solve the geolocation problem and the formation of virtual coordinates using a collaboration of nodes instead of fixed servers? To the author

knowledge, there are no algorithms or architectures that achieve this objective. Algorithms or architectures for this problem must focus on issues such as privacy, security and handling faulty and corrupted nodes.

On a higher level of the ONS, tracking the movement of an object (or, tag) from one geographic location to another is a problem that is attracting a lot of research currently. In this problem, it is assumed that ownership of a tag moves as well from one server to another. Solving this problem efficiently would lead to many interesting applications. (see [KCKT08] and [STF05] for more details about the problem).

Part VI

Bibliography

Bibliography

[AA13] W. A. A. Alsalihy and N. A. Askar, *Enhancement of Redundant Reader Elimination by Using Hybrid Algorithm in RFID Systems*, Wireless Personal Communications (2013), 1–18.

[AAE12] N. A. Ali and M. Abu-Elkheir, *Data management for the Internet of Things: Green directions*, IEEE Globecom 2012 Workshops (GC Wkshps), IEEE, 2012, pp. 386–390.

[AAH11a] K. Ali, W. Alsalih, and H. Hassanein, *Set-Cover Approximation Algorithms for Load-Aware Readers Placement in RFID Networks*, IEEE International Conference on Communications (ICC), June 2011, pp. 1–6.

[AAH11b] K. Ali, W. Alsalih, and H. Hassanein, *Using neighbor and tag estimations for redundant reader eliminations in RFID networks*, IEEE Wireless Communications and Networking Conference (WCNC), mar 2011, pp. 832–837.

[ABD+07] F. Armenio, H. Barthel, P. Dietrich, J. Duker, C. Floerkemeier, J. Garrett, M. Harrison, B. Hogan, J. Mitsugi, J. Preishuber-Pfluegl, et al., *The EPCglobal architecture framework*, Final Version **1** (2007).

[abi13a] *Bluetooth's Role in the Internet of Everything*, Tech. report, ABI Research, 2013.

[abi13b] *With an Installed Base of 10 Billion Devices Expected in 2018, Bluetooth will be an Essential Tool for Building the Internet of Everything*, ABI Research News (2013).

[AH13] C. Aggarwal and J. Han, *A Survey of RFID Data Processing*, ch. 11, Springer, 2013.

[AIM10] L. Atzori, A. Iera, and G. Morabito, *The internet of things: A survey*, Computer Networks **54** (2010), no. 15, 2787–2805.

[Alb01] P. Albitz, *DNS and Bind*, O'reilly, 2001.

[AR04] F. Araújo and L. Rodrigues, *Geopeer: A location-aware peer-to-peer system*, Third IEEE International Symposium on Network Computing and Applications (NCA 2004), IEEE, 2004, pp. 39–46.

[BAHT11] J. Baliga, R. W. Ayre, K. Hinton, and R. S. Tucker, *Green cloud computing: Balancing energy in processing, storage, and transport*, Proceedings of the IEEE **99** (2011), no. 1, 149–167.

[Bar13] B. Bartlett, *Is PRISM's 'Big Data' about Big Money?*, The Fiscal Times (2013).

[BBDC11] R. Bolla, R. Bruschi, F. Davoli, and F. Cucchietti, *Energy efficiency in the future internet: a survey of existing approaches and trends in energy-aware fixed network infrastructures*, IEEE Communications Surveys & Tutorials **13** (2011), no. 2, 223–244.

[BBMP04] S. Basagni, R. Bruno, G. Mambrini, and C. Petrioli, *Comparative performance evaluation of scatternet formation protocols for networks of Bluetooth devices*, Wireless Networks **10** (2004), no. 2, 197–213.

[BFNO03] L. Barrière, P. Fraigniaud, L. Narayanan, and J. Opatrny, *Dynamic construction of bluetooth scatternets of fixed degree and low diameter*, The fourteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2003, pp. 781—790.

[BKFS11] S. Balakrichenan, A. Kin-Foo, and M. Souissi, *Qualitative Evaluation of a Proposed Federated Object Naming Service Architecture*, International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing (iThings/CPSCom), IEEE, 2011, pp. 726–732.

[blu10] *Bluetooth Specifications ver4.0*, Tech. report, Bluetooth Special Interest Group (SIG), June 2010.

[BMSU01] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia, *Routing with guaranteed delivery in ad hoc wireless networks*, Wireless networks **7** (2001), no. 6, 609–616.

[BS11] D. Bandyopadhyay and J. Sen, *Internet of things: Applications and challenges in technology and standardization*, Wireless Personal Communications **58** (2011), no. 1, 49–69.

[BSRS10] M. Bolic, D. Simplot-Ryl, and I. Stojmenovic, *RFID systems: research trends and challenges*, Wiley, 2010.

[CCR+03] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, *Planetlab: an overlay testbed for broad-coverage services*, ACM SIGCOMM Computer Communication Review **33** (2003), no. 3, 3–12.

[CJBM02] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, *Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks*, Wireless networks **8** (2002), no. 5, 481–494.

[CK03] E. Cohen and H. Kaplan, *Proactive caching of DNS records: Addressing a performance bottleneck*, Computer Networks **41** (2003), no. 6, 707–726.

[cla05] *Class 1 Gen2 UHF Air Interface Protocol Standard Version*, Tech. report, 2005.

[CMA04] F. Cuomo, T. Melodia, and I. F. Akyildiz, *Distributed self-healing and variable optimization algoritms for QoS provisioning in scatternets*, IEEE Journal on Selected Areas in Communication **22** (2004), no. 7, 1220–1236.

[CMM02] R. Cox, A. Muthitacharoen, and R. T. Morris, *Serving DNS using a peer-to-peer lookup service*, Peer-to-Peer Systems, Springer, 2002, pp. 155–165.

[CRK+09] B. Carbunar, M. K. Ramanathan, M. Koyutürk, S. Jagannathan, and A. Grama, *Efficient tag detection in RFID systems*, Journal of Parallel and Distributed Computing **69** (2009), no. 2, 180–196.

[CTF00] A. Caprara, P. Toth, and M. Fischetti, *Algorithms for the set covering problem*, Annals of operations research **98** (2000), no. 1-4, 353–371.

[CZQ+05] W. Cai, S. Zhou, W. Qian, L. Xu, K.-L. Tan, and A. Zhou, *C 2: a new overlay network based on CAN and Chord*, International Journal of High Performance Computing and Networking **3** (2005), no. 4, 248–261.

[Dap04] A. Daptardar, *Meshes and cubes: Distributed scatternet formations for Bluetooth personal area networks*, Master's thesis, Washington State University, 2004.

[dBE46] N. G. d. Bruijn and P. Erdos, *A combinatorial problem*, Koninklijke Nederlandse Akademie v. Wetenschappen **49** (1946), no. 49, 758–764.

[DCKM04] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, *Vivaldi: A decentralized network coordinate system*, ACM SIGCOMM Computer Communication Review **34** (2004), no. 4, 15–26.

[DCvKO08] M. De Berg, O. Cheong, M. v. Kreveld, and M. Overmars, *Computational geometry*, Springer, 2008.

[Den05] P. J. Denning, *The locality principle*, Communications of the ACM **48** (2005), no. 7, 19–24.

[DGK10] B. Donnet, B. Gueye, and M. A. Kaafar, *A survey on network coordinates systems, design, and security*, IEEE Communications Surveys & Tutorials **12** (2010), no. 4, 488–503.

[DHM+07] D. Dubhashi, O. Häggström, G. Mambrini, A. Panconesi, and C. Petrioli, *Blue pleiades, a new solution for device discovery and scatternet formation in multi-hop bluetooth networks*, Wireless Networks **13** (2007), 107–125.

[DKK+01] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, *Wide-area cooperative storage with CFS*, ACM SIGOPS Operating Systems Review **35** (2001), no. 5, 202–215.

[DMBG10] V. G. Dhas, R. Muthukaruppan, K. Balakrishnan, and R. Ganesan, *Optimal Solution for RFID Load Balancing*, Recent Trends in Networks and Communications, Springer, 2010, pp. 41–49.

[dns07] *Root server attack on 6 February 2007*, Tech. report, Internet Corporation for Assigned Names and Numbers (ICANN), March 2007.

[DOL07] R. Derakhshan, M. Orlowska, and X. Li, *RFID data management: challenges and opportunities*, International Conference on RFID, IEEE, 2007, pp. 175–182.

[DSP08] S. Deb, A. Srinivasan, and S. K. Pavan, *An improved DNS server selection algorithm for faster lookups*, Third International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE 2008), IEEE, 2008, pp. 288–295.

[DSS+08] Q. Dong, A. Shukla, V. Shrivastava, D. Agrawal, S. Banerjee, and K. Kar, *Load balancing in large-scale RFID systems*, Computer Networks **52** (2008), 1782–1796.

[DW03] Y. Dong and J. Wu, *Three Bluetree formations for constructing efficient scatternets in Bluetooth*, The Seventh Joint Conference on Information Sciences, 2003, pp. 385–388.

[EFG08] S. Evdokimov, B. Fabian, and O. Günther, *Multipolarity for the object naming service*, The Internet of Things, Springer, 2008, pp. 1–18.

[EGHK99] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, *Next century challenges: Scalable coordination in sensor networks*, Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking, ACM, 1999, pp. 263–270.

[epc08] *Object Name Service (ONS) 1.0.1*, Tech. report, GS1 EPCglobal, 2008.

[epc12] *Object Name Service (ONS) 2.0*, Tech. Report 1, EPCglobal, dec 2012.

[Eva11] D. Evans, *The Internet of Things: How the Next Evolution of the Internet Is Changing Everything*, Tech. report, Cisco IBSG, April 2011.

[Fab09] B. Fabian, *Implementing secure p2p-ons*, IEEE International Conference on Communications (ICC'09), IEEE, 2009, pp. 1–5.

[Fei98] U. Feige, *A threshold of ln n for approximating set cover*, Journal of the ACM (JACM) **45** (1998), no. 4, 634–652.

