Technische Universität München
Fakultät für Informatik
Lehrstuhl für Betriebssysteme und Systemarchitektur

Dissertation

Mobile Services: A platform-independent structure for development and usage

von

Ulrich Dümichen

Betreuer: Prof. Dr. Uwe Baumgarten

München, Januar 2008

Institut für Informatik
der Technischen Universität München

Mobile Services: A platform-independent structure for development and usage

Ulrich Dümichen

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Die Dissertation wurde am 30.01.2008 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik
am 20.05.2008 angenommen.

Abstract

The science of computer engineering in wireless networks is growing more and more complex. This is the result of the fast evolving set of new communication technologies. Especially, since the integration of the Bluetooth technology and the IEEE 802.11 standard into cellular phones has taken place, nearly every user is able to build up short living wireless connections and to transfer data. In the near future, these kinds of connections will grow evermore complex and will be omnipresent at any time. Complex networks will be created, which will have to be organized very elaborately. Simultaneously, the usage of such networks for clients must be as simple as possible.

This thesis introduces a software system, named BlueSpot System, designed to organize such underlying wireless networks. By applying these networks, the system makes possible to use platform-independent communicative mobile services. A main aspect hereby is to provide a system with a well defined structure to which these services can connect to and be run independently from any underlying hardware and communication technology. Throughout the development process, the design of software interfaces became important in order to easily add new or adapt existing functionalities, such as various wireless routing protocols or QoS approaches. Additionally, self-management is included into the resulting middleware in order to have a fault-tolerant and easy-to-use system.

Acknowledgments

In the course of working out this thesis, many people played an important role. They greatly conduced to the work needed in order to bring forward the research efforts. Without their support, it would not have been possible to finalize this *Dissertation*.

First of all, I would like to thank Prof. Dr. Uwe Baumgarten for his profound advice and assistance. He was an incessant point of contact for me and always had an open ear for my questions. The same applies to my colleagues, who spent very much time with me at the *Lehrstuhl für Betriebssysteme und Systemarchitektur* of the TU München and were always responsive for interesting and in-depth discussions.

In addition to this, many students helped to finalize this thesis by writing their diploma or bachelor thesis or further kinds of works in the field of the BlueSpot system. Without their help, the implementation of the demonstrator would not have been possible. We all will remember the hours we spent in the meeting room discussing new approaches and how they can be brought to fruition.

A big thanks to Prof. Dr. Gerhard Mensch for hours and hours of brainstorming. But most of all, I am very grateful to my fiancée Tina-Marianne Mensch. She was responsible for the proofreading of this thesis as well as raising my spirits the last months. Whenever she could, she encouraged me to continue.

München/Garching, January 2008

Ulrich Dümichen

Contents

Chapter 1

Introduction

Contents

1.1. Thematic Context

The theory behind wireless networks is growing more and more complex, triggered by new realities from the fast evolution of new communication technologies. Especially, with the recent integration of the Bluetooth [Bluetooth SIG, 2007] technology or the IEEE 802.11 [IEEE 802.11, 2007] standard into cellular phones, nearly every user is able to build up short living wireless connections and to transfer data. In the near future, the trend is that these "emergent"' kinds of connections will grow even more complex. They will be omnipresent in time, especially in the category of so-called wireless information networks, on which we focus in this work.

On the one hand, through the emergence of temporary, short-living wireless connections, even more complex networks will be created which will have to be organized very elaborately. On the other hand, the application of such networks for great numbers of participants must be as simple as possible. The trade-off between these requirements is why we focus our work on this topic, as it necessitates theory building and reality building simultaneously.

1.2. Scope and Contribution

This thesis introduces a software system designed to make the use of platform-independent communicative *mobile services* possible. A main aspect hereby is the providing of a well defined structure in which these services can be placed on top of and be run independently from any underlying hardware. Additionally, this software system is able to build up an autonomous wireless network which operates using the currently widely spread Bluetooth and the IEEE 802.11 standards.

Throughout the development process we emphasized the necessity of the system's easiness of installation and application. Also, we concentrated on designing software interfaces in order to easily add new or adapt existing functionalities. These functionalities are mobile services, which are seen as *external extensions* and adaptive behavior modules, which are seen part of the network self-organization and thus are *internal extensions*.

The name of this software system is the *BlueSpot system*. Chapters 3 and 8 are devoted to its theoretical understanding.

1.3. Aims of this Thesis

Currently available mobile devices are very heterogeneous concerning their hardware, services, and user interface. Almost every device is based on a different hardware platform and has a different configuration concerning its attached peripherals. In this thesis we concentrate on a selected set of these devices such as smartphones, PDAs or embedded devices. We also included x86-compatible desktops as an additional testing environment. To be able to provide a *homogeneous communication platform for mobile services running on those devices*, it was necessary to identify a subset of corresponding supported peripherals and technology standards all applied devices provide. In particular, for the choice of the underlying wireless communication standard, only the Bluetooth standard and the IEEE 802.11 standard came into question, as these two standards are currently the most common ones.

One aim is purpose: the purpose of the BlueSpot system is to provide an environment in which as many capabilities of those devices as possible can be brought to one standardized platform to support mobile services in a mostly general manner. To this end, the resulting system must be viable and adaptable to a wide-reaching spectrum of requirements. It must endeavor to build up a wireless network between those devices, which allows them to communicate with each other, as well as to exchange data. The function of a mobile service is not only limited to the local device it runs on, but it is also extended to be able to communicate with other mobile services running on other devices within the BlueSpot system. The underlying wireless network is held completely transparent to the user of the mobile service, so that he does not need to have any knowledge of network configuration or of organizational issues involved.

Another aim is experimental: in terms of computer science, the BlueSpot system is employed to demonstrate a platform that allows to examine and compare existing as well as new network self-organization concepts. It includes well defined interfaces where additional software modules can be added or existing ones can be replaced. That way, the BlueSpot system is kept as general as possible so that even the underlying communication technology, which is used for networking, can be replaced very easily by another, not yet existent network technology.

Owing to the generality of the concept, the main software module needs no modification, but special features of a new technology can be very easily added via the provided interfaces. E.g. the master-slave allocation of the Bluetooth standard was included into the networking module of the BlueSpot middleware without any loss of generality. In doing so, network forming algorithms can be implemented in a general manner, and they can additionally take advantage of this special characteristic of Bluetooth. On all supported devices we used a general Bluetooth stack. It was very important for the design of the BlueSpot system that the stack does not have any restrictive requirements on existing hardware drivers and supporting software. Hence the used Bluetooth stacks did not need to be modified for application within the BlueSpot system.

A third aim is to understandability: in order to make the BlueSpot system even more clear, we implemented several different kinds of mobile services which are used to represent the varying requirements mobile services can have. The most illustrating service is a small remote controlled car. We replaced its receiver with an embedded computer. This computer was later integrated into the BlueSpot system, which made it possible to steer the car via the entire network. The controlling of the car could also be performed by any device that had a corresponding car steering service installed on it. As an extra demonstration we supplemented a second remote controlled car with a camera mounted onto its roof. The video of the camera is streamed via the BlueSpot system, so that a user can see where the car is driving to and instantly watch the video on the display of his device.

Finally, the aim of this thesis is to present the concept of a generalizable software system: a system that enables the inclusion of a far-reaching spectrum of hard- and software technologies, network mechanisms, and large variety of mobile services. We had to show that the resulting system is viable when generalized. With the resources at hand, the investigation of an all-encompassing software platform that enables the proving of services was made possible. As can be seen in chapter 7, our first attempt of a fairly general description of the BlueSpot system went well, but, admittedly, a more general approach is and will always remain a vision. With the BlueSpot system described in this thesis, we tried to provide a solution that goes one step further in this direction, yet many appear to be possibly feasible at a later time.

In order to demonstrate the current system's capabilities, we investigated the distinctive behavior of Bluetooth-base Scatternets with M/S and S/S bridges in great detail. By use of the system, we were able to show that S/S bridged Scatternets provide a much better scalability than M/S bridged ones. For the detailed description of these experiments we refer to section 9.2.5.

1.4. Structure of this Thesis

This thesis is divided into three parts. Part one provides all the fundamentals needed to understand the BlueSpot system, and is built up as follows:

Starting with the motivation, we will describe the underlying problematic this thesis points at. We will therefore introduce example scenarios the BlueSpot system was designed for. For a better classification of the field of application this thesis aims at, related work will be shown and its accordances and diversities in contrast to the BlueSpot system discussed.

The subsequent chapter 3 will provide necessary definitions that are fundamental for the research and development work in this thesis. Starting with a formal definition of wireless network configurations and how they can be described as mathematical edge graphs, wireless networks will be classified concerning their characteristics. The communication model of the BlueSpot system will be explained by use of the previously presented network classification. In order to understand the software model, a short introduction into a general description of software models will be given. The main software components that came to application within the BlueSpot system will be introduced and formally defined. By transferring these elucidations onto the BlueSpot system, this chapter is rounded up by depicting the software model of the BlueSpot system in detail.

Based on the underlying network topology, each network needs a mechanism that is responsible for the end-to-end communication. Otherwise, communication would only be possible between two neighboring nodes. The fourth chapter contains all information needed to understand such mechanisms. It classifies wireless routing algorithms concerning their properties and shows an example for each class in more detail. The present classes of wireless routing algorithms are pro-active and reactive ones. These two class distinctions characterize such algorithms. In addition, more specialized approaches such as hybrid and flow oriented algorithms will be explained. Finalizing the chapter, further specialized algorithms will be addressed which establish the far-reaching spectrum of possible solution approaches that can be expected to emerge in the near future.

The last chapter of this first part, chapter 5, will present the main characteristics of the two supported communication technologies Bluetooth and IEEE 802.11 WLAN. Especially, in case of Bluetooth, the underlying network structure with its distribution of roles must be understood in order to enable the description of further investigations that will be made in the later chapters of this thesis. In addition to the used communication technologies, the hard- and software products that served as basis for the BlueSpot system will be discussed. Each class of hardware will be presented by describing its architecture, the installed operating system as well as the needed extension software modules.

The second part of this thesis begins by introducing the BlueSpot system and explaining in great detail in chapter 6. Beginning with the definition of the requirements of the BlueSpot system's software architecture, different views of the entire system's schematics will be presented. These serve to describe the points of integration for further adaptivity extension modules as well as the structure of the implemented middleware with its various system layers.

In chapter 7 we will present all aspects concerning the self-organization functionalities that are needed to control the underlying network. Therefore, alternatively possible Bluetooth network topologies will be shown and their properties discussed. In order to provide an algorithm that is responsible for the automatic network establishment, subsequently, general network formation approaches will be shown that are based on the usage of selected edge graph constellations. Once the network is formed, we will demonstrate how this formation can be modified to full-fill changing or new requirements of on-top running mobile services. We will show up the adaptive behavior of the BlueSpot middleware in correlation with the several existing software interfaces for extension modules. The last point in this chapter regards mobility aspects, such as finding new clients or the appearing of handover events.

In chapter 8 we will classify mobile services according to their characteristics and to the requirements they place to the underlying system. After presenting a collection of parameters that categorize the needs of mobile services, we describe the implementation of this collection within the BlueSpot middleware. At the end of this chapter we demonstrate several different example implementations.

Chapter 9 contains selected simulation and benchmarking results we collected in the course of our investigations of the BlueSpot system. firstly, we will introduce the monitoring tool used to display the actual behavior of the BlueSpot system during runtime. Additionally, we will describe our testing environment running on an NS2 simulator with its various different network constellations. Thirdly, comparable benchmark results gained during simulation and real life testing proceedings will be presented and unexpected behaviors of Bluetooth-based networks we detected will be discussed.

The last chapter of this thesis, chapter 10, gives a conclusion summarizing the results of the entire thesis and pointing out the scientific advances made during our theoretical and empirical investigations. It closes this thesis with an outlook on open questions concerning the BlueSpot system as well as future wireless network technologies.

The appendix makes up the third part of this thesis. It contains the short abstracts of all student works conducted in the course of the development of the BlueSpot system. All these works can be seen as milestones that helped to compose individual approaches and ideas to a complex software system. In addition to this, the most important software interfaces are presented in source code. This is used to provide the reader with a better understanding of selected functionalities of the BlueSpot system.

Chapter 2

Motivation and Related Work

Contents

2.1. Motivation

In this chapter we wish to explain the motivation for beginning the research and development that led to the BlueSpot system. Research in the field of wireless networks is plenty. But there are many open questions still to be investigated. With the aid of the BlueSpot system we desired to provide a self-organizing platform, which on one hand enables any research using wireless networks. On the other hand, the system serves as starting point for our own research activities on selected problems within wireless networks and mobile service providing. There are questions such as "How must a system be constructed in order to manage or organize itself?". Another question asked is: "How can the performance be increased during run-time?". Current mobile devices - the main clients of wireless networks - are very heterogeneous concerning their hardware. This leads to questions such as "How can we simplify the providing of mobile services for such kinds of devices?", and "What are the abilities of those devices, if they get connected to a wireless network?".

These and other questions motivated us to construct the BlueSpot system. A preliminary investigation led to the result that the posed questions can be classified into two categories. The first category covers questions concerning the underlying wireless network, the second those related to mobile services. Classified this way, a simple division of the system's components emerged, as can be seen in the figure 2.1.

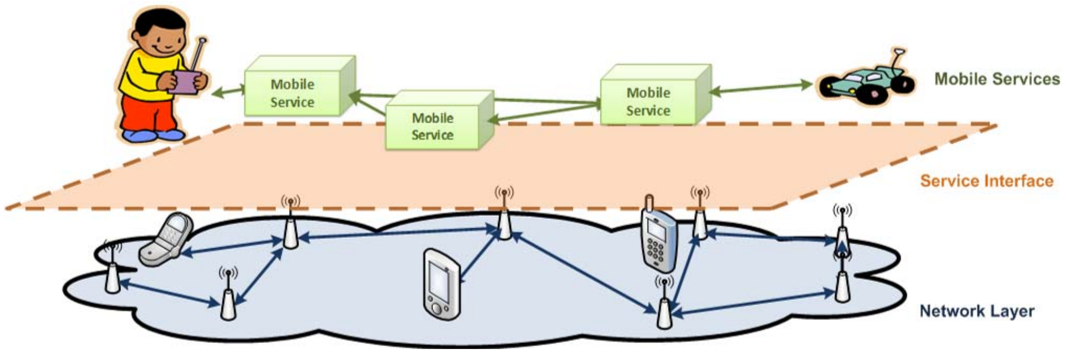


Figure 2.1.: Schematic Diagramm of the BlueSpot system

The diagram shows that the BlueSpot system is divided into two independent layers: the mobile services lying on-top (depicted as green boxes), responsible for the type of utilization, and the underlying network components (depicted as blue cloud) used to build up the formation of the wireless network. The service interface is responsible for providing a standardized platform on each device the mobile services can use to connect to (depicted as orange rectangle).

A whole variety of wireless networks and service systems exist to which the scheme applies. In accordance to the utilization of the system, it is necessary to provide adapted services in order to meet the requirements of a user. When installed in a museum, for example, the BlueSpot system could be used to offer information to visitors about the exhibit a visitor is standing in front of. The visitor's current position within the museum could be calculated using the network connections his mobile device currently holds upright. A second possible mobile service could act as tour guide, leading the visitor on his path throughout the museum. In advance to his tour, the visitor could choose the exhibits he would like to see, which are of special interest to him. After the visitor has completed his choices, the service would calculate the shortest path through the rooms of the museum and start the guided tour.

Another exemplary field of utilization of the BlueSpot system is sensor networking. After equipping every device within such a network with one or more sensors, the measured value would be propagated by the system. If a centralized instance is placed inside the network, all data could be collected and evaluated there. This arrangement could, for example, be used to observe environmental parameters, such as the detection of toxic gas concentrations in the air.

With the BlueSpot system, an elaborate experimental environment can be provided. Our first area of interest was to invent such an experimental environment for comparing new and alternative approaches of wireless networking to existing ones. By adding a platform for mobile services to the system, we were able to simulate any kind of utilization. Normally, this can also be achieved by any Java middleware. But we went further. Exceeding Java, the BlueSpot system has the great advantage of building up and organizing a wireless network entirely on its own. The resulting communication links to all other involved participants are then provided to the on-top running mobile services in a technology-independent manner. This way, the network forming process runs completely transparent to both mobile service and the participants.

The second area of interest is the underlying network layer. With this layer we are able to demonstrate various network topologies, such as a spanning tree formation, or a linear order independent of the communication technology in use. Due to its immanent generality, the BlueSpot system allowed us to add new network organization algorithms. These added algorithms can be tested and compared to exiting ones, and thus be verified. Along with these types of algorithms, the BlueSpot system can also be extended by different routing algorithms or adaptive behavior mechanisms. The latter mechanisms are used to alter the behavior of the entire system in accordance to changes of the mobile service's requirements. An example for such a change can be an increase of the needed bandwidth-throughput after a user has initiated the transfer of a large file.

2.2. Related Work

State of the art wireless networks are currently in use in four fields of application. One field are wireless networks that consist of *wireless integrated network sensors* (WINS) [Asada et al., 1998]. The sensor nodes are highly integrated, and in most cases, they do not have their own power supply. They obtain their energy by induction current that is emitted from a central master.

A second field of wireless networks concerns *wireless sensor networks* (WSN) [Szewczyk et al., 2004]. They consist of many small nodes that are organized in order to cover and observe a large geographical area. Each node contains its own logic and tries to connect to a neighboring node in order to establish a single- or multi-hop network (for wireless network types classifications see section 3.2). They are distinguished by focus: by the dedication to a specialized purpose.

