PhD Dissertation

International Doctorate School in Information and Communication Technologies

DIT - University of Trento

Ad Hoc Network Optimization from the Bluetooth Scatternets and Delay-Tolerant Networks Perspectives

Csaba Kiss Kalló

Advisor:

Dr. Mauro Brunato

Università degli Studi di Trento

March 2005

DIT-05-019

Acknowledgements

This thesis could not have been completed without the precious support of many people. Above all, I want to thank to my advisor Dr. Mauro Brunato who provided me with his advice and feedback on my work. I am thankful also to Prof Roberto Battiti for the environment and support for my studies.

I would like to thank to Dr. Carla-Fabiana Chiasserini for her collaboration on the Bluetooth scatternet optimization part of this thesis, and to Prof Mario Gerla who hosted me for several months in his prestigious research group where I developed a part of my thesis in collaboration with Sewook Jung.

Finally, I owe special thanks to my fiancee, parents, grandparents and brother for being beside me despite the long geographycal distance between us and for their unconditional support also in difficult times.

Thank you all!

Abstract

In the past five years Bluetooth scatternets were one of the most promising wireless networking technologies for ad hoc networking. In such networks, mobility together with the fact that wireless network nodes may change their communication peers in time, generate permanently changing traffic flows. Thus, forming an optimal scatternet for a given traffic pattern may be not enough, rather a scatternet that best supports traffic flows as they vary in time is required.

In this thesis, we propose a novel heuristic algorithm suite capable of dynamically adapting the network topology to the existing traffic connections between scatternet nodes. The periodic adaptation of the scatternet topology to the traffic connections enables the routing algorithms to identify shorter paths between communicating network nodes. This will allow for a more efficient communication in terms of throughput and power consumption.

Secondly, we also present a performance analysis and optimization of delay-tolerant networks (DTNs), i.e. ad hoc networks (e.g scatternets) with no end-to-end path between the communicating nodes. In DTNs heavy buffer usage and long message delivery delays may render the communication inefficient. In this work we present our heuristic routing algorithm for location-aware message delivery, called K2*, based on the k-nearest-neighbors technique.* K2 *uses location information for interlocking the region where the destination of a packet is supposed to be reachable. Simulations show that* K2 *reduces buffer requirements, while maintaining routing delays ap-*

proximately unchanged when compared to a flooding-based routing algorithm.

Keywords
[wireless ad hoc networks, bluetooth scatternets, delay-tolerant networks, heuristic optimization algorithms, simulations]

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This section briefly introduces the key aspects of the research effort presented in this thesis and describes several problems and our solutions to them with an emphasis on the main contributions of this work. The structure of the thesis is also presented at the end of the section.

1.1 The Context

Pervasive networking environments, with always-on wireless connectivity by means of portable computers, PDAs and cellular phones, are becoming part of our everyday experience, at least in office environments. A trend towards the everyday life has also begun. Low-bandwidth systems such as packet-oriented cellular connections (GPRS) have been commercially available for years, and third generation cellular systems, oriented both to voice and data, are being rapidly deployed and commercialized.

Further, wireless LANs are starting to appear in many public places, such as stations, shopping malls department stores, hospitals, universities, entertainment parks, etc., in [1] collectively called *e-villages*. An e-village consists of a limited, mixed indoor/outdoor area with a high user population density served by a sophisticated intelligent environment, which provides context-aware information distribution, wireless connectivity among

mobile users, and location-based applications and services. Taking advantage of short-range radio technologies, like Wi-Fi and Bluetooth, in an e-village the mobile users can access the Internet while on the move or form spontaneous networks without any need for infrastructure. This provides the users with an additional degree of communication freedom enabling spontaneous inter-personal data exchange as well as body area networking. Such networks are generally called *ad hoc networks* or *scatternets* in the Bluetooth terminology.

Due to mobility, interference, battery depletion and alike, ad hoc networks may get partitioned and hence, some network nodes may not be reachable for other nodes through an end-to-end connection. To overcome this problem and, in general, to provide connectivity between two isolated ad hoc networks, a new communication paradigm, known as *delay-* or *disruption-tolerant networks* (DTN), is being developed. Communication through the segmented connections of a DTN is based on a family of store-and-forward techniques.

This thesis presents optimization techniques for Bluetooth scatternets and DTNs. Although DTNs may be used for interconnecting isolated Bluetooth scatternets, these two topics are addressed independently in this thesis. Instead of presenting DTNs as a complement of Bluetooth scatternets, in this work we aim at demonstrating the usability of this state-of-the-art communication paradigm independently of the wireless technology used. The specific problems that we tackled are summarized in the following section.

1.2 The Problem

In this section the problems addressed in the thesis are introduced and the objective of our work is briefly formulated. The problems pertaining to the

Bluetooth scatternet and DTN issues are separated in two subsections.

1.2.1 Dynamic Behavior of Scatternet Nodes

In the past couple of years considerable research effort has been dedicated to find an efficient way for organizing a set of Bluetooth nodes into a connected scatternet. This issue is challenging for two main reasons.

First, network formation should be as fast as possible, despite the time-consuming neighborhood discovery procedure and the time necessary for all nodes to set up a set of links that ensure a connected network.

Second, beside the necessity of having a path between any pair of nodes, it is also important to set up the most appropriate links among the nodes, since this defines the overall performance of the scatternet in terms of throughput, power consumption and communication delay.

As a result of this effort, researchers proposed a wide range of scatternet formation algorithms [7, 9, 23, 33, 42, 59, 68, 70], each of them providing an optimal solution within the frame of their initial set of assumptions. However, mobility and the fact that nodes may change their communication peers in time, as well as similar dynamic factors like interference, battery depletion and node failure, generate permanently changing traffic flows in a scatternet. Therefore, even if we had an ideal scatternet formation algorithm capable of producing optimally interconnected networks of Bluetooth devices, some time after the scatternet formation suboptimal traffic flows would show up. Thus, for an efficient communication it is not enough to form an optimal scatternet, rather a scatternet that best supports traffic flows as they vary in time is required.

The above motivation drove us to study the possibilities for maintaining the optimal scatternet topology even after the network formation phase. Therefore, our first objective is to define a technique for reconfiguring the scatternet topology based on the current traffic flows of the network, such

that the new topology allows a higher overall throughput and a more efficient power consumption in the scatternet.

1.2.2 Buffer Usage and Delays in DTNs

Since in DTNs no end-to-end path is assumed between the communicating nodes, packet delivery happens on a store-and-forward basis. In this networking paradigm an end-to-end path is segmented into two or more parts, conditioned by the reachability of potential relay nodes along the path. Therefore, routing in a DTN is highly dependent on the available knowledge about future connections of the nodes. The availability of such knowledge enables the nodes along a segmented communication path to make smart decisions when selecting the next hop for the packets that they need to forward toward their destinations. Such decisions have direct impact on the efficient use of network resources, such as storage capacity, bandwidth and power consumption. For instance, if a node that needs to forward a packet to some directly unreachable destination knows nothing about the future connections of its neighbors, it floods its whole neighborhood with the packet. However, if it can deduce that one of its neighbors will meet the destination of its packet later on, it will forward the packet to that neighbor only.

One of the most important information that can significantly influence the performance of a DTN is the *trajectory* of a mobile node. If nodes know where their neighbors are heading, they may be capable of figuring out which of their neighbors may successfully deliver their packets to the destination. However, usually the future trajectory of nodes is hard to obtain. Instead, in certain circumstances we may know the position that a target node visited recently and hence interlock an area where it may be reachable now. With such a tool in our hands we could reduce the number of relay nodes involved in the transmission of a packet, hence

saving network resources.

The scenario where location information could be used for target interlocking (i.e. for identifying the area where the destination of a packet is supposed to be reachable) could be an urban transportation network (called *busnet* in the followings) where the buses (and similar vehicles) are outfitted with mobile access points (MAPs) capable of relaying data between mobile users and the Internet. Having the support of a busnet users could continue sending emails even after moving away from a hot spot (i.e. the coverage of a static access point (AP)) because MAPs could collect their messages and take them to the Internet. This way the network connectivity of users could be extended also to places that are not covered by a fixed network infrastructure.

In consequence, our second objective is to define a routing algorithm based on target interlocking that optimizes the resource usage in a DTN in terms of buffer requirements and routing delays compared to a network-wide flooding based approach.

1.3 The Solution

This section present briefly our approach for reaching the objectives set forth in the previous section.

1.3.1 Hop Count Reduction in Scatternets

As a result of our investigation on methods for dynamically adapting the scatternet topology to support more efficiently the currently existing traffic flows in the network, we found that there is a strong relationship between the path length connecting communicating devices and the overall throughput and power consumption of the scatternet. Intuitively, if a packet has to cross many links to reach its destination it will occupy a larger amount

from the limited capacity of the links on its route, which then cannot be used by other traffic flows. Therefore, the lower the number of links crossed by a packet, the higher the unallocated amount of bandwidth. This applies for power consumption as well: the bigger the number of relay nodes for transmitting a packet, the higher the power consumption in the scatternet. To the best of our knowledge we are the first to express this intuition in terms of analytical formulas for Bluetooth scatternets. As a secondary result of our analysis, we also demonstrate the impact of link quality on the overall scatternet throughput, with respect to the average path length of the traffic flows.

After showing the existence of a strong relationship between the path length (or hop count) and scatternet performance, our objective becomes the definition of an algorithm suite capable of reconfiguring the scatternet such that to reduce the hop count. Therefore, we propose a novel heuristic algorithm suite that enables the modification of the network nodes' links and roles based on the length of each node's traffic connections. The algorithms update the topology of the scatternet making it possible for the routing algorithms to identify shorter paths between the communicating peers.

Finally, we use simulations to evaluate the behavior of our algorithm suite in the presence of changing traffic flows and node mobility. We execute the optimization algorithms periodically to track their impact on the overall throughput and power consumption of the scatternet. Our simulations show that the scatternet throughput can be increased significantly even though the higher throughput requires more energy to be consumed. However, this energy is not wasted on carrying data through lengthy paths, but on transmitting useful information along optimized routes.

1.3.2 Using Location Information for Routing in DTNs

For optimizing buffer usage and packet forwarding delays we define and evaluate a novel heuristic routing algorithm for location-aware message delivery, called `K2`. `K2` is based on two variants of the k-nearest-neighbors algorithm (hence its name), both used for interlocking the region where the destination of a packet is supposed to be reachable. First the K nearest static access points (APs) are selected then the C mobile access points (MAPs) that are supposed to meet the target (i.e. the destination node) the soonest are identified too. We evaluate the approach through simulations and compare its performance to the `MFF` (Message Forwarding with Flooding) routing algorithm based on general flooding.

Simulations show that `K2` results in reduced buffer requirements, while routing delays are not increased significantly when compared to `MFF`. Simulations also suggest that adding a DTN-like message forwarding architecture with mobile access points to an existing fixed infrastructure can be an effective way to improve wireless coverage for delay-tolerant end-user applications such as e-mail, message passing and news broadcast as well as for efficiently connecting infostations and resource-constrained sensor nodes to their central management system.

1.4 Innovative Aspects

The contribution of this thesis is multifold on both topics addressed: the optimization of Bluetooth scatternets and delay-tolerant networks.

In general terms, the thesis describes a novel technique for *reconfiguring the topology of Bluetooth scatternets* based on the current traffic flows in the network, such that the new topology allows a higher overall throughput and a more efficient power consumption in the scatternet.

To achieve this goal first we deduced an analytical relationship between

the hop count and the throughput and energy consumption in a scatternet. This deduction also required us to provide a formal description of how the network nodes share their communication capacity among their links. As a secondary result of this analytical approach we also demonstrate the relationship between the wireless link quality and the throughput in the scatternet.

Second, we defined a matrix-based scatternet model which is a very helpful tool when formulating optimization problems or making scatternet-related implementations. The model reflects the links and relationships between the nodes of the scatternet topology. We used this model to provide a formal definition of the optimization problem regarding the hop reduction in a Bluetooth scatternet.

Third, we defined and implemented a set of so-called *moves*, which are elementary modifications that we perform for shaping the scatternet topology. Taking advantage of these moves and the matrix-based scatternet model we defined an algorithm suite, called the *optimization procedure*, capable of reconfiguring the scatternet topology such that the hop count between the communicating peers is minimized. The optimization procedure uses a local search strategy to find an optimized topology that best supports the traffic flows in the scatternet.

Finally, since none of the available Bluetooth scatternet simulators were adequate for our purposes, we implemented a simulator for evaluating the impact of the optimization algorithms on the scatternet performance. Experiments performed with this simulator confirmed our analytical results, that is, the hop count reduction increases the overall scatternet throughput providing a more efficient usage of the available energy of the devices.

As per the second argument of the thesis, this work contributes to the *DTN research* in two different ways.

First, it presents a novel heuristic routing algorithm, `K2`, capable of

reducing the storage capacity requirements at the fixed and mobile access points while maintaining routing delays unchanged. The algorithm reduces the number of transmitted replicas of the packets in the network, hence it reduces the bandwidth occupation as well. `K2` is based on location information and operates similarly to the LAR (Location Aided Routing) algorithm [40]. Our approach is different from LAR because it does not need to set up communication paths based on route-request messages, that can not operate properly in environments where no end-to-end connection exists between the traffic sources and destinations. Furthermore, `K2` uses redundant paths to deliver packets, which can reduce delays on the expense of using a bit more of the network resources, as shown in [51].

Second, we implement a simulator for the presented DTN scenario that helps us to evaluate our routing algorithm and provide a useful analysis of the buffer usage and routing delays in the modeled DTN. In the frame of this analysis we identify the buffers and delays involved in the transmission of packets in the DTN and point out those of them which have a major impact on the performance of the network. In consequence we obtain a clear view on which buffers and delays can be used for improving the efficiency of the communication and, hence, in the development of `K2` we focus on those elements.

1.5 Structure of the Thesis

This work is structured in two main parts and an additional part which concludes the thesis. The first and main part of this work addresses the issue of Bluetooth scatternet optimization, while the second one presents an analysis and optimization of DTNs.

After this introductory chapter that provides a brief overview about the issues discussed in this thesis, Chapter 2 presents the state-of-the-art in

the fields of Bluetooth scatternets and delay-tolerant networks. Since the primary argument of this thesis is related to the Bluetooth scatternets, more space has been dedicated to present the different aspects of this issue than for DTNs. Thus, the first four sections of Chapter 2 present the most relevant aspects of the Bluetooth technology, the state-of-the-art research on scatternet formation and optimization and several publicly available Bluetooth simulators. The last section of the chapter provides an overview on delay-tolerant networks.

Following Chapter 2 the thesis branches, according to the two arguments, in Part I that presents our scatternet-related optimizations and Part II describing our work on DTNs. Part I addresses three inter-related problems in six chapters.

Chapter 3 provides a detailed description of each of the three problems including a formal scatternet optimization problem formulation together with our notations (Section 3.3).

Chapter 4 presents our analytical demonstration regarding the relationship between the hop count, throughput and power consumption in a scatternet (Section 4.4 and 4.5), that shows the importance of scatternet optimization based on hop count reduction. Also, in this chapter we present the link quality model used for the optimizations (Section 4.2) and describe our formal model for communication capacity sharing in Bluetooth scatternets (Section 4.3).

Chapter 5 relies heavily on the problem formulation of Section 3.3 to present our technique for scatternet optimization based on hop count reduction. In Section 5.3 we introduce the so-called "moves", which are the key elements in the development of the hop count reduction algorithms in Sections 5.5–5.7.

Chapter 6 focuses on the application of the hop reduction algorithms, described in Chapter 5, also for dynamic scatternets. In consequence,

the algorithms are embedded in an "optimization process" (Section 6.4) that aims at improving the performance of scatternets in the presence of dynamic traffic flows (Section 6.2) and node mobility (Section 6.3).

Chapter 7 contains the simulations that we performed to verify the solutions for the scatternet optimization problems described in Chapter 4, 5 and 6. Thus, Section 7.1 shows the evolution of the throughput and power consumption calculated with the analytical formulas of Chapter 4; Section 7.2 presents the performance of the hop count reduction algorithms; and finally, Section 7.3 describes the experiments with dynamic scatternets.

Chapter 8, the last one in Part I, contains the related work for the scatternet optimization part.

Part II presents our work on delay-tolerant network optimization and is structured as follows. Chapter 9 describes the DTN optimization problem, while Chapter 10 contains our solution to that problem. This solution consist of a routing algorithm based on "target interlocking", presented in Section 10.4. In Section 10.5 we describe the simulator that we implemented to evaluate this routing algorithm. Simulation results can be found in Chapter 11.

Chapter 12 examines the relevant literature for the DTN optimization problem.

As already mentioned, Part III concludes the two previous parts as well as this thesis and outlines several topics for future work.

Chapter 2

State of the Art

Bluetooth is a short-range, low power wireless communication technology with ad hoc networking capabilities. In this chapter we provide a brief overview of this technology and its potential for ad hoc networking along with research efforts regarding scatternet optimizations. Additionally, we briefly present some open source Bluetooth simulators that we considered for our work. Finally, in Section 2.5 we provide an overview on the state-of-the-art research in the field of delay-tolerant networks, which is the second argument of this thesis.

2.1 The Bluetooth Technology

2.1.1 Introduction

Nowadays the utilization of intelligent devices became part of the everyday experience. Advanced handheld devices like PDAs, mobile phones, digital cameras and alike are widely used today. To enhance the individual capabilities of these devices and enable their use for a wider range of applications, wireless radio technologies were developed, which allow their easy interconnection. One of the earliest wireless technologies designed to allow seamless communication of devices, by replacing cables and the need for

line-of-sight visibility required by the infrared technologies, is *Bluetooth*.

The interesting name of this technology is borrowed from King Harald Blåtand of Denmark who united Denmark and Norway for several years more than 1000 years ago. In a similar manner, Bluetooth aims to unite different type of devices in one network.

The Bluetooth technology was conceived by the Ericsson telecommunications manufacturer in 1994 [49]. In 1998 leading companies like Ericsson, Intel, IBM, Nokia and Toshiba formed the Bluetooth Special Interest Group (SIG). They published the first version (1.0) of the specification in two volumes (specification and profiles) in July 1999. In December of the same year four more companies, 3Com, Lucent, Microsoft and Motorola, became members of the SIG. These companies are only the promoters of the SIG, but there are several thousands of adopting members as well. A new version (1.1) of the specification [12] was published in February, 2001. In March 2002 the major part of the specification was adopted by the IEEE 802.15.1 standard. Finally, the latest version (1.2) of the specification [14] was published in May, 2003.

Bluetooth can provide better solutions than the existing ones in many scenarios. Using Bluetooth, the spaghetti-like image of cabling from behind a personal computer can disappear. The mouse, keyboard, printer, scanner, speakers and joystick can all be connected to the PC through Bluetooth radio links. A mobile phone can use the microphone and speakers of a laptop or car through a Bluetooth link, demonstrating how devices can borrow each others capabilities using the Bluetooth technology.

Using a Bluetooth headset and a mobile phone it is possible to answer and initiate calls without the need to take out the phone from the pocket or briefcase. A laptop can use a mobile phone as a modem for accessing the Internet without any cables. As a combination of the above two examples, a notebook in the briefcase can use the mobile phone in the pocket for

accessing periodically the Internet and download the new emails without requiring any action from the user. This example belongs to the so-called *hidden* or *unconscious* computing set of applications [49].

The list of possible applications of the Bluetooth wireless technology could be continued through many interesting examples. The possibility of organizing a big number of Bluetooth devices in a single connected network would enlarge this list even more. Pervasive and ubiquitous computing are both supported by this technology. Body Area Networking (BAN) and Personal Area Networking (PAN) highly benefit of the Bluetooth technology.

The last version of the Bluetooth specification [14] defines how to organize Bluetooth devices in a network called *piconet* with at most 8 active nodes. Further, it introduces the *scatter mode* for supporting communication among inter-connected piconets, called *scatternet*. However, the problem of efficient scatternet formation, discussed in numerous research papers is still not addressed in this latest version of the specification.

This section aims at presenting the state-of-the-art in the field of Bluetooth scatternets. On this purpose we first present the Bluetooth technology, providing more details on scatternet-related aspects. Second, we provide a literature survey on the issue of scatternet formation. Finally, we present research efforts aimed at rendering Bluetooth networking efficient.

2.1.2 Bluetooth Overview

Bluetooth is a short-range, low power wireless communication technology operating in the free 2400-2483.5 MHz radio frequency band. Its standard communication range is of 10/100 meters at 0/20 dBm transmission power, but with special directional antennas line-of-sight communication at more than 1500 meters is also possible at the same 20 dBm transmit power. The Bluetooth transceiver uses the Frequency Hopping Code Division Multiple

Access (FH-CDMA) modulation and multiple access scheme, spreading the signal on a large frequency spectrum. This scheme offers the possibility of sharing the same frequency among many users at the same time. The hopping frequency is of 1600 hops per second (3200 hops during connection establishment) over the 79 (in some countries 23) channels of 1 MHz.

2.1.3 Bluetooth Data Packets

Bluetooth uses packet-based communication. Every packet consists of a 72-bit long access code, a 54-bit long header and a payload of up to 2745 bits [27].

Bluetooth data communication happens through Asynchronous Connectionless Links (ACL) using time slots of 625μs. Data packets may use 1, 3 or 5 slots and they may be Forward Error Coded (FEC). FEC packets are called DM1, DM3 and DM5 while the non-error coded ones are called DH1, DH3 and DH5 (with the digits indicating the number of slots used). The useful maximum payload of these packets is 136, 968 and 1816 bits for DM and 216, 1464 and 2712 bits for DH packets, respectively. Packets with bigger payloads can achieve higher throughput in error-free environments (i.e. with high link quality). However, if a bit gets corrupted, the whole packet will have to be retransmitted. Therefore, when retransmissions happen often, smaller packets are more efficient. DM packets have smaller payloads than their DH counterparts, but their content is error checked, in contrast with DH packets.

2.1.4 Communication Model

Communication in the Bluetooth technology is based on the master-slave paradigm. Any device can play the role of master or slave, but two devices have to decide who takes which role before they set up a communication

channel to each other. The master role is assumed by the device that starts the channel set up, but later the roles can be switched.

During the channel setup, called *connection establishment* or *pairing*, the devices exchange their globally unique 48-bit device address and synchronize the slave's 28-bit clock and frequency hopping sequence to that of the master. Without this synchronization the devices are not able to communicate even if they are in range.

One master is capable of simultaneously coordinating up to 7 active and 256 passive slaves. Active slaves have an active member address and can communicate with each other through the master. They form a star-shaped network called *piconet*. Since a piconet can have one master only, it is identified by the master's unique device address. Passive slaves should be in one of the three Bluetooth low-power modes (presented later) and can not communicate in the piconet, unless they become active. Among the slaves, the communication capacity of the master is shared on a time division duplex (TDD) basis.

2.1.5 Connection Establishment

A Bluetooth device can be in one of 9 possible states. A state diagram is shown in Figure 2.1, explained next.

- *Standby* – initial, default low power state; only the native (internal) clock working;
- *Inquiry* – the device sends out messages, searching for other devices;
- *Inquiry Scan* – the device listens for and replies inquiry messages;
- *Page* – a device initiates a connection to an earlier discovered (during inquiry) device;

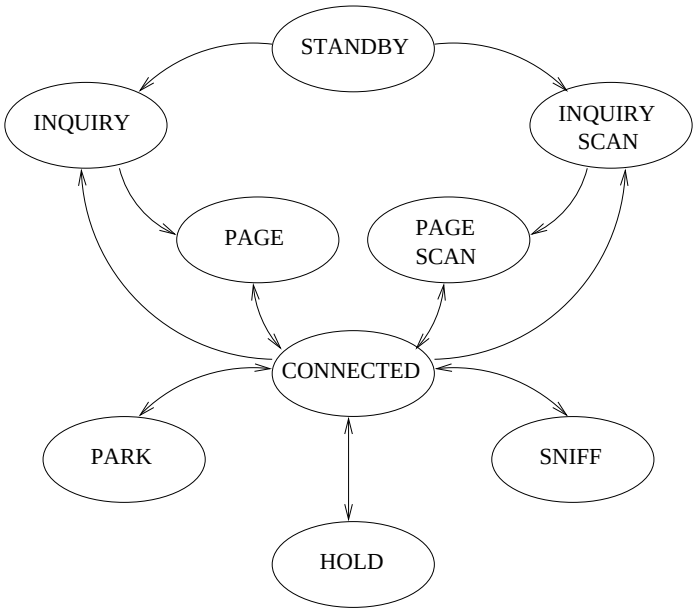

Figure 2.1: Connection establishment in Bluetooth

- *Page Scan* – a device waits for being paged by another device that inquired it earlier;
- *Connected* – active operation state; useful communication takes place in this state;
- *Park* – very low power mode for slaves;
- *Hold* – a slave enters this low power mode for a specified time interval;
- *Sniff* – a slave in this low power mode listens to master transmissions only at given intervals of time.

A typical connection establishment between two Bluetooth devices happens as follows. After two devices, A and B, are powered on they end up in the standby state. They start alternating between the inquiry and inquiry scan states. At a given instant, device B, being in the inquiry scan state, receives the inquiry message of A (in inquiry state) and replies with

an FHS (Frequency Hopping Sequence) packet that contains its device address and clock. Device B moves to the page scan state waiting for being paged (i.e. contacted) by A, which is in the page state. At this moment we know that the slave in the shaping piconet will be B while A will take the role of master. Master A, after receiving the inquiry response of the slave B, enters the page state and sends a DAC (Device Access Code) packet containing only the B's device address. B replies with another DAC packet serving as an acknowledgment. Finally, A sends an FHS packet with its own device address and clock to B. After the reception of this packet B synchronizes its clock and frequency hopping sequence to that of A and both enter the connected state. From this state the devices can go back to the inquiry or page states for contacting other devices or the slaves can go in one of the three low power modes. Even if not shown in Figure 2.1, they can also go back to the standby state.

2.1.6 Inter-connected Piconets

A device can join more than one piconet at the same time, but can be active only in one of them. A device that is connected to more than one piconet at the same time is called *bridge*. A bridge can be master in one of its piconets or it can be slave in all of them and hence we differentiate them as *master&bridge* and *slave&bridge* devices, respectively. Bridge nodes are used to connect piconets by sharing their time among them. The event when a bridge stops being active in one piconet and becomes active in another one is called *switching*. Inter-connected piconets are called *scatternets*. Scatternets are not well defined in the existing specifications, however many research papers address this aspect of the Bluetooth technology. We present the current state-of-the-art on scatternets later in this section.

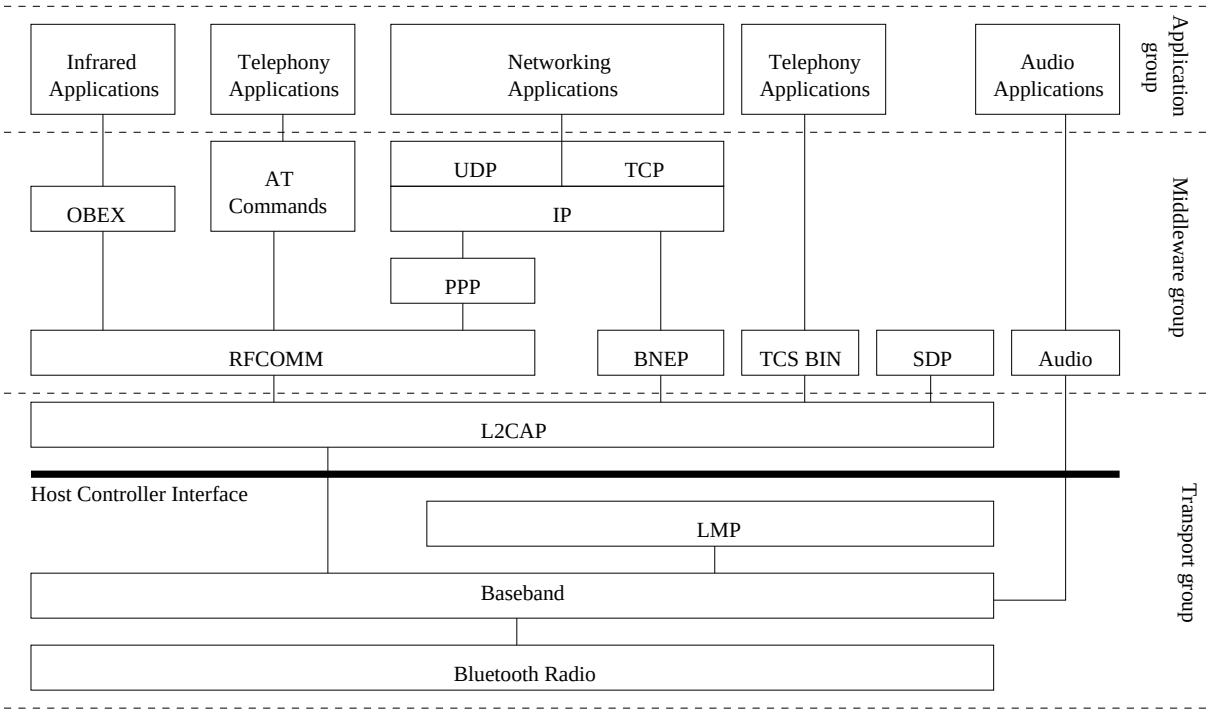

Figure 2.2: Bluetooth protocol stack

2.1.7 Bluetooth Protocol Stack

The bluetooth protocol stack is organized in three main groups (Figure 2.2) [49]:

- Transport protocol group
- Middleware group
- Application group

Transport Group

The transport group contains the protocols developed for carrying the data traffic between Bluetooth devices. These protocols are the Bluetooth Radio, Baseband, Link Manager Protocol (LMP), Logical Link Control and

Adaptation Protocol (L2CAP) and Host Controller Interface (HCI). All Bluetooth devices must implement the protocols of the transport group.