[FG07] B. Fabian and O. Gunther, *Distributed ONS and its Impact on Privacy*, IEEE International Conference on Communications (ICC'07), IEEE, 2007, pp. 1223–1228.

[Fur08] A. Furness, *CASAGRAS and the Internet of Things*, 2008.

[Fur12] C. A. Furuti, *Azimuthal Projections*, December 2012.

[GBA+13] E. I. Gaura, J. Brusey, M. Allen, R. Wilkins, D. Goldsmith, and R. Rednic, *Edge mining the Internet of Things*, IEEE Sensors Journal **13** (2013), no. 10, 3816–3825.

[GFMR11] F. Gandino, R. Ferrero, B. Montrucchio, and M. Rebaudengo, *Probabilistic DCS: An RFID reader-to-reader anti-collision protocol*, Journal of Network and Computer Applications **34** (2011), no. 3, 821–832.

[GHS83] R. G. Gallager, P. A. Humblet, and P. M. Spira, *A distributed algorithm for minimum-weight spanning trees*, ACM Transactions on Programming Languages and systems (TOPLAS) **5** (1983), no. 1, 66–77.

[GL96] C. Gotsman and M. Lindenbaum, *On the metric properties of discrete space-filling curves*, IEEE Transactions on Image Processing **5** (1996), no. 5, 794–797.

[GS69] K. R. Gabriel and R. R. Sokal, *A new statistical approach to geographic variation analysis*, Systematic Biology **18** (1969), no. 3, 259–278.

[GSR+12] C. Gross, D. Stingl, B. Richerzhagen, A. Hemel, R. Steinmetz, and D. Hausheer, *Geodemlia: A robust peer-to-peer overlay supporting location-based search*, IEEE 12th International Conference on Peer-to-Peer Computing (P2P'12), IEEE, 2012, pp. 25–36.

[GYL11] R. K. Ganti, F. Ye, and H. Lei, *Mobile crowdsensing: Current state and future challenges*, Communications Magazine, IEEE **49** (2011), no. 11, 32–39.

[Hal10] S. Haller, *The things in the internet of things*, Internet of Things 2010, November 2010.

[HCC09] J. Hui, D. Culler, and S. Chakrabarti, *6LoWPAN: Incorporating IEEE 802.15. 4 into the IP architecture*, IPSO Alliance White Paper **3** (2009).

[HCCL13] X. Hu, T. Chu, H. Chan, and V. Leung, *Vita: A Crowdsensing-Oriented Mobile Cyber-Physical System*, IEEE Transactions on Emerging Topics in Computing **1** (2013), no. 1, 148–165.

[HCS+05] L. Huang, H. Chen, T. Sivakumar, T. Kashima, and K. Sezaki, *Impact of topology on Bluetooth scatternet*, International Journal of Pervasive Computing and Communications **1** (2005), no. 2, 123–134.

[HCY07] C.-H. Hsu, Y. Chen, and C.-T. Yang, *A Layered Optimization Approach for Redundant Reader Elimination in Wireless RFID Networks*, The second IEEE Asia-Pacific Service Computing Conference, IEEE, December 2007, pp. 138–145.

[HES06] J. Ho, D. W. Engels, and S. E. Sarma, *HiQ: a hierarchical Q-learning algorithm to solve the reader collision problem*, International Symposium on Applications and the Internet (SAINT) Workshops, IEEE, 2006, pp. 4–pp.

[HH06] D. Heutelbeck and M. Hemmje, *RectNet-A Distributed Geometrical Data Structure*, The Seventh International Conference on Mobile Data Management (MDM), IEEE, 2006, pp. 149–149.

[HJS+03] N. J. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman, *SkipNet: A Scalable Overlay Network with Practical Locality Properties*, USENIX Symposium on Internet Technologies and Systems, vol. 274, 2003.

[HKS09] S. Haller, S. Karnouskos, and C. Schroth, *The internet of things in an enterprise context*, Future Internet–FIS 2008, Springer, 2009, pp. 14–28.

[HM98] F. Howell and R. McNab, *SimJava: A discrete event simulation library for java*, Simulation Series **30** (1998), 51–56.

[HP01] Y. Hassin and D. Peleg, *Sparse communication networks and efficient routing in the plane*, Distributed computing **14** (2001), no. 4, 205–215.

[HPP07] V. Hunt, A. Puglia, and M. Puglia, *RFID: a guide to radio frequency identification*, Wiley-interscience, 2007.

[HWH06] L. E. Hodge, R. M. Whitaker, and S. Hurley, *Multiple objective optimization of bluetooth scatternets*, Journal of combinatorial optimization **11** (2006), no. 1, 43–57.

[HYC+12] C.-H. Hsu, C.-H. Yu, C.-Y. Chung, C.-T. Yang, and C.-H. Chou, *An overlap aware technique for redundant reader elimination*, The 9th International Conference on Ubiquitous Intelligence & Computing and The 9th International Conference on Autonomic & Trusted Computing (UIC/ATC), IEEE, 2012, pp. 357–360.

[HYHB06] T.-C. Huang, C.-S. Yang, C.-C. Huang, and S.-W. Bai, *Hierarchical Grown Bluetrees (HGB): an effective topology for Bluetooth scatternets*, International Journal of Computational Science and Engineering **2** (2006), no. 1, 23–31.

[IF09] R. Itsuki and A. Fujita, *Consideration for efficient RFID Information retrieval in traceability system*, IEEE Conference on Emerging Technologies & Factory Automation (ETFA 2009), IEEE, 2009, pp. 1–4.

[IY10] N. Irfan and M. Yagoub, *Efficient algorithm for redundant reader elimination in wireless RFID networks*, International journal of computer science issues **3** (2010), no. 11, 1–8.

[JJZ12] A. Jedda, G.-V. Jourdan, and N. Zaguia, *Towards better understanding of the behaviour of Bluetooth networks distributed algorithms*, International Journal of Parallel, Emergent and Distributed Systems **27** (2012), no. 6, 563–586.

[KCJ+07] C. Kiss Kalló, C.-F. Chiasserini, S. Jung, M. Brunato, and M. Gerla, *Hop count based optimization of Bluetooth scatternets*, Ad Hoc Networks **5** (2007), no. 3, 340–359.

[KCKT08] C. Kürschner, C. Condea, O. Kasten, and F. Thiesse, *Discovery service design in the epcglobal network*, The Internet of Things, Springer, 2008, pp. 19–34.

[KHR+12] A. Karime, M. A. Hossain, A. M. Rahman, W. Gueaieb, J. M. Alja'am, and A. El Saddik, *RFID-based interactive multimedia system for the children*, Multimedia Tools and Applications **59** (2012), no. 3, 749–774.

[KK03] M. F. Kaashoek and D. R. Karger, *Koorde: A simple degree-optimal distributed hash table*, Peer-to-Peer Systems II, Springer, 2003, pp. 98–107.

[KKK+08] A. Katasonov, O. Kaykova, O. Khriyenko, S. Nikitin, and V. Y. Terziyan, *Smart Semantic Middleware for the Internet of Things.*, ICINCO-ICSO **8** (2008), 169–178.

[KLK+07] J. Kim, W. Lee, E. Kim, D. Kim, and K. Suh, *Optimized transmission power control of interrogators for collision arbitration in UHF RFID systems*, IEEE Communications Letters **11** (2007), no. 1, 22–24.

[KLS07] A. Kovacevic, N. Liebau, and R. Steinmetz, *Globase.Kom - a p2p overlay for fully retrievable location-based search*, Peer-to-Peer Computing, 2007. P2P 2007. Seventh IEEE International Conference on, IEEE, 2007, pp. 87–96.

[KLY+05] J. Kim, W. Lee, J. Yu, J. Myung, E. Kim, and C. Lee, *Effect of localized optimal clustering for reader anti-collision in RFID networks: fairness aspects to the readers*, International Conference on Computer Communications and Networks (ICCCN), IEEE, 2005, pp. 497–502.

[KPP93] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos, *Multicast routing for multimedia communication*, IEEE/ACM Transactions on Networking (TON) **1** (1993), no. 3, 286–292.

[Kru13] L. G. Kruger, *Internet Governance and the Domain Name System: Issues for Congress*, Tech. report, Congressional Research Service, April 2013.

[KSS09] V. Kantere, S. Skiadopoulos, and T. Sellis, *Storing and indexing spatial data in p2p systems*, IEEE Transactions on Knowledge and Data Engineering **21** (2009), no. 2, 287–300.

[KW06] M. Knoll and T. Weis, *Optimizing locality for self-organizing context-based systems*, Self-Organizing Systems (2006), 62–73.

[LBNS08] H. Liu, M. Bolic, A. Nayak, and I. Stojmenovic, *Taxonomy and challenges of the integration of RFID and wireless sensor networks*, IEEE Network **22** (2008), no. 6, 26–35.

[LCW02] X. Li, G. Calinescu, and P. Wan, *Distributed construction of a planar spanner and routing for ad hoc wireless networks*, Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02), vol. 3, 2002, pp. 1268–1277 vol.3.

[Lin92] N. Linial, *Locality in distributed graph algorithms*, SIAM Journal on Computing **21** (1992), no. 1, 193–201.

[Liu13] H. Liu, *Latest development status of the Internet of Things in China*, The second OpenChina-ICT Thematic Workshop on Internet of Things and Future Internet, Aug 2013.

[LMS03] C. Law, A. K. Mehta, and K. Siu, *A new Bluetooth scatternet formation protocol*, Mobile Networks and Applications **8** (2003), no. 5, 485–498.

[LPT06] C. Lin, W. Peng, and Y. Tseng, *Efficient in-network moving object tracking in wireless sensor networks*, IEEE Transactions on Mobile Computing **5** (2006), no. 8, 1044–1056.

[LRSC01] C. E. Leiserson, R. L. Rivest, C. Stein, and T. H. Cormen, *Introduction to algorithms*, The MIT press, 2001.