The BlueSpot system presented here belongs into a third category. Wireless networks in this class aspire to provide more complex services; complexity being characterized by: 1) streaming versus message based communication, 2) critical latency times, 3) critical bandwidth, and 4) client-server communication versus peer-to-peer communication. The detailed characteristics of such services are described in section 8.1. Due to their more complex requirements, the underlying hardware needs to be equipped with a larger amount of resources, and the providing wireless infrastructure needs to be managed more extensively. Since about the mid 1990s, wireless networks within this third category are named *wireless information systems* (WIS) [Pahlavan and Levesque, 1995]. As an early example of this type of networks that deliver services based on produced information over an elaborately organized wireless network see Gerla and Tsai [Gerla and Tsai, 1995]. The BlueSpot system provides more potential for *variety* of future mobile services (generalizability at the service level) versus potential for *volume* on the level of one special service (specialization). This laboratory work at TUM focused on as much generalizability as possible.

A fourth field of wireless networks concerns providing of infrastructures for services with a certain degree of geographical coverage and depth. In this case the type of sub-services is not explicitly specified, but is mostly tied to telecommunication services or internet providing services. Examples for this field are GSM and UMTS networks as well as IEEE 802.11 WLAN or IEEE 802.16 WIMAX infrastructures.

A common feature of the entire wireless field of development is its expected unreliability. Traditionally, reliability is attained by more redundancies built into the system; see fundamental theorems about networks' and systems' reliability of systems with unreliable circuits, such as the theorems of Moore and Shannon [Moore and Shannon, 1956] and of Barlow [Barlow, 1968]. Here, in the BlueSpot system, we have implemented an alternative to redundancy: reliability is accomplished in the form of integrating adaptivity and self-organization capabilities into the software system. See chapter 7 for a description of theses proceedings. In this thesis we have reached a certain degree of integration that can be used to easily investigate more detailed explorations.

One intermediate result on our path to more integration was a completely new adaptive middleware approach. Most current approaches aim at extending existent middleware technologies. E.g. Blair et al. [Blair et al., 1997] described how to extend CORBA to gain better network adaptivity for multimedia applications. Their approach is to obtain a look into black-box systems to add special algorithms for different network bindings. A preliminary version of this intermediate result has been published as a contribution to the chapter *Including Adaptive Behavior in a*

Wireless Network System [Dümichen and Baumgarten, 2008] in the book of Huang et al. (editors) *Advances in Communication Systems and Electrical Engineering*, forthcoming Springer, 2008.

As has been sketched in the before mentioned pre-publication, the following three additional intermediate results apply:

In the quest for extending CORBA, a further-going approach than the one presented by Yau and Karim [Yau and Karim, 2004] applies. They extended CORBA for context-sensitive communication in ubiquitous environments. By doing this, the special requirements within ad hoc networks combined with the perception of context-sensitive sensors can be met with. But both approaches of Blair et al. and Yau and Karim have in common that they use an existing CORBA implementation and thus are not able to directly change the behavior of more than one node simultaneously in a network. Their point of view is to examine only one node and to make the best efforts to optimize its situation. The idea of the BlueSpot system is to move beyond the boundaries of such a single node. The complete system is involved in the adaption behavior, and thus a much larger spectrum of possibilities is available. This enlarges the set of feasible requirements of any one single service running on the BlueSpot middleware.

Regarding our advancement in adaptive software, McKinley et al. gave a detailed overview of different processes of composing such a software [McKinley et al., 2004]. This serves to see what we have accomplished. In order to classify the kind of adaption it describes two different processes: the *parameter adaption* and the *compositional adaption*. The first one focuses on an advancement of performance by changing predefined parameters. As described in section 7.4.2, the BlueSpot system provides several possibilities to adjust the current behavior of the system for a better performance. In case one of these adjustments is to load another adaptivity extension module (see section 6.2.2), a new code is loaded and thus new algorithms are included into the middleware. This correlates to the compositional adaption defined by McKinley et al. [McKinley et al., 2004].

Further, McKinley et al. [McKinley et al., 2004] examined the different elements of an adaptive middleware. This idea goes back to Schmidt [Schmidt, 2002]. Schmidt divided a middleware into four layers: *Host-infrastructure middleware*, *Distribution middleware*, *Common middleware*, and *Domain-specific middleware*. McKinley et al. [McKinley et al., 2004] denote this distinction of four types of middleware to be responsible for bridging the gap between an application program and the underlying operating systems, network protocols, and hardware devices. The BlueSpot middleware stack described in section 6.2.2 can also be linked to this decomposition. The Network Adapter Interface and the Network Adaption Layer in the BlueSpot system correspond to the Host-interface middleware. They cloak the heterogeneity of the underlying network devices. The Protocol Layer and the Session Layer fit to both the Distribution middleware and the Common middleware. They handle fault tolerance as well as high-level programming abstraction to enable developers to write distributed applications in a way similar to stand-alone applications. The Service Interface implements parts of the Distribution middleware as well as of the Domain-specific middleware. Consequently, a whole range of already existing and not yet existing services can thus be connected to the BlueSpot middleware by implementing the functions of the service interface. Therefore, even in its current early version, the BlueSpot system's potential application range is larger.

Part I.

Fundamentals

Chapter 3

Definitions and Model

In this chapter, the underlying assumptions and models for this thesis shall be described. The first section includes the abstract presentation of a wireless network as a mathematical graph. The following section provides an overview of how wireless networks can be classified according to their properties. Next, we will describe the underlying model of the BlueSpot system using the terms introduced before. The chapter is rounded up by additional definitions necessary for understanding the BlueSpot system, ending with a short summary.

Contents

3.1. Defining a Wireless Network as an Edge Graph

Analogically to wired networks, a wireless network can be presented as a mathematical non-directed edge graph. Such an edge graph $G = (V, E)$ consists of nodes, which are also called vertexes $V = \{v_1, \dots, v_n\}$, and edges $E = \{e_1, \dots, e_m\}$ where

$n \in \mathbb{N}^+$ is the amount of nodes, and $m \in \mathbb{N}$ the amount of edges of the graph G. All nodes are distributed within a two dimensional euclidean plane. Therefore, each node has an *absolute position* that describes its position in relation to the point of origin and a *relative position* that determines its position in relation to its neighboring nodes. The *geographical distribution* of the nodes is, as a result, defined by use of the absolute positions, whereas the *relative distribution* of nodes describes the distribution in dependence to other nodes.

A node $v_i \in V$ represents a wireless network device, while an edge $e_{ij} \in E$ represents a communication link between the two nodes v_i and v_j. This definition represents a static snapshot of a network situation. Contrary to wired networks, wireless networks are dynamic and can change over time. Therefore, we need to introduce the parameter t as a time factor. With the use of this parameter we are able to extend the definition of the edge graph G to $\mathcal{G} = \{G_{t_0}, \ldots, G_{t_N}\}$ with $G_{t_i} = (V_{t_i}, E_{t_i})$ that represents the snapshot of $\mathcal{G}$ at time t_i. t_N represents the maximum lifetime of $\mathcal{G}$. Without loss of generality, in the following we will use G instead of G_{t_i} for a more simple understanding. This way, G will always represent a snapshot in the following.

Similar to a wired network, when examining wireless networks, we must differentiate between a communication *link* and an established communication *connection*. We must consider the medium "air", which is used here to transfer data from one node to another, as a shared medium. Every node within the range of a communicating network device's radio is able to listen to it. The moment a node perceives a neighbor that is also able to participate in the network, it will indicate that it is able to build up a link to this neighboring node. Considering the ISO-OSI stack [Zimmermann, 1980] of the used network device driver, a link is established by the data-link layer of the driver. After two nodes have established a link, it is necessary for them to build up a connection in order to exchange data. This is done by the network and the transport layers. Taking this into account, it can be see in figure 2.1 in chapter 2 that the nodes have established a entirely connected graph, illustrated in blue color.

Generally, an edge graph can be used to depict a wireless network in three different layers: an edge graph that represents the links, an edge graph that contains the established connections, and an edge graph that illustrates the established communication paths as they will be described in section 4.1. These three types of edge graphs are shown in figure 3.1. We define a link-based edge graph as a non-directed edge graph $G = (V, E)$, as we assume that an edge e_{ij} between the two nodes v_i and v_j exists in the moment in which each node is within radio range of the other. That way, links are seen as symmetrical. An edge graph that illustrates the established connections between the nodes can, by contrast, be directed as well as undirected. To indicate that an edge e is directed from nodev_i to node v_j, we use the following depiction: $\vec{e_{ij}}$. A directed graph is especially useful for describing Bluetooth-based Pico- and Scatternets (see section 7.1.1). In this case, the direction indicates from which node to which node the connection was established. The third layer that concerns communication paths is yet again undirected. It describes established end-to-end communication paths, as they are defined in section 4.1.

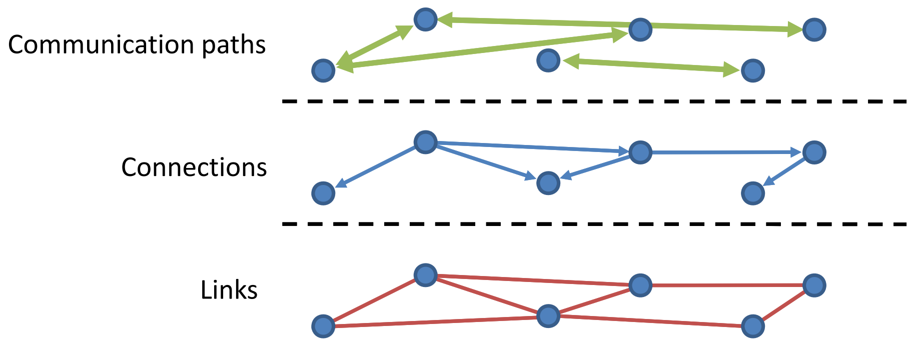


Figure 3.1.: Three layers of presenting a wireless network by use of an edge graph

In the following, the term edge graph is usually used for representing connection establishments, in case nothing else is denoted.

3.2. Classification of Wireless Network Types

In order to be able to classify different types of wireless networks, it is necessary to investigate the properties of such networks. In general, wireless networks can be divided into two categories. One category are *ad-hoc networks* that are the result of having only one kind of nodes. In this case, these nodes are mobile, and they simultaneously act as a client - which is a network customer - as well as as a router - which is responsible for forwarding data to other network nodes. Ad-hoc networks again can also be divided into two different types: the *entirely connected* networks and the *opportunistic* networks. Entirely connected ad-hoc networks are defined by the existence of at least one path with the length $l \geq 1$ between two nodes v_i and $v_j \in V$ in G at any time t_i.

In opposition to this, opportunistic networks consist of nodes that move around, and that build up temporary connections to their neighbors. After two nodes have exchanged their data, they will either close the connection, or one will exit the communication range of the other node, and hence loose the connection. Therefore, connections are always of a short duration. Furthermore, entirely connected networks can be separated into single-hop and multi-hop networks. In case of a single-hop network, all nodes of the network are within communication range of each other, while in a multi-hop network, data must be routed by nodes located within a communication path. As a result, communication paths within a network in general have the length $l = 1$, while in a multi-hop network the paths have the length $l \geq 1$.

Beside ad-hoc networks there is the class of *infrastructured networks*. These are networks established by nodes - the infrastructure nodes - installed especially for this task. These nodes are responsible for the network management, while a second kind of nodes - the client nodes - are the customers of the network. The latter use the network for communication between one another and are not responsible for data forwarding. The infrastructure nodes build up an entirely connected network. The client nodes function as the endpoints of all communication paths with at least one infrastructure node in between. For this reason, a path in G always has the length $l \geq 2$.

Similar to entirely connected ad-hoc networks, infrastructured networks can be divided into single-hop and multi-hop networks.

Hybrid solutions of each of the described networks are possible by combining different networks. The complete described classification can be seen in figure 3.2.

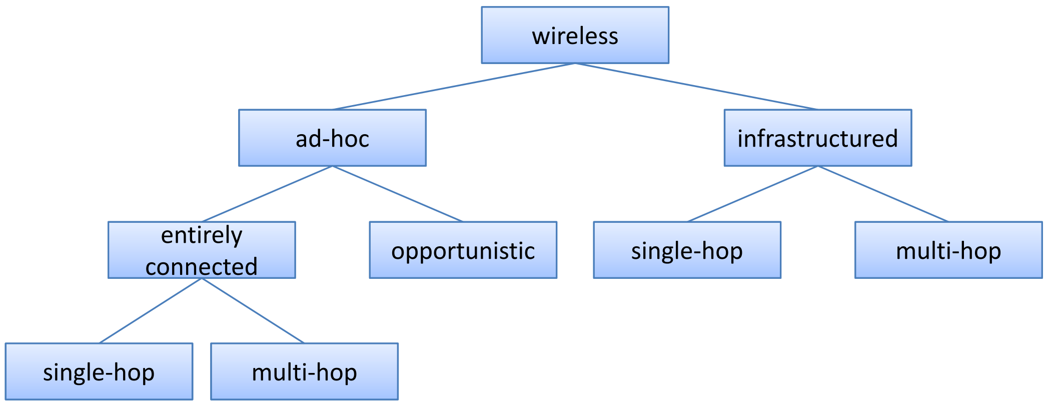


Figure 3.2.: Hierarchical classification of wireless network structures

In literature we can find several additional, more general, ways of classifying wireless networks. E.g. Günes et al. describe the structure of a wireless network as shown in the following [Günes et al., 2007]. They begin the classification by dividing wireless networks into three different categories, depending on their sizes:

- wireless wide area networks like GSM
- wireless local area networks like IEEE 802.11 in infrastructure mode
- wireless networks without infrastructure called MANETs

Within the first two kinds of networks, the only existing wireless links are those between the mobile device of a user and a base station. The remaining network, called infrastructure, is mostly wire based. In contrast to this, the communication in a MANET [Perkins, 2001] is accomplished completely wirelessly. The network is capable of organizing itself, i.e. the forming and routing is done in a decentralized manner and autonomously by each node.

Based on these different types of wireless networks, Günes et al. describe a *wireless mesh network* (WMN) as a composition of the components of all three categories. They introduce a *Backbone Mesh Gateway* (BMG) that acts as gateway between the internet and the mesh network. A BMG is connected to the internet by a wire, while the link to the mesh net is done wirelessly. To span up the mesh infrastructure, several *Backbone Mesh Routers* (BMR) are used. They form the backbone of the network by transporting data between the Clients and the BMG. Also wirelessly connected, the BMR form the coverage area, to which *Routing Mesh Clients* (RMC) as well as *Non-routing Mesh Clients* (NMC) can connect to.

The figure 3.3 points out how Günes et al. compose a mesh net using the described components.

In addition to the different components, Günes et al. describe the most important properties of a mesh network:

- Wireless: The characteristics of the used technologies within a wireless network lead to various restrictions. These include the need of high fault reliability and short communication distances.
- Multi-hop communication: Similar to our definition, multi-hop communication is used for end-to-end communication. It is needed because in most cases the two communication partners are not within each other's communication range. Other network participants are needed to forward the data and thus to build up a communication path. On the one hand, this makes the mesh network more flexible, on the other hand, the restrictions due to the wireless communication

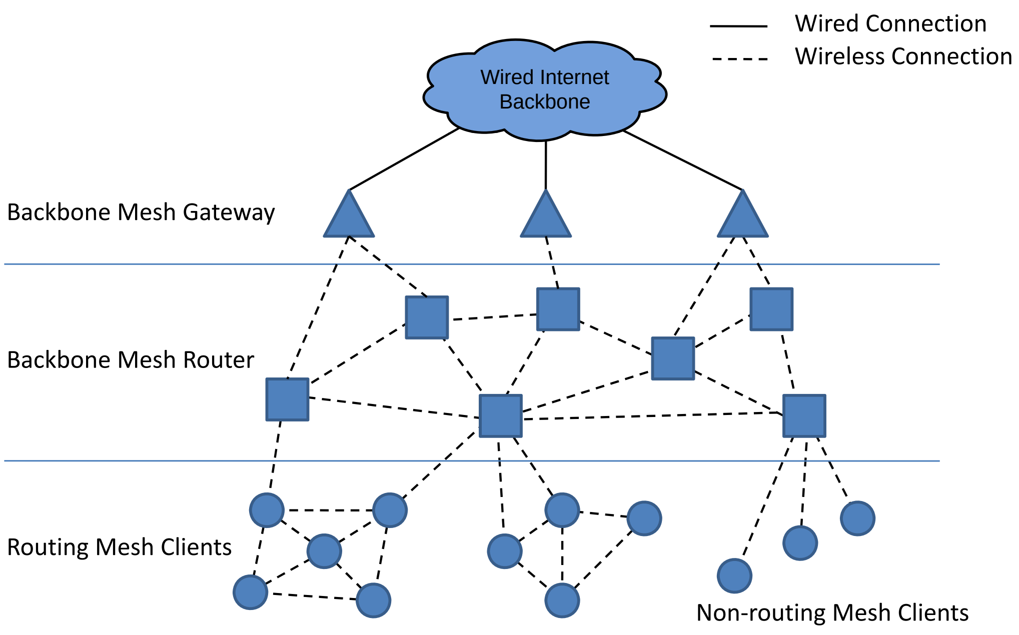


Figure 3.3.: Classification of a mesh net by Günes et al.

technology are enforced.