Middleware Group

The middleware protocols provide standard modes for the applications to access the lower layers. Middleware protocols are the Bluetooth Network Encapsulation Protocol (BNEP), RFCOMM, Service Discovery Protocol (SDP), Object Exchange Protocol (OBEX), Telephony Control Specification Protocol(TCS), Audio protocols as well as the networking protocols that provide network access for Bluetooth devices.

Application Group - Profiles

The application group contains the profiles that provide a standard way to use the features of the lower layer protocols and the applications that drive these protocols. These applications are provided by device manufacturers or other software developers.

A group of profiles is defined in the second volume of the Bluetooth specifications [13]. These profiles proved interoperability among the devices from different manufacturers. Further, they define a series of options in the lower layers which enable the applications to manipulate the entire protocol stack.

2.1.8 Security

Bluetooth offers some basic security features for avoiding unauthorized access to information. In the connection establishment phase an authentication process is performed for verifying the identity of one or both of the devices involved in the communication. Authentication is done by means of a 128-bit link key. This key is generated during an authorization process

when the same identification number has to be entered by the users of the two devices that are about to form a secure link. The link key can be shared by all of the devices participating in the same piconet. Link keys are reusable in time between the same devices therefore the identification numbers do not have to be regenerated for every authentication.

Encryption is also present in the Bluetooth technology, however it is an optional feature [49]. The secret key is derived from the authentication link key. This implies that data can be encrypted on authenticated links only.

2.2 Scatternet Formation

2.2.1 Introduction

A *scatternet* is a group of piconets co-located in the same area [12].

The above definition, taken from the Bluetooth Specification 1.1, states that for a scatternet are necessary two or more piconets in the same geographical area. It does not give any information about the connection between these piconets. In spite of this fact, when dealing with scatternets one of the main problems that should be addressed is how to connect two or more piconets in such a manner that the communication between them shows optimal or good performance.

The Bluetooth Specification [14] does not define a protocol for setting up scatternets[1]. It just presents the way communication between piconets can be performed by the means of bridge nodes on a time multiplexing basis as well as some communication capacity sharing technique for these nodes. This is far from being enough to build an efficient scatternet that can support communication between high numbers of Bluetooth units.

[1]Next in this work the term *scatternet* refers to interconnected piconets

In consequence, many researchers focused their attention on the development of an efficient scatternet formation algorithm. This section provides an overview about the existing solutions by assessing relevant papers from the literature.

When working on efficient scatternets we need to know which are the metrics that enable us to evaluate the performance of such a network. We believe that optimal values of the following parameters provide good performance [42]:

- *energy consumption* – lower is better
- *supported traffic* – higher is better
- *duration of scatternet formation* – shorter is better
- *link scheduling* – lower bridge degree is usually better.

First, low energy consumption is a basic requirement for Bluetooth units. A way to obtain lower energy consumption during the network formation is to reduce the number of control messages. After the scatternet formation, energy can be used more efficiently if the number of intermediate nodes for transmitting a packet is lower.

Second, it is easy to see that if a scatternet can support a higher overall throughput the range of applications that can take advantage of it grows proportionally.

Third, the total time needed for scatternet formation will be experienced by the users. Therefore, this delay should be as low as possible.

Finally, if a bridge is connected to a lower number of other devices then it is be able to dedicate more of its communication capacity to each of them.

The approaches for scatternet formation can be categorized according to the shape of the topology they build. In this section we present the

existing solutions for scatternet formation according to these categories, since they have similar performance characteristics.

2.2.2 Early Scatternets

Low power modes can not only be used for saving energy but also for making possible the handling of more than 7 slaves in a piconet. Scheduling the switching between the active and sniff modes of a slave enables the master to control 255 slaves. This could be seen as a Bluetooth communication model one step before the birth of scatternets. This approach was proposed by Kalia et. al. [33] in an early work, where they also present two other approaches for scatternet formation.

It is obvious that such a solution, although theoretically possible, cannot provide high overall throughput, since every message should go through the master, which becomes a bottleneck with a higher number of nodes. Further, since the slaves share the limited communication capacity of the master, they are allocated a very low bandwith and important slots are wasted also for switching as the number of slaves grows.

2.2.3 Tree-shaped Scatternets

Some of the scatternet formation protocols form hierarchical topologies, having the shape of a tree [33, 70, 66, 52]. In these scatternets leaf nodes are slaves, the root node is a master and all other nodes are master&bridges. Although routing is significantly simplified by tree-shaped topologies, there are some important drawbacks derived from the hierarchical structure of such scatternets, as explained next.

First, if one node leaves the scatternet, all nodes situated under it in the tree structure will be automatically cut off from the network. This compromises the reliability of the scatternet.

Second, the traffic between nodes on different branches of the tree structure has to be routed up and down in the hierarchy. This increases the traffic and power consumption and reduces routing efficiency.

Third, if the tree structure has many levels the root as well as some parent nodes can become bottlenecks. This can happen because all of the traffic between the different sub-trees is routed through the parent node. Therefore, in tree-shaped topologies the root node is a bottleneck.

Finally, as already mentioned, all masters, excepting the root node, are bridges and slaves in the same time. This means that when a master&bridge communicates with its own master as a slave, it cannot fulfill its duty of master in its own piconet. This implies that the communication in its piconet is halted for that period.

All these drawbacks suggest that tree-shaped topologies can operate efficiently only in specific scenarios where the above issues do not cause problems. Thus, we conclude that tree-shaped topologies can not be considered for general-purpose scatternets.

2.2.4 Mesh-shaped Scatternets

In [33] Kalia et al., beside the sniff-based and tree-structured approaches, also propose a mesh-shaped solution using shared slaves to connect two piconets. This solution provides better load balancing and is more robust in the presence of link disruptions. Its drawback lies in the routing complexity, since sophisticated routing algorithms are required to find efficient paths between the communicating network nodes. However, their simulation results regarding throughput and average packet delay clearly show that the mesh-shaped model provides the best solution.

After the early solutions of Kalia et. al. other scatternet formation algorithms were proposed, trying to improve the efficiency of the network formation. Thus, Salonidis et al in [59] presented the Bluetooth Topology

Construction Protocol (BTCP), which forms mesh-shaped scatternets in three phases.

In the first phase a coordinator node is elected by the nodes that, in the second phase, will centrally decide about the structure of the scatternet based on the information that it collects from the nodes. The structure is defined by the links that the nodes will set up and by the roles of the nodes. After the coordinator finished "designing" the scatternet structure, it communicates the results to the other nodes that will act accordingly.

An important issue with this approach is that we can not know immediately the moment when the first phase is over, i.e. when the coordinator is elected. This happens because the nodes do not know how many nodes are going to be connected to the network. Therefore, the elected coordinator will have to wait for some time to ensure that no other nodes can be consider for the coordinator election. This waiting time can extend the duration of the scatternet formation time, which is directly sensed by the users. Although this is an important issue, it can not be considered a weakness of BTCP, since any distributed scatternet formation algorithm has to face this lack of information at the nodes. In our opinion the main weakness of the approach relies in the requirement for the nodes to be all in radio proximity. Further, BTCP limits the number of nodes to 36, therefore it is not scalable. The authors explain this shortcoming saying that a higher number of nodes may not require a fully connected scatternet.

Another mesh-shaped scatternet formation algorithm is proposed and evaluated in [42, 41]. In these works the authors introduce the concept of *component* which can be a single node, piconet or scatternet. At the beginning each node forms a component in which it is the *leader*. The leaders start discovering other nodes and connect to them. After the connection of two nodes the one that inquired the other will be the master and maintains its leadership in the new component, while the other node

will be the slave and stops being a leader. When a leader discovers the slave of another leader the slave is transformed into a bridge while one of the leaders will give up its leadership. The leaders continue the neighborhood discovery until only one leader is left. This is the moment when the scatternet formation ends.

The strength of this protocol is that it is robust in the presence of node failures, since the operations used for shaping the components can also be used for healing the network. However, it still requires the nodes to be all in radio proximity.

Simulation results show that the algorithm assures constant energy consumption for a device even if the number of devices increases. The time needed for the scatternet formation is approximately 10s for 16, 11s for 32 and 30s for 64 devices, which are longer than the $5.5s$ of BTCP [59]. However, this approach has no limitation on the maximum number of scatternet nodes.

Protocols that form mesh-shaped scatternets without the requirement for the nodes to be all in range are proposed in [68, 63, 9, 55]. These works provide good approaches to scatternet formation also from the viewpoint of other important metrics. In particular, they are robust and try to keep the average shortest path between two nodes in the scatternet low, the supported traffic high, the number of nodes in a piconet balanced and the number of piconets at an optimal value. Despite the good characteristics we can still identify some of the weaknesses of these protocols. Thus, the protocol in [9] allows more than 7 slaves in a piconet in the *park* low-power mode. This compromises the network performance since masters need to waste their time on parking and unparking their slaves. The problem is solved in [63] and [55], however [63] has the weakness of requiring the nodes to know their geographic position. This implies additional hardware requirements for the nodes. Finally, although BlueNet [68] seems to provide

good performance, the authors compare it only against Bluetrees [70], a protocol that forms tree-shaped scatternets. This does not give enough insight in the performance of BlueNet with respect to other mesh topology forming protocols.

2.3 Scatternet Optimization

After the presentation of the state-of-the-art scatternet formation algorithms, in this section we assess papers focusing on scatternet optimization.

One of the first studies on the performance of Bluetooth scatternets was presented in [48]. The authors observed that the efficiency of a scatternet depends on the carried traffic, number of piconets and node degree, but the most important parameters are the bridging overhead and the number of links. Their simulations show that the higher the number of piconet switches performed by a node the higher the overhead. According to the paper, a high number of links in a scatternet allows higher traffic only at a first stage of the communication, but later these links degrade the overall performance due to the overhead that increases proportionally with the number of links.

In [6] Marsan et al present a method for minimizing the traffic load of the most congested node in a scatternet. This optimization is important for balancing the power consumption of the nodes. Given that congested nodes transmit and receive a high number of packets, their power consumption is high too. The authors achieve their goal by introducing new master and bridge nodes in the scatternet that take over part of the load of bottleneck nodes. The proposed optimization model relies on a topology having a coordinator (or leader) like the one proposed in [59], thus it is a centralized approach. This is the main weakness of their solution. A decentralized,

improved version of this approach, extended also with a technique that enables the incremental formation of feasible scatternet topologies, can be found in [20].

A Bluetooth bridge node participates in at least two piconets, but it can be present only in one of them at the same time. Therefore, it can happen that a master tries to communicate to the bridge when the latter is not active in that master's piconet. In these cases communication bandwidth is wasted, which motivates the scheduling of inter- and intra-piconet communication.

Methods for scheduling based on so-called *checkpoints* are presented in [8] and [9]. A checkpoint is a moment of time on which two connected nodes agree to be active on the same link. In this manner the nodes can know in advance when the other end of each of their links is available for communication. A similar approach has been adopted in the latest version of the Bluetooth specification [14].

Communication in scatternets often has on-demand nature. Links should be kept active only when required by the applications. If lower layers were aware of the particular needs of applications they could handle link activation/deactivation more efficiently. This can be achieved by cross-layer communication. An evaluation of the communication efficiency, in terms of active link time and overhead, with and without cross-layer communication shows that Bluetooth could benefit of cross-layer communication, despite losing the advantages (e.g easy upgrade, code reutilization) of the layered approach [57].

The authors of [10] argue that an important first step in the research of self-organizing networks like Bluetooth scatternets we first should be able to encompass all the possible topologies that can be of interest. They show that the total number of scatternet topologies even with a low number of nodes is extremely large (e.g. with 10 nodes 2^{45} topologies). In their paper

the authors show that by considering some constraints on the ratio of the nodes with different roles and the number of links in the scatternet, this number can be reduced significantly.

In a piconet a slave can directly communicate with the master only. Slave to slave communication is not defined by the current version of the specification [14]. In [11] Bhagwat et al, beside unicast and broadcast intra- and inter-piconet routing algorithms, also suggest a solution to this shortcoming of the Bluetooth technology. The disadvantage of their solution is that it requires the modification of the standard Bluetooth packet header. However, direct slave-to-slave communication would exclude the possibility for the master to become the bottleneck of a piconet.

2.4 Bluetooth Simulators

For evaluating new Bluetooth protocols simulators are required. Although in the relevent literature many Bluetooth simulators are referenced, the majority of them is not publicly available. Finding a Bluetooth simulator with scatternet support is even harder.

One of the earliest Bluetooth simulators is *Bluehoc*, developed by IBM [2]. Bluehoc is a freely available open source simulator. It is based on the Network Simulator 2 (ns-2) [3] and it implements the Bluetooth Radio, Baseband, LMP and L2CAP layers of the protocol stack. Although Bluehoc has useful features it does not support scatternets. Scatternet support has been added to the simulator thanks to the Blueware project of MIT, which reused the source code of Bluehoc extending it with new features. The upgraded simulator has been named *Blueware* and is presented in [64]. Blueware was designed to support tree-shaped scatternets only, in particular it implements the Tree Scatternet Formation (TSF) [66] protocol. As an additional feature, it also supports the Locally Coordinated Scatternet

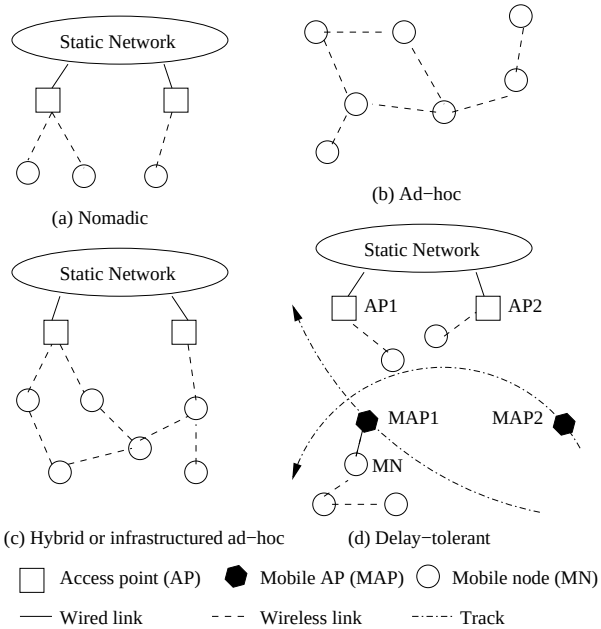

Figure 2.3: Wireless network architectures

Scheduling (LCS) algorithm [65]. Unfortunately it supports static scatternets only and does not support mesh-shaped scatternets. It seems that the development of both Bluehoc and Blueware has been discontinued.

The most promising currently available open source Bluetooth scatternet simulator is UCBT [4] developed at the University of Cincinnati. UCBT has recently been added scatternet support (including mesh-shaped) although only manual scatternet topology formation is possible at the moment. A shortcoming of this effort is that it is not documented yet. Its developers promise the documentation with the version 1.0 of UCBT, currently being at version 0.9.6.

2.5 Delay-Tolerant Networking

With the proliferation of mobile communications, new kinds of network architectures are being defined over the existing static one [46]. The simplest

architecture is called *nomadic* and consists of a traditional static network having access points at its periphery that mobile wireless devices use to access the network, as shown in Figure 2.3.a. On the other hand, the so-called *ad hoc* networks are exclusively formed by mobile components that connect to each other through wireless links, as in Figure 2.3.b.

In a later moment, from the combination of these two basic kinds of mobile wireless network architectures a *hybrid* or *infrastructured ad hoc* network has been defined [44, 50]. The difference between nomadic and hybrid networks is that while in the former case only one-hop connections to the access points are allowed, in the latter architecture multi-hop links toward the access points are also permitted (see Figure 2.3.c).

Due to mobility and to the reduced range of wireless networking technologies, some devices or groups of devices may move away from the main networking area, thus *partitioning* the network in clusters. Devices in a cluster form an isolated network segment, with no access to the resources available in the other parts of the network and to potential communication peers. To make inter-cluster communication possible, two main streams of approaches are available in the literature. First, communication links between two clusters, or a cluster and the Internet, can be set up using specialized gateways with enhanced radio capabilities and/or airborne relay nodes (e.g. airplanes, satellites), like in the systems targeted by [5, 69]. In this architecture devices are organized in hierarchies of two or more layers that take advantage of heterogeneous routing algorithms for managing communication. *Ad hoc network clusters* (i.e. partitions of ad hoc networks) are placed at the lowest layer of the hierarchy. Superior layers may also have the role of *translators*, enabling different communication technologies to be present in the same network. Although this approach provides a straightforward solution for interconnecting ad hoc clusters, airborne or long-range relay nodes may not always be available or they might

be too expensive to use.

In the second approach, mobile nodes that are expected or scheduled to move among isolated clusters physically transport inter-cluster data bundles. Routing in these networks is done on a *store-and-forward* basis, i.e. the data is stored at every node on the route, until another node closer to the destination can overtake it. In these networks an end-to-end path between the source and the destination nodes is not required. This kind of communication paradigm implies both data and device traffic. A communication system following such a model is called a *Delay Tolerant Network* (DTN) [17, 22]. Delay-tolerant network functionality is implemented in the so-called *Bundle Layer* [60], inserted on top of the transport layer in the traditional networking protocol stacks. Such systems are addressed in [19, 67]. Projects like the InterPlaNetary Internet [62], the Saami Network Connectivity [21] and other projects available at [26] build their specific applications on DTNs. An example for delay-tolerant networking is presented in Figure 2.3.d, which shall be fully explained in Chapter 9.

Delay tolerant networks were introduced to solve several problems of traditional networking technologies. One of the main issues to mention here is that current technologies offer poor support, if any at all, for communication between two network elements if no end-to-end path is available. Long propagation delays and high error rates render the proper functioning of the most common networking protocols (in particular IP-based ones) impossible, since frequent retransmissions, the expiration of short time-to-live values, links marked as non-operable, and so on, make them abort all initiated communication channels [22].

The store-and-forward model of DTNs is able to overcome all the above problems, however it is not obvious how to identify the next hop to forward a packet to during the routing process. The authors of [31] recently categorized DTN routing algorithms based on the available knowledge and

confirm through simulations that algorithms taking advantage of a bigger knowledge base outperform the simpler ones. They also provide a sound analytical model for analyzing the traffic in a DTN.

Traditional ad hoc routing schemes based on a general or restricted flooding for route discovery, like [53], AODV [54] or DSR [32], are not usable in DTNs since the long delays will not allow fast connection setup and data transfer between the source and the destination. The set of links of a source node would probably be modified before a route-reply could arrive to the source's route request, rendering route discovery impossible.

Gossiping or its optimized versions like myopic and rumor routing are all based on flooding (independently of its scope's size) [28, 15]. These routing schemes may be useful in DTNs where nothing is known about the behavior of the nodes or when routing is based on some history information that the nodes must collect from each other. However, if some knowledge (like trajectory and schedule in our system) is available about the future behavior of the nodes then more efficient schemes can be developed.

Routing algorithms based on location information like LAR [40] or GPSR [36] are good candidates for being adapted to DTNs since they should not necessarily rely on earlier discovered paths (or links) but on the current location of the destination node. The main problem with these approaches is that they require an end-to-end connection between the traffic sources and destinations. This is an unacceptable requirement for DTNs.

In [71] the authors present an interesting, but very general technique for providing connectivity, using so-called "ferries" (similar to our mobile access points and to Data Mules in [61]), to nodes operating in isolation. In their both proactive and reactive approaches, either the nodes move explicitly to the ferry or the ferry visits sequentially all of the nodes that requested its service through a long range radio. Changing the trajectory of mobile nodes for overtaking data requires either dedicated ferries (i.e.

a mobile infrastructure) or the collaboration of the mobile users. These requirements often cannot be satisfied.

Part I

Bluetooth Scatternet Optimization

Chapter 3

The Scatternet Optimization Problem

The optimization of scatternets based on path length reduction among the communicating peers is the main objective of this thesis. In the following sections we present the main problems that we addressed to achieve this objective.

3.1 General Considerations

As already introduced in Section 1.2.1, mobility and the fact that nodes may change their communication peers in time, generate permanently changing traffic flows in a scatternet. This implies that the routing algorithms continuously search for new routes to meet the changing communication requirements. As we have already seen, in Bluetooth scatternets only those nodes can directly communicate that explicitly set up a physical communication channel (i.e. radio link) between each other. Mere radio proximity is not a sufficient condition to communicate, as it is, for instance, in the case of the 802.11 family of technologies. Further, a higher number of links can only improve the network performance up to a certain threshold, above which the performance decreases [48]. Therefore, the routes that

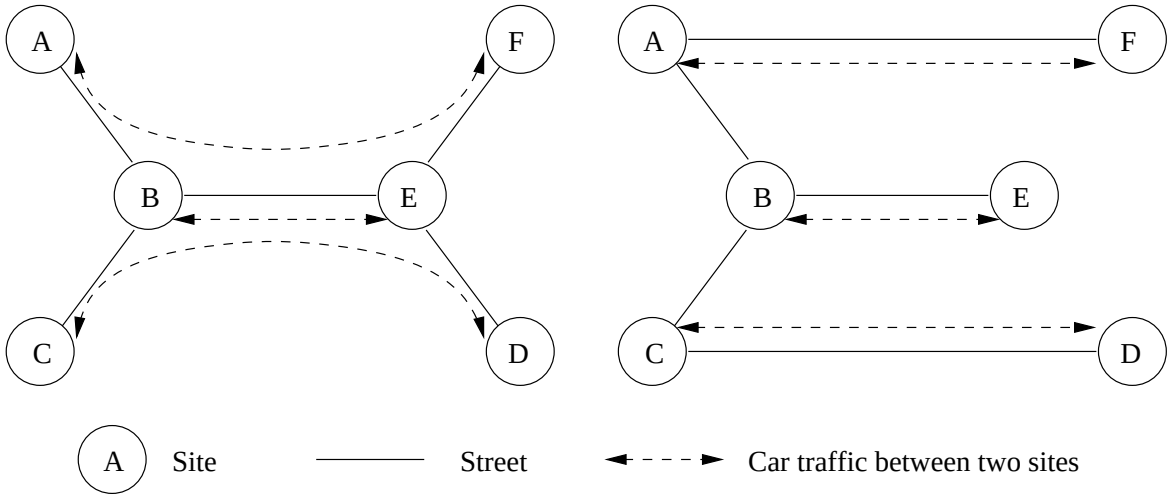

Figure 3.1: Two examples for connecting six sites of a city with five streets

the routing protocols can select are conditioned by that set of links, which currently forms the network topology. Considering the dynamics of nodes and connections in a scatternet, it is clear that these routes will often not be the most efficient ones compared to the set of all possible routes if all possible links would be set up. It follows that the scatternet could benefit of such a topology which has the necessary links for supporting the optimal routes. In consequence, the main objective of this thesis is to define analytical and practical means for periodically finding the optimal topology for the current communication needs of the scatternet nodes.

For a more clear understanding let us consider the example in Figure 3.1. In the figure we show two ways for connecting six sites (A, B, C, D, E, F) on a map of a city, with five streets. We assume that for some reason many inhabitants drive between the sites A and F, B and E, and C and D. From the two figures we can read the followings. On the left side figure the inhabitants drive longer distances and often face traffic congestions between B and E. From this results that they consume more fuel as well. It is obvious that with the street settings as on the right side of the figure,

the inhabitants save time, fuel and avoid congestions.

The situation is similar also in case of computer networks. However, while in wired networks the links (streets) are fixed or in wireless networks based on 802.11x all possible links can be considered as set, in Bluetooth scatternets we have to explicitly select the links to be set up. Therefore, with a particular traffic pattern, in some topologies the transmitted data packets travel more, face congestion more often and consume more of the energy of the forwarding nodes than in some other topologies. In other words, with the same traffic pattern two distinct scatternet topologies provide different overall average throughput, power consumption and routing delays. Consequently, our objective is to find an optimal topology for any given traffic pattern.

Notice that our work does not aim at defining a routing algorithm, but we are searching for a technique for finding scatternet topologies that can support optimal routing. Using the analogy of the car traffic in Figure 3.1, our goal is to build roads, and not to set up traffic lights.

Despite the exceptions, very often the optimal path between two network nodes is the shortest one. Therefore, in our work a shortest path is considered to be optimal. Even if this is not always true, we demonstrate both analytically and through simulations, that by reducing the average path length it is possible to significantly improve the overall throughput and power consumption in the scatternet.

The following subsections describe more specifically the main problems addressed in the first part of this thesis. We can summarize these issues to the following three questions, for which we provide the answers in the subsequent chapters.

- What is the analytical relationship between the hop count and the throughput and power consumption?

- How can we reduce the hop count in a static scatternet?
- What is the impact of periodically optimizing the scatternet topology on the throughput and power consumption, in the presence of mobility and dynamic traffic flows?

Beside the details of the above issues, next in this section we also present our notations and scatternet model, necessary for the formal description of the optimization problem.

3.2 Hop Count and Scatternet Performance

In order to ensure that our intuitive assumptions about the impact of the average hop count on the scatternet performance are correct, we first want to find a relationship between the overall average path length (or hop count) and the total average throughput on all traffic connections of a scatternet. Second, we want to find a similar relationship between the overall average hop count and the total average power consumption.

For our analysis, we identify three fundamental ingredients that are indispensable for calculating the overall throughput in the scatternet.

First, we must take into account the Bluetooth packet types which are used for carrying the data over the Bluetooth asynchronous connectionless links (ACL). As we already know, the specification [14] defines six packet types (DM1, DM3, DM5, DH1, DH3 and DH5) that we must consider. Since these packet types are clearly defined in the specification, we only need to embed them into our analysis.

The situation is more complex with the second ingredient, that is sharing the communication capacity of a node among its links. Although the most recent Bluetooth specification [14] provides additional details on the issue, we still need to provide an analytical deduction of the communication

capacities allocated to each link of a node. In the following chapter we present a detailed analysis on this matter.

Finally, the link quality, that reflects the bit/packet error rate on the wireless channel, needs to be embedded in the analysis as well. On this purpose we use the empirical model of CSR (Cambridge Silicon Radio), the major Bluetooth chip manufacturer.

Using these ingredients, we first want to calculate the overall throughput. As soon as we know the amount of transmitted data, we can focus on the calculation of the overall power consumption. As a secondary objective we also want to find the relation between the overall average throughput and link quality.

3.3 Algorithm Suite for Hop Reduction

The core of our work is the optimization of scatternet topologies by means of hop count reduction along the traffic paths. In this section, after introducing our formal notations and scatternet representation, we formulate the optimization problem that watermarks this thesis.

3.3.1 Notations and Scatternet Representation

Each piconet is composed of a master and up to 7 active slaves. A node participating in more than one piconet is called *bridge*. A node can be a master in only one piconet but it can be a slave in any number of piconets.

Let $\mathcal{N}$ be the *set of nodes* in the scatternet, $\mathcal{M}$ the *set of masters*, and $\mathcal{S}$ the *set of all slaves*. Notice that only pure masters are not elements of $\mathcal{S}$ and $\mathcal{S} \cap \mathcal{M} \neq \emptyset$ if there are master&bridge nodes in the scatternet. We denote with $\mathcal{C}$ the *set of traffic connections* in the scatternet.