[LSW04] X. Y. Li, I. Stojmenovic, and Y. Wang, *Partial Delaunay triangulation and degree limited localized Bluetooth scatternet formation*, IEEE Transactions on Parallel and Distributed Systems **15** (2004), no. 4, 350–361.

[LY94] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, Journal of the ACM (JACM) **41** (1994), no. 5, 960–981.

[Lyn96] N. A. Lynch, *Distributed algorithms*, Morgan Kaufmann, 1996.

[LZ12] Y. Liu and G. Zhou, *Key technologies and applications of internet of things*, Fifth International Conference on Intelligent Computation Technology and Automation (ICICTA'12), IEEE, 2012, pp. 197–200.

[MFH09] G. G. Meyer, K. Främling, and J. Holmström, *Intelligent products: A survey*, Computers in Industry **60** (2009), no. 3, 137–148.

[MJ11] Q. Meng and J. Jin, *The Terminal Design of the Energy Self-Sufficiency Internet of Things*, International Conference on Control, Automation and Systems Engineering (CASE'11), IEEE, 2011, pp. 1–5.

[MM02] P. Maymounkov and D. Mazieres, *Kademlia: A peer-to-peer information system based on the xor metric*, Peer-to-Peer Systems, Springer, 2002, pp. 53–65.

[MNR02] D. Malkhi, M. Naor, and D. Ratajczak, *Viceroy: A scalable and dynamic emulation of the butterfly*, The twenty-first annual symposium on Principles of distributed computing, ACM, 2002, pp. 183–192.

[MO09] J. A. Muir and P. C. V. Oorschot, *Internet geolocation: Evasion and counterevasion*, ACM Computing Surveys (CSUR) **42** (2009), no. 1, 4.

[mob07] *New Mobile Active RFID Reader Goes Anywhere*, RFID Journal (2007).

[MPL11] M. Methfessel, S. Peter, and S. Lange, *Bluetooth Scatternet Tree Formation for Wireless Sensor Networks*, IEEE 8th International Conference on Mobile Adhoc and Sensor Systems (MASS'11), IEEE, 2011, pp. 789–794.

[MR95] R. Motwani and P. Raghavan, *Randomized algorithms*, Cambridge university press, 1995.

[MRSDZ11] Y. Métivier, J. M. Robson, N. Saheb-Djahromi, and A. Zemmari, *An optimal bit complexity randomized distributed MIS algorithm*, Distributed Computing **23** (2011), no. 5-6, 331–340.

[MSDC12] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, *Internet of things: Vision, applications and research challenges*, Ad Hoc Networks **10** (2012), no. 7, 1497–1516.

[Nas04] H. Nasheri, *Economic espionage and industrial spying*, Cambridge University Press, 2004.

[nic08] *Disruptive Civil Technologies*, Tech. report, National Intelligence Council, April 2008.

[NLCQ12] B. Ning, G. Li, Y. Chen, and D. Qu, *Distributed Architecture of Object Naming Service*, Advanced Technology in Teaching-Proceedings of the 2009 3rd International Conference on Teaching and Computational Science (WTCS 2009), Springer, 2012, pp. 251–257.

[Nok11] Nokia, *5 surprising facts about Bluetooth*, May 2011.

[OH12] S. M. Oteafy and H. S. Hassanein, *Towards a global IoT: Resource reutilization in WSNs*, International Conference on Computing, Networking and Communications (ICNC'12), IEEE, 2012, pp. 617–622.

[Pal04] M. Palmer, *Seven principles of effective RFID data management*, Progress Software-Real Time Division (2004).

[PBC03] C. Petrioli, S. Basagni, and I. Chlamtac, *Configuring BlueStars: Multihop Scatternet Formation for Bluetooth Networks*, IEEE Transactions on Computers **52** (2003), 779–790.

[PBC04] ———, *BlueMesh: degree-constrained multi-hop scatternet formation for Bluetooth networks*, Mobile Networks and Applications **9** (2004), no. 1, 33–47.

[Pel90] D. Peleg, *Time-optimal leader election in general networks*, Journal of parallel and distributed computing **8** (1990), no. 1, 96–99.

[PPPW04] K. Park, V. S. Pai, L. Peterson, and Z. Wang, *CoDNS: improving DNS performance and reliability via cooperative lookups*, Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation-Volume 6, USENIX Association, 2004, pp. 14–14.

[PROR99] H. Pasula, S. Russell, M. Ostland, and Y. Ritov, *Tracking many objects with many sensors*, International Joint Conference on Artificial Intelligence, vol. 16, 1999, pp. 1160–1171.

[PRR99] C. G. Plaxton, R. Rajaraman, and A. W. Richa, *Accessing nearby copies of replicated objects in a distributed environment*, Theory of Computing Systems **32** (1999), no. 3, 241–280.

[PRT04] E. Pagani, G. Rossi, and S. Tebaldi, *An on-demand Bluetooth scatternet formation algorithm*, Wireless On-Demand Network Systems (2004), 51–61.

[r108] *Internet of Things in 2020. A Roadmap for the Future*, Tech. report, European Commission Working Group RFID of the ETP EPOSS, 2008.

[RD01a] A. Rowstron and P. Druschel, *Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems*, Middleware 2001, Springer, 2001, pp. 329–350.

[RD01b] ———, *Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility*, ACM SIGOPS Operating Systems Review, vol. 35, ACM, 2001, pp. 188–201.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, *A Scalable Content-addressable Network*, Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (New York, NY, USA), SIGCOMM '01, ACM, 2001, pp. 161–172.

[RMI+10] M. Razalli, M. Mahdi, A. Ismail, S. Shafie, and H. Adam, *Design and development of wireless communication transceiver to support RFID reader at UHF band*, Journal of Electromagnetic Waves and Applications **24** (2010), no. 14-15, 2063–2075.

[Rou06] G. Roussos, *Enabling RFID in retail*, Computer **39** (2006), no. 3, 25–30.

[RRH00] R. Ramanathan and R. Rosales-Hain, *Topology control of multihop wireless networks using transmit power adjustment*, Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'00), vol. 2, IEEE, 2000, pp. 404–413.

[RS04] V. Ramasubramanian and E. G. Sirer, *Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays*, The first conference on Symposium on Networked Systems Design and Implementation, USENIX Association, 2004, pp. 8–8.

[SA04] H. Sreenivas and H. Ali, *An evolutionary Bluetooth scatternet formation protocol*, The 37th Annual Hawaii International Conference on System Sciences, IEEE, 2004, pp. 8–pp.

[San06] N. Santoro, *Design and analysis of distributed algorithms*, vol. 56, Wiley-Interscience, 2006.

[SBTL01] T. Salonidis, P. Bhagwat, L. Tassiulas, and R. LaMaire, *Distributed topology construction of Bluetooth personal area networks*, Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'01), vol. 3, IEEE, 2001, pp. 1577–1586.

[SF03] F. Siegemund and C. Florkemeier, *Interaction in pervasive computing settings using Bluetooth-enabled active tags and passive RFID technology together with mobile phones*, The First IEEE International Conference on Pervasive Computing and Communications (PerCom'03), IEEE, 2003, pp. 378–387.

[She95] S. Shenker, *Fundamental design issues for the future Internet*, IEEE Journal on Selected Areas in Communications **13** (1995), no. 7, 1176–1188.

[SHR06] T. Salminen, S. Hosio, and J. Riekki, *Enhancing bluetooth connectivity with RFID*, Fourth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom'06), IEEE, 2006, pp. 6–pp.

[SLWW05] W. Z. Song, X. Y. Li, Y. Wang, and W. Z. Wang, *dBBlue: Low diameter and self-routing Bluetooth scatternet*, Journal of Parallel and Distributed Computing **65** (2005), no. 2.

[Smi07] J. M. Smith, *A Broader Definition of RFID*, RFID Journal (2007).

[SMK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, *Chord: A scalable peer-to-peer lookup service for internet applications*, ACM SIGCOMM Computer Communication Review, vol. 31, ACM, 2001, pp. 149–160.

[SOTZ05] Y. Shu, B. C. Ooi, K.-L. Tan, and A. Zhou, *Supporting multi-dimensional range queries in peer-to-peer systems*, Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05), IEEE, 2005, pp. 173–180.

[SSYH06] D.-H. Shih, P.-L. Sun, D. C. Yen, and S.-M. Huang, *Taxonomy and survey of RFID anti-collision protocols*, Computer communications **29** (2006), no. 11, 2150–2166.

[SSZ02] I. Stojmenovic, M. Seddigh, and J. Zunic, *Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks*, IEEE Transactions on Parallel and Distributed Systems **13** (2002), no. 1, 14–25.

[STF05] T. Staake, F. Thiesse, and E. Fleisch, *Extending the EPC network: the potential of RFID in anti-counterfeiting*, The ACM symposium on Applied computing, ACM, 2005, pp. 1607–1612.

[Sto12] I. Stojmenovic, *Keynote 1: Mobile Cloud and Green Computing*, Procedia Computer Science **10** (2012), 18–19.

[Sun07] X. Sun, *SCAN: a small-world structured p2p overlay for multi-dimensional queries*, The 16th international conference on World Wide Web, ACM, 2007, pp. 1191–1192.

[Suo13] J. Suomela, *Survey of Local Algorithms*, ACM Comput. Surv. **45** (2013), no. 2, 24:1–24:40.

[SW08] J. Schneider and R. Wattenhofer, *A log-star distributed maximal independent set algorithm for growth-bounded graphs*, The twenty-seventh ACM symposium on Principles of distributed computing, ACM, 2008, pp. 35–44.

[SWRW09] W.-Z. Song, Y. Wang, C. Ren, and C. Wu, *Multi-hop scatternet formation and routing for large scale Bluetooth networks*, International Journal of Ad Hoc and Ubiquitous Computing **4** (2009), no. 5, 251–268.

[SZ06] I. Stojmenovic and N. Zaguia, *Bluetooth scatternet formation in ad-hoc wireless networks*, Chapter 9 in: Performance Modeling and Analysis of Bluetooth Networks: Network Formation, Polling, Scheduling, and Traffic Control (J. Misic and V. Misic), Auerbach Publications (Taylor & Francis Group), 2006, pp. 147–171.