- Redundancy: As a rule, the BMR form a network with a highly redundant infrastructure. This is important to be able to overcome connection failures. A broken communication path can easily be replaced by an existing alternative path without any need of additional resources.
- Mobility: According to the definition of a WMN, the infrastructure (BMG and BMR) of such a network is static. Mobility is added to the network by the clients. They are able to connect to the BMR while moving around within the coverage area of the WMN, or additionally, form an ad-hoc network by connecting to other RMC.
- Dynamic and extensibility: two of the main properties of WMN. These are gained by adding self-capabilities to the network, such as self-configuration, self-organization and self-management. With the aid of self-configuration, a mesh network is able to configure itself automatically. This is necessary, as the costs for configuration increase quickly with a rising number of nodes. The same applies to the self-organization capability. In addition to this, the

self-organization capability is needed to achieve the wanted dynamic of the network. If a communication path brakes, the network itself is responsible of finding an alternative path. The self-management enables the network to handle its available resources itself.

- Infrastructure: According to our definition, a mesh network has an infrastructure. In opposition to ad-hoc networks, this leads to a high reliability for services like DNS and routing. Furthermore, the energy consumption of each node and the network density become less important due to the static placement of the BMR.
- Integration and convergence: Because of the ability to easily connect NMC to the mesh network, these nodes do not need to take over additional tasks for the network. This way, the integration of new client nodes bears no problems.

A property not described by Günes et al. is decentralization. Every wireless network - both ad-hoc and mesh networks - is always managed decentrally. The network has no central instance that is able to coordinate actions for the entire network. This way, decisions made by nodes are mostly local decisions, as nodes usually do not have the over view over the entire network. This should be kept in mind constructing a wireless network.

Mapping the mesh network above onto our definition of wireless networks, the mesh network is mainly an infrastructured multi-hop network, due to the static BMR and the BMG that are responsible for the infrastructure forming. The integration of the NMC matches our understanding of an infrastructured network. By adding RMC, the resulting network is a hybrid form of a wireless network, as RMC form an ad-hoc network within their coverage area. One of the mesh networks' properties is the multi-hop property. According to our point of view, a mesh network can also be a single-hop network. However, the qualifications of such a mesh network remain the same.

3.3. Communication Model of the BlueSpot system

With the definition of the edge graph, the two classifications of wireless networks, and the properties described in the sections above we are able to describe the underlying model of the BlueSpot system.

The network layer (refer figure 2.1) which is responsible for the network forming can be described as an edge graph with non-directed edges. Each node represents a participant of the network, while the edges represent the links built up between the nodes. A node can either be a client device, such as a PDA, a smartphone, or a laptop, or an infrastructure forming node. But they all have in common that they have a low performance and low resources.

Due to the distinction of two different kinds of nodes, we define the BlueSpot system as a mesh network with one restriction: only NMCs are possible. Clients that do routing are not available. The role of a BMG is not especially assigned because of the utilization of the BlueSpot system. Mobile services - that run on top of the BlueSpot system - are only available within the system. If a mobile service needs to communicate with another network or with the internet, the gateway functionality has to be implemented within the mobile service running on the selected node that has access to both networks. A difference to the mesh network definition is, that mobile services also can be run on the BMRs. BMRs are static by definition. They do not have any user interface, so that a user interaction is not possible.

Based on our definition of wireless networks, the BlueSpot system can be specified as a infrastructured multi-hop network. The system tries to form an entirely connected network at any time. It builds-up communication paths that are several hops long. The client nodes are consumers of the network and do not have any other task within the network. This is the definition of an infrastructured network.

The BlueSpot system is built up completely wirelessly. They underlying communication technology is not restricted to any special technology. In order to be able to demonstrate the system, we programmed reference implementations for both Bluetooth and WLAN. As an additional feature, the BlueSpot system supports handover, so that a client does not lose its connection to the network while it leaves the radio vicinity of the currently connected infrastructure node. By adapting to changing requirements of the running mobile services, the system has self-capabilities. E.g. it is able to modify its underlying network topology. This way, the BlueSpot system is dynamic and extensible. In order to integrate a new client node it is necessary to install the BlueSpot middleware on this node. After successful installation the node is immediately able to connect to the system and use the provided services. Network convergence is gained by the abstraction of the underlying wireless network technology. In our reference implementation we are able to interconnect both Bluetooth based nodes and WLAN based nodes into one network.

3.4. Model of the software system

After having described the underlying communication model of the BlueSpot system, we shall define its software system. To understand the concept in detail, it is necessary to take a closer look at software systems in general. After this, we shall give a definition of the most interesting components and map the architecture of the BlueSpot system to them.

3.4.1. General Description of a Software System

At first, it is necessary to discuss the architecture of a software system providing any kind of service in general. See figure 3.4 for a general structure.

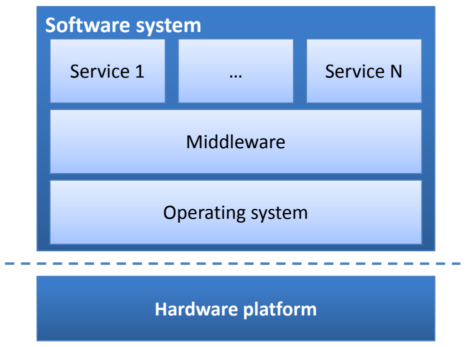


Figure 3.4.: Structure of a software system with services

Every software system is based on a hardware platform. In order to run programs such as applications or services on a hardware platform, an operating system is needed. The operating system supports programs by providing system calls. These system calls facilitate the access of hardware resources to the on top running programs. Therefore, it usually disables direct access to be able to manage race conditions and to prevent deadlocks between different programs.

By adding a middleware on top of the operating system, the access to the hardware resources grows increasingly more comfortable. Programs can function in a more abstract way, meaning that they do not need to be adapted to the underlying hardware platform. This way, one and the same program can be run on different hardware platforms and operating systems without any need of modification or recompilation. The only requirement is, that a ported version of the middleware is available for each used hardware platform.

3.4.2. Formal Definition of the Components

One main field of interest in this thesis is the deployment of a well defined software platform for mobile services. To have a common understanding of the term *service*, we shall quote a definition for it introduced by Dinkel et al. [Dinkel et al., 2006].

Definition 1:
Service

Definition 1: Service

A service is a set of functionalities that is realized with the help of a computer. A service has a set of users, who draw benefit from the offered functionality via a well defined interface.

The main point of this very general definition is that users draw benefit from a service. The meaning of the term *user* can either be a person interacting with the service or another software component communicating with it. A service provides a well defined interface to enable interaction with it, but it also uses an interface to interact with an underlying platform in turn. This interface is part of the software platform. Such a platform is defined by Dinkel et al. as follows:

Definition 2:
Service Platform

Definition 2: Service Platform

A service platform is a collection of infrastructure functionalities that simplifies the implementation of services.

Such a collection of functionalities can be part of the hardware, of the operating system, or of a middleware. In contrast to the functionalities of a service, the platform's functionalities facilitate a comfortable usage of the system resources.

If the underlying hardware is mobile, the service itself must be mobile, too. Dinkel et al. define the term *mobility* in context to a service as follows.

Definition 3:
Mobility of a Service

Definition 3: Mobility of a Service

An entity is mobile if it has the ability to change its geographic position over time.

In the definition, the term *entity* is used pursuant to two different meanings of mobility. On the one hand, a service can be placed statically on top of a mobile hardware platform. In this case the entire software system is mobile. On the other hand, the service itself can be mobile. This way, it can to be moved from one hardware platform to another. For this performance we use the term *service mobility*. In contrast to a mobile agent [White, 1997], such a service does not move itself from one platform to another. Rather, it is moved passively by a user, or by the middleware the mobile service is running on.

In this thesis "both" types of mobile services shall be utilized and investigated.

3.5. Software Model of the BlueSpot System

The main component of the BlueSpot system is its middleware. To be able to support several different hardware platforms, we ported this middleware to each of them. The middleware contains the service platform used to run the services on top of it. The main type of hardware platform the BlueSpot system was developed for are mobile devices, such as smartphones or PDAs. This way, it provides a platform for mobile services.

To enable communication to other network participants for mobile services, the middleware manages the connections to them. By doing this, it allows a mobile service to use these connections in a technology-independent manner. Regardless whether the underlying technology is Bluetooth or WLAN, the communication partner can be accessed by the mobile service in the same way.

If a user wants to use a mobile service which is currently not installed on his device, this service is automatically located within the network. After its localization, the mobile service will be transferred to the user´s device and started automatically by the middleware. This way, service mobility is gained for the BlueSpot system.

3.6. Summary

The main aspect in this chapter is to describe the underlying communication and software models of the BlueSpot system. For this, the structure of a wireless network is mapped onto a mathematical edge graph. Additionally, properties of wireless networks are discussed. These properties are used to classify different types of wireless networks. We shall introduce mesh networks and describe their components.

Afterwards, the communication model of the BlueSpot system shall be explained by the use of the introduced wireless network classification, properties and components.

Next, the software model shall be described. We will describe a software system in general, and then define the most important components for the BlueSpot system. By doing this, the terms *service*, *service platform* and *mobility of a service* are introduced. At the end, we shall explain how the middleware of the BlueSpot system can be understood by the use of the structure defined before.

Chapter 4

Routing Protocols in a Wireless Network

This chapter includes detailed information about routing aspects within wireless networks. After the forming of wireless networks has been explained in the last chapter, the next layer - the network layer - of the ISO-OSI stack shall be regarded. In order to achieve end-to-end communication, a fitting routing protocol is needed here. The routing protocol determines, on which path data is to be sent to reach its destination and how to find this path in the first place. As a result of the many different field of applications, there are various requirements to such a protocol.

First, this chapter points out the properties of wireless routing protocols. Afterwards, these properties are used to classify them. While performing this classification, every type of protocol is explained by use of an example.

Contents

4.1. Properties of Wireless Routing Algorithms

It is the task of the implemented nodes within the network to enable end-to-end communication to any other node in the system. The difficulty is that the node does not know how to find the right communication path to the designated endpoint. This is why there is the need for a mechanism responsible for path-finding through the network. This mechanism is named *routing* and is defined within the ISO-OSI stack on layer three.

End-to-end communication is enabled by the establishment of the before mentioned communication paths. A *communication path* is defined as the result of stringing together all connections established between nodes that lie in-between the endpoints. Prerequisite is, that links already exist among the involved nodes. Otherwise, connections cannot be built up.

Due to the different requirements of routing algorithms in wireless networks, they need to be handled more elaborately than algorithms for wired networks. Especially, because of having nodes that can move nearly without any restrictions through the entire network, the network's structure changes permanently. For this reason, static routing algorithms are not applicable. Such algorithms are not able to react to a topology change and hence are not able to keep established communication paths upright. But this is one of the main tasks of routing algorithms for a wireless network. They need to be able to constantly adapt to changes as fast as possible. This is the only way to always ensure the currently on-going communication.

By moving around within a multi-hop network, a node can loose its connection to its neighboring node anytime. In order to replace this connection it must establish a new one to an other node in range that is also part of the wireless network. Next, this event must be posted to the other participants of the network currently communicating with it or using a communication path the node is part of. Another characteristic of wireless networks are unstable communication links. Because of interferences or disturbances of the radio signal, a communication link can break off anytime. As a reaction to this, the lost link needs to be reestablished as quickly as possible in order to enable further communication. These two scenarios demonstrate that routing algorithms for wireless networks additionally need to have a high fault tolerance. These and further requirements of wireless routing algorithm are described by Al-Karaki [Al-Karaki and Kamal, 2004] in more detail. The results of the described characteristics lead to the following proposition:

The more dynamic a network is, the more frequently its routes change. But, the more frequently the routes change, the more actively the various participants of the *network need to take care of new communication paths.*

Consequence of this proposition is, that we are able to give a first differentiation of wireless routing algorithms. On the one hand, there are *pro-active* algorithms, that try to find routes to all other participants of the network immediately after they have joined the network. All other node recognize the new node and add it to their own routing tables. On the other hand, there are *reactive* algorithms. The moment a nodes must transmit data to another node, it starts to search for the best route. This way, routes are established on demand.

Further distinctions of routing algorithms are the geographic order of the nodes, energy efficiency of the algorithm itself, or the usage of existing information, such as the network topology or transmission statistics. The latter is called *flow oriented routing*. All types of routing algorithms can be combined with hybrid routing approaches, which are mainly the combination of reactive and pro-active routing mechanisms.

When contemplating the published literature concerning wireless routing algorithms, it becomes clear that there will not be one singular routing approach that will fit to all utilizations of wireless networks equally in future. Dependant from the application, it is always necessary to select the right network type, as well as the best fitting class of routing algorithms. During the installation of a wireless network, the properties of the area of application for this network need to be evaluated. These are used in order to find the best fitting algorithm for this configuration. The following section can be used to simplify the finding. With its aid, first the according routing class and second the best algorithm of this class can be found.

4.2. Classification

In the following we shall present the above mentioned classes of wireless routing algorithms and describe them one by one using an example for each. This way, pro-active routing shall be described using the *Destination-Sequenced Distance-Vector* (DSDV) routing. With the aid of *Dynamic Source Routing* (DSR), reactive routing shall be explained. As an example for hybrid routing, we shall introduce the *Zone Routing Protocol* (ZRP). The flow oriented routing we shall be presented by explaining the *Link life Based routing* (LBR).

Other classes are geographical routing, power aware routing and multicast routing. These classes shall be described by pointing out the main idea of each class in a summary at the end of this section.

4.2.1. Pro-active Routing

As the term *pro-active* indicates, these algorithms try to obtain an up-to-date list of paths to each node within the network before the next communication takes place. This way, routes must be determined right after a node has joined the network. The underlying problem can be solved by use of the Bellman-Ford algorithm [Cormen et al., 1990]. This algorithm finds the shortest path between each node in an edge graph. In order to transfer it to wireless networks, it must be modified to a distributed version. In most cases a central management instance is not available.

Considering the ISO-OSI stack, the network layer is normally responsible for routing issues. But in the field of wireless networks, routing algorithms are more and more often implemented within the data-link layer. The big advantage if this is, that wireless routing algorithms often must react on broken or newly established links. Contrary to wired networks, links cannot be seen as stable, due to moving nodes and fluctuating signal strengths. This way, the point of entry for path finding needs to be based on communication links, not on communication connections of the network layer.

As first step in direction of pro-active routing was done for wired networks by Halpern et al. [Halpern and Bradner, 1996] by defining the *Distance-Vector algorithm*. The core statement is: "Tell your neighbors your sight of the world". In detail, the algorithm works with metrics that represent the costs for the use of a connection. The lower the metric is, the better the connection is. A short description looks as follows:

1. produce a matrix of metrics to the other nodes
2. extract a vector with the best metrics and exchange this vector with the other neighbors
3. include the received vectors of the neighbors in the own vector
4. if the best metrics vector has changed goto step 2, else goto step 3

In order to be able to use the distance-vector routing for wireless networks, it needs to be modified. The resulting algorithm is called *Destination-Sequenced Distance-Vector* routing (DSDV) first described by Perkins [Perkins and Bhagwat, 1994]. The DSDV algorithm is the result of adding a sequence number to each entry in the metrics vector. The number indicates the currentness of the corresponding entry. In case of a no longer reachable node, the metric entry for this node is set to infinity, as defined by the distance-vector algorithm. Only if the node receives an updated vector from a neighboring node, where the metric is smaller than infinity and the sequence number is higher than the currently assigned one, the node stores the new entry. The availability of a new path to the up to then not reachable node is indicated this way. Furthermore, the sequence numbers are divided into even and odd numbers. If the node has found a new neighbor on its own, the entry in the vector is always assigned to an even sequence number. In the other case, if the new node was announced by another node, the stored sequence number is odd. When an entry is updated, the sequence number always is incremented by two. This way, the node is always able to differentiate between its own entries and information received from its neighbors.

Considering pro-active routing algorithms in general, they inherently have advantages and disadvantages in common. The advantages are a fast packet switching, as the route to a destination is well known from the beginning. Additionally, topology changes are recognized immediately. If a new node joins the network, its entrance and thus the path to it is announced throughout the entire network. This way, every node is noticed. As a disadvantage the pro-active routing does not scale very well. Because of having an entry in the distance vector for every node that is participant of the network, the size of vectors increase with the number of nodes. Additionally, the vectors must be exchanged with the neighbors. With a great number of nodes and hence large vectors, the traffic resulting from routing management increases dramatically. Closely linked to this disadvantage is the bad convergence of the algorithm that indicates the time needed to reach stable routing tables. Due to the need of extra data exchange to keep the distance vectors up-to-date, it takes disproportionally long time until the vectors of the single nodes are stable and need not be updated anymore.

4.2.2. Reactive Routing

The disadvantages of the pro-active routing algorithms were the motivation for the development of reactive ones. In order to be able to manage larger wireless networks, the needed information for routing must be reduced. Therefore, the path to the destination is determined the moment the data needs to be sent to it. This is called *on-demand routing* [Das et al., 2000]. The path is found by using a variant of the Dijkstra algorithm. Within an edge graph, this algorithm finds the shortest path between two nodes [Dijkstra, 1956]. After the path is found, it is added to the data and transmitted along with it. This way, each node that is part of the path between the source and the destination node is able to find the next hop by searching the additional path information coming with the data. The most common routing algorithm based on this approach is called *Dynamic Source Routing* (DSR) and was first described by Johnson et al. [Johnson et al., 2001]. This routing protocol was designed for the internet protocol and hence the routing information needed to find the destination node is stored within the header of an IP packet.