$\mathcal{R} = \{r_{ij}^{sd}\}$, the *routing matrix*, stores the path between each source-

Table 3.1: Link matrix properties based on nodes' role

Role of node k	**Property**
Master	$\sum_{j=1}^{N} l_{kj} \geq 1$
Pure slave	$\sum_{i=1}^{N} l_{ik} = 1$
Slave&bridge	$\sum_{i=1}^{N} l_{ik} \geq 2$
Master&bridge	$\sum_{j=1}^{N} l_{kj} \geq 1$ and $\sum_{i=1}^{N} l_{ik} \geq 1$
Free node	$\sum_{j=1}^{N} l_{kj} = 0$ and $\sum_{i=1}^{N} l_{ik} = 0$

destination pair $(s, d) \in \mathcal{C}$; we have

$$r_{ij}^{sd} = \begin{cases} 1 & \text{if connection } (s,d) \text{ is routed on arc } (i,j), \\ 0 & \text{otherwise.} \end{cases}$$

$\mathcal{T} = \{t_{sd}\}$ is the *traffic matrix* with $t_{sd} \in [0, 1]$ indicating the intensity of the data flow on the connection (s, d). $t_{sd} = 0$ means that there is no traffic flow between the nodes s and d.

$\mathcal{H} = \{h_{sd}\}$, the *hop matrix*, contains the minimum number of hops between any connection $(s, d) \in \mathcal{C}$.

$\mathcal{P} = \{p_{ij}\}$ is the *radio proximity matrix* with

$$p_{ij} = \begin{cases} 1 & \text{if nodes } i \text{ and } j \text{ are in-range,} \\ 0 & \text{otherwise.} \end{cases}$$

The *link matrix* $\mathcal{L} = \{l_{ij}\}$ is defined as

$$l_{ij} = \begin{cases} 1 & \text{if } i \text{ is master of } j, \\ 0 & \text{otherwise.} \end{cases}$$

The link matrix indicates the master-slave connections in the scatternet. Link matrix properties are explained below and summarized in Table 3.1.

1) A *master* has on its row one entry equal to 1 for each of its slaves.

2) A *pure slave* has one entry equal to 1 on its column corresponding to its master.

3) A *slave&bridge* has on its column exactly one entry equal to 1 for each of its masters.

4) A *master&bridge* node has one entry equal to 1 for each of its slaves on its row and for each of its masters on its column.

5) A *free node* – node not belonging to any piconet – has all 0s on both its row and column.

The link matrix is a square matrix with as many rows and columns as nodes in the scatternet. We assign to each row and column the scatternet node with the coresponding identifier (we assign identifiers from 1 to $|\mathcal{N}|$).

As an example, next we show the link matrix that encodes the scatternet from Figure 3.2.

$\mathcal{L}$	**1**	**2**	**3**	**4**	**5**	**6**
1	0	0	0	1	0	0
2	1	0	1	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	1	0	1
6	0	0	0	0	0	0

From the matrix we can read the roles of the nodes (based on the formulas in Table 3.1), the master-slave relationship between two nodes, the number of masters and slaves of a node (which implicitly gives the number of links of the node) and the total number of links in the scatternet. For example, node 3 is a pure slave since it has only one entry set on its column and no entries are set on its row. Node 2 is a pure master since it has more than one entry set on its row and no entries set on its column. Node 4

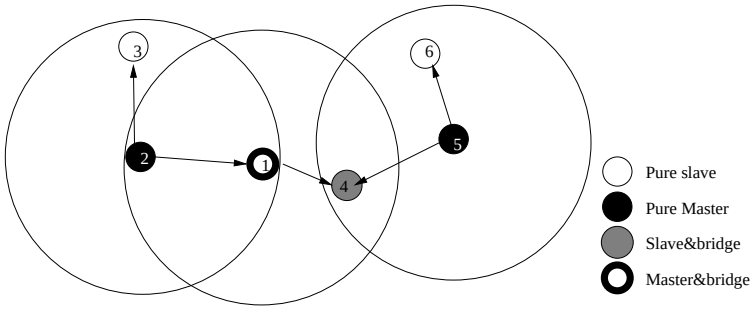

Figure 3.2: Example scatternet with three piconets

has two masters represented by the two 1s on it column and has no slaves. This tells us that 4 is a slave&bridge.

Using the link matrix scatternet representation, link creation/deletion is as simple as switching the appropriate position of the matrix from 0 to 1 and vice-versa. Role modifications happen automatically, as the number of 1s changes on the rows and columns of the nodes.

The above stated properties are very rewarding when modeling scatternet reconfigurations.

3.3.2 Optimization Problem Formulation

Using the notations from the previous subsection now we can define function F as

$$F = \sum_{(s,d)\in\mathcal{C}} t_{sd} h_{sd}. \tag{3.1}$$

F is the sum of the number of hops weighted with the traffic intensity between all source-destination pairs in the scatternet.

Our objective is to solve the following optimization problem,

$$\mathcal{P}: \qquad \min_{H} F \tag{3.2}$$

subject to the following constraints [6].

- A piconet must contain one master and up to 7 slaves.

$$\sum_{j=1}^{N} l_{kj} \leq 7, \forall\, k \in \mathcal{M} \tag{3.3}$$

- There can exist a master-slave relationship between two nodes if and only if they are in radio proximity of each other.

$$l_{ij} \leq p_{ij}, \forall\, i, j \in \mathcal{N} \tag{3.4}$$

- If i is master of j, then j cannot be master of i.

$$l_{ij} + l_{ji} \leq 1, \forall\, i, j \in \mathcal{N}; i \neq j \tag{3.5}$$

- Traffic between source s and destination d can be routed through edge (i, j) only if i and j communicate, i.e. either i is assigned to j, or j is assigned to i.

$$r_{ij}^{sd} \leq l_{ij} + l_{ji} \quad \forall\, (s, d) \in \mathcal{C}, \forall\, i, j \in \mathcal{N} \tag{3.6}$$

- All traffic between two nodes is routed through a minimum length paths, with no loops. The selected path may not necessarily be the same in both directions, if more than one minimum length paths exist.

$$h_{sd} = h_{ds} \quad \forall\, (s, d) \in \mathcal{C} \tag{3.7}$$

$$\sum_{i,j \in \mathcal{N}} r_{ij}^{sd} = h_{sd} \quad \forall\, (s, d) \in \mathcal{C} \tag{3.8}$$

- Standard constraints used for routing should also be considered.

3.4 Evaluation of Dynamic Scatternets

We have in our hands an algorithm suite for reducing the number of hops between communication peers in a static scatternet. Further, we also deduced analytically that hop reduction has positive impact on the overall throughput and power consumption.

The third objective of Part I is the evaluation of the impact of hop reduction on the scatternet performance in the presence of changing traffic flows and mobility, through simulations. Specifically, we want to obtain a confirmation to the previously obtained analytical results through simulations. On this purpose we need to extend our simulator to support mobility and dynamically changing traffic connections, none of them supported in the today's largely available scatternet simulators. Our ultimate goal is to show that the scatternet efficiency, defined as the ratio of the total average throughput and total average power consumption, can be improved by reducing the average hop count in the scatternet. This issue is discussed in Chapter 6.

Chapter 4

Throughput, Power and Path Length Tradeoffs

4.1 Introduction

Bluetooth scatternets with dynamic traffic connections can be found in several application scenarios. Beside the well-known conference-room scenario, we can foresee the use of scatternets in *interfering industrial environments* with machinery that autonomously or semi-autonomously accomplishes its tasks. Components of such an automated environment are static and *mobile* robots, sensors of various type and human supervisors. All these components need to be networked for exchanging the data necessary for accomplishing their tasks. Raw input data required for the tasks, sensor data, progress reports and a whole series of control data are all examples for information that need to be exchanged among the components. Also, each node may have multiple communication peers sustaining *random data traffic sessions* with them, sequentially and/or in parallel.

A data network supporting such a scenario needs to be adaptive for achieving high performance in terms of throughput, power consumption and packet delivery delay. Factors that influence networking predictably in such a scenario, and that in principle can reduce the aggregate system

performance, are mobility, interference and random communication sessions. Bluetooth scatternets are a good candidate for supporting such an ad hoc networking scenario since the technology is robust to interference, given its communication mode based on frequency hopping.

As already presented in Section 3.2, in this chapter we want to determine an analytic relation between the hop count and throughput in a Bluetooth scatternet. On this purpose we need to take into account three fundamental issues: the *Bluetooth packet types*, *link quality* and *link scheduling*.

Bluetooth data communication happens through Asynchronous Connectionless Links (ACL) with six DM and DH packets of different size, with or without FEC (see Section 2.1.3). For calculating the throughput in the scatternet it is important to take into account which of these packets are used.

To calculate the throughput, we also need to model the quality of the radio channel (i.e. the link quality). The link quality reflects the bit and packet error rates on a certain link. If we know the packet error rate then we can calculate the number of retransmissions on the links. We present the link quality model that we used in 4.2

Link scheduling refers to the allocation of time slots in a bridge node to its piconets. A bridge node can be present in one piconet at a time. Therefore, it has to switch continuously between its piconets for being reachable by each of its masters and to relay traffic efficiently. Since we aim at analyzing the scatternet throughput, link scheduling is indispensable for us. Therefore, in the next section we present an analytical model of the link scheduling algorithm that we used in our work.

Link scheduling is a complex task and since it is not the target of our work, we try to keep it as simple as possible. However, we still obtain a quite complicated analytic interpretation of our link scheduling algorithm, described in Section 4.3.

In the remaining part of this chapter we aim at calculating the overall scatternet throughput and power consumption, based on Bluetooth packet types, link quality and link scheduling. Later, in Section 7.1 we evaluate the obtained relations through simulations, also showing several performance aspects regarding packet types and link quality. We published the solutions in this section in [35, 34]

4.2 Link Quality Model

Link quality represents the bit and packet error rates (BER and PER for short) on the radio channels in the network, hence it is related to the packet success rate (PSR) as well, since this latter is the complement of the packet error rate. To calculate the throughput in the scatternet we use PSR to represent the link quality in our formulas.

PSR can be obtained from the packet error rate, as in (4.1). PER, denoted by r, can be calculated as a function of the bit error rate , using the formulas (4.2) and (4.3), for DH and DM packet types, respectively [18], where s is the size of the packet in bits and b is the BER.

$$q = 1 - r \tag{4.1}$$

$$r = 1 - (1 - b)^s \tag{4.2}$$

$$r = 1 - ((1 - b)^{15} + 15b(1 - b)^{14})^{s/15} \tag{4.3}$$

The BER can be obtained from the link quality (LQ) value with some vendor-specific formula. However, [14] states that link quality values should be normalized to the range [0,255]. On this purpose, in our calculus we use the CSR (Cambridge Silicon Radio) model, given in (4.4).

$$BER = (255 - LQ)/40000, \;\; 215 \leq LQ \leq 255$$

$$BER = 32 \cdot (255 - LQ)/40000, \quad 105 < LQ \leq 215 \tag{4.4}$$
$$BER = 256 \cdot (255 - LQ)/40000, \quad 0 \leq LQ \leq 105$$

We use the link quality model presented in this section to calculate the scatternet throughput, in Section 4.4.

4.3 Sharing the Communication Capacity

In our approach each piconet is assigned an overall traffic capacity of 1. (Hence, the traffic rate of a pure master is equal to 1, too.) This capacity is divided among the slaves of that master according to the expressions (4.5)–(4.14), making distinction between the piconet of a pure master and a master&bridge, respectively. For a pure master we simply have:

$$p(pm) = 1, \tag{4.5}$$

where with $p(pm)$ we denoted the communication capacity allocated to the piconet of pure master pm.

Since master&bridge nodes have to switch among different piconets, we assume that each piconet switching takes two slots, 625μs each. We denote the communication capacity of a node wasted for one piconet switching by σ. On average, a node spends in each of its piconets about 40ms, as proposed in [56]. The capacity dedicated to the piconets of a master&bridge node mb is obtained using (4.6).

$$p(mb) = \frac{1}{NrM(mb) + 1} - \sigma, \tag{4.6}$$

where $p(mb)$ is the communication capacity allocated to the masters of a master&bridge mb as well as its own piconet; $NrM(mb) + 1$ is the total number of mb's piconets. Note that the fact that mb is a master&bridge implies that (4.6) is applicable only when $NrM(mb) \geq 1$.

Next we define a scheme for sharing the available capacity among the nodes of a piconet. A simple scheme is to allocate the same amount of bandwidth to each slave of a piconet. The problem with this simplistic approach is that it allocates the same amount of bandwidth also for bridge nodes that can dedicate less of their communication capacity to a particular master since they have to be present also in other piconets. Thus, bandwidth would be allocated to nodes that can not take advantage of it.

To fix the above problem, we define the *available communication capacity*, β, as the capacity that a node can allocate from its whole communication capacity for a piconet P, and the *allocated bandwidth*, γ, as the portion of P's communication capacity that can be dedicated to the node. Then, we denote by α the *availability factor of a node with respect to a piconet* (hereinafter simply availability factor), i.e., the ratio of the allocated bandwidth to the available communication capacity of the node: $\alpha = \gamma/\beta$.

Taking advantage of the availability factor definition, we observe the following properties of a node. A node is said to be *underloaded* with respect to a particular piconet if it can dedicate more bandwidth to that piconet than the amount of bandwidth that the piconet can allocate to the node, i.e., $\alpha < 1$. Clearly, if a node is not underloaded then $\alpha \geq 1$. Therefore in the following we define a node as *overloaded* if either $\alpha = 1$ or $\alpha > 1$.

By using the above notions, next we present the availability factor computation for nodes with different role. We notice that in the formulas defined for this calculus it is not necessary to explicitly consider the slots used for switching, since at this phase it is only important that these slots are busy and is insignificant to emphasize that they are not used for communication.

- *Pure slave (ps)*: γ is the fraction of bandwidth allocated by the piconet master to *ps* out of the piconet's total capacity of 1. β is equal to 1

since *ps* dedicates its whole bandwidth to its master. Thus, for pure slaves $\alpha \leq 1$ for any possible value of γ and β. Therefore we can state that pure slaves are always underloaded. (Notice that $\alpha = 1$ only when the scatternet consists of merely one piconet made of two nodes: a pure master and its pure slave – this case is insignificant from the viewpoint of scatternets.)

- *Pure master (pm)*: since the pure master manages the entire piconet bandwidth, $\gamma = 1$. Similarly, a pure master dedicates all of its bandwidth to its piconet, therefore $\beta = 1$ as well. Thus, for pure masters we always have $\alpha = 1$ i.e., these nodes are always overloaded.

- *Slave&bridge (sb)*: a *sb* is independently allocated a certain bandwidth γ in each piconet it belongs to. On the other hand, a *sb* shares its own capacity among all of its masters. Initially each slave&bridge shares its capacity uniformly among its masters. Therefore, each piconet is allocated a portion of $\beta = 1/NrM(sb)$ from the total capacity of 1 of the *sb*. Thus the availability factor is $\alpha = \gamma \cdot NrM(s)$, where $\gamma \in (0, 1]$ decreases with the increasing number of nodes in the reference piconet. In this case, α may be either smaller or greater than 1. Therefore, slave&bridges may be both, underloaded or overloaded.

  For example, if in a piconet P the slave&bridge *sb* is allocated a bandwidth $\gamma = 0.2$ (which may be the case when there are 5 slaves in P) and *sb* is a slave in three piconets, i.e., $NrM(sb) = 3$, then $\alpha = 0.2/0.3 = 0.66$. In this case *sb* is underloaded with respect to P because it can dedicate more bandwidth to P than P can allocate for *sb*. However, if there were only 2 slaves in P, we would have $\gamma = 0.5$ and hence $\alpha = 0.5/0.3 = 1.66$. In this latter case *sb* is overloaded because P dedicates more bandwidth to it than it can handle.

- *Master&bridge (mb)*: a *mb* shares its bandwidth uniformly among

its masters and its piconet, hence it dedicates to its own piconet a portion of $\beta = 1/(NrM(mb)+1)$. On the other hand, mb manages the whole bandwidth available for its piconet, thus $\gamma = 1/(NrM(mb)+1)$. Therefore, the load factor of a master&bridge with respect to its own piconet is $\alpha = 1$.

Notice that the availability factor of an mb with respect to the piconets of its masters can be calculated similar to the slave&bridge case. The mb is allocated a certain γ by each piconet master while it is able to dedicate at most $\beta = 1/(NrM(mb) + 1)$ of its own bandwidth to each of its masters. Thus the expression of the availability factor is $\alpha = \gamma \cdot (NrM(mb) + 1)$

Thus we can write that the number of underloaded nodes in the piconet of any master m ($NrUN(m)$) is the sum of the pure slaves ($NrPS(m)$) and the underloaded bridges ($NrUB(m)$) (4.7). Then we can calculate the number of overloaded slaves ($NrOS(m)$) through (4.8), where $NrS(m)$ is the number of slaves of m.

$$NrUN(m) = NrPS(m) + NrUB(m) \tag{4.7}$$

$$NrOS(m) = NrS(m) - NrUN(m) \tag{4.8}$$

Once we have computed the number of underloaded and overloaded nodes in a piconet, we can define the link capacities (l) in each piconet as follows.

For overloaded links ($\alpha \geq 1$) from masters to slave&bridges we have:

$$l^o_{sb} = \frac{1}{NrM(sb)} - \sigma \tag{4.9}$$

For overloaded links ($\alpha \geq 1$) from masters to master&bridges we have the same expression as in (4.6), since master&bridges allocate the same

communication capacity for both their masters and piconet:

$$l^o_{mb} = p(mb) = \frac{1}{NrM(mb)+1} - \sigma \tag{4.10}$$

The capacity of a pure master or master&bridge m that is not used by overloaded links is uniformly shared among the underloaded links in its piconet, similarly to the max-min fair technique [45, 30]. For each such master m, the obtained coefficients are stored in a vector $\rho^m = \{\rho^m_i | i = \overline{0, NrUN(m)}\}$. The fraction of the unallocated capacity that is not used by the links is stored in ρ^m_0. Note that if the unallocated capacity can be fully redistributed among the links then $\rho^m_0 = 0$. Equation (4.11) captures the redistributed capacity of an underloaded link, connecting any type of master m to any type of slave s.

$$l^u_s(m) = (p(m) - \sum_{i=1}^{NrOS(m)} l^o_i) \cdot \rho^m_s, \tag{4.11}$$

where $\sum_{i=1}^{NrOS(m)} l^o_i$ gives the total bandwidth allocated for all overloaded slaves of master m. Notice that $p(m)$ should be expressed as in (4.5) or (4.6) for pure masters or master&bridges, respectively. In (4.11) we subtract from the total communication capacity of the piconet the bandwidth allocated for the overloaded nodes (obtaining the total unallocated capacity of m), then we multiply it by the fraction corresponding to the underloaded link connecting master m to its slave s.

Before terminating the capacity allocation, each node compares its own communication capacity of 1 to the total amount of bandwidth received from other nodes. If the received bandwidth is smaller than 1 then the node has some unallocated capacity. Each node having unallocated capacity tries to allocate it to its neighbors. For each node n, these redistributed capacity fractions (i.e. δ^n_i) are stored in the vector $\delta^n = \{\delta^n_i | i = \overline{0, NrN(n)}\}$ where $NrN(n)$ is the number of neighbors (denoted by the index i) of node

n with unallocated capacities. After several iterations of this latter phase all nodes will have allocated as much as possible from their capacities (stored in the vectors δ^n). The corresponding updated formulas for (4.9)–(4.11) are (4.12)–(4.14), respectively.

$$l^o_{sb} = \frac{1}{NrM(sb)} - \sigma + \delta^n_{sb} \tag{4.12}$$

$$l^o_{mb} = p(mb) = \frac{1}{NrM(mb)+1} - \sigma + \delta^n_{mb} \tag{4.13}$$

$$l^u_{ps}(m) = (p(m) - \Sigma_{i=1}^{NrOS(m)} l^o_i) \cdot \rho^m_{ps} + \delta^n_{ps} \tag{4.14}$$

4.4 Throughput Estimation

For our calculus we consider as input variable the total *number of hops* between all communication peers in the scatternet. The outputs of interest are the overall scatternet *throughput* and *power consumption.*

Let $\mathcal{N}$ be the set of nodes, $\mathcal{L}$ the *set of all radio links*, $\mathcal{C}$ the *set of all traffic connections* in the scatternet and h_{sd} the minimum hop count between an $(s, d) \in \mathcal{C}$ source-destination communication pair.

Based on the results in Section 4.3, we can calculate the maximum usable bandwidth, c_{ij}, of a radio link $(i, j) \in \mathcal{L}$, as follows:

$$c_{ij} = \begin{cases} l^o_{sb}, & if \alpha_{ij} \geq 1,\ i \text{ is master, } j \text{ is slave\&bridge,} \\ l^o_{mb}, & if \alpha_{ij} \geq 1,\ i \text{ is master, } j \text{ is master\&bridge,} \\ l^u_{ps}(m), & if \alpha_{ij} < 1,\ i = m \text{ is master, } j \text{ any slave} \end{cases}$$

where α_{ij} is the availability factor of node j with respect to the piconet of master i.

The maximum bandwidth fraction of a link (c_{ij}) is shared by the traffic connections crossing that specific link as shown in (4.15). In (4.15) we denoted by $(s, d) \supset (i, j)$ all the connections (s, d) crossing link (i, j).

$$c_{ij} = \sum_{(s,d)\supset(i,j)} f_{ij}^{sd} \tag{4.15}$$

We use the max-min fair bandwidth allocation algorithm to compute the portion f_{ij}^{sd} that is allocated to each particular connection (s, d) from the available bandwidth on a link (i, j). Let us denote by $F_{ij} = \{f_{ij}^{sd} | (s, d) \in \mathcal{C}\}$ the vector of bandwidth portions allocated to each connection (s, d) on a link (i, j). We can then express the throughput of an $(s, d) \in \mathcal{C}$ traffic connection as in (4.16).

$$\theta^{sd} = C \cdot \min_{(i,j)\in(s,d)} (f_{ij}^{sd} \cdot q_{ij}) \tag{4.16}$$

where C is the maximum capacity of a Bluetooth radio link, specific for each DH and DM packet type, $\min_{(i,j)\in(s,d)} (f_{ij}^{sd} \cdot q_{ij})$ denotes the smallest usable bandwidth portion on the links of a connection (s, d) (i.e. the bottleneck), while q_{ij} is the packet success rate (PSR) of the link (i, j), representing the link quality in our model, as defined in 4.2.

Finally, the aggregate throughput over all traffic connections (i.e. the throughput of the scatternet) can be calculated as:

$$\theta_a = \sum_{(s,d)\in\mathcal{C}} \theta^{sd} = C \cdot \sum_{(s,d)\in\mathcal{C}} \min_{(i,j)\in(s,d)} (f_{ij}^{sd} \cdot q_{ij}) \tag{4.17}$$

Having obtained the expression of the scatternet throughput, we now have to demonstrate the relation between θ_a and the hop count (h) of the scatternet. Notice that h can be calculated as the sum of bandwidth portion vector elements (i.e. connections) on all links:

$$h = \sum_{(i,j)\in\mathcal{L}} |F_{ij}| \tag{4.18}$$

In (4.18) each unitary hop count reduction implies the decrease by one of exactly one bandwidth portion vector's (F_{ij}) number of elements . This,

on turn, implies that one bandwidth portion of the involved link is released. If the link capacity was not fully utilized before the hop reduction then the network throughput remains unchanged. (However, the power consumption decreases, as we will see later in this section.) Secondly, if the link capacity was fully utilized then after the hop reduction the bandwidth used by the old connection is distributed among the remaining ones. In other words, the bandwidth portions f_{ij}^{sd} increases on the involved link. This implies that all connections having their bottleneck on the link in question are allocated new bandwidth, i.e. the minimum f_{ij}^{sd} value grows. It can be seen in (4.17) that this growth has direct positive impact on the aggregate throughput θ_a. This clearly shows why lower scatternet hop counts can produce higher network throughput.

4.5 Power Consumption Estimation

After introducing the main concepts for calculating the throughput, the power consumption is now easier to compute. We assume that the power consumption when transmitting and receiving data at the full capacity of a radio link is P_t and P_r, respectively. Data is transmitted and received by all nodes along a path, excepting the source from one reception and the destination from one transmission. Therefore, all data bits are transmitted and received as many times as the number of hops along the path. Thus, the power consumption of an $(s, d) \in \mathcal{C}$ traffic connection can be expressed as

$$P^{sd} = (P_t + P_r) \cdot h_{sd} \cdot \min_{(i,j)\in(s,d)} (f_{ij}^{sd}) \tag{4.19}$$

where h_{sd} is the hop count between nodes s and d. Notice that the factor $\min_{(i,j)\in(s,d)} (f_{ij}^{sd})$ in (4.19) adapts the power consumption to the bandwidth of the bottleneck link along the path.

The aggregate power consumption through all connections, P_a, is then given by:

$$P_a = \sum_{(s,d)\in\mathcal{C}} P^{sd} = (P_t + P_r) \cdot \sum_{(s,d)\in\mathcal{C}} h_{sd} \cdot \min_{(i,j)\in(s,d)} (f_{ij}^{sd}) \tag{4.20}$$

The dependence of the power consumption on the hop count is easy to see in this case since h_{sd} appears explicitly in the expressions (4.19) and (4.20).

4.6 Conclusion

In this chapter we presented a method for calculating the throughput and power consumption in a scatternet based on the average number of hops connecting communication peers. For our approach we presented the link quality model and defined an analytical model for link scheduling, both necessary for calculating the throughput in the scatternet. The expressions that we obtained for the scatternet throughput and power consumption confirm the intuitive expectations, that is, the scatternet performance depends on the length of the traffic connections in the network.

For evaluating the presented model, we implemented the throughput and power consumption calculus in C++. The results of this evaluation can be found in Section 7.1.

Chapter 5

Optimization Procedure for Hop Count Reduction

5.1 Introduction

After showing the existence of a strong relationship between the path length and scatternet performance, our objective becomes the definition of an algorithm suite capable of reconfiguring the scatternet such that to reduce the hop count between the communicating nodes. General-purpose scatternets have mesh topology therefore we define these algorithms such that to operate on mesh-shaped topologies.

Path length minimization can be achieved by checking all possible radio links of each node for finding those links that ensure a shorter path to the peers of the nodes. This requires us to make available for the hop reduction algorithms a functional scatternet and a set of source-destination traffic connections. The algorithms then can calculate the value of function F (defined in Section 3.3.2) each time they try to reduce the hop count by setting up a new link. The objective is to find the lowest possible F value for the given connection pattern, after checking all of the possible links of each node.

Modifying the links of a node has an impact also on the role of the

nodes. Therefore, slaves and masters need to be handled separately. On this purpose we need to devise different algorithms that reduce the path length of the connections of a slave and those of a master. In this chapter we provide a detailed presentation of the algorithms that we developed for performing the above task. In other words, we present our solution to the optimization problem defined in Section 3.3.2. We have published these solutions in [39, 38].

5.2 Overview

For finding an optimized network topology we first generate a connected and totally functional scatternet, as detailed in Section 5.4. Based on the processing power, we choose one of the masters, the so-called *optimizer*, to coordinate the optimization procedure. The optimizer collects relevant information about all of the scatternet nodes, such as the identity and role of the nodes and their neighbors, masters and communication peers, and feeds it into the optimization algorithm.

The optimizer uses a local search strategy based on a set of possible changes that can be made on the topology, the so-called *moves*. Moves may lead to piconet formation or merging, or just make slaves move from one piconet to another one. In particular, moves targeting slaves typically increase the number of piconets in the network, while moves targeting masters may merge piconets. Thus, in our optimization procedure we try to reduce the number of hops by first moving slaves and then moving masters. As an example, consider the scatternet shown in Figure 5.1. If there is a high traffic flow between slaves 8 and 12, then the scatternet can be optimized by removing node 8 from master 2 and assigning it to master 1 instead.

For each move, the optimizer calculates the new value of F, the function

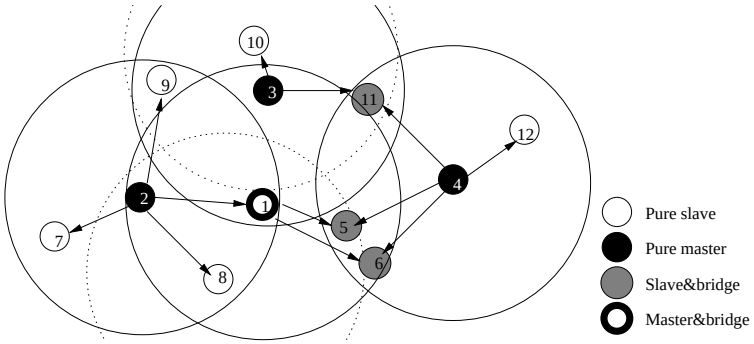

Figure 5.1: Scatternet snapshot

to optimize, as defined in Section 3.3.2. If F decreased after the move then it is stored, otherwise it is dropped. At the end of the optimization the most convenient scatternet configuration, stored during the search, is set.