[tag11] *Tag Data Standard, version 1.6*, Tech. report, EPCglobal, 2011.

[Tel88] G. Tel, *Total Algorithms*, Lecture Notes in Computer Science, Concurrency'88, vol. 335, 1988, pp. 227–291.

[TFH+09] F. Thiesse, C. Floerkemeier, M. Harrison, F. Michahelles, and C. Roduner, *Technology, Standards, and Real-World Deployments of the EPC Network*, IEEE Internet Computing **13** (2009), no. 2, 36–43.

[TG05] D. Tian and N. D. Georganas, *Connectivity maintenance and coverage preservation in wireless sensor networks*, Ad Hoc Networks **3** (2005), no. 6, 744–761.

[TJ02] M. Theimer and M. B. Jones, *Overlook: Scalable name service on an overlay network*, The 22nd International Conference on Distributed Computing Systems (ICDCS'02), IEEE, 2002, pp. 52–61.

[TMGB02] G. Tan, A. Miu, J. Guttag, and H. Balakrishnan, *An efficient scatternet formation algorithm for dynamic environments*, The IASTED Communications and Computer Networks (CCN'02), 2002, pp. 4–6.

[Tou80] G. T. Toussaint, *The relative neighbourhood graph of a finite planar set*, Pattern recognition **12** (1980), no. 4, 261–268.

[TSH09] I. Toma, E. Simperl, and G. Hench, *A joint roadmap for semantic technologies and the internet of things*, The 3rd STI Roadmapping Workshop, 2009.

[TW10] L. Tan and N. Wang, *Future internet: The internet of things*, The 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE'10), vol. 5, IEEE, 2010, pp. V5–376.

[TYL+09] S. Tang, J. Yuan, X.-Y. Li, G. Chen, Y. Liu, and J. Zhao, *Raspberry: A stable reader activation scheduling protocol in multi-reader rfid systems*, IEEE International Conference on Network Protocols (ICNP), IEEE, 2009, pp. 304–313.

[uid06] *Ubiquitous ID Architecture*, Tech. Report 910-S002-0.00.24 / UID-CO00002-0.00.24, Ubiquitous ID Center, T-Engine Forum, Nov 2006.

[VD08] N. Vaidya and S. R. Das, *RFID-based networks: exploiting diversity and redundancy*, SIGMOBILE Mobile Computing and Communication Review **12** (2008), no. 1, 2–14.

[VGSR05] E. Vergetis, R. Guérin, S. Sarkar, and J. Rank, *Can Bluetooth succeed as a large-scale ad hoc networking technology?*, IEEE Journal on Selected Areas in Communication **23** (2005), no. 3, 644–656.

[Vio04] B. Violino, *VeriSign to Run EPC Directory*, RFID Journal (2004).

[Vio05] B. Violino, *The History of RFID Technology*, RFID Journal (2005).

[Wan] Q. Wang, *UCBT - Bluetooth extension for NS2 at the University of Cincinnati*, Tech. report, University of Cincinnati.

[Wan08] Y. Wang, *Topology Control for Wireless Sensor Networks*, Wireless Sensor Networks and Applications (Y. Li, M. Thai, and W. Wu, eds.), Signals and Communication Technology, Springer US, 2008, pp. 113–147.

[Wan11] B. Wang, *Coverage problems in sensor networks: A survey*, ACM Computing Surveys **43** (2011), no. 4, 1–53.

[WBF+11] Y. Wang, D. Burgener, M. Flores, A. Kuzmanovic, and C. Huang, *Towards street-level client-independent IP geolocation*, The eighth USENIX conference on networked systems design and implementation (NSDI'11), 2011, pp. 27–36.

[Wei91] M. Weiser, *The computer for the 21st century*, Scientific american **265** (1991), no. 3, 94–104.

[WES03] J. Waldrop, D. W. Engels, and S. E. Sarma, *Colorwave: an anticollision algorithm for the reader collision problem*, IEEE International Conference on Communications (ICC), vol. 2, IEEE, 2003, pp. 1206–1210.

[WHC05] R. M. Whitaker, L. Hodge, and I. Chlamtac, *Bluetooth scatternet formation: A survey*, Ad Hoc Networks **3** (2005), no. 4, 403–450.

[WLL+10] M. Wu, T.-J. Lu, F.-Y. Ling, J. Sun, and H.-Y. Du, *Research on the architecture of Internet of Things*, The third International Conference on Advanced

Computer Theory and Engineering (ICACTE'10) , vol. 5, 2010, pp. V5–484–V5–487.

[WMMY93] R. E. Walpole, R. H. Myers, S. L. Myers, and K. Ye, *Probability and statistics for engineers and scientists*, vol. 8, Prentice Hall Upper Saddle River^ eNJ NJ, 1993.

[WMZA02] C. Y. Wong, D. McFarlane, A. A. Zaharudin, and V. Agarwal, *The intelligent product driven supply chain*, IEEE International Conference on Systems, Man and Cybernetics, vol. 4, IEEE, 2002, pp. 1–6.

[WNR07] L. Wang, B. Norman, and J. Rajgopal, *Placement of multiple RFID reader antennas to maximise portal read accuracy*, International Journal of Radio Frequency Identification Technology and Applications **1** (2007), no. 3, 260–277.

[WSL04] Y. Wang, I. Stojmenovic, and X.-Y. Li, *Bluetooth scatternet formation for single-hop ad hoc networks based on virtual positions*, The ninth International Symposium on Computers and Communications (ISCC'04), vol. 1, IEEE, 2004, pp. 170–175.

[WT09] T. Winter and R. D. Team, *RPL: Routing protocol for low power and lossy networks*, Networking Working Group Internet drafts (2009).

[WTH09] Z. Wang, R. K. Thomas, and J. Haas, *Performance comparison of Bluetooth scatternet formation protocols for multi-hop networks*, Wireless Networks **15** (2009), no. 2, 209–226.

[WTJ+11] G. Wu, S. Talwar, K. Johnsson, N. Himayat, and K. Johnson, *M2M: From mobile to embedded internet*, IEEE Communications Magazine **49** (2011), no. 4, 36–43.

[WZ04] R. Wattenhofer and A. Zollinger, *XTC: a practical topology control algorithm for ad-hoc networks*, Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International, April 2004, p. 216.

[XWW11] H. Xu, S.-P. Wang, and R.-C. Wang, *P2PONS: A Distributed Object Naming Service Architecture based on P2P for EPC Network*, Advances in Information Sciences and Service Sciences **31** (2011).

[XZ02] Z. Xu and Z. Zhang, *Building low-maintenance expressways for p2p systems*, Hewlett-Packard Labs, Palo Alto, CA, Tech. Rep. HPL-2002-41 (2002).

[Yao82] A. C.-C. Yao, *On constructing minimum spanning trees in k-dimensional spaces and related problems*, SIAM Journal on Computing **11** (1982), no. 4, 721–736.

[YL12] C.-M. Yu and J.-H. Lin, *Enhanced Bluetree: a mesh topology approach forming Bluetooth scatternet*, IET Wireless Sensor Systems **2** (2012), no. 4, 409–415.

[You95] N. Young, *Randomized rounding without solving the linear program*, The sixth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 1995, pp. 170–178.

[YS11] H. Yihua and L. Shilei, *A middleware-based approach for redundant reader elimination*, The fifth International Conference on New Trends in Information Science and Service Science (NISS), vol. 1, IEEE, 2011, pp. 209–214.

[YYL08] K.-M. Yu, C. W. Yu, and Z.-Y. Lin, *A density-based algorithm for redundant reader elimination in a RFID network*, The second International Conference on Future Generation Communication and Networking (FGCN'08), vol. 1, IEEE, 2008, pp. 89–92.

[ZBC01] G. Zaruba, S. Basagni, and I. Chlamtac, *BlueTrees - Scatternet formation to enable Bluetooth-based personal area networks*, The IEEE International Conference on Communications (ICC'01), vol. 1, 6 2001, pp. 273–277.

[ZDS08] N. Zaguia, Y. Daadaa, and I. Stojmenovic, *Simplified bluetooth scatternet formation using maximal independent sets*, Integrated Computer-Aided Engineering **15** (2008), no. 3, 229–239.

[ZGDZ07] Z. Zhou, H. Gupta, S. R. Das, and X. Zhu, *Slotted scheduled tag access in multi-reader RFID systems*, IEEE International Conference on Network Protocols (ICNP), IEEE, 2007, pp. 61–70.

[ZGLB10] M. Zorzi, A. Gluhak, S. Lange, and A. Bassi, *From today's intranet of things to a future internet of things: a wireless-and mobility-related view*, IEEE Wireless Communications **17** (2010), no. 6, 44–51.

[ZHS03] H. Zhang, J. C. Hou, and L. Sha, *A Bluetooth loop scatternet formation algorithm*, IEEE International Conference on Communications (ICC'03), vol. 2, IEEE, 2003, pp. 1174–1180.

[ZLN11] D. Zhang, Y. Liu, and L. Ni, *Rass: A real-time, accurate and scalable system for tracking transceiver-free objects*, IEEE International Conference on Pervasive Computing and Communications (PerCom'11), IEEE, 2011, pp. 197–204.

Part VII

Appendices

Appendix A

Case Study: `BlueMISI` using `OrderedExchange`

This chapter introduces implementations of `BlueMISI` using `OrderedExchange`. These implementations improve the execution time of `BlueMISI`. Recall that, in `BlueMISI`, each node v in this round builds the set `MIS`(v), which is a maximal independent set of the neighbors list $N(v)$. Each node v shall slave all its neighbors in `MIS`(v). The construction of `MIS`(v) for each $v \in V$ may lead to a *symmetry* scenario. That is, a pair of neighbors slave themselves simultaneously. These symmetries are broken using unambiguous local rules. More details about `BlueMISI` are found in Section 5.3 and Figure 5.6.