Similar to the above mentioned pro-active routing algorithms, the reactive ones have advantages and disadvantages. The advantages are - as mentioned before - that there is no need for continuously exchanging large amounts of routing information between the nodes. By needing less information to be stored on each node, these do not need to have such a high-performance as they do when working pro-actively. Additionally, they do not need as much memory for storing the routing information. Due to the reactive behavior the first time for transmitting data takes very long. The time is needed to first find the path to the destination, before the data can be sent. This is the main disadvantage of reactive routing algorithms. An extension of the DSR algorithm named DSRFLOW [Hu et al., 2001] avoids long waiting times while the nodes accomplish the path finding. This algorithm is part of the class of flow oriented routing algorithms that is described in subsection 4.2.4 in more detail. Another disadvantage of the reactive routing algorithm is, that, if the network topology changes, the single nodes are not notified concerning this event. As a result, data that was sent before a change can no longer reach its destination and thus needs to be retransmitted. This leads to a bad behavior for reacting to dynamics within the wireless network.

4.2.3. Hybrid Routing

Hybrid routing is the result of joining pro-active and reactive routing algorithms. By doing so, these approaches try to combine the advantages of both types of algorithms. Hybrid routing shall be explained in more detail using the *Zone Routing Protocol* (ZRP) - first described by Haas [Haas, 1997], [Haas and Pearlman, 2001] - as an example. This algorithm is based on the assumption that the most traffic occurs locally in direct neighborhood of the nodes. For this reason, the wireless network is divided into several smaller routing zones. Within these zones, a pro-active routing algorithm is used to guarantee a fast routing. Haas et al. recommend the use of the standard DSDV algorithm, but other pro-active algorithms can be used alternatively. For longer distances, beyond the borders of a zone, an additional algorithm is necessary. In order to reduce complexity, Haas et al. have developed a reactive routing algorithm, that constructs a hierarchical order of the various zones. The proposed algorithm is able to keep multiple paths to one and the same zone upright. The occurrence of a strong backbone forming is thus prevented.

The advantage of this approach is obvious: due to the forming of small routing zones, the distance vectors within these zones remain small. Therefore, pro-active routing is applicable. The usage of a reactive routing algorithm for inter-zone routing is also useful due to a higher stability of the zones. Even if the nodes within a zone are very mobile, it is unlikely that all nodes will leave the zone at once, which would lead to the disappearance of the zone. It is more likely that only a number of nodes leave the zones while new ones join it. This results in the zone being continuously available any time.

4.2.4. Flow Oriented Routing

As described in subsection 4.2.2, the most flow oriented routing approaches are an extension of reactive routing algorithms (but also the extension of pro-active routing is thinkable). By saving the link state to each neighbor, the reactive route finding is improved. Additionally, historical information based on the statistics of the traffic having occurred before can be used to select the optimal path to a destination node. The historical information can be latency times, throughput of data, or link stability.

As an example for flow oriented routing, we present the *link life based routing* (LBR) [Manoj et al., 2001]. To determine the best route to a destination node, a parameter called *link life* is calculated by the algorithm. This parameter is an indicator for the worst case expected lifetime of a wireless link. I.e. the longer the link has been stable, the better the value of this parameter is, hence the link is assumed to remain stable in future. In order to calculate the link life, the current link quality is measured. This is compared to the measurements accomplished before. By use of this comparison the actual value is calculated. This knowledge can be used to form a communication path through the network that is as stable as possible. Nodes with a high movement rate are omitted, as a result of their bad link life parameter. The underlying routing algorithm can be either pro-active or reactive. It was first described by Manoj et al. based on a reactive one.

Flow oriented routing algorithms enable routing in highly dynamic wireless networks in which a set of nodes exist, that do not move much. The occurring traffic will be handled by these nodes. With the usage of statistical information, predictions of future situations can be made. This way, the path finding for both pro-active and reactive routing approaches can be improved. As a disadvantage, the approach does not work very well when the system is first started. At that moment, no pre-knowledge is available. Therefore, the system has to be running for a while, until enough information has been collected.

4.2.5. Other Routing Approaches

Many other routing algorithms and approaches exist beside those mentioned before. Each implementation is customized to a set of scenarios an algorithm was designed for. The algorithms described before have in common that they do not need any knowledge of their geographical position. This is possible as every node can be identified by its unique address. In case only the geographical position of a node is known, another type of routing is needed. In the course of application of such a network, the position of a node must first be found. There are several ways this can be achieved. The simplest way is to equip the node with a positioning device, such as a GPS receiver. The disadvantage is, that GPS only works outdoors and is very energy consuming. As an alternative, another approach is the triangulation. Here, the node takes the bearings by measuring the direction and the strength of received signals from at least three different neighboring nodes. After this is done, the position can be calculated in relation to these nodes. After the node's position is known, this information can be used to route data in the direction the destination node is expected to be. An example for geographic routing is the *Greedy Perimeter Stateless Routing* (GPSR) [Karp and Kung, 2005]. It uses a greedy algorithm to locate the geographical destination for the data that is to be transmitted. When it reaches the perimeter of its destination, the receiver is able to collect it there.

Another point of interest is the power consumption accompanied by the routing efforts. Since, in many cases, the nodes are mobile and thus have a limited power supply, the power consumption should be as little as possible. This way, a long operating time of all nodes can be obtained. E.g. sending data is power consuming. The more routing information must be sent in order to keep the routes up-to-date, the more energy is used. This was the motivation to develop routing algorithms that need very little energy. An example for energy efficient routing is Span described by Chen et al. [Chen et al., 2002].

Sometimes it is needed to send data to more than one destination node. In this case, bandwidth of the network can be saved by using a multicast routing protocol. A group of receivers can be defined this way and the data needs to be sent just once. The underlying routing algorithm again can be both pro-active and reactive. One of the best known multicast routing approaches is the *Distance Vector Multicast Routing Protocol* (DVMR) [Waitzman et al., 1988].

4.3. Summary

It is seen as given that every kind of wireless network is used in its own area of application. With the different applications running on such a wireless network, the properties the network has vary from case to case. In order to emphasize this situation, we showed up the main differences between wired and wireless routing and point out the corresponding properties. Afterwards, by use of these properties, the best fitting class of routing algorithms can be selected in dependency of the area of application.

In order to enhance the understanding of wireless routing further, the most common classes are shown and an example is given for every class. Therefore, we start with the description of pro-active and reactive routing in detail. This is done by use of the DSDV and the DSR algorithms. In addition to this, hybrid routing approaches are shown. This combination of pro-active and reactive routing mechanisms conjoin the advantages of both algorithm classes. By using additional information, such as the link state between two nodes, the concept of flow oriented routing algorithms is described. At the end, the classification of wireless routing algorithms is rounded up by presenting the main motivations and concepts for geographical, energy efficient and multicast routing approaches.

Chapter 5

Technologies

In this chapter the technologies we used in the course of the development of the BlueSpot system shall be presented. For each technology, the main properties and especially the restrictions will be discussed in detail. The aim of this chapter is to give the reader the background information on what kind of hardware was used and what the main reasons for their selection were. Additionally, the reader will be able to understand some decisions made in the BlueSpot system, by reason of hardware limitations. The handling of these limitations will be described in the next chapter, where also the concrete design of the BlueSpot system shall be explained.

This chapter is divided into two sections: we shall start with the communication technologies used in this project. Here the relevant aspects of Bluetooth and IEEE 802.11 WLAN will be discussed. In the second section, the used hardware platforms shall be described. These descriptions are amended by the corresponding operating systems, running on each presented hardware platform.

Contents

5.1. Communication Technologies

A main objective in the course of the development of the BlueSpot system was to be as technology-independent as possible. But for a proof of concept, we needed to select some technologies for presentation. These technologies are Bluetooth and IEEE 802.11 WLAN. Both provide an implementation of a complete TCP/IP stack. But in case of Bluetooth, we forgo the usage of the TCP/IP protocol. Its utilization would reduce the low available bandwidth furthermore and a satisfactory operation would by unachievable. For this reason, the support for WLAN is implemented in terms of a TCP/IP based extension. In case of Bluetooth, we use the RFCOMM protocol as basis for implementation. By the usage of TCP/IP every other or new technology providing this protocol is automatically supported as well.

Next, the project relevant properties of the Bluetooth standard will be described. Afterwards, IEEE 802.11 WLAN will be introduced, divided into two subsections. The first shows the relevant properties of the communication standard, the second subsection discusses the configuration of TCP/IP according to the BlueSpot system.

5.1.1. Bluetooth

In 1994, a first standard for Bluetooth was published by the Bluetooth Special Interest Group (Bluetooth SIG). This group was formed by the companies Ericsson, Nokia, IBM, Toshiba and Intel in order to define a standard that enables the replacement of wired connections between computers and peripherals (e.g. keyboards, mouses or printer) by wireless ones.

Additionally, the standard included a definition for the formation of a Bluetooth based network that is valid till this day. In order to implement Bluetooth support in a system, the standard is constructed as a stack. Each layer is responsible for its especially defined tasks. This stack is named Bluetooth stack.

Figure 5.1 gives an overview of a Bluetooth stack. Beside various other profiles, the most relevant for the BlueSpot system are shown. The baseband is situated on top of the radio hardware as well as underneath the protocols. Its properties are discussed in the next subsection. The setup of the protocols will be described in subsection 5.1.1.

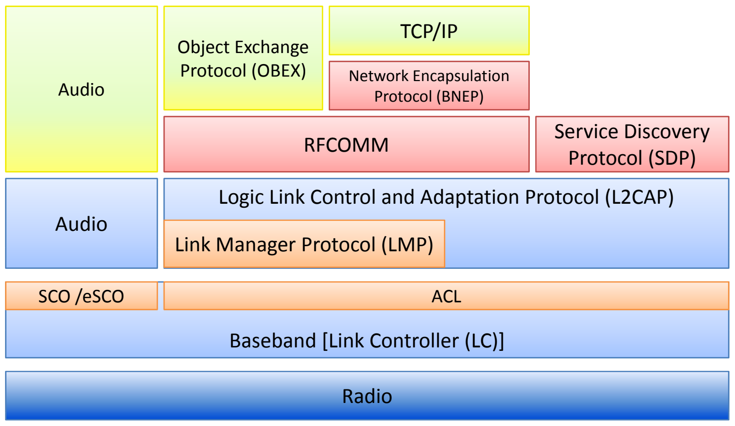

Audio
Object Exchange Protocol (OBEX)
TCP/IP
Network Encapsulation Protocol (BNEP)
RFCOMM
Service Discovery Protocol (SDP)
Audio
Logic Link Control and Adaptation Protocol (L2CAP)
Link Manager Protocol (LMP)
SCO /eSCO
ACL
Baseband [Link Controller (LC)]
Radio

Figure 5.1.: Bluetooth core system stack

Bluetooth Baseband

Bluetooth uses the ISM-Band located at 2,4 GHz, and currently has a maximum data throughput of 2Mbit/s (standard V.2.0+EDR [Bluetooth SIG, 2004]). The technique of communication is a Frequency Hopping Spread Spectrum (FHSS) approach, e.g. explained in detail by Schiller[Schiller, 2003a].

In order to construct a network, a node must actively build up a link to another node. After the successful link-establishment, the first node is named *master* node (M), the latter *slave* node (S). The link-establishment is divided into three steps. First, the master node broadcasts an *inquiry request*. This is answered by other nodes within communication range by sending an *inquiry reply* including their 48-bit Bluetooth-address. This step is named *scanning*. In the next step - named *paging* - the master creates a unique frequency hopping sequence calculated in dependency of the earlier received Bluetooth-addresses. The master sends this sequence to the other nodes. By use of this sequence, the other nodes are now able to synchronize to the master and communicate with it. This is the third step, named *connected*. The nodes are ready to communicate. The construction they have formed is named *Piconet*. It consists of at least one master and up to seven slaves. This way, the size of a Piconet is limited to eight nodes. Slave nodes are not able to communicate directly with one another. Each data exchange must be done via the master node.

If a node is connected, it can be in four different states in turn. These states are either *active*, *sniff*, *hold* or *park*. The first state is the standard state. If a node is in active state, it is able to partake the Piconet regularly. Therefore it obtains a 3-bit address, named *Active Member Address* (AMA) used to identify the node. Due to the length of this address - 3-bit - the size of a Piconet is limited to eight members.

The other three states are used to reduce the energy consumption of the node. If a node is in sniff state, it reduces the intervals in which it is listening to the master. This way, it can save power and bandwidth is freed for other nodes. In the hold state all active links are closed. This state is also used when the node leaves a Piconet to temporarily join another Piconet. The third state is the park state. If a node is in this state, it obtains an 8-bit long *Parked Member Address* (PMA) from the master instead of the AMA. Now the node is no longer able to communicate with other nodes in the Piconet, but it keeps the hopping sequence of the master, thus remaining synchronized to it. This is done by listening to the master periodically, in order to receive the clock signal. The freed AMA can be assigned to another node, such as a currently parked one. This way, it can switch back to active state. By use of the 8-bit PMA a Piconet can be extended to eight active and up to 256 parked nodes. All the Bluetooth states a node can be in can be seen in figure 5.2.

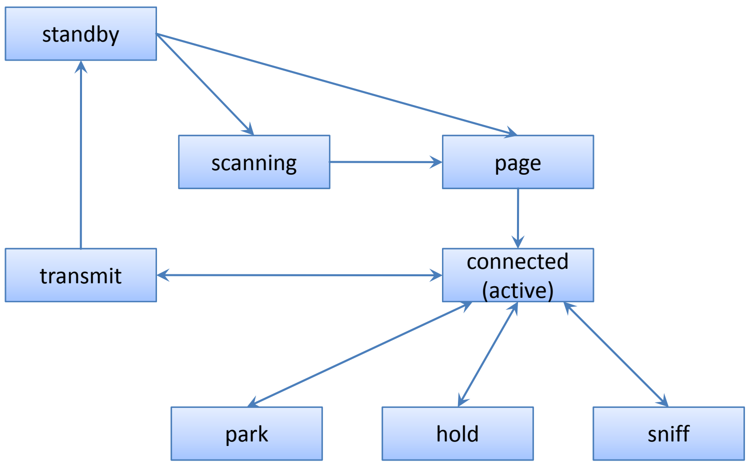


Figure 5.2.: Main states of a Bluetooth device at baseband [Schiller, 2003b]

In order to connect to more than eight active nodes simultaneously, several Piconets can be combined to a *Scatternet*. Therefore, a Scatternet is a composition of two or more Piconets. In order to make communication between two Piconets possible, a *bridging node* is needed. Such a bridging node is part of both Piconets and can either be master in the one and slave in the other (M/S), or slave in both Piconets (S/S). Due to the Bluetooth standard´s restrictions, a node can be active in only one Piconet. Therefore, it must be set in hold state in the first Piconet, in order to switch to the other. But this is precisely one of the main disadvantages of

Bluetooth Scatternets. Because of this restriction, the communication throughput within a Scatternet is reduced dramatically. Even worse, if the bridging node is a M/S node and it is currently active in the Piconet in which it functions as a slave, all nodes of the other Piconets are completely inactive, due to the missing master. In contrast to this, if the node is a S/S node, both Piconets can act regularly. But if the benchmarking results are compared, the communication throughput over a small amount of Piconets is higher by using M/S bridging nodes. But this changes the more Piconets are involved into communication. The results can be seen in chapter 9.2.5. This behavior can be explained as follows: in case a node joins a Piconet, it needs to synchronize its own clock to the master's. By acting as an S/S bridging node, it cannot be synchronized to both Piconets simultaneously. By switching to the other Piconet, it needs to resynchronize to the clock signal of the other master. Therefore, it is very time consuming, until the node is able start communication in the new Piconet. In another case - considering a M/S node - the node is the master providing the clock signal to one of the two Piconets. In case it switches to the second Piconet, within which the node acts as a slave node, the system behaves the same as a S/S bridged Scatternet. But the moment the node switches back to the first Piconet, in which it is the master, it does not need to synchronize anymore. The slave nodes of this Piconet are still synchronized to it and hence are able to immediately receive data from it. This way, the overall throughput of the Scatternet with a bound number of bridging nodes is higher by using M/S nodes than by using S/S bridges.

Bluetooth links can be divided into two different types: the *Synchronous Connection Oriented* (SCO) links and the *Asynchronous Connectionless Links* (ACL). As the names indicate, one type is connectionless, the other connection-oriented. SCO links are used for audio transmission. Therefore, it is important to have a constant data stream without any interruption. SCO links are installed to guarantee the bandwidth used for such a constant data stream. They provide a guaranteed data rate of 64 kbit/s. In contrast to this, ACL links have a data rate of up to 723,2 kbit/s (Bluetooth standard v1.1) or 2Mbit/s (Bluetooth standard v2.0+EDR). ACL links work packet based and thus the throughput rate cannot be guaranteed. But due to the higher data rate, ACL links are usually used. Additionally, currently there is no completely functioning implementation of a Bluetooth stack that supports SCO links in its full complexity. This can be traced back to the fact that even the Bluetooth standard itself leaves open essentials in the definition of SCO links. For example, most of the currently available headsets for mobile phones work with an ACL based protocol due to the missing SCO support. It transpired that the throughput rate is high enough to reach appropriate voice quality.