The optimization algorithm can be executed periodically. We call the time between two executions *optimization period*. Implicit feedback from the scatternet, like the gain of previous optimizations, can be used for dynamically determining the optimization period. Thus, in a scatternet with dynamically changing traffic connections and high node mobility the optimization period will be short, while in quasi-static environments the optimization will be rarely executed. Note, however, that in this chapter we only study single executions of the optimization procedure. The behavior of periodically optimized scatternets is presented in Chapter 6.

5.3 Move Types

A *move* is a set of modifications on the master-slave relationship between nodes in the network. Such modifications are made by link creation, deletion and/or by master-slave role exchange. If, due to these modifications, some nodes get disconnected, the operations necessary to reconnect them to the scatternet are considered as parts of the same move.

We identify four kinds of possible moves:

Slave to Slave (SS) – a slave connects to a different master or establishes a new piconet with a node which then exchanges its role from slave to master. Since moving bridge nodes influences considerably the routing scheme of the scatternet we are not moving bridge nodes but only pure slaves.

Example (based on Figure 5.1): we want to remove slave 8 from master 2 and assign it to master 1. To this end we set $l_{28} = 0$ and $l_{18} = 1$; i.e., first we cancel the link between master 2 and slave 8, then we create the link between master 1 and slave 8.

Slave to Master (SM) – a slave creates a new piconet by paging another node.

Example (based on Figure 5.1): we want to remove slave 8 from master 2, change its role into master and assign slave 5 to it. To this end we set $l_{28} = 0$ then $l_{85} = 1$. This means that we cancel the link between master 2 and slave 8, then we create the link between node 8 and slave 5.

Master to Slave (MS) – a master becomes a pure slave. Such a move is possible only if the slaves of the moving master (i.e. the node giving up its role of master) can be assigned to other nodes in the scatternet. The optimizer takes the abandoned slaves (i.e the slaves of the moving master) one-by-one and assigns them to an already existing master, using SS and SM moves.

Master to Master (MM) – merging two piconets: a master overtakes all the slaves of another master. Such a move can take place when any node in the two piconets is in the range of the persisting master (i.e. the node maintaining its role of master after the move) and the total number of nodes in the two piconets is not greater then 8. This move can be done by removing from $\mathcal{L}$ all the 1s from the row of the master that is about to become a slave and adding them to the corresponding positions in the

row of the persisting master. The old master should be connected to the persisting one through an additional operation. For instance, if node i gives up its role of master and joins the piconet of master j, the additional operation would be $l_{ji} = 1$.

5.4 Scatternet Generation

For our simulations we generate an initial scatternet based on the algorithm proposed by Basagni and Petrioli in [9]. We generate N nodes randomly positioned in the network area. The size of the network area is chosen so that the radio topology results to be connected with high probability.

For each node we generate a random weight that indicates the willingness of the device for assuming special roles (master or bridge) in the scatternet. After all nodes and weights have been generated, we select the *init masters* (the nodes with the biggest weight in their radio proximity [9]). Slaves get connected to one of the init masters in their radio proximity (at less than 10 meters). If a slave is located in the radio proximity of more than one init master, a random choice is made. We believe that parking and unparking slaves (like in [33]) when scatternet formation is possible, is an inconvenient operation since it does not give the possibility to all of the devices to be active at the same time and requires extra operations from the master. Therefore, in the case where more than 7 nodes are in the radio proximity of an init master, only 7 of them become members of its piconet while the others will organize themselves in one or more other piconets. This happens in a subsequent step of the scatternet formation when the masters and their salves are selected in the same manner as in the case of init masters but taking into account also the nodes that have gotten a role already.

Once all nodes have been assigned to a piconet, the algorithm proceeds

with selecting firstly the one-hop bridge nodes between neighboring masters. Only one bridge is placed between two masters and, if possible, a bridge gets connected to two masters only. If there are more than one potential bridge nodes between two masters, we select the one that has the smaller number of masters (possibly 1) and whose physical distance from the two masters is smaller. By doing so we reduce the bridge scheduling overhead and select bridges that receive the strongest signal from the masters.

As the last step of the scatternet formation, we connect all of the two-hop pure masters if they were not already connected through the scatternet. In [9] all two-hop masters are connected. Although this is necessary for ensuring the formation of a connected scatternet, it introduces redundant links between the nodes that consume device resources. Since our work concentrates mainly on scatternet optimization and not formation, we assume that some time after the network formation, nodes abandon these redundant links and maintain the shortest path only to those neighboring nodes that are at least 6 hops away from them. (Notice that the minimum distance between the pure slaves of two two-hop pure masters is 5, hence the value of 6 above.) At this time, a master node may also become a bridge.

If physically possible, at the end of the algorithm we obtain a connected scatternet. The simulation environment generation ends with selecting a predefined number of traffic connections between random source and destination nodes.

5.5 The Optimization Procedure

The *optimization procedure* is the core of our work. It coordinates the various kind of modifications performed on the scatternet topology, aimed

at reducing the number of hops between communicating nodes. The optimization procedure should run on a selected node, possibly with strong computational power, capable of collecting all the necessary data about the scatternet nodes.

The optimization algorithm is presented in Figure 5.2. It consists of a main body from which three different optimization modules, namely SS, SM (either one denoted by SX in Figure 5.2), MS and MM, can be called. At the beginning of the main body several initializations are performed. First the initial state of the link matrix $\mathcal{L}$ is saved (line 2). In line 3 with the function *ActualNrHops()* we retrieve the number of weighted hops between all the source-destination pairs and assign it to F, our function to be minimized. In lines $4-5$ all pure slaves and masters are selected and put in *slavelist* and *masterlist*, used later by the *slave optimization* (i.e. when SS or SM moves are performed) and *master optimization* (i.e. when MS or MM moves are performed) modules, respectively.

In line 6 we cycle through the optimization procedure *nr_diversifications* times. Inside the cycle one of our three optimization modules is executed. Each of these modules takes advantage of a local search [29] based on the corresponding moves for finding a scatternet configuration that reduces the value of F. The operation of the optimization modules as well as the role of the aforementioned cycle is explained in details later in this section.

After the optimization module has been selected and executed, we obtain a scatternet configuration with an F value that we compare with the initial (or previously stored) value of F. If this new value is smaller than the initial one, we have a more optimal scatternet configuration. Therefore, we save this configuration in $\mathcal{L}_{opt}$ and update the value of F (lines $8-10$). Before the next iteration of the **for** loop we set $\mathcal{L}$ to its initial value (line 11). We also update the hop matrix $\mathcal{H}$ since the moves made during the optimization modified it. This is performed by the *HUpdate()*

```
1.  OPTIMIZER
2.  L_init ← L
3.  F ← ActualNrHops()
4.  slavelist ← list of slaves (for SX module)
5.  masterlist ← list of masters (for MS and MM)
6.  for k ← 1 to nr_diversifications do
7.  ⎡ call SX or MS or MM optimization module
8.  ⎢ if F > ActualNrHops() then
9.  ⎢ ⎡ F ← ActualNrHops()
10. ⎢ ⎣ L_opt ← L
11. ⎢ L ← L_init
12. ⎢ HUpdate()
13. ⎢ shuffle slavelist (for SX module)
14. ⎣ shuffle masterlist (for MS and MM module)
15. L ← L_opt
16. HUpdate()
17. end OPTIMIZER

18. SX optimization module
19. for each slave i in slavelist do
20. ⎡ ssbestmove ← SS(slavelist[i])
21. ⎢ smbestmove ← SM(slavelist[i])
22. ⎢ if ssbestmove.localF < smbestmove.localF then
23. ⎢    SSmover(slavelist[i], ssbestmove.bestid)
24. ⎣ else SMmover(slavelist[i], smbestmove.bestid)
25. MS optimization module
26. for each master i in masterlist do
27. ⎡ moves ← MS(masterlist[i])
28. ⎢ if moves is not empty then
29. ⎣    MSmover(moves)
30. MM optimization module
31. for each master i in masterlist do
32. ⎡ mmbestmove ← MM(masterlist[i])
33. ⎢ if mmbestmove.localF < F then
34. ⎣    MMmover(masterlist[i], mmbestmove.bestid)
```

Figure 5.2: Pseudo code of the optimizer

function based on $\mathcal{L}$. Finally, the nodes in *slavelist* or *masterlist* are re-ordered (i.e. a *diversification* is done) and the algorithm proceeds with the next iteration of the **for** loop. After the termination of the **for** loop $\mathcal{L}$ is set to the best configuration found, stored in $\mathcal{L}_{opt}$, and the hop matrix is updated.

Our optimization modules operate as follows. The *SX optimization module* evaluates one-by-one each node from *slavelist* (line 19). The SS and SM algorithms are called one after the other in this module, since they both provide an alternative for moving the same slave. As we will see, this is not the case when moving masters.

Although not explicitly mentioned in the pseudo code, during the search slaves can change their role to master. Therefore, as the **for** loop in line 19 cycles through the slaves in *slavelist*, each node should be checked whether it is still a slave. All the pure slaves are then evaluated using a series of SS and SM moves (lines $20 - 21$) for possible reductions of F. The functions SS() and SM(), implementing this series of moves, get as input the identifier of a slave from *slavelist*. They both return a pair (*bestid*, *localF*), containing the identifier (*bestid*) of the node to which the slave should be moved for obtaining the biggest reduction of F, held by *localF*. These pairs are stored in the variables *ssbestmove* and *smbestmove* for SS() and SM(), respectively. The return value of these two functions can be interpreted as "*the best* F *value of* sxbestmove.nrhops *for this slave can be obtained by connecting it to node* sxbestmove.id". If no move of the slave reduces the value of F, the identifier of its current master is returned accompanied by F's initial value. A detailed description of the SS and SM algorithms can be found in Section 5.6.

The optimization algorithm continues by evaluating the outcome of the SS() and SM() algorithms (lines $22 - 24$). Depending on the hop reduction provided by SS() and SM() the *SSmover()* or the *SMmover()* function is

executed. *SSmover()* and *SMmover()* act on the link matrix $\mathcal{L}$, therefore after their execution the hop matrix should be updated. Indeed, this is what the *HUpdate()* function is used for in line 12.

A secondary task of *SSmover()* and *SMmover()* is to update the variable storing the role of each node (not shown in the pseudo code). We prefer to use such a variable instead of recalculating every time the roles from the link matrix, in order to reduce the execution time.

The two master optimization modules operate in a similar manner, but they are somewhat simpler since the MS and MM moves have separate, dedicated modules. This is required since they do not provide alternative moves for the same master.

The *MS optimization module*, however, differs on a further point from all other modules since the MS() algorithm returns a list of already executed moves instead of one single move that is to be executed later. This is because for transforming a master into slave we need to move all its slaves to some other master (as detailed in Section 5.7). After this series of moves has been executed inside MS() it would be a waste of CPU time resetting them at the end of MS() then setting them again in the frame of the *MSmover()*. Thus, the only task of the *MSmover()* function is to update the variable (not shown) storing the roles of the nodes in the *moves* list. The impact of this choice on the system's modularity is minor.

Thus, we call the MS() algorithm for each master in the master list. If it returns a non-empty list of moves (meaning that a move has been performed) that reduces F, the *MMmover()* function is called for updating the roles of the moved nodes.

In the *MM optimization module* (line 30) the MM() algorithm shows similar behavior to that of the SS() and SM() algorithms. It returns a node identifier (*mmbestmove.bestid*) and the number of weighted hops (*mmbestmove.localF*) that the MM move with that identifier would produce. If

mmbestmove.localF is less than the value of F, it means that a more optimal scatternet configuration has been found.

Returning to the for cycle in line 6, next we explain why is that necessary to repeat the optimization with different ordering of the slaves and masters when we already found a configuration with a lower F. The reason is that even if this configuration has lower number of hops between all its source-destination pairs than the original one, it is high the probability that other, better solutions exist. For example, in case of slave optimizations better solutions can exist due to the fact that during the execution of the algorithm slaves can change their masters. Since pure slaves do not forward data, SS and SM moves directly influence only the number of hops between the moving slave and its destinations. Pure slaves take part only as target nodes in any communication. However, indirectly they can affect also communication links where they do not play neither the role of source nor that of destination. In particular, this is possible when a slave moves to another master and another slave could have used it in a latter iteration to shorten the path to its own destinations.

On the other hand, if two slaves, A and B, communicate they will mutually try to move closer to each other. The number of hops that can be cut off from the route between A and B is not always symmetric. In many cases if we make the hop reduction from the point of view of A, it is not possible to carry out also the reductions from B, because of the route modifications.

We can conclude that the order in which slaves are analyzed by the optimizer is important. Therefore, it is reasonable to suppose that repeating the same procedure (lines $6-14$) but with a different ordering of the slaves in the *slavelist* could produce a better solution. This is what we do in our algorithm. We reset $\mathcal{L}$ to its saved initial value (line 11), update the hop matrix $\mathcal{H}$, generate a random ordering of the slaves (line 13) and re-execute

the search.

The reason why we generate randomly the order of slaves is that we want to examine only a few possibilities from the huge search space. Recalculating the minimum paths between all scatternet nodes would be extremely time-consuming. For example, in a scatternet with only 30 nodes the number of pure slaves is typically 12. All possible permutations of 12 slaves are almost 480 million. Trying all these permutations requires an unacceptably big amount of time. Therefore, we prefer to repeat the search for several times only and choose the best solution found. The importance of the cycle from line 6 is to specify the number of, so-called, *diversifications* of the search trajectory [29]. This means that we randomly reorder the slaves in $slavelist$ $nr\_diversifications$ times. The local minimum found in a diversification is compared against the former optimum stored in F. If the new local minimum is smaller, F is assigned this new value and the link matrix is also saved in $\mathcal{L}_{opt}$ (line $8 - 10$).

The situation is similar also with masters. The central idea is that at every iteration of the optimization algorithm we have to chose only one move from a set of mutually excluding moves. Therefore choosing one move from the set we may eliminate the possibility of performing moves that could produce a lower F value. Thus, repeating the same search trying different moves from the same set of mutually excluding moves raises the probability of finding a more optimal configuration. This is the reason why we repeat the search $nr\_diversifications$ times.

The optimization algorithm can execute the optimization modules sequentially, combining them in different ways. For instance, if we perform the SX, MS and MM optimization modules, we obtain an optimization algorithm that we refer to as SX_MS_MM. The SX module can also be replaced by an SS or SM module giving the so-called SS_MS_MM and SM_MS_MM optimizations, respectively.

Regardless of the optimization modules used, after executing the optimization algorithm a certain number of times, we find a scatternet configuration with fewer hops connecting traffic sources to destinations. Our algorithm can guarantee a global optimal configuration only if each optimization module is called for all possible permutations of nodes in the corresponding *slavelist* and *masterlist*. This would take an unacceptably long period of time. Therefore, a good trade-off between the number of diversifications and execution time should be found for achieving acceptable performance in real environments.

5.6 Reduction of Hops by Moving Slaves

As already mentioned in the previous sections, slave optimization aims at finding the best possible SS or SM move for reducing the number of weighted hops between a slave and all of its communication peers. During the optimization, slaves are assigned one-by-one to each node in their radio proximity and the produced reduction of hops is evaluated. After all of the neighbors of a slave were checked, the move that produced the biggest hop reduction is selected. Next the slave optimization algorithms, SS() and SM(), are detailed. Since these two algorithms are similar we discuss them together, outlining the differences where it is the case.

The slave optimization algorithms start by several initializations (Figure 5.3). *localF* stores the best solution found until the current instant for the current ordering of the slaves (line 2). With function *MyMaster()* the master of the evaluated slave is retrieved and stored in m (line 3). The same initial value is assigned also to the variable *bestid*, which keeps the identifier of the best potential target node.

The cycle starting at line 4 evaluates one-by-one all of the neighbors i of slave *id*. Not all neighbors are used for the optimization. In the lines

```
1.  SS (id)
2.  localF ← ActualNrHops()
3.  bestid, m ← MyMaster(id)
4.  for each neighbor i of slave id do
5.    if h[id][i] > 2 and
6.      (i is not master or (for SM: NA)
7.      (i is master and (for SM: NA)
8.      NrSlaves(i) < 7 )) then (for SM: NA)
9.      l[m][id] ← 0
10.     l[i][id] ← 1 (for SM: l[id][i] ← 1)
11.     if localF > ActualNrHops() then
12.       localF ← ActualNrHops()
13.       bestid ← i
14.     l[m][id] ← 1
15.     l[i][id] ← 0 (for SM: l[id][i] ← 0)
16. return (bestid, localF)
17. end SS
```

Figure 5.3: Pseudo code of the SS and SM algorithms

$5-8$ we impose a series of requirements for the neighbors that are to be evaluated. Thus, the condition in line 5 filters out all neighbors that are in the same piconet in which the evaluated slave, *id*, is. This way we avoid creating piconets inside another piconet. The conditions in lines $6-8$ are used only in the SS algorithm to avoid moving new slaves to a master that already has 7 of them. Therefore, only those neighbors are evaluated, which are either pure slaves, bridges or masters having less than 7 slaves. These constraints are not applicable (NA) for SM moves where the number of slaves that a master has is not important, since after the move the slave *id* creates its own piconet.

Before the execution of an SS move the slave *id* is connected to its master m while the target neighbor, i, is not connected to that same master (recall the condition in line 5). Therefore, the corresponding positions in the link

matrix $\mathcal{L}$ are $l[m][id] = 1$ and $l[i][id] = 0$. The SS and SM moves alter these settings as shown in lines $9 - 10$. In case of an SS move the neighbor i pages slave id, while for SM moves slave id becomes a master and pages neighbor i. Once the move has been executed, its effect on the number of hops is evaluated and, if there is any improvement, the newly calculated number of hops and the identifier of the target node, is stored in *localF* and *bestid*, respectively (lines $11 - 13$). After the evaluation the original links are restored, differentiating again between SS and SM moves (lines $14 - 15$).

Once the evaluation of the whole neighborhood was terminated, the pair (*bestid, localF*), containing the identifier (*bestid*) of the target node that produced the biggest hop reduction (*localF*), is returned (line 16).

5.7 Reduction of Hops by Moving Masters

Master moves are more complicated than slave moves since they may involve more than two nodes. In this section we present MS and MM, the two master moves.

The MS Algorithm

After moving slaves, we try to reduce the number of hops between source and destination nodes by moving also masters.

The MS optimization algorithm (Figure 5.4) gets as input the identifier of a master, *id*, and returns the list of moves that the algorithm was able to identify for reducing the value of F (see (3.1)), after a predefined number of iterations.

The MS algorithm has three main parts. The first part (line $3 - 16$) concerns the reassignment of *id*'s slaves to new masters while the second part (line $17 - 26$) finds the new master of *id*, based on the number of hops between all source-destination pairs in the scatternet. Finally, the last 9

lines re-execute the series of moves that produce the highest hop reduction and return the *movelist* to the optimizer. Next we present the algorithm in details.

In line 2 two variables are initialized: F contains the lowest number of weighted hops found up to the current moment while *initnrhops* is the value of F at the beginning of the algorithm. After the initializations a cycle is started (line 3) for reassigning to new masters all of the slaves of *id*, i.e. pure slaves, bridges and master&bridges. In this cycle the slaves of *id* are analyzed one-by-one. The algorithm handles differently the pure slaves on one hand and bridges and master&bridges on the other hand.

The *SuggestMaster()* function (line 5) finds for each pure slave a master in the slave's radio proximity, that can overtake it from *id*. Further, it finds the master that reduces the most the number of weighted hops between the slave and all of its destinations. If no such a master is found, the function returns the identifier of the slave's old master. However, if the returned value, stored in the variable *sugmast*, is a master's identifier, other than *id*, then the slave is moved to this master (lines $7-8$) and the move is stored in a *movelist*. The elements of *movelist* are made of node pairs indicating which node has been moved to which master.

If there is one single slave that cannot be moved to another master, then it is not possible to perform any optimization with the master *id*, thus the algorithm returns the control to the optimizer (line 13). It would be possible to transform a slave into master, however we do not want to increase the number of masters in the scatternet while performing the MS algorithm. Moreover, it is among the tasks of this algorithm to compensate the increase in the number of masters produced by SS and SM moves. Therefore, the MS algorithm restores the link matrix $\mathcal{L}$ (line 11) based on the *movelist* and returns an empty *movelist* to the optimizer if no MS move can be performed with the master *id*.

```
1.  MS(id)
2.  F, initnrhops ← ActualNrHops()
3.  for each slave j of master id do
4.      if j is a pure slave then
5.          sugmast ← SuggestMaster(j,id)
6.          if sugmast ≠ id then
7.              l[sugmast][j] ← 1
8.              l[id][j] ← 0
9.              append move to movelist
10.         else
11.             Restore(movelist,id)
12.             clear movelist
13.             return movelist
14.     if j is a bridge or master&bridge then
15.         l[id][j] ← 0
16.         append move to movelist
17. for each potential target master i do
18.     if CommonMaster(i,id) = 0 then l[i][id] ← 1
19.     append move to movelist
20.     HUpdate()
21.     if CheckConnectivity(movelist) ≠ 0 and
22.         F > ActualNrHops() then
23.         F ← ActualNrHops()
24.         bestmaster ← i
25.     if CommonMaster(i,id) = 0 then l[i][id] ← 0
26.     remove last move from movelist
27. if initnrhops > F then
28.     if CommonMaster(bestmaster,id) = 0 then
29.         l[bestmaster][id] ← 1
30.     append move to movelist
31. else
32.     Restore(movelist, id)
33.     clear movelist
34. HUpdate()
35. return movelist
```

Figure 5.4: Pseudo code of the MS algorithm

If the role of a slave of *id* is bridge or master&bridge (line 14), the situation is easier because we do not have to look for a master that can overtake it, since these kind of nodes already have at least one other master. Therefore, all the algorithm does is to remove the link of the slave from *id* and store this move in *movelist*. Since bridge and master&bridge nodes can always be moved, we do not have to perform any related verifications.

If the algorithm does not return the control to the optimizer in line 13, in line 17 we proceed with its second phase. In this case we can assume that every slave of *id* has successfully been moved to some other master. Therefore, we start searching for a master that could accommodate also *id* in its piconet. Usually in big and dense scatternets there will be more than one such master, however we want to find the one reducing the most the value of F. The second phase (line $17-26$) concentrates on this issue.

We take one-by-one each potential target master that could accommodate *id*. In line 18, using the *CommonMaster()* function, we check whether *id* and the evaluated master i have a master in common. We do so in order to avoid creating "triangles", i.e. piconets in another piconet. A new link between i and *id* is created only if they do not already have a master in common. Anyway it be, in *movelist* will be recorded (line 19) that node *id* has been moved to node i, even if no modification on the link matrix has been performed. This operation is necessary for the function *CheckConnectivity()* that verifies whether the connectivity of the scatternet was damaged by the moves. If it finds a path between each pair of nodes in *movelist*, it can be stated that the moves have not damaged the connectivity, given that the scatternet was connected at the beginning of the algorithm. Notice that the function *HUpdate()* is called before *CheckConnectivity()*, to recalculate the shortest paths between every node of the scatternet. *HUpdate()* takes the necessary input data from the link matrix $\mathcal{L}$, uses Floyd's algorithm for solving the *all-shortest path problem* [24], and

stores the result in $\mathcal{H}$ (recall from Section 3.1).

Beside the connectivity check the hop reduction obtained by moving *id* to i is also evaluated (line 22). The *ActualNrHops()* function returns the current number of weighted hops in the scatternet, which is compared to F. If the current number of weighted hops is less than the one stored in F, the value of F is updated and so is *besttagetmaster*, the variable containing the identity of the master that produced the lowest number of weighted hops so far.

For any outcome of the connectivity check in line 21, the move of *id* to master i is undone (line $25 - 26$) and the algorithm continues with a subsequent iteration (line 17), evaluating the following potential new master of *id*.

The second phase of the algorithms terminates when the evaluation of all the masters is finished. At this point, in F we have the lowest number of weighted hops found during the search, while *bestmaster* stores the identity of the master that produced this F. Note that all moves performed with the slaves of *id* are stored in *movelist*, but there is no move for *id* itself yet (see lines $25 - 26$). In the third phase, if F is smaller than the initial number of weighted hops (line 27), the best move found is performed again and this move is appended to *movelist* as well (lines $28 - 30$).

If no better solution was found by moving master *id*, all moves performed with *id*'s slaves are undone (lines $32 - 33$).

A final update of $\mathcal{H}$ is done in line 34 then the algorithm terminates by returning *movelist* to the optimizer. If a more optimal position for *id* has been found, *movelist* will contain the corresponding sequence of moves that lead to that configuration. Otherwise an empty list is returned.

The MM Algorithm

Although the MM moves target masters, the structure of the MM algorithm is similar to the SS and SM ones and not MS as one would expect.

```
1.  MM(id)
2.  localF ← ActualNrHops()
3.  tm ← TotNrMasters()
4.  nrs ← NrSlaves(id)
5.  bestid ← id
6.  store all slave links of id
7.  for each neighboring master i of id do
8.    if NrSlaves(i) + nrs ≤ 6 and Collocated(i, id)
9.      store all slave links of i
10.     for each slave j of id do
11.       l[id][j] ← 0
12.       l[i][j] ← 1
13.     l[id][i] ← 0
14.     l[i][id] ← 1
15.     HUpdate()
16.     if localF > ActualNrHops() or
17.       (localF = ActualNrHops() and
18.       tm > TotNrMasters())
19.       bestid ← i
20.       localF ← ActualNrHops()
21.       tm ← TotNrMasters()
22.     restore all links of id and i
23. return (bestid, localF)
```

Figure 5.5: Pseudo code of the MM algorithm

It basically consists of checking every neighboring master of a master *id*, saving their links to their slaves, checking the hop obtained reduction and resetting the original links. Next the MM algorithm is presented in details.

At the beginning of the algorithm we initialize several variables. Thus, *localF* is set to the current number of weighted hops between the communication peers; *tm* will hold the total number of masters in the scatternet using the *TotNrMasters()* function while *nrs* is set to the number of *id*'s slaves. *bestid*, the variable storing the identifier of the master that could

overtake all the nodes in the piconet of *id*, is initialized to *id*. Finally, in line 6 all the links that connect *id* to its slaves are stored.

The main part of the algorithm starts in line 7 with a loop that takes one-by-one every neighboring master i (i.e. masters in radio proximity) of *id* with the goal of piconet fusion. These masters are further filtered in line 8. First, it is verified whether the total number of slaves in the piconets of *id* and i is less than 6. This condition is necessary to observe the specification requirement of 8 nodes in the new piconet. Second, it is also checked (using the *Collocated()* function) whether all of the slaves of *id* are in the range of master i. If such a master is identified, the MM move described in Section 5.3 is executed, as follows. The loop in lines $10 - 12$ moves the slaves of *id* to i then *id* itself is connected to i (lines $13 - 14$), too. (The assignment in line 13 is useful only when a link exists between *id* and i.) After the move the hop matrix $\mathcal{H}$ is updated to enable hop counting (line 15), and the move is checked (lines $16 - 19$) in the following manner. If the number of weighted hops in the scatternet is less than it was before the move or it is the same but the number of masters (i.e. piconets) decreased then the identity, i, of the target master is stored, as well as the number of weighted hops and piconets. In any case, after this verification the original state of the link matrix $\mathcal{L}$ is restored (line 22) and the search for another potential target master is continued.

After checking all of the neighboring masters, the identifier of the target master that could overtake all of the nodes in *id*'s piconet, reducing the most the number of weighted hops (*localF*), is returned.

5.8 Conclusion

In this chapter we presented a centralized method to dynamically adapt a Bluetooth scatternet topology to the actual traffic conditions. We de-

veloped an algorithm suite that reconfigures the nodes' role and links so as to minimize the number of hops between communicating nodes in the scatternet. As part of the algorithm suite, we presented two algorithms for reducing the length of connections belonging to slaves and other two for master connections. These algorithms are based on the four move types (i.e. elementary modules for reconfiguring scatternets) that we defined at the beginning of this chapter. The four algorithms are managed through the optimization algorithm.