The original specifications of `BlueMISI` in [ZDS08] state implicitly that `BlueMISI` is constructed using one communication round of `RandomExchange`. This can be noted also from the pseudocode of `BlueMISI` in Algorithm 5.1. Given that `OrderedExchange` outperforms `RandomExchange` in Bluetooth networks in execution time (see Section 5.5.3), the intuition indicates that an implementation of `BlueMISI` using `OrderedExchange` leads to an improved execution time also. Simulation experiments in Section 5.6 confirms this intuition. However, using `OrderedExchange` with `BlueMISI` requires a redesign of `BlueMISI`. Thus, two designs (or, implementations) of `BlueMISI` using `OrderedExchange`, called `ComputeMISI` and `ComputeMISII`, are given in the following.

`ComputeMISI:` is a simple algorithm that consists of two `OrderedExchange` communication rounds. In the first round, each node v exchanges its neighbors list $N(v)$ with all its neighbors. Then, each node v locally constructs a maximal independent set `MIS`(v) of its neighbors, using a similar procedure to that used in `BlueMISI`. Initially, `MIS`(v) is empty.

The neighbors of v are passed by in order from smallest to largest. A node u is inserted into $\texttt{MIS}(v)$ if there is no neighbor of u in $\texttt{MIS}(v)$. An additional communication round is needed herein to break the symmetries found in $\texttt{MIS}(v)$ for each node v. Recall that a symmetry occurs if $v \in MIS(u)$ and $u \in MIS(v)$ for a pair of neighbors v and u. Thus, each node v exchanges the set $\texttt{MIS}(v)$ with all its neighbors in a second communication round. Then, each node v locally breaks the symmetries found in $\texttt{MIS}(v)$ and $\texttt{MIS}(u)$ for all $u \in N(v)$, using the same rules of `BlueMISI` (that is, v becomes the master of u and u becomes the slaves of v if v is larger than v). Note that in `ComputeMISI`, the communication rounds are used to exchange the lists $N(v)$ and $\texttt{MIS}(v)$ for each node v. Thus, the order of contacting the neighbors is not important. However, the order of building locally the set $\texttt{MIS}(v)$ should be preserved as it is the case in `BlueMISI` (see Remark 5.3.4).

`ComputeMISII:` is another algorithm that implements `BlueMISI` using `OrderedExchange`. The algorithm constructs the set $\texttt{MIS}(v)$ at each node v on-the-fly, and requires two `OrderedExchange` communication rounds.

The basic idea is to divide the neighbor list $N(v)$ at each node v into the subsets $N_s(v)$ and $N_l(v)$, where $N_s(v)$ consists of the smaller neighbors of v and $N_l(v)$ consists of the larger neighbors of v. The set $\texttt{MIS}(v)$ is also divided into $\texttt{MIS}_s(v)$ and $\texttt{MIS}_l(v)$, where $\texttt{MIS}_s(v)$ are the smaller neighbors of v in $\texttt{MIS}(v)$ and $\texttt{MIS}_l(v)$ are the larger neighbors of v in $\texttt{MIS}(v)$. The set $\texttt{MIS}_s(v)$ can be constructed from $N_s(v)$ directly using the same rules of `BlueMISI` (i.e., by replacing $N(v)$ by $N_s(v)$). The construction of the set $\texttt{MIS}_l(v)$, however, depends on the contents of $\texttt{MIS}_s(v)$. This is because $\texttt{MIS}_l(v)$ must not have any node u that is neighbor to a node $u' \in \texttt{MIS}_s(v)$. Let $\mathcal{N}(\texttt{MIS}_s(v))$ be the set that consists of the union of all the neighbors of each node in $\texttt{MIS}_s(v)$. That is, $\mathcal{N}(\texttt{MIS}_s(v)) = \bigcup_{u' \in MIS_s(v)} N(u')$. Then, the set $\texttt{MIS}_l(v)$ is constructed from the set $N_l(v) \setminus \mathcal{N}(MIS_s(v))$ using the same rules of `BlueMISI` (by replacing $N(v)$ in `BlueMISI` by $N_l(v) \setminus \mathcal{N}(\texttt{MIS}_s(v))$). Thus, the union of $\texttt{MIS}_s(v)$ and $\texttt{MIS}_l(v)$ is $\texttt{MIS}(v)$.

`ComputeMISII` implements the rules of constructing $\texttt{MIS}_s(v)$ and $\texttt{MIS}_l(v)$ using two communication rounds. The first round constructs $\texttt{MIS}_s(v)$. The set $\{N_l(v) \setminus \mathcal{N}(MIS_s(v))\}$ is locally built at the beginning of the second round. Then, the set $\texttt{MIS}_l(v)$ is constructed. $\texttt{MIS}(v)$, therefore, is set to $\{\texttt{MIS}_s(v) \cup \texttt{MIS}_l(v)\}$. The symmetries in $\texttt{MIS}(v)$ are broken locally in `ComputeMISII`. The pseudocode of `ComputeMISII` is shown in Algorithm A.1.

Algorithm A.1 `ComputeMISII` at node v

1: $E \leftarrow N(v)$, $\texttt{MIS}(v) \leftarrow \emptyset$, $M(v) \leftarrow \emptyset$, $p \leftarrow 0$
2: **while** $p \leq 1$ **do**
3: $L \leftarrow \emptyset$
4: **if** $p = 0$ **then**
5: $A \leftarrow N_l(v)$, $B \leftarrow N_s(v)$
6: **else**
7: $A \leftarrow N_s(v)$, $B \leftarrow N_l(v)$
8: **while** $A \neq \emptyset$ **do**
9: wait for receipt of messages of all nodes $u \in A$
10: **if** message received from u is *attack* **then**
11: $M(v) \leftarrow \{M(v) \cup v\}$
12: $A \leftarrow \{A \setminus u\}$
13: **while** $B \neq \emptyset$ **do**
14: $v \leftarrow \min(E)$ {v is the node with minimum identifier in E}
15: $\texttt{MIS}(v) \leftarrow \{\texttt{MIS}(v) \cup u\}$
16: send a message to v, call the message *attack*
17: $B \leftarrow \{B \setminus \{u \cup N(u)\}\}$
18: $E \leftarrow \{E \setminus \{u \cup N(u)\}\}$
19: $L \leftarrow \{L \cup \{N(v) \cap N(u)\}\}$
20: **for** each $u_i \in L$ **do**
21: send a message to u_i, call the message *dummy*
22: $p \leftarrow p + 1$
23: locally break symmetries found in $\texttt{MIS}(v)$ and $M(v)$

Appendix B

Details of Simulation Experiments

This chapter gives the detailed numerical results of the simulation experiments performed in this thesis. The chapter is divided into sections. Each section considers experiments performed in a certain chapter.

B.1 Experiments Numerical Values and Confidence Interval Calculations

The numerical values of the thesis experiments are given in the following. Each figure that illustrates an experiment result is reperesented as a table. Each table gives the detailed numerical values of the experiments results figures given in the thesis. Each experiment is represented by the pair $(avg, std, rng, p - rng)$, where:

1. *avg* is the average of all the experiment runs. That is, if an algorithm is run n times and each run gave a different performance $x_1, \ldots, x_n$, then:

$$avg = \sum_{i=1}^{n} x_i/n \tag{B.1}$$

2. *std* is the standard deviation of all the experiment runs, calculated as follows:

$$std = \sqrt{\frac{\sum_{i=1}^{n}(x_i - avg)^2}{n-1}} \tag{B.2}$$

3. *rng* is the half of the 95% confidence interval. That is, the simulation experiment result is expected to have a value within the interval $[avg - rng, avg + rng]$ with a probability equal to 0.95. It is calculated as follows:

$$rng = t_{\alpha/2,n-1}\frac{std}{\sqrt{n}} \tag{B.3}$$

where $t_{\alpha',n'}$ is the t-value that leaves $1-\alpha'$ to its right with a degree of freedom equals to n'. This is because the random sample (i.e., the experiment runs results) are assumed to have a t-distribution with $n-1$ degree of freedom. See [WMMY93] for more details.

4. *p-rng* is the percentage of *rng* with respect to the average. That is:

$$p - rng = \frac{rng}{avg} \times 100\% \tag{B.4}$$

B.2 Reader-Tag RFID Coverage Algorithms

Table B.1: Impact of the number of tags on the number of non-redundant readers in uniform geometric topologies (Figure3.1)

Num Tags	DRRE				RRE				RANDOM				MAXMIN				GDE				LIM-GDE(1)				NTE			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
100	32.02	2.58	0.51	1.58	27.87	2.59	0.51	1.82	29.35	2.26	0.44	1.51	26.23	1.80	0.35	1.35	21.73	1.37	0.27	1.24	22.49	1.68	0.33	1.46	23.47	1.51	0.30	1.26
250	42.61	4.31	0.84	1.98	38.81	3.40	0.67	1.72	34.80	2.72	0.53	1.53	30.51	2.25	0.44	1.45	25.91	1.43	0.28	1.08	29.12	2.50	0.49	1.68	27.54	1.50	0.29	1.07
500	47.14	4.56	0.89	1.90	45.11	3.98	0.78	1.73	36.12	2.44	0.48	1.32	32.12	2.15	0.42	1.31	28.10	1.50	0.29	1.05	32.75	2.88	0.56	1.72	29.28	1.69	0.33	1.13
750	48.89	4.68	0.92	1.88	47.84	3.65	0.72	1.50	37.42	2.88	0.56	1.51	32.63	2.24	0.44	1.35	28.96	1.43	0.28	0.97	33.87	3.07	0.60	1.78	30.22	1.66	0.33	1.08

Table B.2: Impact of the number of readers on the number of non-redundant readers in uniform geometric topologies (Figure 3.2)