During the implementation of the BlueSpot system, we had to use several different Bluetooth stacks due to the different hardware platforms. On Linux based systems we used the BlueZ stack[Holtmann and Krasnyansky, 2007]. This stack is the one used most commonly for Linux based systems. On PocketPC based PDAs, a Bluetooth stack provided by the company Widcomm (today Broadcom [Broadcom, 2007]) is preinstalled. The Nokia smartphones include a stack implemented by Symbian [Sym, 2007]. All three stacks have in common that implementation of SCO links is either not included or not usable due to implementation bugs. For this reason, we had to forgo the usage of SCO links.

Bluetooth Protocols

Beside several other features, the Bluetooth baseband provides the features link establishment and network formation. In order to use a link, there is the need to select one of the many profiles defined for Bluetooth. A profile is used to combine a protocol with the configuration information needed to run the protocol properly. By exchanging of the list of the supported profiles, a communication partner knows the supported profiles as well as the corresponding configuration. In the following, we will focus on the protocols that are important the utilization of Bluetooth for the BlueSpot system. In addition to the applied protocols, the OBEX and the BNEP protocols shall be discussed. They are candidates for usage, but are currently not integrated. The reason for this will be given after they have been described later. In order to get an overview of a Bluetooth stack and the on top running protocols see figure 5.1.

The supported Bluetooth protocols take place on top of the baseband. They are in a hierarchical order in dependency to the main protocol, the *Logic Link Control and Adaptation Protocol* (L2CAP). Every connection, that is based on an ACL link is established via the L2CAP. As the name indicates, it is responsible for the logic link control. On top of the L2CAP, there is the *Radio Frequency Communication* (RFCOMM) protocol. It enables the emulation of up to 60 serial ports. The RFCOMM protocol is the standard protocol used for communication between two nodes. The same applies to the BlueSpot system.

Beside the RFCOMM protocol there are two further common protocols for Bluetooth. These are the *Object Exchange* protocol (OBEX) and the *Bluetooth Network Encapsulation Protocol* (BNEP). The OBEX protocol was first specified within the IrDA standard [Millar et al., 1998] in 1998. It was designed to enable mobile devices to easily exchange contact information in form of visiting cards (vcard) or calender entries. The main advantage of this protocol is that the format of data, to be exchanged is defined very strictly. As a result, the interoperability between several different devices currently available on the market can be guaranteed. This is done by defining objects that include the information to be sent. These objects can be pushed to the other device. However, this is the big disadvantage for utilization in the BlueSpot system. Especially streaming service cannot be implemented by use of a protocol based on object pushing. But one of the main targets of our research was to support every kind of mobile service.

The second common protocol is named BNEP protocol. It enables TCP/IP support for Bluetooth by simulating a virtual Ethernet device. It is used for the establishment of standard TCP/IP based socket connections via Bluetooth. The utilization of TCP/IP would seem to be obvious. But it has the great disadvantage that the overhead used to form the needed TCP/IP header for each packet is huge. The bandwidth, that was small to begin with, is thus reduced additionally. As the BlueSpot middleware is also responsible for the network organization - normally one main task of the TCP/IP protocol - the BlueSpot system can go without the support of the BNEP protocol.

A special part of the Bluetooth stack is the *Service Discovery Protocol* (SDP). It is responsible for scanning a communication partner for its supported protocols. This scanning takes several seconds due to its complexity, which is a rather long time. For this reason, the SDP protocol does not take part in the BlueSpot system. Instead, a node tries to build up a connection via a predefined link configuration. The moment the other node accepts the connection, the BlueSpot system is supported by this node. Otherwise, the node will reject the connection request. Therefore, the only supported protocol is the RFCOMM protocol. The used parameters are hard-coded into the middleware.

5.1.2. (W)LAN with TCP/IP

Beside Bluetooth, IEEE 802.11 WLAN was used for the BlueSpot project. Unlike Bluetooth, the WLAN stack implies that the main area of application is TCP/IP. For this reason, this subsection is divided into two parts. The first part describes all of the properties of the IEEE standard relevant for the BlueSpot system relevant in general. The second part aims at the TCP/IP layer of the IEEE 802.11 ISO-OSI stack. In this part all the important properties of TCP/IP will be pointed out. It shall also be described how it can be mapped onto Bluetooth so that the resulting network behaves like a Scatternet.

IEEE 802.11 WLAN

In the last years, WLAN has evolved to the most common wireless communication standard in small network organizations like home networks. But also nearly every company provides its own WLAN network to support their employees with independent mobile working within the company's premises. In most cases, the covering of the entire premises with only one radio device is not possible. Therefore, additional radio devices are needed to form a network.

The IEEE 802.11 standard describes two different possibilities to do this. These are the *infrastructure mode* and the *ad-hoc mode* [Schiller, 2003c]. If the network is run in infrastructure mode - this is the common way - a specialized node, named *access point* (AP), is needed. The type of the formed network is a single-hop infrastructured network with the access point as gateway between the wireless LAN and a wired LAN (BMG). By adding additional AP, the coverage area of the network can be extended. Clients of the network are NMC. They always have to communicate via the AP. The AP can communicate among one another by using the wired backbone. They are not able to exchange data directly via air.

The second mode, supported by IEEE 802.11, is the ad-hoc mode. Within a network run in this mode an AP is not needed. All nodes can communicate with each other directly. The resulting network is a single-hop ad-hoc network. In terms of the IEEE standard, such a formation of nodes is named *Independent Basic Service Set* (IBSS). Principally, it is possible to overlap two or more IBSS. A node can just be member of one IBSS at once. Finally, the IEEE standard does not include a definition for dedicated nodes that enable routing or the exchange of data between two IBSS. Summarized, there are no bridging nodes for IBSS.

For utilization in the BlueSpot system, WLAN is run in infrastructured mode. As a result of the missing bridging nodes, WLAN in ad-hoc mode is not applicable, because of the restricted scaling possibilities. An IBSS with more than five nodes will have a bad performance. WLAN has a transfer rate that is up to 150 times faster (IEEE 802.11n standard) than Bluetooth, which is WLAN's big advantage. The described overhead for TCP/IP is thus no longer relevant. As a disadvantage, every node must be configured singularly in order to connect to a TCP/IP subnet. Therefore, its IP address and subnet mask must be specified. An additional disadvantage is the energy consumption of WLAN. Bluetooth devices work much more energy efficiently and thus have a longer operating time.

In the course of our testings in the laboratory, we detected that running Bluetooth and WLAN at the same time with a high density of overlapping Piconets (five or more) lead to problems with Bluetooth. The used IEEE 802.11b/g standards work at the frequency of the ISM-Band at 2.4 GHz. This is the same frequency Bluetooth works on. The result are instable Bluetooth connections. For this reason, we extended the BlueSpot middleware with an auto-reconnect functionality. With aid of this extension, a node tries to reconnect immediately if it detects that a connection that was established before has been closed. But experiments have shown that a reconnection can take several seconds. In this time, data transfer is not possible and thus large buffers are needed to bridge over the downtime of the connection.

TCP/IP

In the beginning, the IEEE 802.11 standard was introduced to replace wired cables for TCP/IP based networks. It defines the physical and the data-link layer of the ISO-OSI stack. TCP/IP concerns the network and the transport layer of the stack. Additionally, for the last ten years, the TCP/IP protocol in version IPv4 has developed to the most widely spread network protocol. Many other network standards provide support for TCP/IP. For this reason, this protocol sets a good basis for the BlueSpot system. By implementing support to it, the BlueSpot system can be run on several different network types; even future developments will be usable.

Connections via TCP/IP are built-up via sockets. The destination addressing is performed by the combination of the destination's IP address and a port number. Nodes are organized in subnetworks. Mapped onto a WLAN that is run in infrastructured mode, all nodes within this network are organized in one subnet. If the network is enlarged by adding additional AP, these AP are included in the same subnet. Principally, it would be possible to span a new subnet with each AP. This entails, that every client needs to be reconfigured when it connects to another AP. By partitioning the network into several subnets, the configuration would be apparently closer to a Bluetooth Scatternet. But because of the contemplated configuration overhead, the network could not be used in a for user transparent manner any longer. For this reason we decided to form a BlueSpot system based on WLAN in one subnet. The resulting BlueSpot topology is a single-hop infrastructured wireless network. The requirements made to this network otherwise remain unchanged.

5.2. Hardware Platforms and their Operating Systems

To be able to demonstrate the results of the BlueSpot project, various hardware components were used. These were selected to represent each known category of mobile devices. The categories are smartphones, PDA, desktop PCs and embedded computers. The selected devices are two different NOKIA smartphones, a FSC Pocket Loox PDA, various Gumstix and desktop PCs. Next in this section, these hardware components are discussed in more detail. We shall describe the properties of each hardware platform and its on top running operation system. Furthermore, the additionally needed software components shall be specified.

5.2.1. Gumstix with Embedded Linux

The Gumstix [Gumstix, 2007] is a small computer in size of a chewing-gum stick, therefore its name. Its dimensions are 80mm x 20mm x 6.3mm. The Gumstix consists of a Intel XScale processor, a flash ROM and SDRAM memory placed on a PCB. There are various different versions available. They either have a PXA 255 or a PXA270 processor with 200MHz, 400Mhz or 600MHz clock speed, and different sizes of memory between 64 and 128MB. Additionally, there are versions equipped with a Bluetooth chip. These are the ones we used for our project. All Gumstix can be enhanced with extension boards via two connectors, placed on its top-side as well as its bottom-side. This way, the Gumstix can be equipped with WLAN and LAN, audio support, an LCD display, various types of memory flash cards or a GPS module. Additionally, it supports many current connection standards such as USB, I2C, NSSP or UART interfaces. See figure 5.3 that depicts a Gumstix.

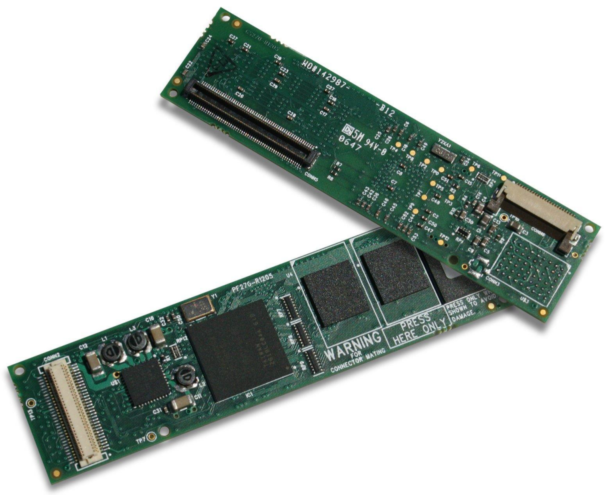


Figure 5.3.: A photo of two Gumstix

The operating system used to run the Gumstix is Linux. The applied distribution is an embedded version that uses the Busybox toolkit [Busybox, 2007] - a replacement for many small UNIX commands which are part of the POSIX standard [Josey et al., 2004]. The standard GNU C library is replaced by the uClibc library [uClibc, 2007]. In contrast to the standard library, this library requires less memory. Additionally, the uClibc library supports processors that do not have a floating point unit (FPU), as the processors of the XScale series do. The library provides floating point operations by emulating such a unit.

To be able to implement mobile services for the Gumstix a toolchain is available. This toolchain is supported by the manufacturer of the Gumstix and is kept up-to-date concerning the Linux kernel and standard Linux software. Beside native C, there is the possibility to implement mobile services in Java (see subsection 8.2 for details). Therefore, a Java Virtual Machine (JVM) is needed to be run on the Gumstix. In this case, the applied JVM is Sun's J2ME (Java 2 Micro Edition) [J2ME, 2007]. This is the most common JAVA version for Linux based systems.

To enable a more comfortable programming, the standard template library (STL) was used. This library supports more complex data types, such as vectors or maps. Additionally, strings and various string operations are included. This library was developed for Linux and can be easily used on the Gumstix platform.

5.2.2. Nokia Smartphones (6600/6630/E60) with SymbianOS

The smartphones used for the BlueSpot project demonstrator were made by Nokia. These are the Nokia 6600 and the Nokia 6630. The Nokia E60 is not supported currently, but support is planned in the future, due to the integrated WLAN support. Each smartphone has a different processor type. The resolutions of the displays are between 176 x 208 pixel (Nokia 6600) and 352 x 416 pixel (Nokia E60). Each smartphone has at least 6MB of internal memory and can be extended by an SD or MMC flash card.

In comparison to the Gumstix, smartphones are more difficult to be programmed. This is due to the installed operating system, which is Symbian OS [Sym, 2007]. The version of the operating system differs from phone to phone. E.g. the Nokia 6600 runs with version 7.0s, whereas the Nokia 6630 comes with Symbian OS in version 8.0. The Nokia E60 is equipped with version 9.1. Due to the different versions, there was a need to adapt the BlueSpot middleware to each smartphone individually. E.g. major changes were made in the implementation of the Bluetooth support. The Nokia 6600 as well as the Nokia 6630 support Bluetooth version 1.1. But between Symbian OS 7.0s and 8.0 the Bluetooth API needed to implement

Figure 5.4.: The smartphones used in the project from left to right: Nokia 6600, 6630 and E60

Bluetooth support was completely revised. Therefore, a complex redesign of the Bluetooth interface within the BlueSpot middleware became necessary. But all versions of the BlueSpot middleware have in common that the generic interface for mobile services are standardized. As a result, any mobile service can be run on the smartphones, regardless of the used Symbian OS version. Additionally, there is the restriction that all API used to develop applications for Symbian OS are closed source. A developer depends on the documentation provided by Symbian and by Nokia.

All three smartphones come with Java preinstalled. The Java version is the *Mobile Information Device Profile 2.0* (MIDP 2.0) [Bloch and Wagner, 2003]. This is a profile of J2ME that is especially adapted to the requirements of smartphones and cellular phones. To enable an easy implementation of applications for such devices, MIDP is kept very simple in its object model. This way, it allows a straight forward development analogous to the desktop and the enterprise versions of Java. Despite its simple object model, MIDP lacks support for modularizing the implementation of technical concerns such as data persistence, screen management, session management, security management, etc. All these aspects are normally not needed for the development of applications for mobile devices. For the BlueSpot system, the support of MIDP based mobile services allows to have services that are portable onto all other hardware platforms used in this project. E.g. a MIDP based service can easily be transferred to a Gumstix and executed there.

Considering the STL, there is no equivalent implementation available for Symbian OS based smartphones. For this reason, we decided to newly design an implementation of the data types and functions used in the BlueSpot middleware on our own.

This way, we managed to port the middleware to the smartphones with moderate efforts.

Our choice to support Symbian OS based smartphones representative for cellular phones can be explained by several properties immanent to these kinds of smartphones. First of all, these smartphones can be extended by extra software that is not part of original configuration. In order to enable such a software, there are various SDK available to make the development of applications possible. Furthermore, the hardware resources theses smartphones provide suffice to run more complex software, such as the BlueSpot middleware. Finally, the support of standard technologies that are used by the BlueSpot system is most advanced in comparison to other cellular phones. E.g. all devices have a MIDP 2.0 Java VM that comes close to the standardization introduced by the Sun Corporation.

5.2.3. PDA FSC Pocket Loox 720 with PocketPC 2003

Another large group of mobile devices are PDAs. In order to present the class of PDAs, we chose the FSC Pocket Loox 720 [cognitas, 2005], depicted in picture 5.5. It was manufactured by Fujitsu-Siemens and is equipped with an XScale PXA 272 processor running at 520 MHz. The memory sizes are 128 MB RAM and 64 MB ROM. The display has a size of 3,6" and a resolution of 480 x 640 pixel (QVGA). Bluetooth is supported in the version 1.2. Additionally, the PDA is equipped with an IEEE 802.11b WLAN network adapter.

Figure 5.5.: FSC Pocket Loox 720 with Windows Pocket PC 2003 SE

The operating system that runs on the device is Windows Mobile Software 2003 for Pocket PC Second Edition (PPC2003SE). This is a Windows CE based operating system developed by Microsoft. The used Bluetooth stack was implemented by Broadcom (former Widcomm). A corresponding SDK needed to implement applications with Bluetooth support is also provided by Widcomm. Unfortunately, the available documentation was tenuous, which strongly impeded our development efforts. A second difficulty was to find a Java Virtual Machine based on PPC2003SE. There are several machines available, but none of them provides all features defined by the standard introduced by Sun. The result of an evaluation showed that the best fitting implementation would be IBM J9, which is part of the WebSphere Everyplace Micro Environment [IBM WEME, 2007]. This version comes close to the Sun standard and supports all main features of J2ME and MIDP 2.0.

This device is especially suitable for the BlueSpot system due to its technical parameters. It supports WLAN as well as Bluetooth, and additionally the manufacturer provides well-designed SDKs to enable programming of both communication standards. Additionally, there is a large community providing support concerning implementation issues, which proved very helpful in the course of development.

5.2.4. Desktop PC with Linux

In order to reach a stable testing environment, we decided to port the BlueSpot middleware to standard desktop PCs and laptops. The used operating system is Linux, based on a Linux kernel in version 2.6. On this type of system, the development is simplified by well-engineered debugging tools. E.g. these can be used to find memory leaks or to debug deadlocks. The usage of the STL showed no problems due to the sophisticated implementations available for Linux. In order to enable Bluetooth support, we used USB dongles made by the companies Acer and Broadcom. These are operated with the BlueZ stack [Holtmann and Krasnyansky, 2007] placed on top of the *Host Controller Interface* (HCI) standard and hence function similarly to the Bluetooth devices installed on the Gumstix. In order to develop Midlets - a Java based application especially developed for MIDP - we used the standard J2ME version provided by Sun [J2ME, 2007]. The MIDP 2.0 profile is enabled by an emulator, also provided by Sun. Part of the emulator is a virtual mobile device, which pops up on the screen when a Midlet is started. This virtual device has a telephone keypad and softkeys, with which the Midlet can be controlled. Outputs are displayed on the virtual display of the device, that pops up on the screen.