For evaluating our solution, we implemented a scatternet simulator in C++. The simulation results can be found in Section 7.2.

Chapter 6

Tracking the Optimal Configuration

6.1 Introduction

In Chapter 5 we presented an algorithm suite that enables us to dynamically find that scatternet topology that provides the shortest paths between the existing communication peers in the network. The main objective of that chapter was to define a tool for reducing the hop count in a given scatternet with static traffic flows, but the behavior of this tool in dynamic scenarios was not considered.

In this chapter we aim at extending this scatternet optimization tool to enable the evaluation of our hop reduction algorithms also in the presence of dynamic traffic connections and node mobility. In other words, instead of considering a single execution of the optimization procedure, we execute it repeatedly as the nodes change their locations and choose new communication peers in time. In this manner we can evaluate, through simulations, the impact of the optimization algorithms on the scatternet performance during a longer period of operation. The main performance metrics that we consider are the overall scatternet throughput and power consumption.

Further, in this chapter we also fix some side-effects of our algorithms that could not be realized with single executions of the optimization procedure. Such side-effects regard the tendency of slave and master oriented

optimizations of slightly increasing the number of masters and links, respectively.

Concluding, in this chapter we improve our optimization procedure to enable the evaluation of dynamic scatternets in time, through simulations. In the upcoming sections we present how dynamic traffic flows and mobility were embedded in the optimization process. Simulation results will follow in a subsequent chapter, in Section 7.3.

6.2 Dynamic Connections

To simulate the changing communication behavior of nodes we implemented a more complex traffic generation scheme than the one in the previous version of the optimizer (presented in Chapter 5). While previously traffic intensity between communication peers was generated in a random fashion preceding the optimization, in the current model the available bandwidth is allocated on a max-min fair basis [45, 30], taking into consideration also the constraints imposed by the master-slave communication model of the Bluetooth technology.

The max-min fair algorithm consists of gradually and uniformly incrementing the bandwidth allocated to each connection until a link gets saturated. At this point all connections using the saturated link are assigned the bandwidth obtained so far (i.e. they get satisfied) and are eliminated from the allocation process. The max-min fair allocation proceeds with incrementing the bandwidth allocated for the remaining connections in the same manner until each connection gets satisfied or the available bandwidth is entirely allocated.

In our model, connections are assigned a life time during which we assume that they communicate on the full bandwidth that has been allocated to each of them. A connection is removed as soon as its associated life time

expires. For generating new connections, two methods were considered. Originally connections were generated according to the Poissonian distribution. Later we switched to a more simple connection generation method which simply replaces the expired connections with new ones. Although the Poissonian method is more realistic, it does not guarantee a constant number of connections during the entire simulation. Since the number of traffic connections in the scatternet has a major impact on the aggregate throughput and power consumption, for obtaining a clear view on the performance improvement caused by our optimization algorithms, we need to keep this number constant. Therefore, in our evaluation we used the latter approach, without loss of generality. By repeatedly replacing expired traffic flows with new ones (i.e. with communication sessions between different end-nodes) we obtain a permanently changing traffic pattern in the scatternet, similar to the traffic flow dynamics in real ad hoc networks. This motivates the periodic exectution of the optimization procedure, which reconfigures the scatternet such that to support more efficiently the newly evolved traffic pattern.

The details of how dynamic traffic connections are embedded in the optimization process can be found in Section 6.4.

6.3 Mobility

The second element that we consider for dynamic scatternets is mobility. In our mobility model, during each time unit of the optimization process the location of mobile nodes may change. Nodes move with walking speed (of about $1.6m/s$) in random directions. Each mobile node moves in a certain time unit with a probability generated at the beginning of the optimization process. This probability expresses the degree of mobility of each node with extreme values 0 and 1 meaning that a node is static or

moving in each unit of time, respectively.

In this work we only study connected networks (i.e. no partitions are allowed), therefore in our mobility management procedure we introduced a protection against position changes that would lead to network disruptions. Naturally, link disruptions are allowed. Investigating on the effects of network disruptions (i.e. partitions) on the throughput and energy consumption might be an interesting topic for future work.

Before changing the position of a node we check whether any connections could be broken due to the increasing distance between the mobile node and its masters/slaves. In case of a broken link we attempt to reattach the node to one of its neighbors. If it is impossible to find a path between the mobile node and the other end node of the broken link, the moving direction is changed or, as a final resort, the node will not change its position in that time unit. After updating the position of each node, new routes are found for the mobility-affected traffic connections.

When we evaluate the scatternet performance without using our hop reduction algorithms, a node, that is disconnected from one of its masters due to mobility, sets up a new link to the first node that it discovers in its neighborhood. On the other hand, when our optimization algorithms are used, the node sets up a new link to that neighbor that reduces the most the length of its traffic connections, taking advantage of our hop reduction algorithms.

The details on how we embedded mobility in the optimization process can be found in Section 6.4.

6.4 Optimization Process

After describing our approach to the dynamic aspects of the scatternets in the previous sections, in the followings we present the operation of a

```
1.  OPTIMIZATION PROCESS
2.  set time_unit, sim_time, T
3.  read node parameters
4.  build topology
5.  read initial connection set
6.  while t < sim_time do
7.      call scatternet optimization procedure
8.      calculate link capacities
9.      perform max-min fair bandwidth allocation
10.     tper ← 0
11.     while tper < T do
12.         calculate throughput and power consumption
13.         forall connections c do
14.             if t ≥ c.expiration_t then
15.                 remove c
16.                 generate new connection
17.         forall nodes n do
18.             update location of n
19.             fix possible link disruptions of n
20.             update role of n and of its neighbors
21.             reset broken connections of n
22.         remove unused links
23.         calculate link capacities
24.         perform max-min fair bandwidth allocation
25.         tper ← tper + time_unit
26.         t ← t + time_unit
27.         if t ≥ sim_time then stop
28. end OPTIMIZATION PROCESS
```

Figure 6.1: Pseudo code of the optimization process

scatternet in time, based on our optimization procedure and in the presence of dynamic traffic flows and node mobility. We refer to this series of scatternet optimization and management operations as the *optimization process* (Figure 6.1).

The optimization process starts out by setting and initializing several important parameters. In particular, we set the *time_unit* between two consecutive evaluations of the scatternet state; the time *sim_time* during which we analyze/simulate the behavior of the network; and the optimization period T, which separates two subsequent optimizations. Further, we read all the a priori generated node-specific parameters (including node position and degree of mobility) and after building an initial topology in line 4 (based on the algorithm in [9] as described in Section 5.4) we also read from a file an initial set of traffic connections, that has been generated in advance.

In line 6 the *optimization loop* starts which also provides the timestamp, denoted by t, for the optimization process. At each iteration of the optimization loop t is incremented by one *time_unit* (line 26) until *sim_time* is reached, moment when the optimization process terminates.

The topology is optimized for the first time at the beginning of the optimization loop (line 7), when the optimization procedure is called (recall Section 5.5). Since the optimization procedure modifies the topology of the scatternet, it should always be followed by the re-calculation of the link capacities (line 8). Link capacities are calculated using the technique presented in Section 4.3 of this thesis. The modified link capacity pattern requires also the re-allocation of the available bandwidth among the connections of the scatternet, hence the max-min fair bandwidth allocation algorithm is performed (line 9), as explained also in Section 6.2.

After the re-allocation of the bandwidth the optimized scatternet is ready for efficient operation. However, the changing traffic patterns and node mobility gradually modify the configuration for which the network was optimized, reducing the efficiency of the scatternet in time. This requires that the optimization procedure be repeated periodically. The *dynamicity management loop*, starting in line 11, is aims at managing the

aforementined dynamic aspects of the scatternet.

The *dynamicity management loop* starts after the execution of an optimization procedure and when it ends the subsequent optimization procedure is called. This loop has its individual timestamp, denoted by *tper*, initialized each time before entering the loop (line 10) and synchronized to t, the timestamp of the optimization loop (lines $25 - 26$). Each optimization period (i.e. the time between two optimizations) is T seconds long.

The purpose of the optimization process is to study the variations of the scatternet throughput and power consumption, as well as other parameters. Therefore, in line 12 we calculate the values of these parameters, which enable us to obtain a clear view on the performance of the scatternet at every instant of the optimization process.

Two *for* loops follow after the performance calculations. In lines $13-16$ the connection expiration times (denoted by *c.expiration_t*) are checked and each expired connection is replaced with a new one, with different (random) end nodes. Each newly created connection is initialized with an end-to-end path, expiration time and is allocated a fair amount of bandwidth.

On the other hand, the second *for* loop (lines $17 - 21$) manages the mobility of nodes. At every iteration of the dynamicity management loop each node is moved to a new location, based on the preset walking speed, the length of the *time_unit* and a random direction (line 18). In case a link is disrupted due to node displacement and the limited Bluetooth radio range, a new link is searched for to reset the connectivity of the node. If such a link is not available, the node relocation is cancelled, as explained in Section 6.3. However, if the node can be reconnected and its new location is acceptable, the role of the nodes that are involved in the new link creation should be updated (line 20). Finally, if a link used by any connection was

disrupted, and hence removed, the affected connections should be repaired as well, by finding new paths between their end nodes (line 21).

Before recalculating the link capacities and reallocating the bandwidth among the new connections, in line 22 a simple mechanism is executed to remove those links from the scatternet that are not used for a predefined time interval by any connections. This mechanism is useful to avoid wasting node CPU time on links that are not used anyway, as well as to counterweight the side-effect of master moves that tend to increase the number of links in the scatternet.

Another side-effect of our optimizations is that slave moves tend to generate new masters. This is not a problem when the optimization procedure is executed only once in a scatternet, because slave moves produce only a couple of new masters. However, when the optimization procedure is executed periodically, slave moves may gradually transform all nodes into masters, which is not desirable. Although not explicitly shown in the pseudo code of Figure 6.1, we counterweight this shortcoming by executing slave moves followed by master moves. As explained in Section 7.2, master moves reduce the number of masters because they tend to merge piconets.

The optimization process alternates between scatternet optimizations and dynamicity management periods until *sim_time* expires.

6.5 Conclusion

In this chapter we extended our optimization procedure to enable the evaluation of scatternets in time and in the presence of dynamic traffic flows and node mobility. The resulting optimization process provides a systematic approach to scatternet optimization not only right after the network formation, but also later when the scatternet performance may degrade because of the unpredictable behavior of the users. Finally, the optimiza-

tion process also comprises some solutions to the side-effects of our hop reduction algorithms.

Chapter 7

Experimental Results

After presenting our approach to the problem of scatternet optimization through hop reduction, in this chapter we provide experimental support to the described solution.

7.1 Evaluation of Analytical Results

For evaluating our throughput and power consumption calculus presented in Chapter 4, we implemented our model in C++. We performed experiments with 50 scatternets, each made of 100 randomly positioned nodes with communication range of 10m, on a 66x66m^2 area. On all these scatternets we generate 15 to 50 random bidirectional traffic connections. The number and length of the connections are fixed for each particular experiment. We perform experiments varying the length of connections from 1 to 10 hops as well as modifying the link quality value in the range of $[215, 255]$. The lower bound of 215 corresponds to the maximum bit error rate of 0.1% allowed by the Bluetooth Specification at the distance of 10m with no obstacles. Finally, during our experimentation we set $P_r = 150mW$ and $P_t = 170mW$. The experimental results shown in Figure 7.1–7.3 are averaged over the 50 different scatternets.

In our first experiment (Figure 7.1) we calculate the average throughput

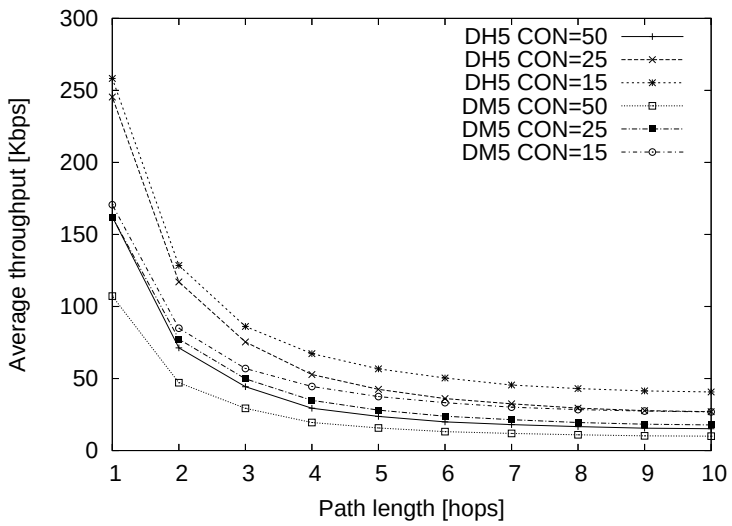

Figure 7.1: Throughput versus connection length

on 15, 25 and 50 bidirectional traffic connections. In this figure we show one of the main objectives of our work, i.e. the throughput decreases with the number of hops. The results show the maximum achievable average throughputs, since we use the two biggest packet types, (i.e. DH5 and DM5) and the link quality is set to 255 (i.e. no packet loss). As we expected, the highest average throughput per connection is achieved with 30 connections, with the DH5 packets, since in this case more bandwidth can be allocated to each connection. The curves then follow each other in the order of number of connections and the packet size.

In our second experiment (Figure 7.2) we show the dependence of the throughput on the link quality. In this experiment the number of bidirectional connections is fixed to 50 and we use DH5 packets only. On the other hand, the connection length is different on each curve. In the figure, shows the expected result: the throughput increases with the link quality. Again, shorter connections are less affected by the link quality, while the longer ones have a very low throughput.

In our third experiment (Figure 7.3) we tested the average power con-

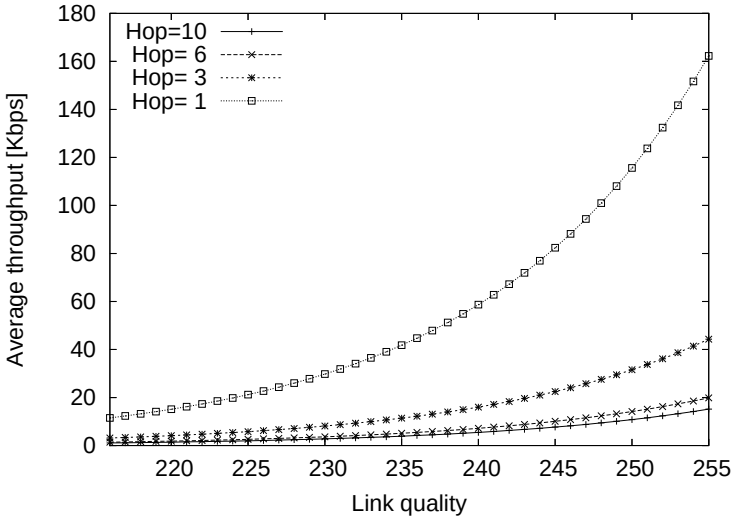

Figure 7.2: Throughput versus link quality

sumption on 15, 25 and 50 bidirectional connections. The packet type in this case has no importance since power is consumed at the same extent by both, useful payload bits and error coding bits.

We can observe in the figure that initially the power consumption decreases, then it starts increasing again. This is explained by the fact that when the connections are short the throughput is high, therefore a higher amount of power is consumed. In other words, power consumption is high because more traffic is transmitted and not because it is less efficiently used. However, after increasing the number of hops and the throughput goes down, the real tendency of the power consumption shows up. It can also be seen that the highest amount of power is consumed when we have 15 connections, since in this case the throughput is higher. This, on turn, makes the power consumption increase faster, as it can be also seen in the figure.

Finally, the power consumption does not depend on the link quality since power is consumed at the same extent for transmitting new packets or repeating the old corrupted ones.

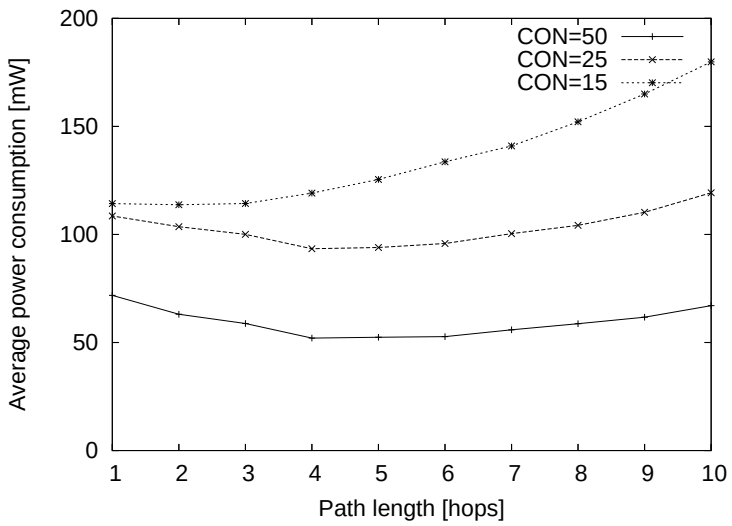

Figure 7.3: Energy efficiency versus connection length

7.2 Hop Count Reduction

To evaluate the performance of our hop reduction algorithms presented in Chapter 5, we implemented a Bluetooth scatternet simulator in C++. Since the algorithms operate on the scatternet topology, in our simulator we mainly considered topology-related aspects like physical links, nodes' roles, radio proximity and multihop connections, omitting protocol stack details and the problem of connection establishment delays.

We tested the optimizer by generating 50 scatternets, of 100 nodes each, over an area of 66x66 m^2. We set the nodes' radio range to 10 m and $nr\_diversifications = 1000$. We randomly generated 100 source-destination bidirectional (200 unidirectional) communication pairs (elements of $\mathcal{C}$). The traffic intensity (t_{sd}) on these connections is 0.1, 0.25 and 0.5 for 50%, 30% and 20% of (s, d) connections, respectively. All the simulations in this section were run on a Linux PC with a 1.7 Ghz CPU and 256 Mb RAM.

We performed experiments combining the SS, SM, MS and MM opti-

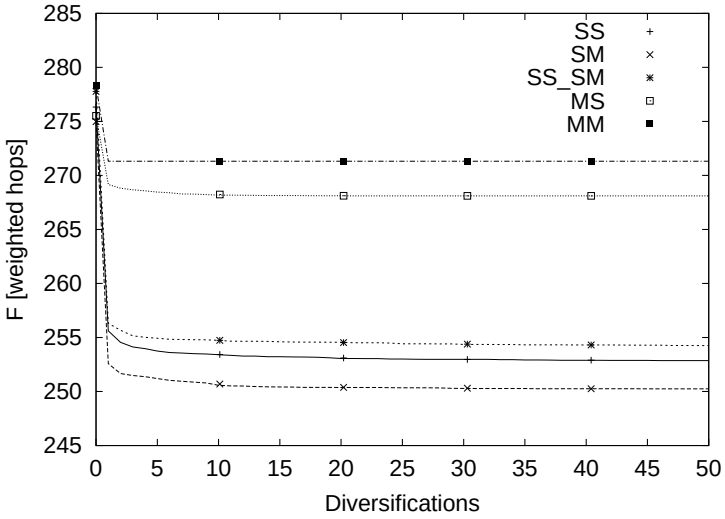

Figure 7.4: Optimization with simple moves

mizations in many different ways. In this section we present results derived from the most illustrative sample optimizations. In particular, we consider three groups of experiments referring to the case where the optimizer is called one, two and three times, respectively, during the same simulation. Each call to the optimizer corresponds to a different optimization module. Figures 7.4-7.6 show the evolution of function F during these experiments, against the number of diversifications.

Then, for each group of experiments, the one giving the best performance is evaluated in more details in Tables 7.1-7.3. In each table, the following metrics are presented. The *Slaves*, *Slave&bridges* and *Total master* parameters report the number of pure slaves, slave&bridges and total number of pure masters and master&bridges, respectively. The parameter *Links* represents the number of links in the scatternet. The *Weighted hops* row shows the overall optimization achieved after each module of the optimization is terminated, while the *Hops* row presents the corresponding hop count. The distance weighted (i.e. multiplied) by the traffic intensity (t_{sd}) is expressed in *weighted hops* ($[wh]$), and the distance is measured in *hops*

Table 7.1: Optimization with SM moves

EXP #1	**Before**	**After**	**Diff.**	**%**
Pure slaves	32.92	14.32	-18.60	-56.50
Slave&bridges	32.48	33.42	0.94	2.89
Total Masters	34.60	52.26	17.66	51.04
Links	121.32	121.32	0.00	0.00
SM optimization [wh]	274.97	250.22	-24.75	-9.08
Hops [h]	1222.08	1121.68	-100.40	-8.26
Weighted hops [wh]	274.97	250.22	-24.75	-9.08

Table 7.2: Optimization with SM_MM moves

EXP #2	**Before**	**After**	**Diff.**	**%**
Slaves	33.34	21.20	-12.14	-36.41
Slave&bridges	32.34	38.02	5.68	17.56
Total Masters	34.32	40.78	6.46	18.82
Links	121.04	131.12	10.08	8.33
SM optimization [wh]	274.48	249.81	-24.67	-9.09
MM optimization [wh]	249.81	234.75	-15.06	-6.14
Hops [h]	1221.88	1054.92	-166.96	-13.76
Weighted hops [wh]	274.48	234.75	-39.73	-14.64

([h]). The rows referring to the SM, MS and/or MM optimizations in the *Before* and *After* columns contain values of the weighted distance in the scatternet configuration exactly before and after that specific phase of the optimization. All other values in these two columns refer to the beginning and the end of the entire optimization procedure. The last two columns indicate the differences between the values in the *Before* and *After* columns, expressed in the appropriate unit (*Diff.*) as well as in percents (*%*).

First, let us consider Figure 7.4 and Table 7.1. In Figure 7.4, for the sake of readability we present only the first 50 diversifications out of 1000 (the curves remain flat from that point onward). The plot shows that

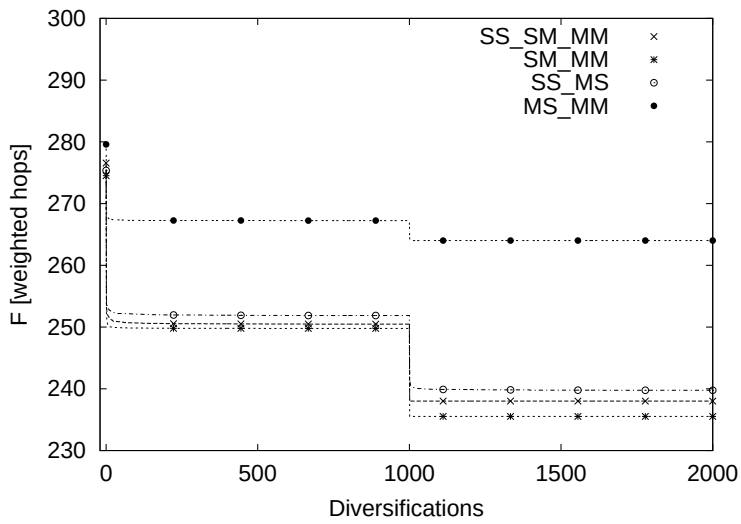

Figure 7.5: Optimization in two phases

the SM optimization produces the greatest weighted hop reduction among the simple moves. Master moves (MS and MM), instead, produce small hop reduction. This confirms that in our initial scatternet masters were selected with care. Table 7.1 presents the results of the SM optimization, i.e the most performant among the simple moves. Observe that the SM optimization gives a weighted hop reduction of 9.08%, at the expense of 51% increase in the number of masters (i.e piconets). In fact, according to their definition, SM moves transform the moving slave into a master creating a new piconet. The number of links instead is unchanged, since SM moves always tear off a link for another.

To improve performance in terms of weighted distance and keep the number of piconets small, we perform also master moves after having moved the slaves (i.e after the 1000th iteration). The results are presented in Figure 7.5. It can be seen that the largest hop reduction (14.64%) is obtained through the SM_MM optimization, whose performance is reported in Table 7.2. Table 7.2 shows that the 14.64% gain in hop reduction corresponds to 18.82% and 8.33% increase in the number of masters and links, respec-

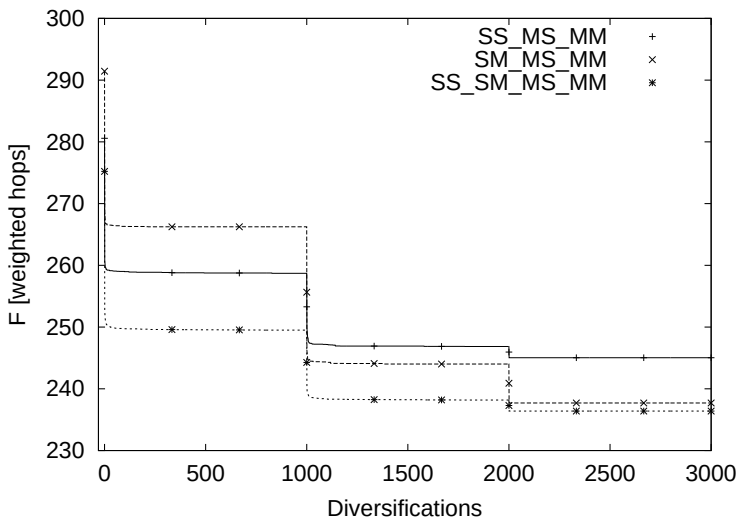

Figure 7.6: Optimization in three phases

tively. We would like to mention that the SS_MS optimization produces a lower hop reduction (12.8%) but it increases the number of masters and links by only 1.49% and 0.54%, respectively. This highlights that master moves counterweight the increase in number of masters produced by slave moves.

Figure 7.6 presents the results of our third experiment, composed of slave optimizations followed by both MS and MM moves (diversifications $1000 \div 1999$ and $2000 \div 3000$, respectively). We obtained the best results with the SM_MS_MM optimization (see Table 7.3). This optimization gives also the best overall performance. However, if we take into account the average optimization execution times, we have: 26.94, 42.95 and 82.43 minutes for SM, SM_MM and SM_MS_MM, respectively. Thus, we can conclude that it is not worth performing both, MS and MM moves for additional 1-2% of hop reduction.

Finally, we highlight that the step-like behavior of function F in all of the three plots in Figures 7.4-7.6 suggests that most of the hop reductions happen at the beginning of each call to the optimizer, namely within the

Table 7.3: Optimization with SM_MS_MM moves

EXP #3	**Before**	**After**	**Diff.**	**%**
Pure slaves	33.02	24.82	-8.20	-24.83
Slave&bridges	32.84	35.68	2.84	8.65
Total masters	34.14	39.50	5.36	15.70
Links	121.42	125.26	3.84	3.16
SM optimization [wh]	278.74	254.47	-24.27	-8.86
MS optimization [wh]	254.47	241.36	-13.11	-4.48
MM optimization [wh]	241.36	235.06	-6.30	-2.67
Hops [h]	1228.24	1047.88	-180.36	-14.34
Weighted hops [wh]	278.74	235.06	-43.68	-15.26

first 10-50 diversifications. Therefore, we can drastically reduce the number of diversifications and, thus the execution times without any significant impact on the overall performance. For example, repeating the SM, SM_MM and SM_MS_MM optimizations with $nr\_diversifications = 10$, the (execution time, reduction) pairs, expressed in [s,%], will be of (15.33, 8.67), (21.71, 13.82) and (42.8, 14.29), respectively.

7.3 Dynamic Scatternets

7.3.1 General Considerations

To evaluate the performance of scatternets also in dynamic environments we embedded in our simulator support for dynamic traffic flows and node mobility. We performed experiments similarly to those presented in the previous sections of this chapter, with 50 scatternets of 100 nodes each, over an area of $66x66m^2$, radio range of $10m$, DH5 packets, link quality of 255 and 50 bidirectional traffic connections. Further, we set the number of diversification $nr\_diversifications = 5$, the simulation time $sim\_time = 3600s$ with $time\_unit = 1s$, duration of connections in the range of $60 -$

$120s$ and optimization period $T = 120$.

After performing the first experiments with this extended simulator we observed several issues that need special attention when the performance of our hop reduction algorithms is evaluated with the optimization process described in Section 6.4.

Slave optimizations work according to the expectations, reducing the hop count between communication peers. However, they tend to transform all pure slaves into masters, as already noticed in Section 6.4. This was not a major problem when we were executing the optimization only once during an experiment (as in Section 7.2). However, with repeated executions slave moves (especially SM) slowly lead to the transformation of all slaves into masters. Therefore, after a number of optimizations there will be no pure slaves left in the scatternet to perform slave moves with. This problem can be counterweighted by using slave moves in conjunction with master moves (especially MS), which reduce the number of masters. Therefore, in the experiments presented in this section we always perform slave and master optimizations together.