Num readers	DRRE				RRE				RANDOM				MAXMIN				GDE				LIM-GDE(1)				NTE			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
75	38.64	3.33	0.65	1.69	36.30	2.86	0.56	1.54	34.13	2.42	0.47	1.39	30.36	2.11	0.41	1.36	26.47	1.37	0.27	1.01	28.23	1.83	0.36	1.27	27.52	1.62	0.32	1.15
100	43.33	4.27	0.84	1.93	38.80	3.35	0.66	1.69	34.41	2.74	0.54	1.56	30.32	2.15	0.42	1.39	25.98	1.26	0.25	0.95	28.78	2.41	0.47	1.64	27.77	1.51	0.30	1.07
125	45.61	4.72	0.93	2.03	41.11	3.37	0.66	1.61	34.95	2.39	0.47	1.34	30.37	1.99	0.39	1.28	25.91	1.54	0.30	1.16	29.02	2.32	0.45	1.57	27.48	1.74	0.34	1.24
150	46.72	5.00	0.98	2.10	41.12	3.59	0.70	1.71	34.29	2.37	0.46	1.35	30.01	2.01	0.39	1.31	25.40	1.50	0.29	1.16	29.17	2.71	0.53	1.82	27.16	1.67	0.33	1.21

Table B.3: Impact of the number of readers on the number of non-redundant readers in uniform geometric topologies (Figure 3.3)

Num readers	DRRE				RRE				RANDOM				MAXMIN				GDE				LIM-GDE(1)				NTE			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
75	42.58	4.18	0.82	1.92	43.09	3.39	0.66	1.54	36.25	2.56	0.50	1.38	32.44	2.00	0.39	1.21	29.37	1.45	0.28	0.97	33.35	2.40	0.47	1.41	30.89	1.74	0.34	1.10
100	49.26	5.02	0.98	2.00	49.03	3.75	0.74	1.50	37.57	2.86	0.56	1.49	32.68	2.12	0.42	1.27	29.21	1.36	0.27	0.91	35.09	2.92	0.57	1.63	30.69	1.60	0.31	1.02
125	54.69	5.85	1.15	2.10	53.27	4.65	0.91	1.71	38.46	2.90	0.57	1.48	33.26	2.20	0.43	1.30	28.84	1.60	0.31	1.09	35.81	3.27	0.64	1.79	30.98	1.76	0.34	1.11
150	58.95	6.20	1.22	2.06	57.13	5.09	1.00	1.75	38.78	3.13	0.61	1.58	33.05	2.15	0.42	1.28	29.10	1.42	0.28	0.96	37.69	4.05	0.79	2.11	30.35	1.71	0.34	1.10

Table B.4: Impact of the number of tags on the number of non-redundant readers in region-based topologies with $z = 25$ (Figure 3.5)

Num tags	DRRE				RRE				RANDOM				MAXMIN				GDE				LIM-GDE(1)				NTE			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
100	26.82	4.35	0.85	3.18	22.60	3.70	0.73	3.21	12.60	2.30	0.45	3.58	10.64	2.21	0.43	4.07	7.72	1.58	0.31	4.01	13.38	3.16	0.62	4.63	8.58	1.70	0.33	3.88
250	38.46	5.56	1.09	2.83	34.04	5.56	1.09	3.20	14.48	3.03	0.59	4.10	11.30	2.31	0.45	4.01	8.70	1.45	0.28	3.27	18.64	4.26	0.83	4.48	9.85	1.98	0.39	3.94
500	46.38	6.40	1.25	2.70	42.92	6.64	1.30	3.03	15.86	3.67	0.72	4.54	13.00	2.86	0.56	4.31	10.12	1.71	0.34	3.31	24.20	6.27	1.23	5.08	11.39	2.37	0.46	4.08
750	52.98	6.49	1.27	2.40	48.72	5.40	1.06	2.17	17.38	3.48	0.68	3.92	13.42	2.70	0.53	3.94	10.76	1.34	0.26	2.44	27.96	5.49	1.08	3.85	11.46	2.01	0.39	3.44

Table B.5: Impact of the number of tags on the number of non-redundant readers in region-based topologies with $z = 125$ (Figure 3.6)

Num tags	DRRE				RRE				RANDOM				MAXMIN				GDE				LIM-GDE(1)				NTE			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
100	33.68	3.41	0.67	1.98	26.28	3.11	0.61	2.32	20.32	3.12	0.61	3.01	16.68	2.33	0.46	2.74	12.32	1.91	0.37	3.04	20.20	3.61	0.71	3.50	13.90	2.67	0.52	3.76
250	49.00	4.55	0.89	1.82	40.20	4.70	0.92	2.29	24.00	4.20	0.82	3.43	19.72	3.42	0.67	3.40	15.16	1.95	0.38	2.52	30.52	4.88	0.96	3.13	17.37	3.04	0.60	3.43
500	59.20	5.80	1.14	1.92	49.84	6.87	1.35	2.70	26.36	4.33	0.85	3.22	21.52	4.92	0.96	4.48	17.24	3.31	0.65	3.76	38.72	6.13	1.20	3.10	18.92	3.23	0.63	3.35
750	63.52	6.79	1.33	2.10	57.24	6.63	1.30	2.27	29.32	5.68	1.11	3.80	23.56	4.36	0.85	3.63	17.80	3.05	0.60	3.36	45.00	5.78	1.13	2.52	19.34	3.59	0.70	3.64

Table B.6: Impact of the number of tags on the number of non-redundant readers in Gaussian geometric topologies (Figure 3.7)

Num tags	DRRE				RRE				RANDOM				MAXMIN				GDE				LIM-GDE(1)				NTE			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
100	19.85	7.43	1.46	7.34	15.02	3.39	0.66	4.42	9.52	2.36	0.46	4.86	7.82	2.22	0.44	5.56	6.05	1.47	0.29	4.76	11.80	3.41	0.67	5.66	6.52	1.70	0.33	5.11
250	29.68	14.10	2.76	9.31	23.45	6.73	1.32	5.63	12.08	5.68	1.11	9.22	9.50	4.18	0.82	8.62	7.60	2.62	0.51	6.76	18.50	5.33	1.04	5.65	8.10	2.66	0.52	6.44
500	33.42	14.17	2.78	8.31	27.92	6.56	1.29	4.61	11.75	4.00	0.78	6.67	9.50	3.31	0.65	6.83	7.30	2.20	0.43	5.91	23.62	4.91	0.96	4.07	7.72	1.94	0.38	4.93
750	39.05	15.59	3.06	7.82	31.80	6.96	1.36	4.29	13.82	4.35	0.85	6.17	9.58	2.85	0.56	5.83	7.75	1.79	0.35	4.53	26.48	6.45	1.26	4.77	8.37	2.25	0.44	5.27

Table B.7: Impact of the edge probability on the number of non-redundant readers in arbitrary topologies (Figure 3.8)

Edge prob.	DRRE				RRE				RANDOM				MAXMIN				GDE				LIM-GDE(1)				NTE			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
0.20	23.74	1.66	0.33	1.37	22.10	1.59	0.31	1.41	24.80	1.65	0.32	1.30	22.62	1.52	0.30	1.32	14.96	0.56	0.11	0.73	22.28	1.48	0.29	1.30	15.94	0.90	0.18	1.11
0.40	12.48	0.75	0.15	1.18	11.28	1.11	0.22	1.93	12.28	0.87	0.17	1.39	11.68	0.99	0.19	1.66	7.90	0.30	0.06	0.74	11.24	1.24	0.24	2.16	7.98	0.24	0.05	0.60
0.60	7.66	0.79	0.15	2.02	7.14	0.80	0.16	2.20	7.80	0.80	0.16	2.01	7.40	0.75	0.15	1.99	5.00	0.01	0.00	0.04	6.80	0.82	0.16	2.36	7.20	0.98	0.19	2.67
0.80	4.75	0.62	0.12	2.56	4.52	0.59	0.12	2.56	4.80	0.56	0.11	2.29	4.55	0.55	0.11	2.37	3.02	0.16	0.03	1.04	4.10	0.62	0.12	2.96	4.52	0.53	0.10	2.30

Table B.8: Impact of the number of tags on the number of non-redundant readers in uniform geometric topologies with probabilistic edges (Figure 3.9)

Num tags	DRRE				RRE				RANDOM				MAXMIN				GDE				LIM-GDE(1)				NTE			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
100	36.28	3.08	0.60	1.66	29.50	2.64	0.52	1.75	34.33	2.28	0.45	1.30	30.31	2.26	0.44	1.46	23.68	1.43	0.28	1.18	24.70	1.84	0.36	1.46	25.92	1.65	0.32	1.25
250	51.55	4.34	0.85	1.65	43.42	3.58	0.70	1.62	44.60	2.64	0.52	1.16	39.02	2.58	0.51	1.30	30.88	1.92	0.38	1.22	35.11	3.16	0.62	1.76	33.26	1.96	0.38	1.16
500	59.90	4.25	0.83	1.39	51.89	3.63	0.71	1.37	50.92	3.29	0.64	1.27	46.09	2.55	0.50	1.08	37.01	1.88	0.37	1.00	45.05	3.02	0.59	1.31	39.79	2.18	0.43	1.07
750	64.53	4.17	0.82	1.27	57.63	3.40	0.67	1.16	56.00	3.23	0.63	1.13	50.24	3.07	0.60	1.20	41.04	1.96	0.38	0.94	52.30	3.83	0.75	1.44	44.11	2.06	0.40	0.92

Table B.9: Convergence rate of `LIMITED-GDE` in uniform geometric topologies (Figure 3.11)

Num tags	LIM-GDE(0)				LIM-GDE(1)				LIM-GDE(2)				GDE			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
100	22.02	3.68	0.72	3.28	13.42	3.01	0.59	4.40	9.13	2.25	0.44	4.83	7.76	1.55	0.30	3.91
250	33.73	5.76	1.13	3.35	19.51	5.04	0.99	5.06	12.43	3.41	0.67	5.38	9.07	1.66	0.33	3.59
500	42.99	6.76	1.32	3.08	24.88	5.94	1.16	4.68	15.93	4.38	0.86	5.39	10.11	1.88	0.37	3.64
750	48.49	6.42	1.26	2.60	27.84	5.40	1.06	3.80	18.14	4.46	0.87	4.82	10.89	1.75	0.34	3.15

Table B.10: Convergence rate of `LIMITED-GDE` in region-based topologies (Figure 3.12)