Due to the more refined background available when running the BlueSpot system on a desktop PC or a laptop, it is possible to analyze the proceedings as they happen during runtime. This way, the development efforts can be accelerated and a better quality of code can be achieved.

5.3. Summary

In this chapter we have shown the main ideas behind the Bluetooth standard. We have introduced the terminology used by the standard and explained how to form Bluetooth based networks. The IEEE 802.11 standard was then investigated. Analogously to the Bluetooth section, the terminology was introduced and resulting network topologies discussed. In the second part of this chapter we described the different embedded devices, smartphones, and handheld devices we used for demonstrating the BlueSpot system. Here, especially the operating system, the used software extensions, and the Java support were of interest due to their usage within the BlueSpot system.

The support of Java especially proved to be a great challenge, as Java functions differently on every device. As a result of our efforts, it can be seen that even Java, which has the main intension to be absolutely hardware independent, is not able to bridge the gap between the different kinds of mobile devices. With the BlueSpot system, we demonstrate that there is a way to support all the above mentioned device classes with one and the same system software.

Part II.

The BlueSpot System

Chapter 6

BlueSpot System

After presenting the basics, now the concept of the BlueSpot system itself will introduced. This chapter shall especially describe the architecture of the underlying network view and the corresponding software model in detail. This chapter is therefore structured as follows: first, the software architecture of the BlueSpot middleware shall be explained by depicting the requirements made to the system. After that, the network view and the software model shall be discussed in depth. Within this section, the different abstraction layers of the network view will be explained. These are needed to explain the scope of adjustments done in the system during runtime to meet the requirements of mobile services. In order to describe the software model, the different layers of the middleware will be presented and their main tasks discussed. The interfaces needed to extend the middleware by adaptivity models shall be additionally shown.

Contents

6.1. Requirements to the Software Architecture

The designing of a new complex software system is usually a challenge for a developer. All problems that need to be solved must be determined, registered, and categorized. Furthermore, additional requirements need to be found and described. The problem handling and the description of the requirements are the basis needed in order to construct the software concept. In our case, the BlueSpot system turned out to be very complex. The best solution in course of development therefore was to divide the entire implementation into several modules. These are ordered into a software stack consisting of five layers. For this very common approach, the ISO-OSI stack served us as model.

A second point of interest are the underlying network structures. Each network has different requirements according to the network technology it bases on. In order to gain a network technology independent solution, a subset of functionalities that contains the minimum of the required capabilities must be defined. By use of this subset, the required similarities can be located and transferred into the implementation. The wireless technology independence of the BlueSpot system is gained by providing an interface that forms an abstraction layer on top of the underlying network. Above this layer, no more knowledge about the type of the underlying network technology is needed. The details of this abstraction layer is described in section 6.2.2.

As further requirement in the concept of the BlueSpot system, the integration of adaptive behavior extensions must be made possible, even during runtime. They shall be provided as extra modules, and thus their integration must not imply the need to change the source code of BlueSpot system's middleware itself. In order to solve this problem, the resulting software architecture is equipped with an interface that lies transversely to the rest of the software stack. It is responsible for opening each layer to enable the inclusion of additional functionalities during runtime. The name of this interface is *adaptive behavior extension*. The detailed description of it can be found in section 6.2.2.

In order to integrate the most current extension approaches, the previously mentioned interface must be divided into several sub-interfaces. Most of the current state of the art approaches concern more than one layer of the software stack. They are named crosslayer approaches. A resulting requirement of the BlueSpot system's software architecture is to enable access to each layer individually and even more important, simultaneously. This is succeeded by the subdividing of the adaptive behavior extension interface into sub-interfaces.

The concept of the BlueSpot system requires the enabling of a large spectrum of mobile services running on the middleware. Analogously to the adaptive behavior extensions, there must be a point of connection available that allows the connection of a service to the middleware during runtime. The code base of mobile services, that are provided as stand alone software packets, can be constructed in different manners. They can be implemented in native C as well as in Java source code (see section 8.2 for a detailed description). The resulting requirement the software architecture must meet with is to provide a connection for all possible types of mobile services, independently from the underlying programming language.

6.2. System Description

In order to understand the BlueSpot system in its full complexity, it is necessary to determine different views of the system. By aid of these views, the description of the system can be structured, making it better understandable for the reader.

There are basically two different views used to describe the system. One view concerns the network model the system bases on, the other displays the middleware structure and shows up the tasks of each layer. The latter can be used to describe the different functionalities of the system, whereas the network view is used to describe the behavior of the system as it evolves in the course of its dynamic behavior. It is used to explain how the system reacts to changes according to the requirements mobile services have when they are run on the system. E.g. when a new service is started, it brings with it the requirement of having a defined minimum of free bandwidth. In order to run the service properly, this bandwidth must be provided, and contingently certain adjustments must be made by the system in order to fulfill these requirements. Beside a required minimum, various other parameters must be considered for describing the requirements of a mobile service. A description of these can be found in section 8.2.1.

Beside the previously mentioned adjustments, various adaptive behavior extensions can be applied to the system that all concern different parts of underlying software architecture. In order to categorize these approaches and describe their point of contact concerning the system's middleware, it is necessary to have various views of differing layers of abstraction. These are defined by use of the network view.

6.2.1. Network View

The network of the BlueSpot system is an infrastructed multi-hop wireless network. As a result, there are two different types of nodes available. One type are NMC nodes, which are named *clients*. Within the BlueSpot system, they are responsible for the user interaction. In most cases this type of node is a PDA or a smartphone.

The support of clients with routing abilities is not provided.

Nodes that form the infrastructure (BMR) are named BlueSpots. They are deployed by use of the Gumstix. Gateway nodes (BMG) are not provided, but the gateway functionality can be achieved by use of mobile services. In this case, the service must be run on a node that has access to another network beside the BlueSpot system. An example for this is a client that is also connected to the internet. The BlueSpot system works in a completely decentral manner. I.e. there is no central server that is responsible for network organization.

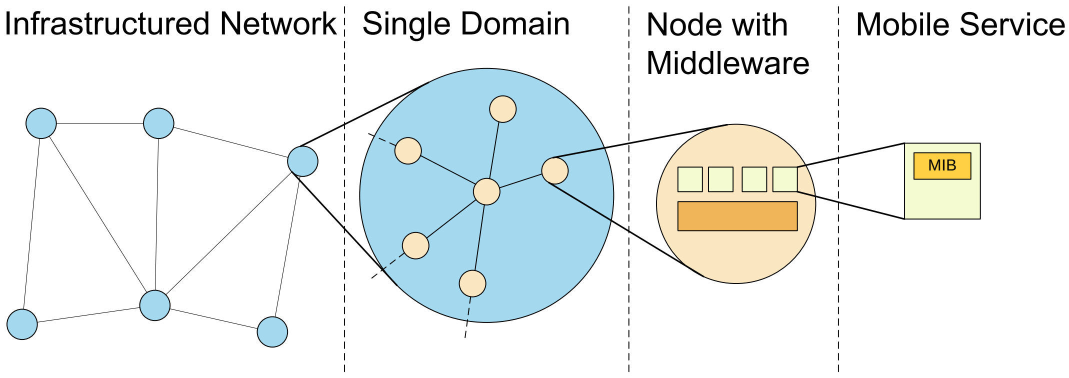


Figure 6.1.: Network Layers for abstraction

By use of the network model, the network view can be described in more detail. Consider figure 6.1 for a classification. The view can be divided into four subviews, here named areas for clarity. The first area is the *infrastructured network*. It represents the global view of the entire network and consists of *single domains* and the connections between them. Each single domain represents a connected number of nodes that are in close interaction with each other. It consists of nodes that form the infrastructure as well as client nodes. In figure 6.1, the infrastructure forming nodes are presented by an additional dotted line that illustrate a connection to another domain. It is import to additionally know that client nodes are mobile within the BlueSpot system. Therefore, they are able to move between single domain areas and connect to the provided infrastructure at this area.

Considering Bluetooth as the underlying network technology, the infrastructured network is represented by a Scatternet, while a domain is represented by a Piconet. The connection between two domains is performed by bridging nodes that belong to both domains. The same applies to IEEE 802.11 WLAN. Here, a domain is represented by an access point with all its connected client devices. By bridging several access points together, an infrastructured network is created. The differentiation between these two areas is needed in order to distinguish the different approaches of system behavior during runtime. That way, the infrastructured network area can be used to describe the global behavior of the system, while local approaches that appear in the vicinity of a node can be explained by use of the single domain area.

In order to investigate the behavior of a single node, an additional area is needed. It represents the area of the *node with the middleware* running on it. By aid of this area, all adjustments made for a node can be described. Unlike the first two areas, this area not only concerns the network but also the software and hardware adjustments. That way, the resources a node provides can be reassigned as a result of changing demands of mobile services. The last area of the network view concerns the *mobile services*. Each service contains an additional set of information named *Meta Information Base* (MIB). The MIB contains all parameters needed to run the service properly. The architecture of the MIB and the content of these parameters are described in section 8.2.1.

In the further proceedings, the term *view* will be used instead of *area*. Here, the term area was used in order to distinguish between the summarized terms *network view* and *middleware view*, and the different layers of abstractions within the network view.

6.2.2. Middleware View

In contrast to the network view, the middleware view is used to describe functionalities the BlueSpot system includes. With its aid their position within the software architecture can be described. As mentioned before, the middleware is constructed as a software stack. Each layer combines various tasks ordered by their common bond. This way, the complexity of the entire system is split into several parts that can be easily implemented. By defining software interfaces that are placed between the layers, these layers are made more independent of each other. See figure 6.2 for the software architecture.

In order to extend the middleware with adaptive behavior, several software interfaces are included, one for each layer. These functionalities are grouped within the adaptivity module, which lies transverse to the entire stack. The *adaptive behavior extensions* are needed for doing adjustment during runtime. This way, the middleware reacts to changing demands of the mobile services. By aid of the software interfaces adaptivity extensions can be included into every single layer. The layers themselves remain completely untouched by these extensions and thus do not need to be modified, if a new adaptive mechanism is inserted.

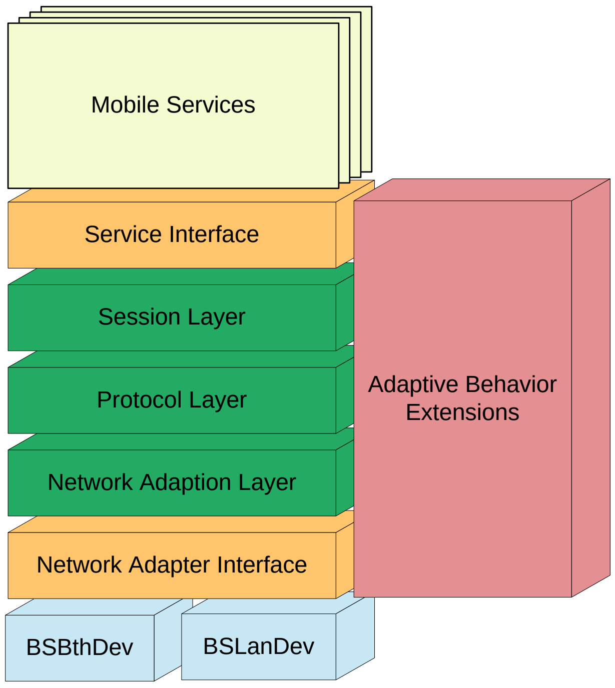


Figure 6.2.: The middleware used in the BlueSpot system

In the following, the software stack shall be described first. Each layer shall be presented in detail and its included functionalities explained. The description shall begin with the lowest layers, the network devices' libraries, continued with the layers in between, and finished with the top layer, named the service interface. Later, the adaptive behavior extensions shall be discussed. In this section, the process of extending the middleware by additional software modules shall be explained using several example extensions.

Stack

As described in the end of section 5.1.1, the BlueZ stack is used to provide Bluetooth support to the Linux based systems. The BlueZ stack includes a well documented API that helped us enormously to add Bluetooth functionality to the BlueSpot system.

In the case of the BlueSpot system, the access to the different network devices is standardized. As a result, the access to each network device takes place in identical manner. The network access is implemented within the *network adapter interface* and the *network adaption layer*. The latter is responsible for the management of the network adapters as well as of the communication handling of each adapter. In order to enable common access to Bluetooth as well as to TCP/IP, the network adapter interface is used. It defines a set of functions that must be provided by the connection network library, such as the *BSBthDev* or the *BSLanDev*, which will be described in the next paragraph. Here is the list of functions to be provided by extending libraries:

- scanning for new devices
- connecting and disconnecting to other devices
- initialization of a master - slave role switch (only if needed)
- reading of the received signal strength indicator (RSSI) (for future use)
- reading of the link quality (for future use)
- sending and receiving of data

The scan process is threaded. Therefore, the process can be started in an asynchronous way. Hence, it does not block the responsiveness of the middleware. A second thread is started after a new connection has been established. This thread is responsible for receiving data and for forwarding it to the upper layers. In order to determine the quality of a link, Bluetooth provides two indicators. One is the *link quality*. A higher link quality value indicates a better link signal quality. The second parameter is the received signal strength indicator (RSSI), which is used to describe the optimal distance between two nodes. In case of Bluetooth, the standard defines an optimum distance of approximately one meter. In case of a shorter distance, the signal strength gets too strong, otherwise, if the distance is too large the signal strength gets too weak. These parameters are very dependent on exterior influences and thus must be used very carefully. As a result, these parameters are a constituent part of the network adapter interface, but are currently not used within the middleware.

In order to support Bluetooth functionality, the *BSBthDev* library is part of the middleware stack. It is responsible for connecting the Bluetooth API functionalities provided by the manufacturer of the Bluetooth hardware to BlueSpot middleware, and thus enables the communication via the Bluetooth hardware. Therefore, it implements all of the previously mentioned functions of the network adapter interface.

In order to access the WLAN adapter hardware, the *BSLanDev* library is applied, which works similarly to the BSBthDev one. It also implements the required functions of the network adapter interface. The role switch is implemented as an empty function, due to the lack of roles within the IEEE 802.11 standard and the TCP/IP protocol. In these two technologies, links are defined to be symmetrical. A differentiation between nodes after a link has been established is not necessary. An indicator for the link quality is available for IEEE 802.11, but it is not provided for the BlueSpot system. Experiments have shown that this indicator cannot be used reasonably due to the high reflection rate of walls and other objects within the radio range of an WLAN device. All these objects reflect the signals transmitted by a sender. On the receiver side, the signal arrives from more than one direction and at different points of time. The latter is the result of different signal propagation delays. Therefore, the link quality indicator is falsified and hence useless.

The scanning process for new devices is accomplished by use of a broadcast message. This message is inserted into a UDP packet and sent to the broadcast address of the IP subnetwork the node is placed in. Other nodes within the vicinity of the broadcasting node answer with their IP address. After this, the node is able to establish a connection to them.

When a node is detected - no matter whether via Bluetooth or WLAN - it must be entered on a list of known devices, named `DeviceDetails`. The source code and a description of this list can be found in appendix B.1. This list must be provided to the middleware stack in order to enable the middleware to manage all connections. By use of it, a node gains knowledge of all nodes that are currently connected to it. This is needed the moment the nodes wants to interact with another node, and thus must know which connection must be used.

On top of the network adaption layer, the *protocol layer* is situated. It uses the `DeviceDetails` structure in order to handle the connections to the neighboring nodes. The protocol layer correlates with the network layer and the transport layer within the ISO-OSI stack. Therefore, it is responsible for the end-to-end communication and thus controls the routing algorithms. The communication of the BlueSpot system is divided into two types of messages: control messages and data messages. Control messages are sent exclusively by use of a first routing algorithm that is integral part of the BlueSpot middleware. This algorithm is a flooding approach with duplicate detection that is enhanced in order to guarantee delivery of messages [Urrutia, 2002]. Without this guarantee a consistent state of the entire network could not be achieved.

Data messages are delivered by use of extending routing algorithms that are responsible for regular communication. These algorithms are exchangeable, and therefore the protocol layer has the ability to load a routing extension module and forward all data messages to it. Subsequent to this, the routing algorithm is responsible for selecting the next hop node in order to forward the data. The selection and exchange of the routing protocols is done by the *protocol interface*, which is part of the adaptive behavior extensions. The currently implemented routing protocols available for the BlueSpot system are DSDV and DSR, which are representatives of the two most common classes of routing algorithms. A general classification of wireless routing protocols is given in section 7.3. The application and integration of such protocols within the BlueSpot system is described in section 7.3 and section 7.4.

The next higher layer is the *session layer*. It is responsible for the mobile services dependent session handling, analogously to the session layer of the ISO-OSI stack. After a service has been started, the session layer creates a new session with a unique 128 bit long session identifier. This identifier is used to address the mobile service run on the corresponding communication partner node. If the communication path between the two partners breaks down, the session is kept upright for a predefined timespan. The moment a new connection path has been reestablished between the two nodes, the still existing session ID is detected and the mobile services are reconnected automatically. This behavior is part of the mobility aspects that will be described in detail in section 7.5.