MS and MM moves can reduce the number of masters, however in their case it is the number of links that increases gradually. This is again an unacceptable situation since our goal is to reduce the number of hops between communication peers without increasing significantly the number of links. Otherwise we would create unacceptably high overhead with managing the links and finding communication routes. The solution that we used to overcome this side-effect of master moves is the periodic removal of all those links that have not been used by any connections during a predefined time window (e.g. $20s$), provided that this way no node gets isolated or the scatternet partitioned. This reduces back the number of links providing efficiency to the optimization process.

Another issue that we realized after the first experiments with the op-

timization process regards the traffic intensity on the connections. While in the experiments of Section 7.2 we considered the weighted hops (defined as hop count weighted with the traffic intensity, recall also (3.1)) as the key metric for our optimizations, now we shift to relate the network performance with respect to the pure hops. The reason for this choice is that with static scatternets (i.e. static connections and nodes) we fixed the value of the traffic intensity for the entire simulation. Therefore, weighted hop values were changing with the hop count only because the traffic intensity was constant. However, in dynamic scatternets the traffic intensity also changes, in our case with the available bandwidth for each connection. Therefore, after an optimization the hop count is reduced, implying that the bandwidth that can be allocated for the connections increases (as explained in Section 4.4). Thus it may happen that the number of weighted hops increases while the hop count decreases. Since in dynamic scatternets there is no regularity in the variation of the weighted hops, it can not be used for evaluating our hop reduction algorithms, although in static scatternets this metric is usable. However, we still consider the traffic intensity, and hence the weighted hops, when reducing the hop count on the connections in the same manner as we did before. In case when a choice has to be made between two or more mutually excluding moves, that connection path is reduced that supports higher traffic. This way more energy can be saved and more bandwidth is released for other connections on the link that is excluded from the optimized path.

After fixing the shortcomings of the optimization algorithms we evaluate their performance with the optimization process presented in Section 6.4. In the following subsections we present experiments in the presence of dynamic connections with static nodes, with static connections and mobile nodes and finally in the the presence of both of them, respectively. The scatternets with dynamic connections and static nodes are less dynamic

than the other two variations, therefore it is expected that the impact of periodic optimizations will last longer in the former case, hence producing better performance. On the other hand, mobility implies that the scatternet topology changes very frequently and, hence, the optimizations can not produce that high performance improvements. The experiments involving mobility are preliminary only therefore we discuss them briefly, both in Subsection 7.3.3.

7.3.2 Dynamic Connections

In the first experiment that we performed with the optimization process we used dynamic traffic connections and static nodes. Averaged simulation results are shown separately for each kind of optimization in Table 7.4. The most important performance metrics of a sample optimization (SS_MS) are compared to the non-optimized scatternet performance in Figure 7.7.

Table 7.4: Scatternet performance with dynamic connections and static nodes

EXP #1	**Hops**	**Throughp.**	**Power**	**Eff.**	**Gain [%]**	**Masters**	**Links**
No optim.	7.10	18.10	50.14	0.362	–	31.00	101.66
SS_MS	5.08	25.47	43.15	0.591	+63.26	34.46	103.46
SS_MM	5.17	25.16	43.33	0.581	+60.50	35.62	103.98
SM_MS	5.60	24.72	44.42	0.557	+53.87	50.09	100.81
SM_MM	5.16	24.78	43.11	0.575	+58.84	46.47	106.99
SS_MS_MM	5.04	25.12	42.49	0.592	+63.54	33.34	103.61
SM_MS_MM	5.14	25.38	42.58	0.596	+64.64	43.74	104.03

In Table 7.4 the performance of non-optimized scatternets is shown along with the performance of six different types of optimization in terms of hop count, average throughput, average power consumption and energy efficiency. We define the energy efficiency of the scatternet as the ratio of

the throughput and power consumption. The energy efficiency expresses the amount of bits that can be transmitted consuming one energy unit.

As secondary metrics we also show in the table the average number of masters and number of links during the simulations. In the *Gain* column we calculated the efficiency improvement of the different optimizations with respect to the simulations with no optimization. This column clearly shows that all of the six optimization types can provide performance improvements of about 60% with respect to the non-optimized scatternets. We should also notice that the secondary metrics are on the favor of the non-optimized scatternets. The number of masters and links grow during the optimizations for the reasons explained in Section 7.3.1. With optimizations containing SS moves this growth is insignificant. However, master moves can not fully counterweight the master generation tendency of SM moves.

On the other hand, the number of links does not grow significantly. This means that by removing the unused links it is possible to keep the number of links in optimized scatternets close to that in the non-optimized networks.

All of the six optimizations provide roughly the same performance improvements in the mentioned experiments. If we were to select the most performant optimization type, according to the efficiency metric we would choose SM_MS_MM. However, considering that this type of optimization has three phases, while the SS_MS optimization achieves a very close efficiency in two phases only, we should select this latter one. Recall from Section 7.2 that the optimization phases consist of many time-consuming operations. Therefore, it is important to consider that a two-phase optimization can obtain very close performance improvements to the three-phase one, about 30% faster.

Since the scatternet performance with all of the six optimizations is

similar and also for better visibility, in Figure 7.7 we only show the performance metrics of the SS_MS optimization.

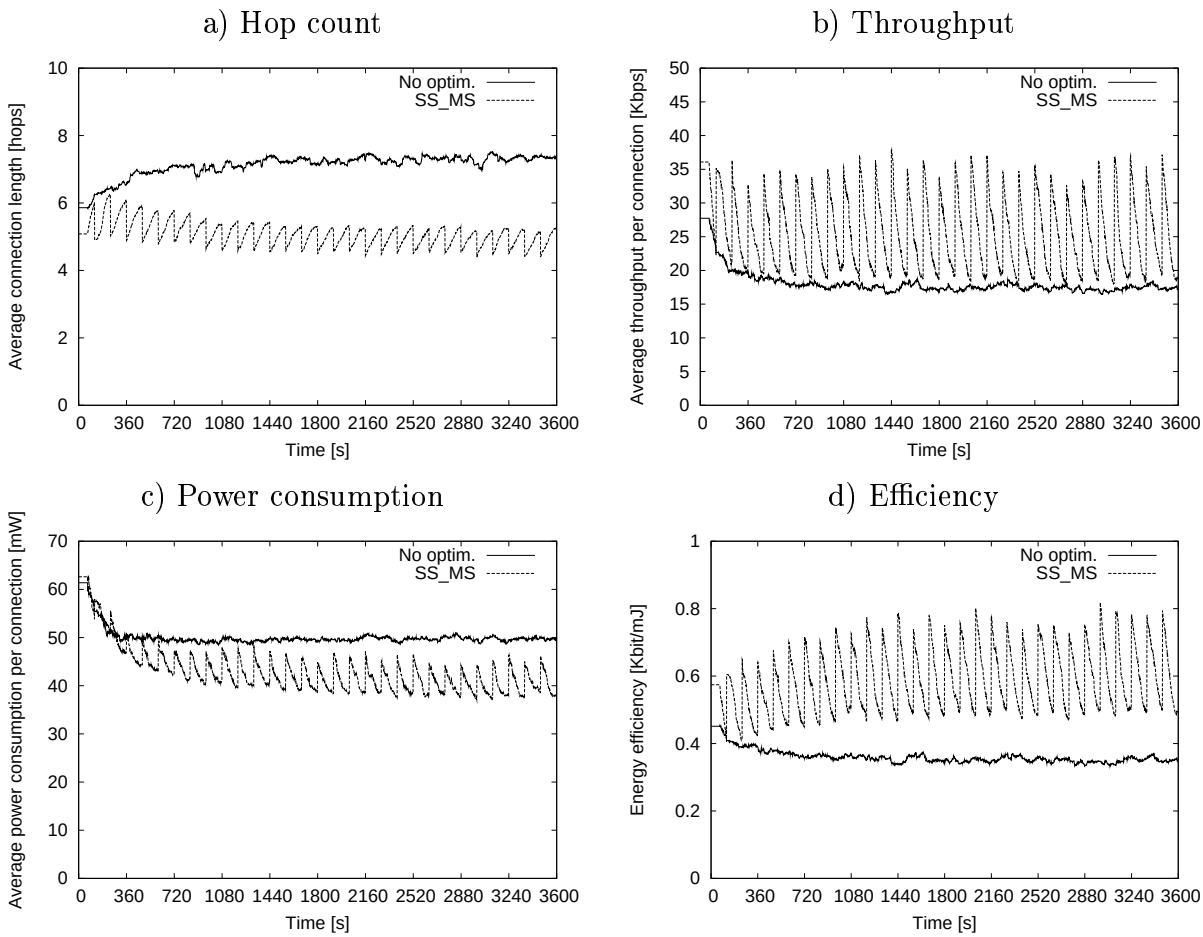

Figure 7.7: Scatternet performance in the presence of dynamic connections with and without optimization; a) Average hop count; b) Average throughput; c) Average power consumption; d) Energy efficiency.

The optimization process starts with the scatternet formation and the selection of the 50 connections. Since the duration of each connections is between $60 - 120s$, at the beginning all of the metrics are constant until the first connection expires. (Notice the short constant segment at the beginning of each curve in Figure 7.7). As the old connections expire and new ones take their places the performance of the non-optimized scatter-

nets degrades. The throughput (Figure 7.7.b) decreases and hence the power consumption (Figure 7.7.c) decreases too. The fact that the power consumption decreases does not mean that the performance of the scatternet improves, because this decrease is caused by the lower number of bits that need to be transmitted, given that the throughput decreased. This is clearly explained by the energy efficiency metric (Figure 7.7.d), which decreases too.

On the other hand, if we now consider the performance of the optimized scatternets (with the SS_MS optimization in this case), we can see that all of the metrics return to a more convenient value each time we execute the optimization. The direct impact of the optimizations can be seen on the evolution of the hop count (Figure 7.7.a), which then is transmitted to the other performance metrics on the basis of the formulas presented in Chapter 4.

In all of the diagrams of Figure 7.7 it can be clearly seen that the performance of the scatternets between two optimizations gradually degrades due to the changing traffic flows. By periodically executing our hop reduction algorithms the network performance can be augmented by about 60%

7.3.3 Dynamic Connections and Mobility

The preliminary experiments with scatternets made of all mobile nodes did not provide that good results as those with dynamic connections. Partially this is motivated by the very rapidly changing scatternet topology, since the 100 mobile nodes cause frequent link disruptions. It can be seen in the *Gain* columns of Table 7.5 and 7.6 that most of the optimizations provide insignificant performance improvements of several percents and some of them perform somewhat worse even than the non-optimized simulation run.

Only the SS_MM and SM_MM optimizations resulted in significant performance improvements, however they achieved this result with a very high number of new links (about the double of the non-optimized case) and masters. This outcome can be explained as follows.

The number of masters in the presence of mobility increases even without optimizations since the mobile nodes often get disconnected from their masters and hence they have to form their own piconet made of two nodes, when there is no other master in their range. Thus, if a slave A gets disrupted from its master and forms a new piconet with bridge node B, this small piconet can be dissolved with MS moves as soon as a master C enters the radio proximity of master A. Note that bridge B does not need be reconnected to some master since it participates in multiple piconets. However, with MM optimizations the slaves of A must all be in the range of the same master C. Therefore, with MM optimizations is harder to reduce the number of masters in the scatternet. If both the MM and MS modules are present in an optimization, the MS module will dominate the MM one because it has looser constraints. This explains the high number of new masters generated by the two optimizations containing only MM out of the two possible master modules.

Table 7.5: Scatternet performance with static connections and mobile nodes

EXP #2	**Hops**	**Throughp.**	**Power**	**Eff.**	**Gain [%]**	**Masters**	**Links**
No optim.	7.76	23.34	58.94	0.396	–	55.48	121.34
SS_MS	8.53	25.20	64.73	0.389	−1.77	58.48	111.02
SS_MM	5.56	31.53	58.01	0.543	+37.12	73.95	207.75
SM_MS	8.73	25.44	65.74	0.387	−2.27	58.10	107.59
SM_MM	5.47	31.79	57.61	0.552	+39.39	74.65	213.09
SS_MS_MM	8.05	25.56	62.99	0.406	+2.52	59.80	118.16
SM_MS_MM	8.02	25.50	62.89	0.406	+2.52	60.19	119.02

Table 7.6: Scatternet performance with dynamic connections and mobile nodes

EXP #3	**Hops**	**Throughp.**	**Power**	**Eff.**	**Gain** [%]	**Masters**	**Links**
No optim.	7.94	22.21	59.29	0.375	–	55.60	121.34
SS_MS	8.62	23.97	64.67	0.371	–1.07	58.46	110.99
SS_MM	5.54	31.33	57.78	0.542	+44.53	74.11	209.10
SM_MS	8.96	23.80	66.06	0.360	–4.00	58.09	107.46
SM_MM	5.46	31.53	57.60	0.547	+45.87	74.42	211.38
SS_MS_MM	8.18	24.46	63.21	0.387	+3.20	59.93	118.38
SM_MS_MM	8.09	24.47	62.59	0.391	+4.27	60.08	119.11

The high number of links originates from the tendency of master optimizations of generating new links. It looks like the most of these newly generated links are used by the traffic connections, because otherwise our mechanism that removes unused links would have eliminated them from the topology. Indeed, the high energy efficiency obtained by the SS_MM and SM_MM optimizations can be explained only with the fact that in mesh topologies with many links shorter paths can be found between the communicating nodes than in scatternets with fewer links.

According to the preliminary experiments of this section, we can state that our hop reduction algorithms could not improve significantly the performance of very dynamic scatternets, however, in such dynamic scenarios the link management overhead becomes so high that communication is compromised anyway. For the future we propose to evaluate our algorithms also in function of the mobility intensity.

7.4 Conclusion

After implementing the analytical formulas that define the relationship between the hop count, throughput and power consumption, experimental

results showed that with a lower number of hops separating communication peers in a scatternet, it is possible to obtain a much higher throughput while power is used more efficiently. Further, with link quality-oriented experiments we demonstrated the dependence of the throughput on the packet loss.

The evaluation of our hop reduction algorithms showed that slave reconfigurations (i.e *moves*) increase the number of piconets, but this can be compensated by master moves, thus obtaining an overall hop reduction between all communication peers of about 15%. However, this may not always hold in very dynamic scenarios.

Our optimization algorithms provided mixed preliminary results in dynamic scenarios. Experiments with dynamic traffic connections and static nodes performed above the expectations, improving the energy efficiency by about 60%. However, our optimizations could not provide significant improvement, with no side-effects, when the nodes were mobile. Thus, for the future we propose to improve our algorithms as well as to evaluate them in function of the degree of node mobility.

Chapter 8

Scatternets Related Work

One of the most important features of the Bluetooth technology is its support for ad hoc networking. The Bluetooth Specification 1.2 [14] defines how to organize Bluetooth-enabled devices in a piconet. Further, it introduces the *scatter mode* for supporting inter-piconet communication. In particular, it describes how time division should be implemented on nodes participating in multiple piconets, earlier discussed also in [56, 7]. However, the problem of efficient scatternet formation as well as many relevant optimization issues, are still not addressed in this latest version of the specification. Part I of this thesis addresses the issue of efficient scatternet topology maintenance in dynamic environments with mobile nodes and randomly changing traffic flows.

As we presented in Section 2.2, many scatternet formation algorithms are available in the literature by now, each building on different assumptions. We believe that the most general algorithms are decentralized and build mesh-shaped topologies without the requirement for the nodes to be all in radio proximity (see also Section 2.2.4).

One of the most interesting approaches for mesh-shaped scatternet formation, at the time when we started this work, is presented in [9]. In this protocol each node is assigned a weight based on the nodes' predisposi-

tion to become a master. In each neighborhood the node with the highest weight is selected to be a master, called *init master* and neighboring nodes are connected to it as slaves. This way, each node is connected to an init master in its neighborhood in a distributed fashion, forming a number of piconets. Later, all these piconets get connected through slave&bridges selected by the init masters and, where necessary, master&bridges are created, forming new piconets.

For all simulations in Chapter 7 we use this algorithm to generate initial scatternets that we perform our optimizations on. In our implementation we make small modifications on the algorithm in order to correct some of its weaknesses and improve its performance.

One of the modifications that we made regards the fact that this algorithm, when there are more than 7 nodes in the neighborhood of an init master, uses low-power modes to accommodate them in the init master's piconet. This is an inefficient solution, as also noted and fixed by the algorithms in [63, 55], since the master will have to waste bandwidth on switching its slaves between active and low-power modes. In our implementation we fix this shortcoming by selecting new masters in such dense neighborhoods, such that to accommodate all of the nodes in a piconet.

Another improvement that we made is that when selecting slave&bridge nodes between two piconets, we chose that node which is closer to both of the masters among all potential bridge nodes, instead of picking one randomly. In this manner the quality of the two links connecting the slave&bridge to its masters will be better and, hence, the inter-piconet traffic will suffer less for packet losses.

Decentralized scatternets impose less constraints on selecting the roles of the nodes and on how to set up links between them. In change, routing and link scheduling becomes more complicated than in the case of tree- and ring-structured topologies, with major impact on the overall performance.

To overcome this shortcoming of mesh-shaped topologies, optimizations are necessary.

Miklós et.al. in [48] are the first to observe that setting up a high number of links in a mesh-shaped scatternet can increase the overall throughput only to a certain limit. Above that limit the link scheduling overhead will lower the overall performance. In our work we consider this finding when during the hop reduction procedure, presented in Sections 5.5, 5.6 and 5.7, we try to keep the number of links unchanged. Since the master optimization modules tend to increase the number of links in the scatternet, we use a simple mechanism for removing the unused links. This way we can keep the number of links almost constant during the intire optimizations.

Finding the topology that best supports the actual traffic flows in a scatternet is not a simple task, since the number of topologies even with a low number of nodes is extremely large. In [10] the authors show that even with 10 nodes the total number of topologies that can be formed is 2^{45}. They also demonstrate that, with constraints on the ratio of the nodes with different roles and on the number of links, this number can be reduced significantly. However, the number of topologies remains still large, especially with many nodes. Therefore, finding an exact solution to our optimization problem formulated in Section 3.3.1 is not feasible, rather a heuristic approach should be used, capable of providing acceptable quality results in a relatively short time.

Local search is a simple combinatorial optimization technique that can be easily adopted to specific optimization problems [29]. The local search heuristic technique does not provide a-priori result quality and computational time guarantees. However it has been shown that with local search heuristics a reasonable solution can be found with the possibility of adapting the computational time to the concrete circumstances of the optimization. In our optimization procedure (Section 5.5) we use a *best-fit*

local search strategy for finding that potential link of each node which reduces the most the number of hops between all communication peers in a Bluetooth scatternet. For achieving better results, we diversify the search trajectory by rearranging the evaluation order of the nodes. We adapt the computation time to our needs by limiting the number of diversifications that we perform.

Since one of our main objectives with scatternet optimization through hop reduction is to increase the overall network throughput we must consider a bandwidth allocation algorithm that will define the amount of traffic on the connections of the scatternet. On this purpose we used the max-min fair bandwidth allocation algorithm that shares the network resources in a fair manner among the users [30]. This algorithm consists of gradually and uniformly incrementing the bandwidth allocated to each connection until a link gets saturated. At this point all the connections using the saturated link are assigned the bandwidth obtained so far (i.e. they get satisfied) and are eliminated from the allocation process. The max-min fair allocation proceeds with incrementing the bandwidth allocated for the remaining connections in the same manner, until each connection gets satisfied or the available bandwidth is entirely allocated.

To shorten the run time of our simulations in Sections 7.2 and 7.3 we use Floyd's algorithm [24] to calculate the shortest paths between all communicating nodes in the scatternet. This algorithm scales with the number of scatternet nodes as $O(n^3)$ and is easy to implement.

The Bluetooth specification [14] defines six packet types for asynchronous data transmission (see Section 2.1.3). In our work, especially in Sections 4.4 and 4.5, we consider all these packet types for calculating the scatternet performance in terms of throughput and power consumption.

To formulate our optimization problem we provide an analytical representation for scatternets in Section 3.3.1. The core of our model is the

link matrix, which encodes the master-slave relationship among the nodes, uniquely defining the entire scatternet topology. A similar representation has been developed simultaneously to our work in [47]. However, that model does not support master&bridge nodes, thus having a limited applicability. As it can be seen in Section 3.3.1, our model does not have this limitation.

For protecting our optimization from creating unfeasible scatternet topologies we defined a series of constraints conform to the Bluetooth specification. We formulated these constraints based on [6], which presents a technique for reducing the load of the most congested node in a scatternet.

In [18] the authors show a technique for modeling link quality and packet loss on Bluetooth radio links. In Section 4.2 of this thesis we adopt that model for the analytical calculation of the scatternet throughput and power consumption. Further, in our simulator we implement the same model, therefore the experiments in Chapter 7 also take advantage of it.

We implemented our optimization algorithms in C++. We opted for implementing a simulator from scratch because at the time when we started our work there were no publicly available Bluetooth simulators. Later, Blueware [2] was developed, however it only supports tree-shaped scatternets and master&slave bridges. Further, it does not support mobility and dynamic traffic connections, both necessary for our work. Adding all these features to the simulator would have required us an unaffordable implementation effort.

Recently UCBT [4], a new open source, ns-2-based scatternet simulator has been released. UCBT supports mesh-shaped scatternets as well as master&bridges. Although support for mobility and dynamic connections is not present also in this simulator, in the future it can be adapted to our goals with reasonable effort.

Both, Blueware and UCBT implement the Bluetooth protocol stack

therefore they are appropriate for specification-conform simulations in the frame defined by their limitations. Since our objective is to simulate scatternets, in our simulator we only implemented scatternet-related aspects of the Bluetooth technology. Although this is a limitation of our tool, it can still provide useful insight in the operation of scatternets.

Part II

Delay-Tolerant Networking

Chapter 9

The DTN Optimization Problem

In the second part of this thesis we focus our attention on the analysis of delay-tolerant networks. Our main interest is the performance of a DTN in an environment with scheduled message forwarder nodes, i.e. nodes with known trajectory and schedule. In such a network a set of mobile access points (MAPs) "ferry" the data back-and-forth between the mobile nodes (i.e. the end-users of the system) and the static infrastructure. We are mainly interested in the delays between the moment of sending and receiving a message from one mobile user to the other and the size of the different buffers that temporarily store the messages along the traffic paths.

Figure 9.1 illustrates how a mobile node MN (e.g a wireless PDA owner walking on the street), within an ad hoc cluster, places a query on a mobile access point MAP1 (installed, for instance, on a bus). After a while, MAP1 moves close to the static access point AP1 (set up in a bus stop), and forwards the message of MN toward the destination, e.g. an e-mail server. After the reception of the message the e-mail server elaborates a response and multicasts it to AP1 and AP2. A short time after, MAP2 (another gateway-equipped bus), passes close to AP2 and takes the message back to the area where MN placed its query on MNP1. When arriving in communication range with MN, MNP2 hands the response data, i.e. several

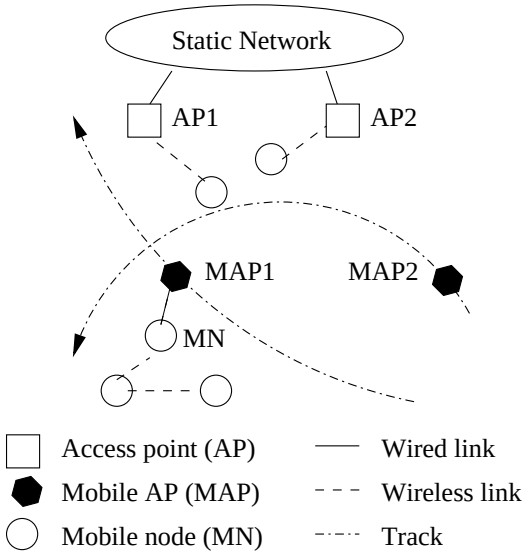

Figure 9.1: The busnet scenario

email messages.

The above scenario can be extended to an urban transportation network where the buses (and similar vehicles) are outfitted with mobile access points (MAPs) capable of relaying data between mobile users and the Internet. Next in this thesis we refer to this extended scenario as *busnet*. Having the support of a busnet users can continue sending emails even after moving away from a hot spot because MAPs collect their messages and take them to the Internet. This way network connectivity can be extended also to places that are not covered by a fixed network infrastructure.

Fixed network nodes may also benefit of such a network connectivity. In a sensor network deployed on the area of operation of a busnet some tasks (for example, periodical network reconfigurations, battery replacement pre-alerts and other maintenance signals) are not time-critical, and some of the related actions are better organized by a centralized server with no power and computational constraints. If the sensor network is large (e.g. thousands of low-cost, low-power gadgets monitoring the civil infrastruc-

ture of a town: pollution, traffic, sewers, bridge and building stresses...) a thorough information-collecting tour would be too expensive, while multi-hop ad hoc communications would be too demanding for batteries. On the contrary, from the sensors' point of view a DTN is a single-hop system and no node would have to spend power in order to relay someone else's information. Clearly, only non-time-critical data can be handled this way.

On the other hand, communication between infostations [25] (aka information kiosks) and vending machines and their management center requires network connectivity. Laying cables or setting up a wireless network connection for each unit is often very expensive. Since many kinds of data can be cached at the infostations, most applications on such machines (e.g. downloading maps, administrative forms, tourist information, news, etc.) do not require always-on network access and hence, a busnet can satisfy their communication needs very well.

A final benefit from a busnet system that we mention here is that its infrastructure can be used for providing on-board services, too. Over the fact that bus travelers would sense very reduced delays while using networking applications, they can be offered services like digital bus ticket purchase. Such services help the investors of such an infrastructure to regain their investment much faster.

The usage of mobile access points with augmented capabilities for collecting and delivering data from/to mobile users can provide network connectivity also in places where the deployment of static access points is unprofitable. Indeed, the advantage of this network architecture is its capability of expanding the coverage of a wireless network on a considerably bigger area than with the same number of static gateways, at the same cost. The price payed for this is the intermittent connectivity of such a network, which implies longer response times sensed by mobile users in user-initiated transactions. The length of these response delays is one of

the issues that we investigate in this work. Since these delays lead to the accumulation of data on both mobile and static gateways, we also want to obtain from our experiments a clear view upon the appropriate size of the buffers on these devices that can assure the proper operation of the system.

The objective of Part II of this thesis is to reduce the buffers on relay nodes of a scheduled DTN with no significant negative impact on the message delivery delays sensed by the users. On this purpose:

- we define `K2`, a novel routing algorithm based on target interlocking, and
- we implement a simulator for an application scenario that enables us to evaluate the `K2` algorithm and obtain numeric estimations about the various delays and buffers present in the DTN.

For the simulations we take advantage of the urban transportation network scenario (i.e. busnet) presented above.

Chapter 10

Delay-Tolerant Networks

10.1 Introduction

As already mentioned, the objective of this thesis regarding delay-tolerant networks, is to evaluate the buffer usage and routing delays in a network where the data of mobile users is transported to the static network by mobile access points (MAPs). Further, we also aim at defining such a routing algorithm that uses some location information history of the destination node to reduce the number of static and mobile access points that receive the packets for routing purposes. This will reduce energy consumption and buffer needs as well as save bandwidth.

We focus our attention on DTNs, such as busnet, described in Chapter 9, more specifically on that network segment of such a DTN which is dedicated to provide the connection between a mobile user and the static infrastructure. In this work we do not consider delays or networking issues such as routing behind the static access points (i.e. in the static network), making the assumption that these times are usually much shorter than those introduced by the MAPs. Our main concern is to move data between the mobile users and the access points[1].

[1]Infrastructure, static network and the access points in the generic sense are used as synonyms in this work, if not specified otherwise.

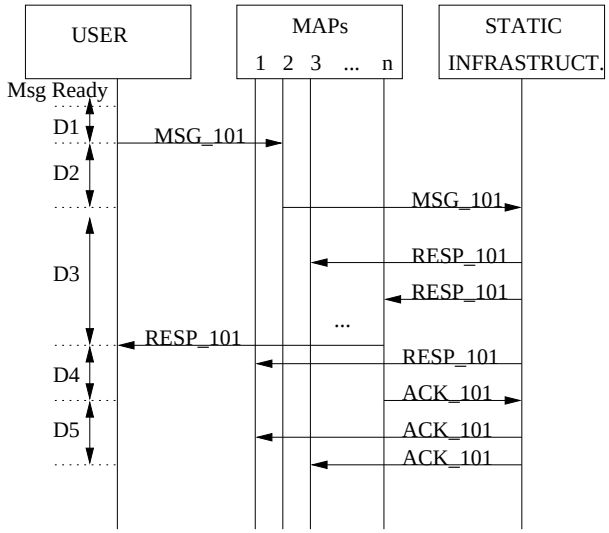

Figure 10.1: Message flow between users and the infrastructure

We analyze the network performance with respect to the routing delay and buffer length in the case of two routing algorithms, *Message Forwarding with Flooding* (`MFF`) and our algorithm based on location information called `K2`. Both of these two algorithms follow the generic communication scheme presented in Figure 10.1, as explained in Section 10.2. `MFF` and `K2` are described in details in Section 10.3 and 10.4, respectively. We published this approach in [37].