Num tags	LIM-GDE(0)				LIM-GDE(1)				LIM-GDE(2)				GDE			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
100	22.02	3.68	0.72	3.28	13.42	3.01	0.59	4.40	9.13	2.25	0.44	4.83	7.76	1.55	0.30	3.91
250	33.73	5.76	1.13	3.35	19.51	5.04	0.99	5.06	12.43	3.41	0.67	5.38	9.07	1.66	0.33	3.59
500	42.99	6.76	1.32	3.08	24.88	5.94	1.16	4.68	15.93	4.38	0.86	5.39	10.11	1.88	0.37	3.64
750	48.49	6.42	1.26	2.60	27.84	5.40	1.06	3.80	18.14	4.46	0.87	4.82	10.89	1.75	0.34	3.15

Table B.11: Convergence rate of `RANDOM`$^+$ in uniform geometric topologies (Figure 3.13)

Num tags	RANDOM				MAX-MIN				RANDOM-2				RANDOM-3			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
100	29.35	2.26	0.44	1.51	26.23	1.80	0.35	1.35	28.21	2.20	0.43	1.53	27.37	2.05	0.40	1.47
250	34.80	2.72	0.53	1.53	30.51	2.25	0.44	1.45	32.62	2.35	0.46	1.41	31.75	1.80	0.35	1.11
500	36.12	2.44	0.48	1.32	32.12	2.15	0.42	1.31	34.73	2.31	0.45	1.30	33.16	2.14	0.42	1.26
750	37.42	2.88	0.56	1.51	32.63	2.24	0.44	1.35	35.46	2.41	0.47	1.33	33.89	2.27	0.44	1.31

Table B.12: Total number of reads in uniform geometric topologies (Figure 3.16 - Part(1))

Num tags	DRRE				RRE				RANDOM				MAX-MIN				GDE			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
100	983.90	50.01	9.80	1.00	491.95	25.00	4.90	1.00	491.95	25.00	4.90	1.00	644.47	27.61	5.41	0.84	1383.24	121.62	23.84	1.72
250	2478.08	67.83	13.29	0.54	1239.04	33.92	6.65	0.54	1239.04	33.92	6.65	0.54	1676.13	50.55	9.91	0.59	3844.58	370.99	72.71	1.89
500	4930.62	93.53	18.33	0.37	2465.31	46.77	9.17	0.37	2465.31	46.77	9.17	0.37	3390.60	89.22	17.49	0.52	7829.74	558.74	109.51	1.40
750	7402.14	119.91	23.50	0.32	3701.07	59.96	11.75	0.32	3701.07	59.96	11.75	0.32	5104.23	122.70	24.05	0.47	11812.58	935.08	183.28	1.55

Table B.13: Total number of reads in uniform geometric topologies (Figure 3.16 - Part(2))

Num tags	LIM-GDE(1)				LIM-GDE(2)				RANDOM-2				RANDOM-3			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
100	1150.52	67.34	13.20	1.15	1347.63	104.58	20.50	1.52	643.47	30.28	5.93	0.92	791.15	36.24	7.10	0.90
250	3007.72	119.70	23.46	0.78	3673.24	262.49	51.45	1.40	1670.54	49.75	9.75	0.58	2076.90	58.67	11.50	0.55
500	6045.10	174.86	34.27	0.57	7463.72	406.06	79.59	1.07	3392.84	91.07	17.85	0.53	4223.10	145.18	28.46	0.67
750.00	9100.54	269.91	52.90	0.58	11259.43	668.48	131.02	1.16	5106.39	114.90	22.52	0.44	6388.36	191.14	37.46	0.59

Table B.14: Total number of overwrites in uniform geometric topologies (Figure 3.17 Part(1))

Num tags	DRRE				RRE				RANDOM				MAX-MIN				GDE			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
100	710.36	32.56	6.38	0.90	219.00	9.93	1.95	0.89	218.89	9.34	1.83	0.84	321.40	16.66	3.27	1.02	309.41	25.42	4.98	1.61
250	1787.74	43.81	8.59	0.48	548.47	16.24	3.18	0.58	548.60	15.99	3.13	0.57	832.74	27.12	5.32	0.64	858.68	75.53	14.80	1.72
500	3557.88	60.54	11.87	0.33	1092.88	23.40	4.59	0.42	1092.75	20.73	4.06	0.37	1688.45	46.97	9.21	0.55	1766.93	125.82	24.66	1.40
750	5340.29	79.34	15.55	0.29	1646.93	29.77	5.83	0.35	1639.15	28.98	5.68	0.35	2531.05	75.17	14.73	0.58	2662.72	210.96	41.35	1.55

Table B.15: Total number of overwrites in uniform geometric topologies (Figure 3.17 Part(2))

Num tags	LIM-GDE(1)				LIM-GDE(2)				RANDOM-2				RANDOM-3			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
100	293.25	19.14	3.75	1.28	308.17	25.36	4.97	1.61	242.98	12.77	2.50	1.03	265.20	16.43	3.22	1.21
250	786.60	44.58	8.74	1.11	851.69	67.48	13.23	1.55	622.50	27.39	5.37	0.86	668.82	33.88	6.64	0.99
500	1602.81	72.64	14.24	0.89	1751.84	119.20	23.36	1.33	1238.61	59.94	11.75	0.95	1348.51	73.57	14.42	1.07
750	2416.42	105.41	20.66	0.85	2635.23	192.13	37.66	1.43	1859.39	62.56	12.26	0.66	2028.61	108.33	21.23	1.05

B.3 Reader-Reader RFID Coverage Algorithms

Table B.16: Impact of the number of tags on the average number of iterations (Figure 4.8)

Num tags	IOB-COVERAGE(nt)				IOB-COVERAGE(nr)				IOB-(RCA+COV)(nt)				IOB-(RCA+COV)(nr)				GA-1			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
250	7.36	1.20	0.24	3.20	14.64	1.60	0.31	2.14	46.00	5.05	0.99	2.15	42.28	3.69	0.72	1.71	4.44	0.25	0.05	1.10
500	7.72	0.78	0.15	1.98	16.52	2.08	0.41	2.47	46.48	3.67	0.72	1.55	42.72	2.91	0.57	1.34	4.96	0.36	0.07	1.42
750	7.56	0.80	0.16	2.07	16.88	1.77	0.35	2.06	45.52	4.17	0.82	1.80	43.04	3.21	0.63	1.46	5.04	0.28	0.05	1.09
1000	7.68	0.79	0.15	2.02	17.48	1.84	0.36	2.06	45.28	3.80	0.74	1.64	42.36	2.70	0.53	1.25	5.12	0.35	0.07	1.34

Table B.17: Impact of the number of readers on the average number of iterations (Figure 4.9)

Num readers	IOB-COVERAGE(nt)				IOB-COVERAGE(nr)				IOB-(RCA+COV)(nt)				IOB-(RCA+COV)(nr)				GA-1			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
50	7.84	1.38	0.27	3.45	12.24	1.11	0.22	1.78	23.12	2.70	0.53	2.29	20.72	1.51	0.30	1.43	5.72	0.60	0.12	2.06
100	7.68	0.79	0.15	2.02	17.48	1.84	0.36	2.06	45.28	3.80	0.74	1.64	42.36	2.70	0.53	1.25	5.12	0.35	0.07	1.34
150	8.12	0.95	0.19	2.29	21.24	2.60	0.51	2.40	68.20	4.72	0.93	1.36	63.20	3.77	0.74	1.17	4.64	0.39	0.08	1.65
200	7.80	0.89	0.17	2.24	23.20	1.90	0.37	1.61	90.44	5.64	1.11	1.22	84.56	4.50	0.88	1.04	4.68	0.30	0.06	1.26

B.4 Bluetooth Scatternet Formation Algorithms: Eliminate

Table B.18: Execution time of `Eliminate`, `BlueStars`, and the different implementations of `BlueMISI` (Figure 5.9)

Num nodes	ComputeMIS I				ComputeMIS II				BlueMIS I				Eliminate				BlueStars			
Nodes	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
30	1.30	0.27	0.05	4.07	1.05	0.20	0.04	3.78	2.87	0.75	0.15	5.12	0.62	0.15	0.03	4.74	0.87	0.21	0.04	4.72
50	2.95	0.57	0.11	3.79	2.35	0.49	0.10	4.09	5.02	1.72	0.34	6.71	1.45	0.32	0.06	4.33	1.39	0.25	0.05	3.53
70	5.42	1.10	0.22	3.98	3.95	0.96	0.19	4.76	6.90	2.13	0.42	6.05	2.64	0.52	0.10	3.86	1.91	0.35	0.07	3.59
90	8.18	1.32	0.26	3.16	6.98	1.88	0.37	5.28	11.18	2.42	0.47	4.24	4.10	0.64	0.13	3.06	2.40	0.38	0.07	3.10
110	11.75	1.81	0.35	3.02	11.01	2.45	0.48	4.36	13.24	1.90	0.37	2.81	5.88	0.90	0.18	3.00	2.95	0.45	0.09	2.99

B.5 Bluetooth Scatternet Formation Algorithms: BSF-UED

Table B.19: Comparison of the execution time of major BSF algorithms (Figure 6.4)

Num nodes	BlueStars				BlueMesh				BlueMIS I				BlueMIS II				BSFUED			
Nodes	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
30	0.90	0.21	0.04	4.58	4.43	1.07	0.21	4.73	0.94	0.20	0.04	4.22	2.47	0.77	0.15	6.11	1.53	0.51	0.10	6.53
50	1.42	0.25	0.05	3.44	10.02	1.56	0.31	3.05	2.19	0.49	0.10	4.39	5.75	1.64	0.32	5.59	3.32	0.75	0.15	4.43
70	1.92	0.35	0.07	3.57	17.82	2.98	0.58	3.28	3.94	0.96	0.19	4.78	9.55	2.32	0.45	4.76	5.52	1.21	0.24	4.30
90	2.45	0.38	0.07	3.04	25.63	4.22	0.83	3.23	6.61	1.88	0.37	5.58	15.14	3.88	0.76	5.02	7.52	0.98	0.19	2.55
110	3.05	0.45	0.09	2.89	35.92	4.58	0.90	2.50	10.43	2.45	0.48	4.60	22.71	4.90	0.96	4.23	10.17	1.20	0.24	2.31