The binding of mobile services to the BlueSpot middleware is accomplished by the *service interface*. Mobile Services are implemented in an extra software packet and loaded to the system during runtime. Therefore, the service interface provides an API, which is used to control it. For a more detailed description of the service interface API see appendix B.2.

Mobile services can be implemented in two different programming languages. On the one hand, a service can be implemented in native C language. After compiling the source code, a software packet is created that can be loaded into the same process space of the BlueSpot middleware. The set-up is depicted on the left hand side of figure 6.3.

On the other hand, a service can consist of Java source code. In this case, the service is started in a Java virtual machine and thus runs in an extra process space. In this case, the communication between the mobile service and the BlueSpot middleware is established via a socket connection that uses the localhost device. Therefore, two connector classes are provided by the BlueSpot middleware that implement this connectivity as seen on the right hand side of figure 6.3: one is placed within the service interface, and thus runs in the process space of the middleware. The other is situated on top of the JVM, and therefore, it is run in the JVM's process space. The moment a mobile service is started, it is integrated into the same process space of the JVM and the *BS connector class*, and thus is able to communicate with the BlueSpot middleware. In that way, the needed inter-process communication is solved. Normally, the standard way to connect the service would be to use a JNI interface. But in this case, JNI is not available due to its lacking within the MIDP 2.0 Java standard. The detailed mobile service description for the BlueSpot system can be found in chapter 8.

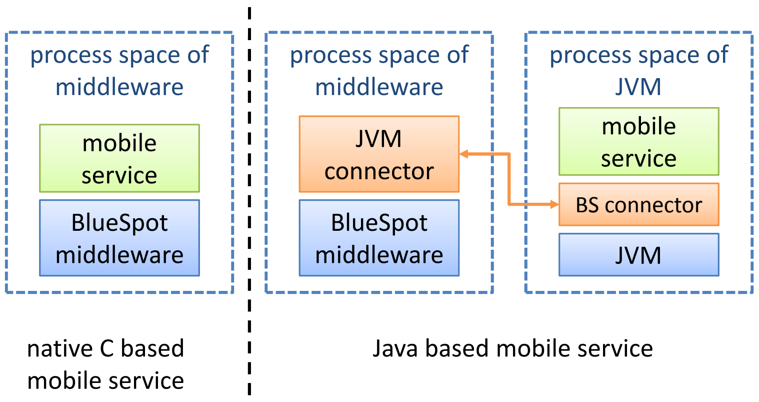


Figure 6.3.: Comparison of the connection of a native C based and a Java based mobile service to the middleware

This way of structuring the BlueSpot middleware was very useful in the course of development due to the resulting possibility to divide the development into separate work packages. These could be accomplished by student works: the Bluetooth support for the Gumstix, and thus the basis of the BSBthDev library was implemented by Steinle and Eiband [Steinle and Eiband, 2006]. The BSBthLan library, the network adapter interface, network adaption layer, and protocol layer and was implemented by Hacker [Hacker, 2006]. The session layer, the service interface and the connector classes were developed by Schupfner [Schupfner, 2007]. Langhammer and Metzger were responsible for the porting of the whole middleware to the smartphones and the PDA [Langhammer and Metzger, 2007].

Adaptive Behavior Extensions

As described before, the middleware can be extended by additional software modules. In order to enable these extensions for adaptive behavior, interfaces are needed, one for each layer of the middleware stack. These interfaces are organized within the *adaptive behavior extensions* module. The detailed source code description can be found in section B.3. For a more general description see figure 6.4.

Depicted on the left side are the layers of the middleware stack. Each layer has

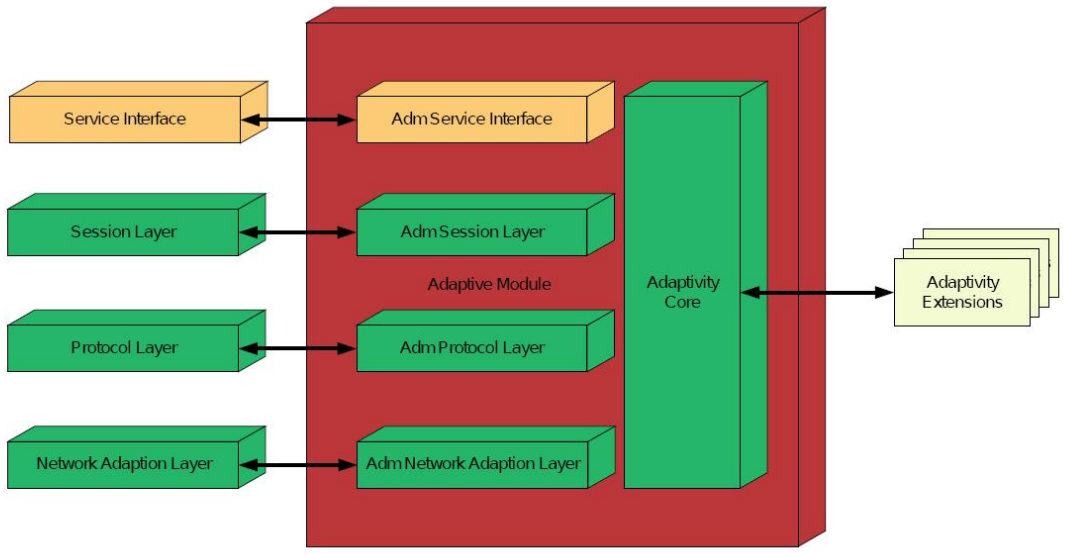


Figure 6.4.: Structure of the adaptivity module

a corresponding interface marked with the prefix *Adm*. In most cases, an approach yet to be implemented concerns more than one layer of the stack, and thus is a crosslayer approach. As a result, the interfaces of the middleware stack can be used to insert desired functionalities on the level of the stack the functionality belongs to. To simplify the usage of theses interfaces, their functions and methods are standardized within the *adaptivity core* class. For further information see Steinle [Steinle, 2007].

When a mobile service is to be started, the general work flow can be described as follows: before the service is started, the service interface requests the MIB (see section 8.2.1) of the service. This is provided to the adaptivity module that must decide whether or not the requirements can be met with. This is done by comparing the required resources to the own available ones. Each node has a database that is used to store its current occupancy rate. This rate is made up of a vector of parameters similar to those stored in the MIB of the service. By comparing the values of each parameter, the middleware is able to find out if the resources the service requires are available. For a detailed description of this approach we refer to section 8.2.1. In case the requirements cannot be met with at the first, available extension modules are tested to see if they can be used to adjust the system in order to gain the needed resources. If a positive result is found, the corresponding extension modules are loaded and started. After this, the service can be started and the allocated resources registered in the local resources database. The adaptivity module therefore is always the central instance making crucial decisions, and as such it is an essential part of the adaptive behavior processes made by the BlueSpot system.

6.3. Summary

This chapter commenced with the explanation of the used software architecture of the BlueSpot system. Due to this, the middleware is implemented as a stack with layers. The adaptivity module lies transversely to the layers in order to compose the provided interfaces of each of them. To explain the tasks of the BlueSpot system in more detail, two different views of the system were introduced. One view deals with the underlying network model, the other is responsible for showing the software stack in great detail, and therefore, it is used to explain the functionalities of each layer. Starting with the underlying network device libraries and the corresponding network adapter interface, the available software interfaces are presented. The service interface positioned on top was used to round up the overall picture. Afterwards, the adaptive behavior extensions module were illustrated. Without giving any concrete examples, the general work flow for starting a mobile service is explained. The understanding of this work flow is important for categorizing the available adjustments for adaptive behavior, that will be introduced in the next chapter together with an overview of possible extension modules.

Chapter 7

Network Self-Organization

In the previous chapter, the different views as well as the structure of the BlueSpot system were introduced. Now, the mentioned adaptive behavior of the system shall be explained. This chapter is therefore structured as follows: before the system can go productive, an infrastructure must be established. For this reason, possible topologies in terms of Bluetooth based networks shall be introduced. The advantages and disadvantages of each possible topology shall be discussed, then basic approaches for automated network forming introduced.

After a wireless network has been formed, it is started and mobile services can go active. But due to the changing demands the system must meet with, it needs to be adjusted continuously.

The available adjustments shall be explained commencing with approaches concerning the modification of the topology. Afterwards, the possibilities of adjusting the used routing protocol shall be described.

The complete exchange of the routing protocol as well as simple configuration adjustments shall be shown. In order to provide a general approach that adds adaptive behavior to a wireless network system, the BlueSpot system's adaptive capabilities shall be introduced and discussed in detail. In addition, the description of the Blue Spot system shall be rounded up by explaining the functionalities available for supporting mobile clients.

Contents

7.1. Network Forming

In order to establish a new wireless network, a network topology on basis of connections must be found that fulfills the minimum requirements the on top running services have. Factors such as the latency times and the throughput within the network must be kept in mind. But the organization of the network's topology is also crucial for a later successful network operation. E.g. a network with a greater amount of nodes is more difficult to organize than a flat topology due to a bad scalability. A result would be huge routing tables and long communication paths with high hopping lengths in average. A better solution here would be to have a hierarchical order into which nodes are grouped into single domains with a bound maximum size.

All these discussions are very fundamental. The implementation in real life with real hardware is much more difficult. Usually, the main idea is to enable the nodes to organize themselves automatically. But in the initialization phase of a wireless network, a node has no knowledge of the actual situation of any other node. Even if the design of the network was made manually, nodes must be able to react to events that harm the network. In general, it is common to completely separate the network forming problem from the network management and controlling performed during runtime. As a result, the forming process is simplified, since it can be considered as an isolated problem. Beside the deliberations that shall be made in the following, we like to refer to Langhammer [Langhammer, 2007] for an overview of related approaches on this sector made by other researching groups.

7.1.1. Bluetooth Topologies

The main communication technology used by the BlueSpot system is Bluetooth. For this reason, network forming shall be discussed by use of Bluetooth. The following description of Bluetooth topologies is taken from Dümichen and Baumgarten [Dümichen and Baumgarten, 2007].

An issue of interest are points of contact, at which clients are able to connect to the infrastructure. As described in the Bluetooth fundamentals in section 5.1.1, an infrastructure node that acts as an master node can only accept one additional connection request. After accepting, it turns to a M/S bridging node and is only able to establish further connections by its own initiative. A way out is to initiate a role switch in order to integrate the client node into the Piconet. The infrastructure node turns back to a master and can accept another connection request. A slave node can accept one further connection. Moreover, it can neither accept nor establish any further connections as it turns to a S/S bridging node.

Generally, two parameters are of interest. On the one hand, the more points of contact are provided by the infrastructure, the better. On the other hand, these points must be spread evenly over the entire network in order to gain a balanced structure. These two issues are very fundamental requirements. Currently, there is no standard mechanism that allows to easily solve these issues simultaneously.

Another aspect to be considered is the average size of the Piconets within the Scatternet. Large Piconets come along with a small number of bridging nodes that are evidently the bottlenecks of the network. As a result, the average throughput will be higher than in networks with sparsely populated Piconets. But in contrast to that, large Piconets offer only a small number of points of contact. In summary, a high throughput rate stands in contrast to the number of available points of contact. Therefore, the middle course is strongly required.

Due to the Bluetooth system's immanent restrictions, there is only a minimal amount of topology types possible. Concerning the BlueSpot system, these topologies only concern the infrastructure. A client node is connected after the formation process was finished successfully by accessing an available point of contact. Before a formation process can be started, an entirely linked network is assumed. The resulting topology is the product of a directed conduction of connection establishments between nodes. The distribution of nodes must be considered in two ways: their *geographical distribution*, where the exact position is of interest, and their *relative distribution*, where the position is seen in relation to the node's vicinity (see section 3.1). The possible resulting topologies and their characteristics will be described next.

The most common topology is the *random* Scatternet. It is the result of establishing a network without any coordination. All types of nodes occur randomly. Piconets are connected by M/S bridging nodes as well as by S/S bridging nodes. See figure 7.1 for an example.

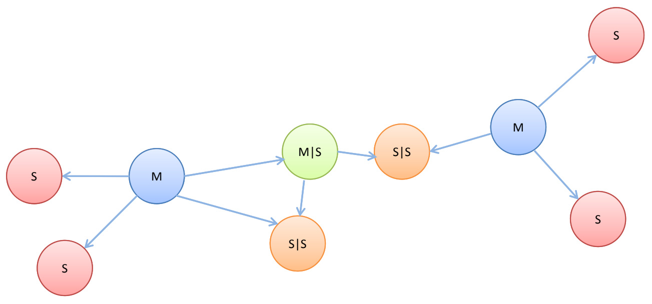


Figure 7.1.: Random Scatternet topology

A randomly organized network has the big advantage that no forming management is needed. But as a result, no predictions can be made whether or not the resulting network is entirely connected. Additionally, it is unlikely that the network is balanced concerning the points of contact. An even distribution of points of contact can also not be guaranteed.

A completely different approach is a Scatternet put into *linear* order. This can be constructed in two ways: by use of M/S bridging nodes or by use of alternating master and S/S bridging nodes. The forming management is not very costly and thus can be easily implemented. An illustration can be seen in figure 7.2.

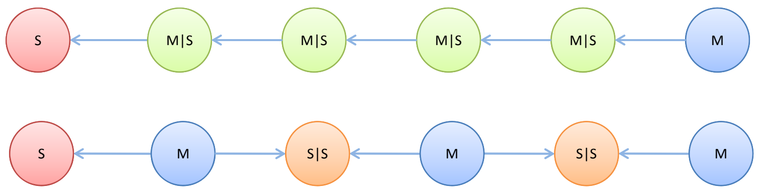


Figure 7.2.: Linearly ordered networks

Both types of linearly ordered Scatternets have a very even distribution of points of contact. However, the M/S bridged type provides more points of contact due to the fact that S/S bridging nodes can neither accept nor establish any further connections (see section 5.1.1), and thus only the master nodes are able to establish new connections or to accept incoming connection request. In contrast to this, M/S bridging nodes are not able to accept any further connection requests. If such a configuration is used, the network must permanently poll for new possible client nodes in order to contact them. The performance of the two types of linearly ordered topologies is very different and will be discussed in sec 9.2.5 in detail.

A disadvantage of both types of linearly ordered networks is, that the moment a node drops off, the entire network will be split up into two parts. In addition to this, all nodes communicate only via the one existing path, and thus alternative paths are not available. This results in long communication paths as well as in a high throughput concentration at the nodes placed in the center. Therefore, the expected scaling behavior will be very bad.

An advancement of the linearly ordered Scatternet topology is the *spanning tree* formation. Due to its definition, no S/S bridging nodes occur within this topology. The root node is always a master node. The leaves are slaves. All other nodes are M/S bridging nodes. In figure 7.3 an example of a spanning tree Scatternet is depicted.

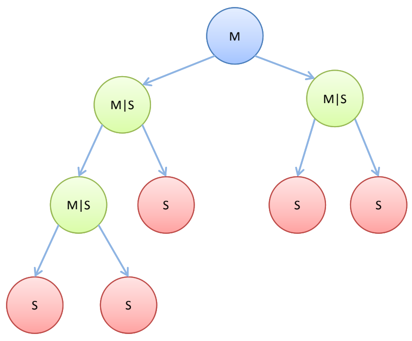


Figure 7.3.: Spanning tree topology

Analogously to a linearly ordered Scatternet, this topology provides many points of contact for clients. The forming of the network is simple due to several forming algorithms already existing for this topology, e.g. the Bluetree algorithm introduced by Zaruba et al. [Zaruba et al., 2001]. In comparison to a linearly ordered Scatternet, the spanning tree formation has a reduced backbone appearance. The length of communication paths is shortened additionally, but nonetheless remains not ideal. If a node drops off, the complete network will fall apart, as it would in a linearly ordered one. The reason for this is, that still only one path from one node to another exists. The throughput and latency times will be higher than in random Scatternets due to the large number of Piconets.

In order to construct a network with redundant paths between the nodes, a *S/S bridged* Scatternet can be used. This topology configuration connects several Piconets by S/S bridging nodes; M/S nodes do not appear. The master nodes are the only available points of contact for connecting clients to the network. The redundant paths originate from Piconets with more than one S/S bridging node, as can be seen in figure 7.4.

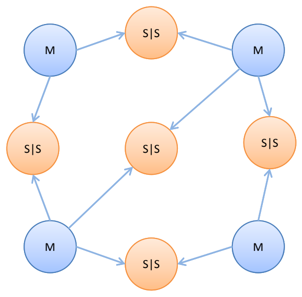


Figure 7.4.: Slave/Slave bridged topology

It is the nature of the master nodes that they are the only points of contact, resulting in their even distribution. The client accepting capacity can be managed easily: the more connection points are needed, the more single domains must be installed (see domain view in section 6.2.1). The throughput of the network is increased by adding supplementary S/S bridges. Clients are connected to the network either by initiative of a master node or of their own. In the second case, it is necessary to initiate a role switch between the two involved nodes to integrate the client into the Piconet of the master node. As a disadvantage, the S/S nodes are not able the handle any further connections. At this point the only function of a S/S node is to forward the data between two Piconets. Additionally, communication paths will be long, as a S/S node is needed for connecting two Piconets.

By adding supplementary slave nodes to a S/S bridged Scatternet, we will obtain a network with nodes we have named *satellite nodes*. See figure 7.5 for an example.