10.2 Routing Delays

Let us suppose that a mobile user wants to send a request to a server in the static network (Figure 10.1). After the request has been issued on the user's device, at the first opportunity, i.e. when a MAP arrives in his communication range, the message is sent to the MAP with a unique identifier (`MSG_101`). The MAP holds the message until a static access point appears in its range. At this moment `MSG_101` is forwarded to the infrastructure and a response (`RESP_101`) is generated by the appropriate server. This

response is replicated and forwarded to a set of MAPs that pass next to a static access point until a receipt acknowledgment (`ACK_101`) is received. A receipt acknowledgment is generated by the MAP that actually hands `RESP_101` to the user and is sent to the infrastructure at the first opportunity. The infrastructure informs all the MAPs about the acknowledgment, so that they can remove `MSG_101` from their buffers. After the reception of the acknowledgment the response message is removed from the infrastructure. A Time-To-Live parameter can also be used to eliminate old messages from the buffers of static and mobile access points.

During this message exchange sequence we encounter at least five delay types, indicated in Figure 10.1. The first delay (D1) takes place from the moment of generation of a request by an application until a MAP overtakes it from the user device. During this period the request message is stored in the bundle layer [17] or a corresponding middleware buffer on the user's device. During the second delay (D2) the message is in the MAP input buffer. This delay lasts until the MAP meets the first static access point. At this moment the all messages collected by the MAP are relayed to the static network. The round trip of the message between the static network boundary and the appropriate server, as well as the time necessary for the server to elaborate a response for the request introduce a new delay. Discussion on this delay is not in the scope of this work, and we shall assume that the response is nearly instantaneous when compared to the other delays.

The consequent delay (D3) lays between the moment when the response arrived back from the server to the access points and the one when the user receives the response.

The delays following D3 are not sensed by the user anymore. They are only important from the system's point of view, in particular when we want to know how long the already delivered messages occupy the system

resources. There are two delay periods that need to be mentioned after this moment. During the first one (D4) the infrastructure has not yet been informed that the response was already delivered to the user. Therefore, in this period the infrastructure keeps propagating RESP_101 messages among the MAPs, occupying communication bandwidth and storage needlessly. Finally, in the last delay period (D5) the infrastructure propagates short acknowledgment messages enabling the MAPs to free their buffers. Numeric values of these delays can be found in Chapter 11.

Clearly, the mentioned delays have been introduced to take into account a complete query-response transaction originated by the user towards a server in the static network. If a message is originated within the static network and must be simply sent to the user (e.g., the user has a subscription to a news service) the total delivery time is just the D3 delay, since nothing has been generated by the user.

From the above presentation of the delay types, it turns out that in our architecture the most critical segment of the data flow is the one from the infrastructure to the mobile gateways. In fact, the number of message replicas grows significantly on this segment. Over the physical constraints that inevitably impose long delays, the message propagation policy on this segment has a significant impact on the round trip times of user messages. If the response message is replicated on many mobile gateways, it is expected that the user will receive it earlier. However, high number of replicas have negative impact on our second parameter, the buffer occupancy, as well as on the communication bandwidth. Therefore, a trade-off should be found that ensures acceptable delays with reduced number of replicas. In the upcoming sections we present two routing algorithms that address this issue in two different ways.

10.3 Message Forwarding with Flooding (MFF)

Message forwarding with flooding (MFF) is a simple but resource consuming technique. The main characteristic of this technique is that the infrastructure does not make any attempt to reduce the number of replicas of RESP_101 that it propagates through the DTN, but it simply forwards these messages to each MAP passing in its radio proximity. Every static access point receives a replica of the message and, if no acknowledgment is received for a long enough period of time, all of the mobile gateways will have the message in their output buffers. Thus, replicas of the message are stored also in the buffer of gateways that physically have no possibility to deliver it to the destination. However, the big advantage of MFF is that if a single opportunity exists for the message to be delivered to the destination, this technique will not miss it.

10.4 Routing with Target Interlocking

In this section we propose a new technique for routing packets to their destinations in a scheduled DTN. First we describe our technique for localizing the target node then we present our routing algorithm called K2.

10.4.1 Target Interlocking

The target interlocking technique is based on the last known position of a node, which is used for identifying an area, called *interlocking region*, where that node is located with high probability. The area identification, called *interlocking* is done by selecting the K nearest APs and the C closest MAPs to the location where the node has last been seen by the system. Placing the data on these APs and MAPs ensures with high probability that the node will receive them.

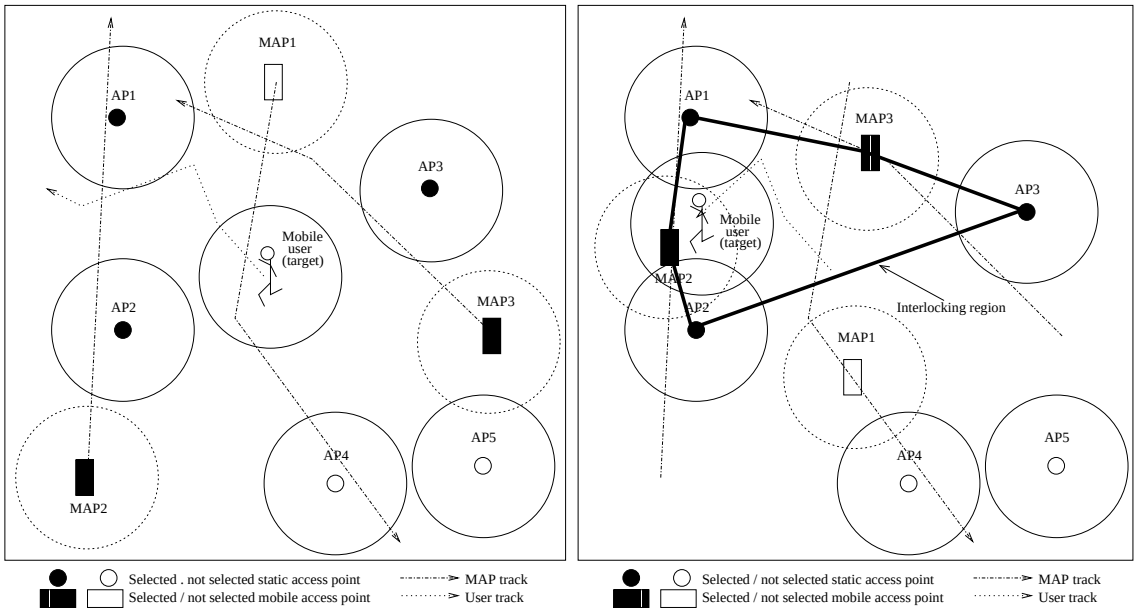

Figure 10.2: Target interlocking scenario with K=3, C=2. The figure shows the moment when the Selection of (M)AP is done (left) and the moment when the message is delivered to the target. The interlocking region is the area hammed in by the 3+2 (M)APs (right).

Location information can be collected either by the means of GPS or some location estimation technique based on the wireless technology used, like the ones in [16, 8, 58]. Notice that it is not required for the mobile nodes to have location estimation capabilities, but the (M)APs register their own location where they meet a certain node. Naturally, if the mobile node has optional positioning capabilities, it may communicate that information to the system, improving this way the interlocking accuracy when a message is to be routed to that node (i.e. the node can help the system to localize it more accurately).

Figure 10.2 shows a scenario where a mobile user to which a packet is to be routed is interlocked by $K = 3$ APs and $C = 2$ MAPs. On the left, the scenario is shown at the moment when a packet is to be routed to the user. The user is placed in the position where it has last been seen by the system. In this case AP1, AP2, AP3, MAP2 and MAP3 are selected as

the most probable (M)APs that will meet the user. Notice that MAP1 is not selected because it will not meet any APs to overtake the message that has to be delivered, before meeting the user himself. On the right, the user meets MAP2 and receives the message. The interlocking region is drawn with thick borders. Notice that the (M)APs may not always surround the target node if the AP locations and MAP tracks are so that this is not possible (e.g. at the borders of the system coverage). Also in these cases those (M)APs are selected that the mobile user will encounter with the highest probability.

10.4.2 The `K2` Algorithm

`K2` is a routing algorithm designed for improving the way resources are used when routing data in a DTN with scheduled mobile access points. In `K2` we use the *k-nearest neighbors* (`KNN`) technique in two different ways, hence the name of the algorithm.

The `K2` algorithm is presented in Figure 10.3. In order to reduce the number of replicas of the response message, the algorithm reduces the set of gateways that will receive the message. This is achieved by interlocking a region selecting the K nearest APs and C closes MAPs to the last known position of the mobile user, as presented in section 10.4.1.

We shall suppose that a message contains, among others, the fields *id* (a unique message identifier) and *destination*. Moreover, a list of mobile APs (*forwardList*) is associated to each message and shall be treated as a contained field.

The algorithm in Figure 10.3 is composed of three asynchronous parts:

Lines 1–4 are executed when a new message appears in the input buffer of a fixed AP directed to a user. The AP sends the message to the output buffers of the K APs that are nearest to the last known po-

```
1.  when incoming message m
2.  ┌ foreach AP a ∈ KNN (m.destination,K)
3.  │    put m into a.outputBuffer
4.  └ m.forwardList ← KCC (m.destination,C)

5.  when user u within range
6.  ┌ foreach message m ∈ outputBuffer
7.  │ ┌ if m.destination = u
8.  │ │ ┌ hand m to u
9.  │ │ │ remove m from outputBuffer
10. └ └ └ put m.id into ackBuffer

11. when MAP a within range
12. ┌ foreach message a in outputBuffer
13. │ ┌ if a ∈ m.forwardList
14. └ └    hand m to a
```

Figure 10.3: Pseudo code of the K2 algorithm

sition of the destination (lines 2–3). Then it determines the C most suitable mobile APs and appends their list to the message (line 4). This happens in contrast with the MFF algorithm, where the message would be forwarded to all APs and the *forwardList* would contain all of the mobile APs. The selection of the K nearest APs is performed by a common k-nearest-neighbors algorithm. On the other hand, for choosing the C most promising MAPs we define a variation of the k-nearest-neighbors technique, called KCC for K-closest-connections. KCC finds the list of MAPs which will pass next to the user's last known location after traveling past a fixed AP that holds the relevant message.

Lines 5–10 are executed whenever a user enters the coverage of an AP. If any message directed to the user is stored on the infrastructure, it is handed and deleted from the AP output buffer. Moreover, the

message identifier is inserted in the AP's ACK buffer for removing delivered messages from the other APs and MAPs.

Lines 11–14 are executed when a mobile AP enters the coverage of an AP. In this case, messages that contain the mobile AP in their *forwardList* are handed to the MAP. The handed messages are not deleted from the AP buffer until their delivery is acknowledged.

The removal of messages from the AP and MAP buffers happens in similar manner to that of regular message sending. When delivered by a MAP, a message is removed from the output buffer of the MAP and an acknowledgement message is placed in its input buffer. Later, this acknowledgment is transmitted to the infrastructure, which will inform also the other MAPs and all APs to remove that particular message from their buffers. If the message is delivered by an AP the other APs and MAPs can be informed earlier about the delivery, since the acknowledgment is already at the infrastructure.

We implemented the busnet scenario for evaluating the performance of the `K2` algorithm. Further information about the operation of the algorithm are provided in Section 10.5 together with the implementation details.

10.5 Scheduled DTN Simulator

To test our algorithm and to study the buffer lengths and the message delivery delays we implemented a simulator in C++ for an urban bus network that we call *BusNet*. Next in this section this simulator is presented in details.

We start out by generating the elements of our scenario. In particular we generate the bus (i.e. MAP) tracks on an area of $5000 \times 3000m^2$. Bus tracks consist of bus stop positions connected with straight lines. For each

bus stop we assign a time between 10s and 60s that the buses following that track will spend there. Further, we set the average speed of $3 \div 10$m/s that the buses will travel with on each segment between two bus stops. Finally, for each specific line we set the frequency the buses will follow each other with. This frequency varies from 10 to 30 minutes.

User tracks are generated in a similar manner with some differences. A random starting point is generated for each user track, then new points are added close to the previous one that the user will reach with a walking speed of $1 \div 3$m/s. The angle formed by two consequent track segments is selected with care in order to simulate users that follow straighter paths or others that move back and forth on a small area. During the simulation users generate messages destined randomly to some other user or to the infrastructure that will answer them. Each user generates messages according to the Poisson distribution, with different average frequency. Notice that even though we generate data packets in a uniformly distributed manner, the traffic in our network will be bursty because packets are accumulated in the different buffers (i.e. at the bundle layer [60]) of our system and forwarded altogether when the right connection becomes available.

Figure 10.4 shows the setup of some experiments. On the left, bus tracks are reported along with user tracks and static AP positions. On the right, a grayscale plot of coverage frequency is shown. White circles indicate full coverage, darker regions are covered for less than 100% of the total simulation time. Note that the scale is nonlinear in order to enhance the visibility of darker regions.

Our simulator starts out by loading the bus and user track data. From this data we first calculate the position of the fixed access points by placing one access point every third bus stop on each line, avoiding superposition in stops that are common to more lines. We also calculate the *round duration*, the time necessary for a bus to travel from one end of a line to the other and

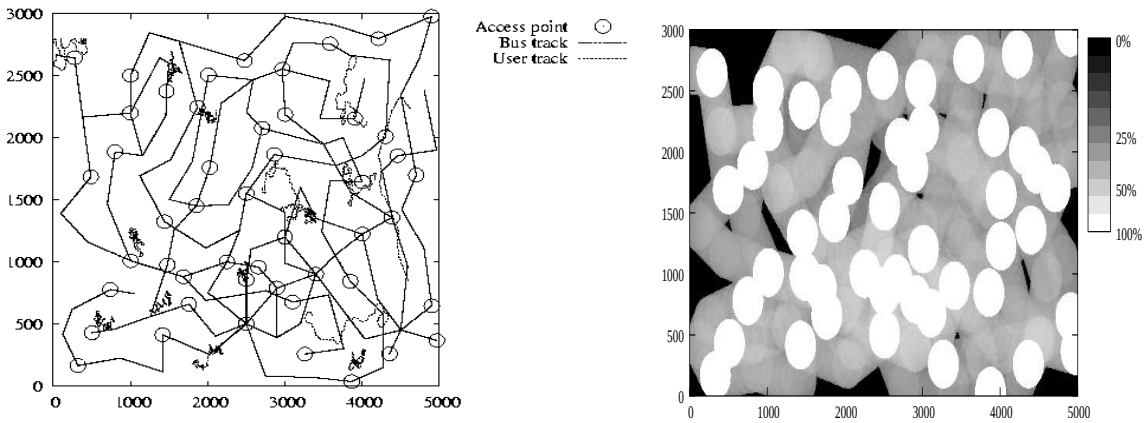

Figure 10.4: Bus lines and user tracks on the map (left) and corresponding time of coverage during the simulation (right). Note that the greyscale is not linear. Map coordinates are given in meters.

back. From the round duration value and the bus frequency we calculate the number of buses that will cycle on each line during the simulation.

Before users start to move on the map and send out messages, buses are moved on their tracks for the duration of the longest round trip time, so that a steady mode of operation is reached, with as many buses per track as needed.

When the bus system is fully deployed, the main part of the simulation, handling users and message exchange, begins. The state of the system is evaluated every 0.5s of simulated time until the preset number of messages were received and removed from all buffers, or until a preset simulation time expires. Each step of the simulation starts by updating the locations of buses and users then user output buffers are checked for newly generated messages. If in the buffer of a user u some messages are found, the system checks whether u is connected to any (M)AP. In case of available connection(s) all pending messages of u are forwarded to the appropriate (M)AP, giving higher priority to fixed access points.

MAP buffers are also checked at each iteration. If a bus arrives in

the range of an AP it forwards all of the messages from its input buffer ($MAPin$) to the input buffer of the AP ($APin$). In $MAPin$ there may be data or acknowledgment messages. The latter have no payload, therefore they only occupy a small space in the buffer, while messages containing user data are all of the same length. Acknowledgment message identifiers are stored at the APs, in order to keep track of acknowledged messages, and then deleted from $APin$. After the forwarding operation each bus will delete from its $MAPout$ all messages in the acknowledged message list of the infrastructure.

At this point messages are processed by the infrastructure and then forwarded or replied. Messages from the infrastructure to the users are routed using the `MFF` or the `K2` algorithm. Some messages are handed directly to the destination by the APs while others are replicated on a number of MAPs. Messages from the output buffer of MAPs are handed to their destination when that user will be in range of the MAP. If this never happens, then that message will remain in the (M)AP buffers until the end of the simulation. In a future version of the simulator a time-to-live value may be attached to the packets enabling the automatic removal of old messages. Further, with the modification of the `K2` algorithm new attempts to deliver the message to its destination could be triggered by a timeout or some other event. These issues are subject for future work.

A message is deleted from the $MAPout$ of a MAP when it is handed to its destination. At the same time a new acknowledgment message is added to the $MAPin$ of the same MAP.

The BusNet system permanently monitors the location of its clients. The `K2` algorithm uses this information to select the K nearest access points and the C closest connections for each particular user. Simulation results are presented in Chapter 11.

10.6 Conclusion

In this chapter we defined `K2`, a location-aware routing algorithm for scheduled delay-tolerant networks. `K2` reduces the number of replicas of the messages that are being forwarded to mobile users by selecting a reduced set of access points that receive the messages. This is achieved with the target interlocking technique that selects the K nearest static access points and C closes mobile access points to the last known position of the mobile user.

Furthermore, we presented a scheduled DTN simulator that enables us to evaluate the buffer usage and message delivery delays while using our algorithm. We also categorized the delays in our system and identified the one that has the most significant impact on the network performance. `K2` focuses on this delay to achieve a more efficient usage of network resources.

Chapter 11

Experimental Results

To evaluate `K2` and get an overall view about the buffer sizes and message delivery delays we performed simulations with numerous different settings of the bus network system. The most important input parameters that we acted upon were the bus map (i.e. the set of bus tracks), K (the number of APs) and C (the number of MAPs).

We built three bus maps. First, we designed an irregular map, referred as *mesh*, such that to cover approximately the whole simulation area (Figure 10.4), trying to imitate the real bus network of a city. Later, for comparison purposes and for evaluating the influence of the coverage on the buffers and delays, we generated two regular maps as well. In the first one vertical and horizontal bus lines form a uniformly spaced (400m) *grid* (see Figure 11.1 left). Finally, the third map consists of uniformly spaced (400m), concentric octagonal bus lines as well as seven *radial* lines connecting the center to the angles of the heptagons (see Figure 11.1 right). This way we obtained a map (grid) with uniform coverage and another one with better coverage in its center and poorer coverage at the margins . On all of these maps static access points were placed at every third bus stop.

Figure 11.2 traces the execution of two simulation runs using the `MFF` (left) and `K2` (right) algorithms, respectively. During both of the runs 1000

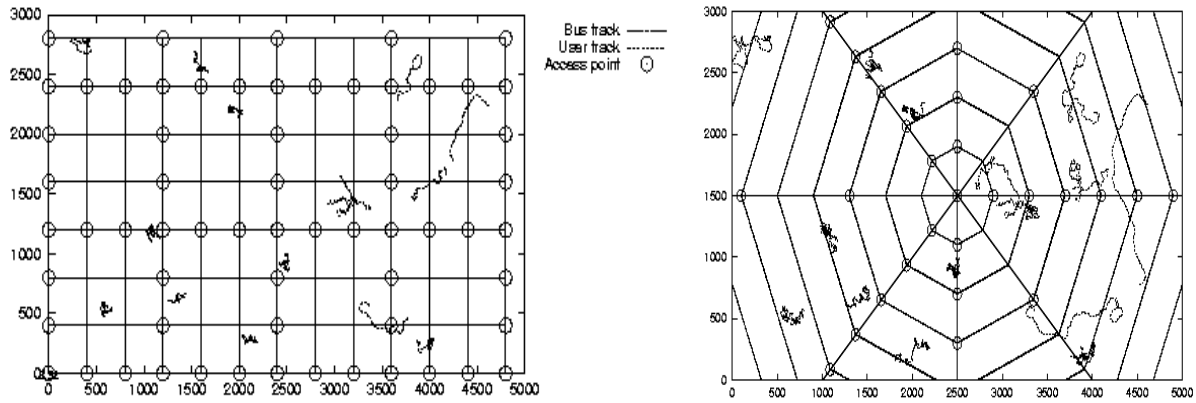

Figure 11.1: Grid map example (left). Radial map example (right)

messages among 15 users were sent, using the the urban setting shown in Figure 10.4, with 9 bus lines. The figure shows the total load, in number of messages, of the most interesting buffers of BusNet, namely the output buffers of the mobile and static APs. The other buffers are all smaller than the MAP output buffer and are not shown in the figure for better visibility.

In the figure, with both algorithms, the AP output buffer is much bigger than the MAP buffer. This happens for two resons. First, since static APs are always connected, they can receive immediately the messages that they need to forward as soon as the messages are available. MAPs, in change, receive these messages only when they get in the range of an AP that has the message. Second, some of the messages are delivered to the destination directly by the static APs, when the users are in their range. Thus, these messages are not forwarded to the MAPs.

Comparing the two buffers in question obtained with the two routing algorithms we find that the MAP and AP buffers are approximately 14.5

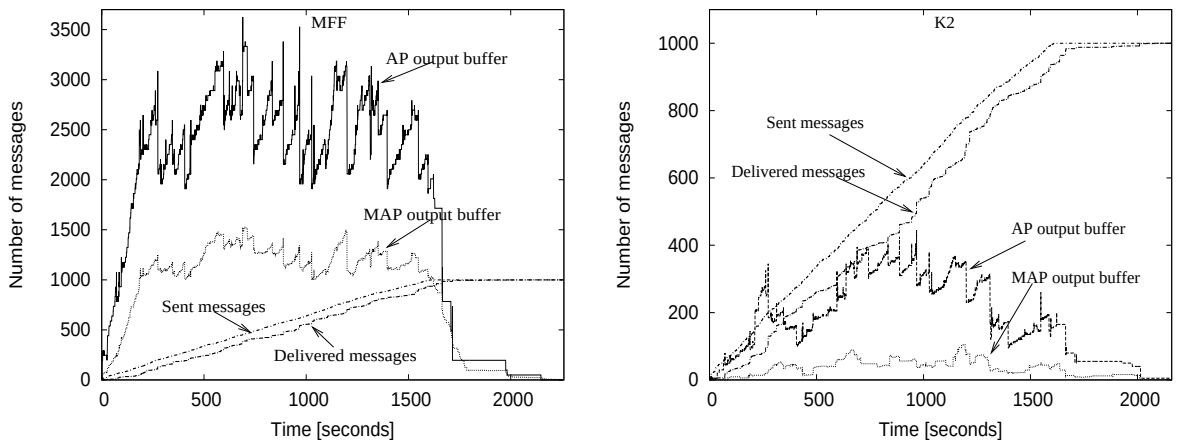

Figure 11.2: Traces of simulation runs with `MFF` (left) and `K2` (right)

and 8 times smaller, respectively, in case of `K2`, which is a considerable reduction in the favor of this latter technique.

We can also see in Figure 11.2 that the number of delivered messages closely follows the number of sent messages. The average time delay between the two curves is D1+D2+D3, which, according to Table 11.2 is equal to 133.96s with `MFF` and 141.1s with `K2`.

Note that the system reaches its "steady state" after only about 200s. This is due to an initial "warm-up" period that we introduced in the simulations. Without this initial period the transient state would last about 1000s.

The simulation stops when the last message is deleted from the last MAP buffer, after the delivery. With the current settings all messages are delivered before the end of the simulation.

For evaluating the performance of the `K2` algorithm we performed seven experiments on each map, fixing $K = 4$ and setting $C = 2, 4, 6, 8$, and vice-versa. Selected simulation results are presented in Figure 11.3.

Before discussing the results in Figure 11.3 let us specify also the fixed settings of our simulations. During each experiment there are 15 users

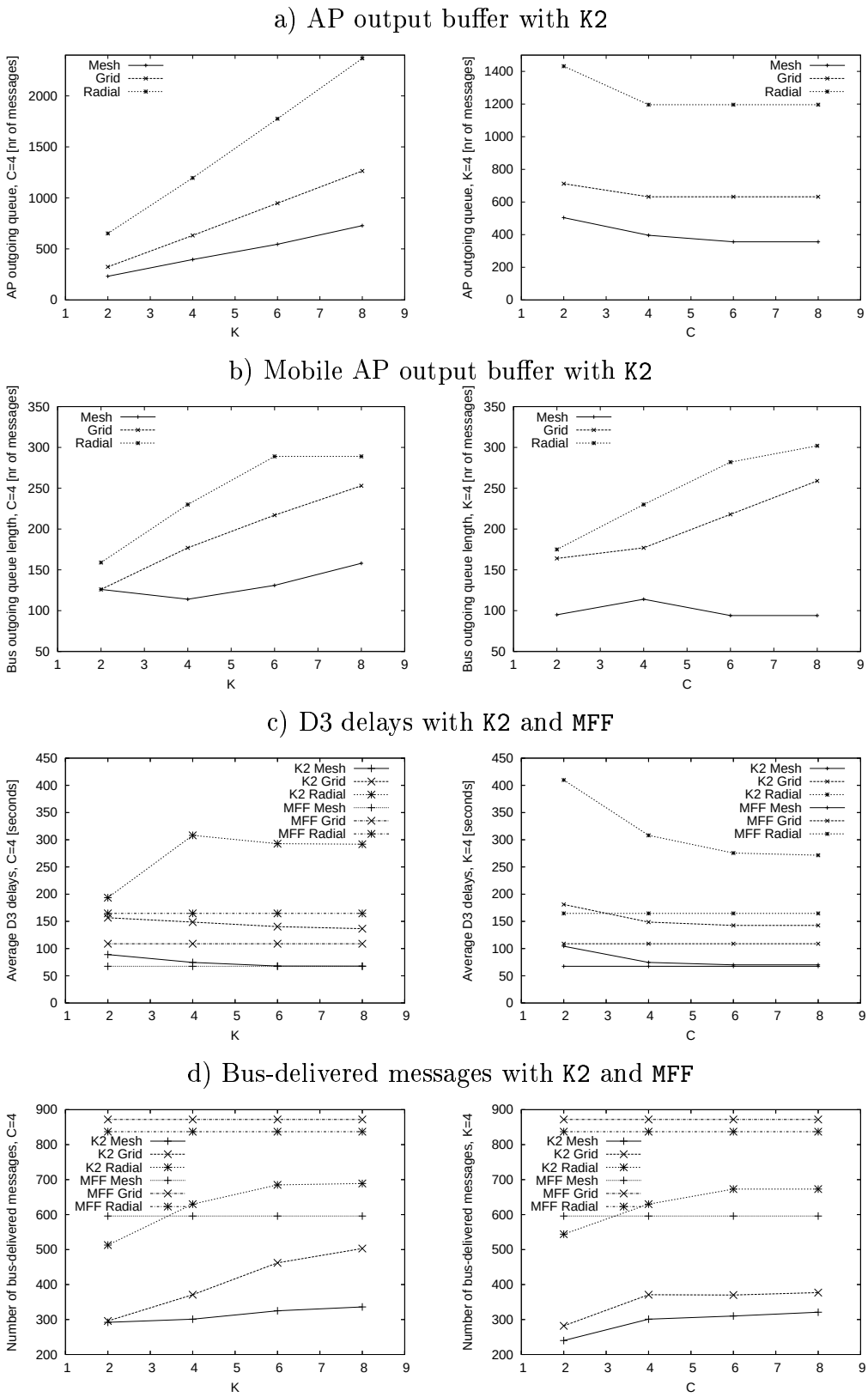

Figure 11.3: Disseminated simulation results for different values of K and C

moving on the map generating 1000 messages. Further, we have the following number of access points generated on the different maps. On the 9 bus lines of the mesh map there are 24 MAPs and 49 APs. On the grid map there are 21 lines with 57 MAPs and 64 APs while on the radial one the number of lines is 22 with 49 MAPs and 42 APs. The number of APs and MAPs in our simulations follows from the length of the bus lines and the number of stops on each line (see also Section 10.5).