Table B.20: Comparison of the number of piconets (masters) of major BSF algorithms (Figure 6.5)

Num nodes	BlueStars				BlueMesh				BlueMIS I				BlueMIS II				BSFUED				BSFUED-H1			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
30	8.51	1.45	0.28	3.34	11.04	1.80	0.35	3.20	26.33	1.07	0.21	0.80	13.42	2.17	0.43	3.17	13.49	2.18	0.43	3.16	13.49	2.17	0.43	3.15
50	10.25	1.84	0.36	3.52	22.85	2.47	0.48	2.12	46.58	1.11	0.22	0.47	22.15	3.84	0.75	3.40	26.92	2.84	0.56	2.07	26.92	2.84	0.56	2.07
70	11.15	2.07	0.41	3.64	37.99	2.74	0.54	1.41	66.88	1.30	0.25	0.38	29.76	6.29	1.23	4.14	42.80	3.78	0.74	1.73	43.43	3.93	0.77	1.77
90	11.46	1.96	0.38	3.35	54.87	2.83	0.55	1.01	87.04	1.22	0.24	0.27	38.78	8.31	1.63	4.20	55.82	4.88	0.96	1.71	59.23	4.96	0.97	1.64
110	11.72	2.07	0.41	3.46	73.27	3.08	0.60	0.82	107.16	1.33	0.26	0.24	46.09	11.69	2.29	4.97	66.22	5.60	1.10	1.66	77.03	6.14	1.20	1.56

Table B.21: Comparison of the number of M/S bridges of major BSF algorithms (Figure 6.8)

Num Nodes	BlueStars				BlueMesh				BlueMIS I				BlueMIS II				BSFUED				BSFUED-H1			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
30	2.28	1.15	0.23	9.89	3.27	1.22	0.24	7.31	12.93	2.31	0.45	3.50	10.56	2.51	0.49	4.66	6.81	1.66	0.33	4.78	6.80	1.66	0.33	4.78
50	3.32	1.43	0.28	8.44	6.74	1.59	0.31	4.62	20.86	2.91	0.57	2.73	16.93	2.89	0.57	3.35	16.28	2.91	0.57	3.50	16.28	2.90	0.57	3.49
70	4.01	1.59	0.31	7.78	10.23	2.17	0.43	4.16	26.53	4.08	0.80	3.01	22.00	4.18	0.82	3.72	29.99	4.24	0.83	2.77	30.28	4.32	0.85	2.80
90	4.01	1.58	0.31	7.73	13.46	2.87	0.56	4.18	31.92	3.99	0.78	2.45	26.09	3.98	0.78	2.99	41.86	5.05	0.99	2.36	44.70	5.44	1.07	2.39
110	4.21	1.74	0.34	8.10	17.95	3.12	0.61	3.41	36.55	4.52	0.89	2.42	29.52	3.96	0.78	2.63	51.92	5.93	1.16	2.24	61.20	6.75	1.32	2.16

Table B.22: Comparison of the number of S/S bridges of major BSF algorithms (Figure 6.9)

Num Nodes	BlueStars				BlueMesh				BlueMIS I				BlueMIS II				BSFUED				BSFUED-H1			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
30	6.01	1.55	0.30	5.06	6.68	1.78	0.35	5.22	2.20	0.99	0.19	8.82	10.41	2.44	0.48	4.59	4.12	1.35	0.26	6.42	4.12	1.35	0.26	6.42
50	8.09	1.82	0.36	4.41	13.19	2.61	0.51	3.88	2.16	1.11	0.22	10.07	20.54	5.44	1.07	5.19	7.33	1.79	0.35	4.79	7.36	1.79	0.35	4.77
70	9.46	2.05	0.40	4.25	20.35	2.60	0.51	2.50	2.30	1.27	0.25	10.82	33.65	7.45	1.46	4.34	11.44	2.85	0.56	4.88	11.11	2.70	0.53	4.76
90	10.52	2.01	0.39	3.74	25.01	2.79	0.55	2.19	2.22	1.06	0.21	9.36	43.89	10.10	1.98	4.51	19.38	3.59	0.70	3.63	16.17	3.30	0.65	4.00
110	11.09	2.11	0.41	3.73	28.63	2.84	0.56	1.94	2.10	1.17	0.23	10.92	56.16	13.82	2.71	4.82	29.99	5.25	1.03	3.43	19.79	4.44	0.87	4.40

Table B.23: Comparison of the number bridges of major BSF algorithms (Figure 6.7)

Num nodes	BlueStars				BlueMesh				BlueMIS I				BlueMIS II				BSFUED				BSFUED-H1			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
30	8.29	1.82	0.36	4.30	9.95	2.05	0.40	4.04	15.14	2.21	0.43	2.86	20.98	2.82	0.55	2.63	10.93	2.08	0.41	3.73	10.93	2.08	0.41	3.73
30	11.41	2.32	0.45	3.98	19.92	2.79	0.55	2.75	22.25	2.78	0.54	2.45	37.47	5.62	1.10	2.94	23.65	2.20	0.43	1.82	23.64	3.05	0.60	2.53
30	13.46	2.87	0.56	4.18	30.59	3.38	0.66	2.17	28.83	3.78	0.74	2.57	55.65	7.40	1.45	2.61	41.43	4.91	0.96	2.32	41.41	4.93	0.97	2.33
30	14.52	2.75	0.54	3.71	38.47	3.95	0.77	2.01	34.14	3.76	0.74	2.16	69.97	10.11	1.98	2.83	61.25	5.20	1.02	1.66	60.94	5.20	1.02	1.67
30	15.30	2.83	0.55	3.63	46.58	3.63	0.71	1.53	38.66	4.26	0.83	2.16	85.68	13.82	2.71	3.16	81.92	6.04	1.18	1.45	80.99	6.33	1.24	1.53

Table B.24: Comparison of the average role per node of major BSF algorithms Figure 6.6

Num nodes	BlueStars				BlueMesh				BlueMIS I				BlueMIS II				BSFUED				BSFUED-H1			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
30	1.37	0.01	0.00	0.14	1.61	0.03	0.01	0.37	2.37	0.01	0.00	0.08	1.94	0.02	0.00	0.20	1.46	0.09	0.02	1.21	1.46	0.09	0.02	1.21
50	1.31	0.01	0.00	0.15	2.06	0.01	0.00	0.12	2.73	0.01	0.00	0.07	2.24	0.01	0.00	0.09	1.67	0.11	0.02	1.29	1.67	0.11	0.02	1.29
70	1.25	0.01	0.00	0.16	2.56	0.02	0.00	0.13	2.98	0.01	0.00	0.05	2.45	0.14	0.03	1.12	1.93	0.15	0.03	1.52	1.94	0.15	0.03	1.52
90	1.22	0.01	0.00	0.16	2.96	0.08	0.02	0.53	3.11	0.02	0.00	0.13	2.57	0.02	0.00	0.15	2.17	0.18	0.04	1.63	2.21	0.19	0.04	1.69
110	1.18	0.00	0.00	0.07	3.27	0.02	0.00	0.09	3.21	0.02	0.00	0.12	2.67	0.01	0.00	0.07	2.38	0.21	0.04	1.73	2.48	0.23	0.05	1.82

Table B.25: Comparison of the average shortest path of major BSF algorithms (Figure 6.10)

Num nodes	BlueStars				BlueMesh				BlueMIS I				BlueMIS II				BSFUED				BSFUED-H1			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
30	3.55	0.30	0.06	1.66	3.42	0.33	0.06	1.89	3.04	0.30	0.06	1.93	3.04	0.30	0.06	1.93	4.62	0.49	0.10	2.08	4.62	0.49	0.10	2.08
50	3.73	0.19	0.04	1.00	3.35	0.21	0.04	1.20	2.98	0.16	0.03	1.05	2.97	0.16	0.03	1.06	4.86	0.45	0.09	1.81	4.86	0.45	0.09	1.81
70	3.78	0.15	0.03	0.78	3.42	0.50	0.10	2.87	2.95	0.12	0.02	0.80	2.94	0.12	0.02	0.81	4.61	0.40	0.08	1.70	4.61	0.40	0.08	1.70
90	3.86	0.14	0.03	0.71	3.14	0.08	0.02	0.50	2.96	0.10	0.02	0.68	2.95	0.10	0.02	0.66	4.32	0.31	0.06	1.40	4.32	0.31	0.06	1.40
110	3.91	0.13	0.03	0.65	3.08	0.10	0.02	0.64	2.95	0.08	0.02	0.53	2.95	0.08	0.02	0.53	4.12	0.25	0.05	1.19	4.12	0.25	0.05	1.19

Table B.26: Comparison of the total number of sent messages of major BSF algorithms (Figure 6.11)

Num nodes	BlueStars				BlueMesh				BlueMIS I				BlueMIS II				BSFUED				BSFUED-H1			
	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng	avg	std	rng	p-rng
30	356	30	6.05	1.70	1251	174	34	2.73	480	82	16	3.36	1064	225	44	4.15	693	164	32	4.64	693	164	32	4.64
50	854	62	12	1.44	3968	460	90	2.27	1752	411	80	4.61	3610	693	135	3.76	2112	417	81	3.87	2112	417	81	3.87
70	1565	103	20	1.29	8943	911	178	2.00	4629	1277	250	5.41	8209	1599	313	3.82	4353	678	132	3.05	4353	678	132	3.05
90	2484	143	28	1.13	16129	1475	289	1.79	10634	3506	687	6.46	16431	4110	805	4.90	7288	896	175	2.41	7288	896	175	2.41
110	3605	185	36	1.01	26376	1968	385	1.46	20803	5773	1131	5.44	29497	6510	1275	4.33	11293	1259	246	2.19	11293	1259	247	2.19