The range of the (M)APs is 200 meters. Simulations with a range of 100 meters confirmed our expectations regarding much bigger buffers and delays. Therefore, for such a network longer-range wireless technologies are favored against shorter-range technologies. For instance, the WiFi family with the range of above 100m will provide better performance when compared to the Bluetooth technology, which has a range of 10m in handheld devices.

Returning to Figure 11.3 we can see the evolution of the most interesting parameters of the simulations. Note that in the diagrams on the left we set $C = 4$ while on the right we set $K = 4$. Further, it is good to know in advance that the average coverage of the system, computed with the Monte Carlo method, is 46.1%, 63.0% and 37.8% for the mesh, grid and radial maps, respectively.

The biggest buffers of our system are the total output buffers of the APs and MAPs (i.e. *APout* and *MAPout*). In Figure 11.3.a, showing results obtained with `K2`, it can be seen that for a fixed number of MAPs, *APout* increases as we increase the number of APs without any sign of stabilization. This is due to the fact that messages destined to users out of all fixed AP ranges fill up the AP buffers. On contrary, when we fix K and gradually increase C the *APout* decreases to a certain value, making obvious the importance of MAPs.

On the other hand, *MAPout* grows in both cases (Figure 11.3.b), when

K2 is used. The reason why *MAPout* increases with K is that if the messages are available from more APs, more MAPs can receive it.

The other buffers of our system are smaller than the ones presented above and they offer scarce possibility for reduction as well, therefore they are of secondary importance for the optimizations. Peak values of these buffers obtained with both the MFF and K2 algorithms (this latter with $K = 4$ and $C = 4$) on each of the three maps are presented in Table 11.1. Generally, all K2 buffers are smaller than those obtained with MFF. The difference is usually not big, however there are also cases (e.g. AP input buffer with the radial map) when the MFF buffer is more than 4 times bigger than the corresponding K2 buffer.

Table 11.1: Peak values for secondary buffer usage obtained with K2 and MFF. For K2 K=C=4.

Buffers	**Mesh**		**Grid**		**Radial**	
	MFF	K2	MFF	K2	MFF	K2
User output	64	64	89	89	121	121
MAP input	106	109	177	106	221	183
Acknowledgments	54	74	86	77	131	119
AP input	74	29	119	52	241	53

For better visibility of the more interesting K2 curves, we do not not show in Figure 11.3.a and 11.3.b the values obtained with MFF, because they are too big with respect to the K2 values. Since MFF results do not depend on the K and C variables, their graphical representation is flat, with the following values. The MFF AP output buffers, in number of messages, are of 3626, 7616, 10122, while the MAP output buffer values are 1526, 6015, 9993, for the mesh, grid and radial maps, respectively.

Among the five delay types we identified D3 as being the longest one and, hence, the most important. Generally, D3 decreases as the number of (M)APs grows. The lower limit of D3 can be obtained with MFF since this

Table 11.2: Average and peak delays obtained with MFF and K2. For K2 K=C=4.

Delay	Alg.	Mesh		Grid		Radial	
		Avg	Peak	Avg	Peak	Avg	Peak
D1	**K2**	51.04	530.5	88.85	856.4	118.50	1175.8
	MFF	51.04	530.5	78.90	856.4	108.52	1175.82
D2	**K2**	15.48	111.0	47.73	293.5	75.07	450.0
	MFF	15.48	111.0	43.43	293.5	68.97	438.00
D3	**K2**	74.58	1189.0	148.68	2193.0	308.12	2630.5
	MFF	67.44	540.5	108.84	856.5	164.60	1337.50
D4	**K2**	10.41	109.0	32.07	391.0	83.18	622.5
	MFF	15.59	160.0	56.14	430.5	109.34	822.50
D5	**K2**	2.41	116.0	6.15	214.0	30.66	436.5
	MFF	45.20	166.0	88.95	304.5	102.29	758.00

algorithm uses all of the available (M)APs. Notice that in Figure 11.3.c we represented also MFF results for an easier comparison. The obtained values are 67.44, 108.84, 164.6 for the mesh, grid and radial maps, respectively.

The lowest average D3 value (67.44s) that we obtained with MFF for the above presented settings was for the mesh map. However, with only 6 MAPs and 4 APs the value of 74.58s can be obtained, using K2. This shows that is useless to raise the number of (M)APs above a certain limit, because the gain that can be obtained is insignificant.

Regarding the other delays, D1 and D2 remain constant during all experiments with the same map. This is due to the fact that messages are forwarded by the user to the first bus that passes in his range, while the bus will pass on the message to the first AP it reaches. D4 and D5 are very small with respect to D3. Not even in the K2 case D4 depends on K or C, but on the distance between the places where the MAP hands the message to the destination and the first AP it reaches thereafter. In change, D5 decreases with K and C since the number of messages to be deleted from

the buses grows with the two constants. Sample average and peak values for these delays for both algorithms are shown in Table 11.2, where the $K2$ algorithm was run with $K = C = 4$ on the three different maps.

Finally, it can be seen in the Figure 11.3.d that the number of MAP-delivered messages is very high if `MFF` is used. This is due to the fact that with many MAPs users receive their packets before they could reach a static AP. Packets are delivered by APs almost exclusively when they are within their range at the moment when the packet is uploaded to the fixed infrastructure from the source side.

On the other hand, with `K2` the number of MAP-delivered messages increases with K and C. When the number of MAPs (i.e. C) is gradually increased the probability for a user to meet a mobile AP earlier than a static one increases because MAPs can cover a bigger area while moving than the static APs. Instead, when packets are replicated to a higher number of static APs (i.e. K is increased), MAPs can benefit again because they can grab the packets from more APs bringing them to the destination node before it can reach a static AP. This may not happen if K is much bigger than C and the APs are deployed densely on the territory.

We can conclude that `K2` can reduce drastically the buffer lengths with respect to the basic flooding algorithm, `MFF`, without significant negative impact on the message latency. The simulations demonstared that by taking advantage of the D3 delay, that we identified as being the most important one, it is possible to route messages in a more efficient way. Moreover, the results regarding messages that were actually delivered by mobile APs to the users show that a DTN approach can be effective for many applications where extensive static coverage is unprofitable.

Although the experimental results seem promising, more detailed simulations are required for evaluating the behavior of `K2` in case of many more users, detailed data transfer models between the users and buses,

multihop access to MAPs through ad hoc networks and so on. Letting occasional MAPs (i.e. vehicles with no schedule) transport a part of the traffic intended for scheduled MAPs could be a further extension of this early architecture. Finally, a reactive scheme for automatically setting the K and C parameters in concordance with the users' moving speeds would also improve the performance and usability of our algorithm. All these issues are subject for future work.

Chapter 12

DTN Related Work

The issue of communicating between isolated ad hoc network partitions has recently received high attention in the research community. Due to the dynamic behavior of mobile wireless devices and their limited radio range it often happens that different network nodes can not reach each other through a multihop path. Therefore, such devices can not communicate nor they can take advantage of each other's resources. In order to make communication possible between such devices, two main streams of approaches are available in the literature.

First, communication links between two ad hoc network partitions or isolated clusters, can be set up using specialized gateways with enhanced radio capabilities and/or airborne relay nodes (e.g. airplanes, satellites), like in the systems targeted by [5, 69]. Although such network architectures provide a straightforward solution for interconnecting ad hoc clusters, airborne or long-range relay nodes may not always be available or they might be too expensive to use. Further, such a solution fits only scenarios where an enhanced gateway can serve a group of devices, but it is not feasible to associate such a gateway to any node operating in isolation.

The second stream of approaches takes advantage of the delay-tolerant networking technology [17, 22] that exploits node mobility for providing

connectivity among isolated network partitions. Part II of this thesis targets this approach for interconnecting nodes that have no end-to-end path among them.

In our approach the functionality of message transportation is filled in by mobile access points (MAPs) carried by vehicles that operate on a particular territory. MAPs are capable of transporting data bundles (i.e. a collection of packets) among isolated devices. Data bundle management happens at the *Bundle Layer* that sits on the transport layer, as described in [60]. The MAPs in our network architecture implement such bundle layer functionality for managing the data that they relay between isolated mobile users and the static infrastructure (see Section 10.2).

The absence of communication support between isolated nodes in the traditional networking architectures turns them useless when there is no end-to-end path available among the nodes. Long propagation delays and high error rates make proper functioning of the most common networking protocols (in particular IP-based ones) impossible, since frequent retransmissions, expiration of short time-to-live values, links marked as non-operable, and so on, make them abort all initiated communication channels [22].

The store-and-forward model of DTNs is able to overcome all the above problems, however is not obvious how to identify the next hop to forward a packet to during the routing process. The authors of [31] recently categorized DTN routing algorithms based on the amount of knowledge about the nodes that they use and confirm through simulations that algorithms taking advantage of a bigger knowledge base outperform the simpler ones. For the development of the `K2` algorithm, presented in Section 10.4.2, we also used the available knowledge about the timestamped location of MAPs for reducing the number of relay nodes between the static infrastructure and destination nodes, when transmitting packets.

Traditional ad hoc routing schemes based on a general or restricted flooding for route discovery, like [53], AODV [54] or DSR [32], are not usable in DTNs since the long delays do not allow fast connection setup and data transfer between the source and destination. The set of links of a source node would probably be modified before a route-reply could arrive to the source's route request, rendering route discovery impossible.

Gossiping or its optimized versions like myopic and rumor routing are all based on flooding (independently of their scope's size) [28, 15]. These routing schemes may be useful in DTNs where nothing is known about the behavior of the nodes or when routing is based on some history information that the nodes must collect from each other. However, if some knowledge (like trajectory and schedule in our system) is available about the future behavior of the nodes then more efficient schemes can be developed.

Routing algorithms based on location information like LAR [40] or GPSR [36] are good candidates for being adapted to DTNs since they should not necessarily rely on earlier discovered paths (or links) but on the current location of the destination. Indeed, in Section 10.4 we present a technique similar to the one in [40], based on the identification of a region where the destination of a message is supposed to be. Our approach is different from LAR because it does not need to set up communication paths based on route-request messages, that can not operate properly in environments where no end-to-end connection exists between the traffic sources and destinations. Furthermore, our `K2` algorithm uses redundant paths to deliver packets, which can reduce delays on the expense of using more network capacity. The usability of redundant paths for delay reduction has been demonstarted in [51]. Naturally, packet transmission on redundant paths uses more of the system resources, however good tradeoffs can be found.

For connecting isolated network clusters the authors of [71] propose the

usage of so-called "ferries". Ferries have the functionality of collecting data in one network cluster and transporting it to other clusters where the destination of the data is reachable. Ferries are a similar concept to that of mobile access points or buses, presented in this thesis, or Data Mules described in [61]. This approach may be a good solution for scenarios where such ferries are dedicated to serve as interconnection means between network partitions. In both of the proactive and reactive approaches presented in [61], the ferry moves to the nodes operating in isolation or vice-versa. This requires the ferry or the node to modify its original trajectory and a long range radio link for the service request. In contrast to that, in our system none of these network elements change their original trajectory for delivering or receiving data, nor the users have to cooperate with the system using long-range signaling messages. This is an additional constraint for our algorithm.

The disadvantage of trajectory modification can be found also in the approach presented in [43]. In that work the communicating nodes estimate meeting points that they then physically have to reach to meet other nodes to which they can forward their messages for propagating them closer to the destination. This solution, beside the trajectory modification, also requires knowledge about the position of other, potential relay nodes.

Part III

Conclusion

This thesis presents our work on Bluetooth scatternet and delay-tolerant network optimizations. Part I discusses the issue of scatternet optimization based on the reduction of path length between the communicating nodes. Node mobility, changing traffic sessions and similar dynamic factors, generate permanently changing traffic flows in a scatternet. Thus, for an efficient communication it is not enough to form an optimal network topology, rather a scatternet that best supports the current traffic flows as they vary in time is required.

Intuitively, the scatternet performance, in terms of throughput and power consumption, highly depends on the hop count separating the communicating peers. In this thesis we demonstrated the above intuition analytically, which required us to define an analytical model also for sharing the communication capacity of nodes in a Bluetooth scatternet.

Having obtained the analytical confirmation about the interdependence of the hop count, throughput and power consumption, simulations were required to get an insight in the actual scatternet performance variations in function of the hop count. On this purpose we developed a heuristic algorithm suite that reconfigures the roles and links of the nodes so as to minimize the number of hops between communicating nodes in the scatternet. As part of the algorithm suite, we presented two algorithms for reducing the length of connections belonging to slaves and other two for master connections. These algorithms are based on four "move" types (i.e. elementary modules for reconfiguring scatternets) and a matrix-based scatternet model, both defined in this thesis.

Finally, to enable the evaluation of scatternets in time we extended our optimization algorithms to support also dynamic traffic flows and node mobility. The resulting optimization process provides a systematic approach to scatternet optimization not only right after the network formation, but also later when the scatternet performance degrades because of the dy-

namic behavior of the users.

Simulation results show that our hop reduction algorithms can reduce the aggregate hop count in the scatternet by about 15%. This, on turn, can generate a performance improvement of above 60% in moderately dynamic scenarios.

The centralized nature of our optimizations are their main weakness since they require to collect all scatternet information to a single node and distribute the solutions to all of the other nodes. This operation may introduce significant delays in the optimizations. To avoid this shortcoming, for the future we propose to find also a decentralized solution for our optimization problem.

Part II of this thesis addresses a routing optimization issue in scheduled delay-tolerant networks. Routing in DTNs is highly dependant on the knowledge about future connections of the nodes, since these connections are the only communication means in the store-and-forward paradigm of DTNs. In this thesis we presented `K2`, a heuristic routing algorithm that uses location information to interlock the neighborhood of traffic destination nodes where they are supposed to be reachable. In consequence, the packets need not flood the entire network, but they are routed only to the nodes that interlock the area of the destination.

We used two metrics, buffer usage and routing delay, to evaluate `K2` and compared its performance to that of `MFF` (Message Forwarding with Flooding). For the evaluation we implemented a simulator of an urban transportation network with buses outfitted with mobile access points (MAPs) that relay data packets back and forth between mobile users and the Internet.

The simulations enabled us to identify those buffers and delays of the simulated system that have a major impact on the efficiency of the communication. In consequence, we obtained a clear view on which buffers

and delays can be used for improving the performance of the simulated DTN and hence we developed `K2` with those elements in mind. Simulation results show that `K2` can reduce the buffer usage on the MAPs by about 14 times with respect to `MFF`, without significant impact on the routing delays.

This DTN optimization work is in its early phase, therefore many improvements can be imagined for the future. Simulations with more users and more diversified moving speeds, detailed data transfer models between users and mobile access points, multihop access to the MAPs, letting occasional vehicles (i.e. vehicles with no schedule) transport part of the bus network's data traffic using their own on-board GPS-enabled MAPs, are all subject for future work.

Bibliography

[1] ICT e Sesto Programma Quadro della ricerca europea: Il progetto AHLLBA. http://www.euroways.org/iaf/2003/reisoli.pdf, September 2003.

[2] Bluehoc. http://oss.software.ibm.com/bluehoc/, Last accessed: December 2004.

[3] The network simulator - (ns-2). http://www.isi.edu/nsnam/ns/, Last accessed: December 2004.

[4] The ucbt simulator. https://www.ececs.uc.edu/ cdmc/ucbt/, Last accessed: December 2004.

[5] Mohiuddin Ahmed, Son Dao, and Randy Katz. Positioning range extension gateways in mobile ad hoc wireless networks to improve connectivity and throughput. In *Proceedings of the IEEE Military Communications Conference (MILCOM 2001)*, Washington, D.C., USA, October 2001.

[6] Marco Ajmone Marsan, Carla-Fabiana Chiasserini, Antonio Nucci, Giuliana Carello, and Luigi De Giovanni. Optimizing the topology of Bluetooth wireless personal area networks. In *INFOCOM*, New York, USA, June 2002.

[7] Simon Baatz, Matthias Frank, Carmen Kohl, Peter Martini, and Christoph Scholz. Adaptive scatternet support for Bluetooth using sniff mode. In *LNC*, Tampa, Florida, 2001.

[8] Paramvir Bahl and Venkata N. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *Proceedings of IEEE INFOCOM 2000*, pages 775–784, March 2000.

[9] Stefano Basagni and Chiara Petrioli. A scatternet formation protocol for ad hoc networks of Bluetooth devices. In *IEEE Vehicular Technology Conference (VTC)*, pages 424–428, 2002.

[10] Pravin Bhagwat and Srinivasa P. Rao. On the characterization of Bluetooth scatternet topologies. http://www.winlab.rutgers.edu/ pravin/ publications/papers/bt-top.ps, August 2002. Submitted for publication.

[11] Pravin Bhagwat and Adrian Segall. A routing vector method (RVM) for routing in Bluetooth scatternets. In *Proceedings of the Sixth IEEE International Workshop on Mobile Multimedia Communications (MOMUC'99)*, November 1999.

[12] Bluetooth SIG. *Specification of the Bluetooth System, v1.1.* February 2001.

[13] Bluetooth SIG. *Bluetooth Specification, Profiles v1.2.* May 2003.

[14] Bluetooth SIG. *Bluetooth Specification v1.2.* May 2003.

[15] David Braginsky and Deborah Estrin. Rumor routing algorithm for sensor networks. In *Proceedings of WSNA 2002*, Atlanta, GA, USA, September 2002.

[16] Mauro Brunato and Csaba Kiss Kalló. Transparent location fingerprinting for wireless services. In *Proceedings of the Med-Hoc-Net Workshop*, Cagliari, Italy, September 2002.

[17] Vinton Cerf, Scott Burleigh, Adrian Hook, Leigh Torgerson, Robert Durst, Keith Scott, Kevin Fall, and Howard Weiss. Delay-tolerant network architecture. Internet draft, IETF, March 2003.

[18] Ling-Jyh Chen, R. Kapoor, M. Y. Sanadidi, and Mario Gerla. Enhancing bluetooth TCP throughput via link layer packet adaptation. In *International Conference on Communications (ICC'04)*, Paris, France, June 2004.

[19] Xiangchuan Chen and Amy L. Murphy. Enabling disconnected transitive communication in mobile ad hoc networks. In *Proceedings of the Workshop on Principles of Mobile Computing, collocated with PODC'01*, pages 21–27, Newport, RI, USA, August 2001.

[20] Carla-Fabiana Chiasserini, Marco Ajmone Marsan, Elena Baralis, and Paolo Garza. Towards feasible distributed topology formation algorithms for Bluetooth-based WPANs. In *36th Annual Hawaii International Conference on System Sciences*, Big Island, Hawai, January 2003.

[21] Avri Doria and Maria Uden. Providing connectivity to the saami nomadic community. In *Proceedings of the 2nd International Conference on Open Collaborative Design for Sustainable Innovation (DYD 02)*, Bangalore, India, December 2002.

[22] Kevin Fall. A delay-tolerant network architecture for challenged internets. Technical Report IRB-TR-03-2003, Intel Research, March 2003.

[23] Chun-Choong Foo and Kee-Chaing Chua. Bluerings - Bluetooth scatternets with ring structures. In *International Conference on Wireless and Optical Communications*, Banff, Canada, July 2002.

[24] Ian Foster. *Designing and Building Parallel Programs*, chapter 3.9. On-line edition, 1995.

[25] Phyllis G. Frankl and David J. Goodman. Delivering information "To Go" via infostations. Technical Report WICAT-TR-02-008, Polytechnic University, NY, USA, 2002.

[26] Delay-Tolerant Networking Group. http://www.dtnrg.org/, Accessed: July 2004.

[27] Jaap C. Haarsten. The Bluetooth radio system. *IEEE Personal Communications*, February 2000.

[28] Zygmund Haas, Joseph Y. Halpern, and Li Li. Gossip-based ad hoc routing. In *Proceedings of INFOCOM 2002*, New York, USA, June 2002.

[29] Johann Hurink. *Introduction to Local Search.* Lecture notes, 2001.

[30] Jeffrey Jaffe. Bottleneck flow control. *IEEE Transactions on Communications*, 29(7):954–962, July 1981.

[31] Sushant Jain, Kevin Fall, and Rabin Patra. Routing in a delay tolerant network. In *Proceedings of ACM Sigcomm 2004*, Portland, Oregon, USA, September 2004.

[32] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[33] M. Kalia, S. Garg, and R. Shorey. Scatternet structure and inter-piconet communication in the Bluetooth system. In *IEEE National Conference on Communications*, New Delhi, India, 2000.

[34] Csaba Kiss Kalló, Sewook Jung, Ling-Jyh Chen, Mauro Brunato, and Mario Gerla. Throughput, energy and path length tradeoffs in Bluetooth scatternets. Technical Report TR040031, University of California, Los Angeles, 2004.

[35] Csaba Kiss Kalló, Sewook Jung, Ling-Jyh Chen, Mauro Brunato, and Mario Gerla. Throughput, energy and path length tradeoffs in Bluetooth scatternets. In *Accepted for publication at the IEEE International Conference on Communications (ICC'05)*, Seoul, Korea, May 2005.

[36] Brad Karp and H.T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of MobiCom 2000*, Boston, MA, USA, August 2000.

[37] Csaba Kiss Kalló and Mauro Brunato. Scheduled message delivery in delay-tolerant networks. Technical Report DIT-04-14, University of Trento, Italy, 2004.

[38] Csaba Kiss Kalló, Carla-Fabiana Chiasserini, Roberto Battiti, and Marco Ajmone Marsan. An algorithm suite for minimizing the number of hops between communication peers in a Bluetooth scatternet. Technical Report DIT-04-101, University of Trento, Italy, 2004.

[39] Csaba Kiss Kalló, Carla-Fabiana Chiasserini, Roberto Battiti, and Marco Ajmone Marsan. Reducing the number of hops between communication peers in a Bluetooth scatternet. In *IEEE Wireless Communications and Networking Conference (WCNC'04)*, Atlanta, GA, USA, March 2004.

[40] Young-Bae Ko and Nitin H.Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. In *Proceedings of MobiCom 1998*, Dallas, TX, USA, October 1998.

[41] Ching Law, Amar K. Mehta, and Kai-Yeung Siu. Performance of a new Bluetooth scatternet formation protocol. In *Proceedings of ACM Symposium on Mobile Ad Hoc Networking and Computing 2001*, Long Beach, CA USA, October 2001.

[42] Ching Law and Kai-Yeung Siu. A Bluetooth scatternet formation algorithm. In *IEEE Symposium on Ad Hoc Wireless Networks 2001*, San Antonio, TX, USA, November 2001.

[43] Qun Li and Daniela Rus. Sending messages to mobile users in disconnected ad-hoc wireless networks. In *Proceedings of ACM Mobihoc 2004*, Tokyo, Japan, May 2004.

[44] Anders Lindgren and Olov Schel. Infrastructured ad hoc networks. In *Proceedings of the International Conference on Parallel Processing Workshops (ICPPW'02)*, Vancouver, B.C., Canada, August 2002.

[45] Qingming Ma, Peter Steenkiste, and Hui Zhang. Routing high-bandwidth traffic in max-min fair share networks. In *SIGCOMM*, pages 206–217, Stanford, CA, USA, August 1996.

[46] Cecilia Mascolo, Lucia Capra, and Wolfgang Emmerich. Middleware for mobile computing (a survey). In Springer Verlang, editor, *Proceedings of the International Conference of Networking, Lecture Notes in Computer Science*, May 2002.

[47] Tommaso Melodia and Francesca Cuomo. Locally optimal scatternet topologies for Bluetooth ad hoc networks. In *Wireless On-Demand*

Network Systems 2004 (WONS'04), LNCS 2928, Springer-Verlag, Madonna di Campiglio, Italy, January 2004.

[48] György Miklós, András Rácz, Zoltán Turányi, András Valkó, and P. Johansson. Performance aspects of Bluetooth scatternet formation. In *MobiHoc*, pages 147–48, August 2000.

[49] Brent A. Miller and Chatschik Bisdikian. *Bluetooth Revealed. The Insider's Guide to an Open Specification for Global Wireless Communications.* Prentice Hall, 2nd edition, 2002.

[50] Matthew J. Miller, William D. List, and Nitin H. Vaidya. Hybrid network implementation to extend infrastructure reach. Technical report, University of Illinois, January 2003.

[51] Michael J. Neely and Eytan Modiano. Improving delay in ad-hoc mobile networks via redundant packet transfers. In *Proceedings of the Conference on Information Science and System*, Baltimore, MD, USA, March 2003.

[52] Elena Pagani, Gian Paolo Rossi, and Stefano Tebaldi. An on-demand Bluetooth scatternet formation algorithm. In Roberto Battiti, Marco Conti, and Renato Lo Cigno, editors, *Wireless On-Demand Network Systems*, pages 130–143. Springer-Verlag, 2004.

[53] Charles Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.

[54] Charles E. Perkins and Elizabeth M. Royer. Ad hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on*

Mobile Computing Systems and Applications, pages 90–100, New Orleans, LA, USA, February 1999.

[55] Chiara Petrioli, Stefano Basagni, and Imrich Chlamtac. BlueMesh: Degree-constrained multi-hop scatternet formation for Bluetooth networks. *Mobile Networks and Applications*, 9:33–47, February 2004.

[56] András Rácz, György Miklós, Ferenc Kubinszky, and András Valkó. A pseudo random coordinated scheduling algorithm for Bluetooth scatternets. In *ACM Symposium on Mobile AD Hoc Networking and Computing*, volume 99, pages 1–100, Long-Beach, CA, USA, 2001.

[57] B. Raman, P. Bhagwat, and S. Seshan. Arguments for cross-layer optimizations in Bluetooth scatternets. In *Symposium on Applications and the Internet*, San Diego, CA, 2001.

[58] Teemu Roos, Petri Myllymäki, and Henry Tirri. A statistical modeling approach to location estimation. *IEEE Transactions on Mobile Computing*, 1(1), January 2002.

[59] T. Salonidis, P. Bhagwat, L. Tassiulas, and R. LaMaire. Distributed topology construction of Bluetooth personal area networks. In *IEEE INFOCOM*, Anchorage, April 2001.

[60] Keith Scott and Scott Burleigh. Bundle protocol specification (v3). Internet draft, IETF, April 2004.

[61] Rahul Shah, Sumit Roy, Sushant Jain, and Waylon Brunette. Data MULEs: modeling a three-tier architecture for sparse sensor networks. In *Proceedings of IEEE SNPA Workshop*, Anchorage, AK, USA, May 2003.

[62] Internet Society IPN SIG. Interplanetary network project, Accessed: July 2003. Project homepage: http://www.ipnsig.org/.

[63] Ivan Stojmenovic. Dominating set based Bluetooth scatternet formation with localized maintenance. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, Fort Lauderdale, FL, USA, April 2002.

[64] Godfrey Tan. Blueware: Bluetooth simulator for ns. Technical Report MIT-LCS-TR-866, MIT, Cambridge, MA, USA, October 2002.

[65] Godfrey Tan and John Guttag. A locally coordinated scatternet scheduling algorithm. In *Proceedings of the 27th Annual IEEE Conference on Local Computer Networks (LCN)*, Tampa, FL, USA, November 2002.

[66] Godfrey Tan, Allen Miu, John Guttag, and Hari Balakrishnan. An efficient scatternet formation algorithm for dynamic environments. In *Proceedings of IASTED Communications and Computer Networks (CCN)*, Cambridge, MA, USA, November 2002.

[67] Amin Vahdat and David Becker. Epidemic routing for partially-connected ad hoc networks. Technical Report CS-2000-06, Duke University, 2000.

[68] Z. Wang, R.J. Thomas, and Z. Haas. Bluenet – a new scatternet formation scheme. In *35th Annual Hawaii International Conference on System Sciences*, Big Island, Hawaii, 2002.

[69] Kaixin Xu and Mario Gerla. A heterogeneous routing protocol based on a new stable clustering scheme. In *Proceedings of the IEEE Military Communications Conference (MILCOM 2002)*, Anaheim, CA, October 2002.

[70] Gergely. V. Záruba, Stefano Basagni, and Imrich Chlamtac. Bluetrees - scatternet formation to enable Bluetooth-based ad hoc networks. In *ICC 2001*, volume 99, pages 273–7, 2001.

[71] Wenrui Zhao, Mostafa Ammar, and Ellen Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *Proceedings of ACM Mobihoc 2004*, Tokyo, Japan, May 2004.