The resulting advantage is a better geographical distribution. Clients connect to these nodes as a master or get connected as a slave. In the first case, it is useful to initiate a role switch since a satellite node acting as a S/S bridge is not able to establish any further connections. In the second case the satellite note runs as a M/S node and is able to connect to six additional clients at maximum. In most cases the enhancement of a S/S bridged Scatternet is necessary to connect all nodes to the infrastructure. Otherwise, it is possible that nodes that could not be integrated into the network will be left behind after completion of the network forming process.

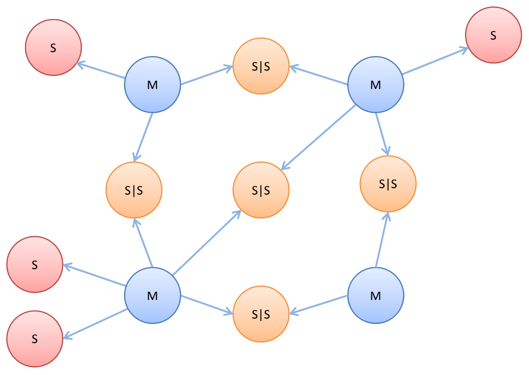


Figure 7.5.: Slave/Slave bridged topology with satellite nodes

An example for a forming algorithm that produces a S/S bridged Scatternet with satellite nodes is the *Bluetooth Topology Construction Protocol* (BTCP) that was first introduced by Salonidis et al. [Salonidis et al., 2001].

After one of the satellite nodes has initialized a role switch and becomes a master, a *S/S bridged Scatternet with satellite nodes and extension* of point of contact is obtained. It structure can be seen in figure 7.6.

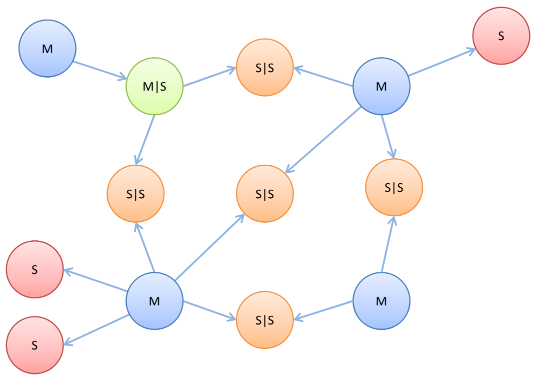


Figure 7.6.: Slave/Slave bridged topology with satellite nodes and extension

The aim of the extension is to be able to satisfy an increasing amount of connection requests occurring momentarily within a particular area of the Scatternet. At this point the resulting master node is able to accept one and to build up several additional connections, meaning an improvement of coverage in this area. After these connections have been terminated, the node can be switched back to regular satellite node. This process can be used as an easy-to-implement feature for the adaptive behavior of the BlueSpot system.

7.1.2. Topology Forming Approaches

The forming of the above described topologies is very difficult and complex especially for the three latter topologies. For these, the most common approaches are based on the application of selected graph theory-based theorems. A detailed overview of the most promising theorems for our gives Li [Li, 2004]. He introduced the application of *high-degree yao graphs* with $k = 7$, where k is the degree of the node's established connections, in order to construct Scatternets that are relatively distributed in an even manner. His deliberations as well as the meaning of high-degree yao graphs for the BlueSpot system, as it is the most promising for network forming, shall be described at the end of this section.

We will start by introducing the *unit disk graphs* that are the common entry point for modeling network formation approaches for wireless networks on basis of the graph theory. On the basis of these unit disk graphs, we shall introduce the *relative neighborhood graph* as well as the *gabriel graph*. These two geometrical structures are applied on top of unit disk graphs and are common approaches that could be used to form networks topologies.

After this, the directed and undirected versions of the *yao graph* are introduced. As an advancement to the other two geometrical structures, the yao graph allows to limit the maximum amount of inbound and outbound connections of a node. In addition to this, two special versions of the yao graph shall be discussed, due to their ability to model the main restrictions made by the Bluetooth standard.

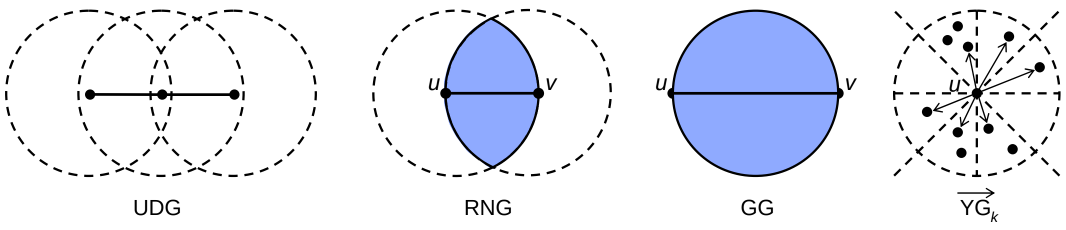


Figure 7.7.: Examples of UDG, RNG, GG and a yao graph.

In the following we consider a wireless network on the level of connections as an edge graph $G(V, E)$ with the nodes V distributed in a euclidean plane as defined in section 3.1. Now, we additionally assume that all nodes have the same maximum transmission range, which is set to one unit. In order to mark this maximum transmission range, a disk is depicted with the node in its center. Two nodes are only able to establish a connection between each other if both nodes are situated within each other's depicted disks. The distance between the two nodes is named *length stretch factor* and is described by the euclidean distance function ($len(e_{ij}) = \|uv\|$). A resulting graph is named *unit disk graph* $UDG(V)$ (see figure 7.7 for an example) that is formally defined as follows:

Definition 4:
Unit Disk Graph

Definition 4: Unit Disk Graph

A unit disk graph is defined by considering an edge graph $G(V, E)$, where all edges E have the maximum length equal to one unit. As a result, there is an edge $e_{ij} \in E$ between two nodes v_i and $v_j \in V$ if only and only if their euclidean distance is at most one.

Further definitions of the $UDG(V)$ that differ from this definition can be found (e.g. Clark et al. [Clark et al., 1990] defines the maximum length stretch factor equal two). But concerning wireless networks, this version makes the most sense, as any two nodes must be within radio range of each other in order to be able to communicate.

A geographical structure that bases on the $UDG(V)$ is the *relative neighborhood graph* $RNG(V, E) \subseteq G(V, E)$. It first was introduced by Toussaint with pattern recognition as its main field of application [Toussaint, 1980]. This geometric concept bases on a set of nodes V that are distributed on a euclidean plane analogously to the definition of $UDG(V)$. Within this graph only those edges exist that have the shortest distance defined by the length stretch factor between two nodes. The formal definition looks as follows:

Definition 5:
Relative Neighborhood Graph

Definition 5: Relative Neighborhood Graph

A relative neighborhood graph $RNG(V)$ consists of all edges e_{uv} such that there is no node $w \in V$ with e_{uw} and e_{wv} satisfying $\|uw\| < \|uv\|$ and $\|wv\| < \|uv\|$.

By use of this structure, the resulting $RNG(V)$ structure is sparser than the underlying $UDG(V)$. Additional edges, that would "overload" the network, and thus would highly influence the performance of an applied routing algorithm are eliminated.

The second geometrical structure that can be considered as interesting for network forming algorithms is the *Gabriel graph* $GG(V)$. It was first introduced by Gabriel and Sokal [Gabriel and Sokal, 1969], who defined the structure as follows:

Definition 6:
Gabriel Graph

Definition 6: Gabriel Graph

Let $disc(u, v)$ be the disk with diameter $\|uv\|$. Then, the Gabriel graph $GG(V)$ contains an edge $e_{uv} \in E$ if and only if $disk(u, v)$ contains no other node $w \in V$ inside.

Analogously to the $RNG(V)$, the resulting graph $GG(V)$ is sparser than the underlying $UDG(V)$. Therefore, the same results apply to the $RNG(V)$ and the $GG(V)$. The area defined by the $disk(u, v)$ is smaller than the intersection of the two disks defined by the $RNG(V)$ (compare figure 7.7). As a result, more edges will be integrated into the resulting graph than by applying a $RNG(V)$ structure. Li [Li, 2004] showed that $RNG(V)$ is a subgraph of the Gabriel graph $GG(V)$. He deduces furthermore that the relative neighborhood graph and the Gabriel graph only contain the edges in UDG satisfying the respective definitions.

An example for the usage of an $GG(V)$ structure was given by Karp and Kung [Karp and Kung, 2005]. By use of their approach a spanning tree topology is established that is the result of a $GG(V)$ based algorithm. They proposed this approach in order to obtain a network topology their GPSR routing algorithm (see section 4.2.5) is working on.

A more promising geometrical structure concerning Bluetooth-based Scatternets is the *Yao graph*. By use of this structure, the disk with a radius of the maximum length stretch factor and the node in its center is divided into k separated cones bordered by rays originating at the node itself. For an illustration of a yao graph see the right most illustration in figure 7.7.

Definition 7:
Yao Graph

Definition 7: Yao Graph

Say, k is an integer value with $k \geq 6$. A directed yao graph $\overrightarrow{YG_k}(V)$ based on k is defined as follows: at each node $u \in V$, any k equally separated rays originated at u define k cones. In each cone, choose the shortest edge e_{uv}, if there is any, and add a directed link $\overrightarrow{uv}$.

The great advantage of the yao graph is that the maximum number of the outbound connections are define by the factor k. This is especially important for Bluetooth Scatternets, where the size of a Piconet is limited to eight nodes: the master and seven slaves.

In order to describe a Scatternet, this structure is not sufficient, due to the lacking of inbound connections. Here the most promising algorithms are the *symmetric Yao graph* and the *high-degree Yao graph* which are special versions of the Yao graph. By adding the oppositely directed links $\overleftarrow{uv}$ to a Yao graph the resulting structure is named *reverse Yao graph*, denoted as $\overleftarrow{YG_k}(V)$. In case an edge is only added to a graph if the link $\overrightarrow{uv}$ as well as the link $\overleftarrow{uv}$ exist, the resulting graph is undirected. It is denoted as *symmetric Yao graph* $YS_k(V)$. Peng and Lu [Peng and Lu, 2000] showed, that a $YS_k(V)$ is strongly connected if $UDG(V)$ is connected and $k \geq 6$. The resulting Scatternet consists of Piconets not larger than seven nodes (six cones plus the node itself) in size in case of $k = 6$.

By using a *high-degree yao graph* the master-slave relation can be taken into account additionally. An algorithm that uses this type of geometric structure as basis for network forming was proposed by Li et al. [Li et al., 2004]. Here, only the main steps of this algorithm are described. For a detailed description we refer to the citation given above. This algorithm consists of two steps. The first step is to prepare all nodes in order to satisfy some properties such as the resulting structure must be planar. In this case, planar subgraphs are formed using the available nodes.

During the second step, the amount of outbound connections of each node is limited to seven by applying the yao structure with $k = 7$. The subgraphs created before, are used in order to assign master-slave relations to all nodes that are members of the subgraph. In addition, each node creates a key, which can be the identity of the node, its degree or a combination of both. In latter case, the key can be used for comparison with its neighbors. Step two is iterated as long as there are no more unassigned nodes. The final structure is denoted as $YH_k(V)$.

7.2. Modification

After a wireless network has been formed, the mobile services to be provided can be started. As a result, the entire system is set active. By this, the demands made to the system begin to vary. New services are started and thus the needed bandwidth must be provided in order to run them properly. Therefore, the entire system must permanently be adjusted in a highly dynamical manner. Additionally, new clients the system needs to attend to can appear. These are able to move throughout the entire coverage area, which results in the usage of permanently changing points of contact. New connections must be established, while connections no longer needed are closed. Both, the changing demands as well as the mobility of the clients require a permanent modification of the network topology. This results in the following events:

- adding of new nodes and thus establishing of new connections
- dropping off of nodes or actively closing of connections
- restructuring the topology, due to reacting to new demands

The first two events always effect a single connection concurrently. The third event needs to be handled more elaborately. The moment the demands change and thus the system decides to trigger modifications, a sequence of events will occur. These events must be thought through carefully and their executing must be well organized.

A restructuring event always concerns more than one connection. Therefore, in order to perform the restructuring event, it is split into various adding and dropping events. This is done at the originating node which has previously detected that the demands cannot be met with anymore, due to changes that are the results of an exogenous event (see section 7.4.2). Afterwards, tasks describing the single atomic events are spread to the concerning nodes, and are executed after one another.

An example for a restructuring event is the *path isolation*. Consider a mobile service that requires a high and constant bandwidth, such as telephony service, that shall be run on the system. The moment the mobile service is started, the used communication path can be taken out of the network topology. All connections that are not part of the communication path are closed. As a result, bandwidth previously assigned to other services is revoked and provided to the new service. Additionally, the number of connections to be handled is reduced to the minimum.

By triggering a path-isolation event, the network falls apart into at least two pieces. Subsequently, the isolated communication path is exclusively responsible for providing the resources the service demands. The rest of the network must reorganize itself. In case the network was divided into three or more parts, the remaining ones must be reconnected in order to reestablish a entirely connected network.

The main usage of modification within the BlueSpot system is to enable the mobility of clients. Here, modification events are the result of handover requests, the joining of new, or the disappearing of existing clients. All the procedures concerning the mobility of clients shall be described in section 7.5 in more detail.

7.3. Routing

If a wireless network contains redundant communication paths, changing demands can be additionally satisfied by adjusting the used routing protocol. These adjustments can be divided into two kinds: the exchange of the complete routing algorithm itself, and adjustments made to a currently running protocol. In case the complete routing algorithm is exchanged, a broadcast message must be sent to all other nodes. This is used to announce the pending exchange event. The moment a node has received an exchange request, it immediately stops transmitting any data and performs the exchange. All upcoming data is queued in order to prevent data loss, due to timeouts or currently unreachable destination faults. After the new routing protocol is started and all needed communication paths are reestablished, the queued data is transmitted.

The BlueSpot system currently provides reference implementations of a DSDV and a DSR algorithm. Providing these two algorithms, it contains an implementation for the proactive as well as for the reactive routing protocol classes. Additional routing protocols are easy to implement due to the protocol interface that is part of the adaptive behavior extensions module. As a result, all functionalities needed to accomplish an exchange of the routing protocol are already included within the BlueSpot middleware. The detailed description of the protocol interface and the two reference implementations of routing protocols for the BlueSpot system was authored by Hacker [Hacker, 2006].

The second kind of adjustments are those made to the currently running routing protocol. These adjustments can be very manifold: beginning with the possibility to change the *maximum transmission unit* (MTU) size that is responsible for the used packet size, over alternative route finding, up to multi-path routing. Any type of adjustment is thinkable. E.g. multi-path routing can be used in order to enhance the available bandwidth of a desired communication path.

A promising example therefore is the *split multi-path routing* protocol (SMR) introduced be Lee and Gerla [Lee and Gerla, 2001]. With their protocol, which is based on an on-demand routing approach, they enable multiple communication paths between a source and a destination node for wireless networks. By this, two advantages are gained: on the one hand the stability of communication paths is increased dramatically. If a path breaks apart, an alternative path that can be used instead immediately. On the other hand, all alternative paths can be combined and used for communication simultaneously. As a result, the throughput of the network can be increased.

Due to the generality of the interfaces of the adaptive behavior extensions model, such an algorithm can be easily implemented for the BlueSpot system. Especially the low bandwidth of Bluetooth can be increased in order to support more demanding mobile services.

7.4. Adaptive Behavior

All the adaptive mechanisms so far described in this chapter concern only one single field of application. They are placed on a specific layer of the BlueSpot middleware. The network adaption layer as well as the implementations of the network adapter interface are responsible for the network forming. Therefore, it is their task to administer all modifications and thus to put the events into action. The routing is implemented at the level of the protocol layer, analogously to the ISO-OSI stack.

Next, resource reservation mechanisms shall be explained. These mechanisms are used to control the globally occurring data traffic as well as that in a local domain.

All approaches have very different ways of adjusting the behavior of the middleware. As a result, they tie on different levels of the software stack. Also, not every mechanism can be used to obtain the desired target, and thus they need to be managed elaborately. How to manage these mechanisms shall be described in the second part of this section. For further information we refer to Dümichen and Baumgarten [Dümichen and Baumgarten, 2008].

7.4.1. Resource Reservation

Wired networks have many mechanisms to enable resource reservation. Some of these mechanisms can also be applied to wireless networks. E.g. the standard way of allocating bandwidth to services is the *best-effort* approach [Clark and Fang, 1998]. By its usage, no limitations are made to any service. As long as there is bandwidth left, services can use it. Other services that have a higher demand must either wait until enough bandwidth is freed or they cannot be run. Alternatives to best-effort are the DiffServ [Nichols et al., 1998] and the IntSrv [Bernet et al., 2000] approaches. These approaches are examples for the *global resource reservation protocols* group. All these approaches can be transferred to a wireless network and deployed there.

Another group of resource reservation protocols concerns the *local assignment of resources*. These approaches are known from local area networks. Examples are very manifold, such as token reservation, dynamic allocation of resources, or any type of credit approaches.

The *token reservation* approach is known from the local area networking. One token is provided to a bound amount of nodes. Only the node the token is currently given to is allowed to send data. After a defined amount of data has been transmitted or a timeout has occurred, the node must pass the token to the next node.

The token reservation is a very fair mechanism. Each node obtains the same conditions for sending its data. But as a disadvantage, the performance of this mechanism is not very good, resulting from the providing of the token to nodes that currently do not need to send anything.

The *dynamic allocation* requires a central instance that is responsible for observing the actual status of the connected nodes. Usually, this instance is the master in a Bluetooth Piconet. All nodes must announce that they want to send data as well as the amount of data to be sent. The central instance collects all requests and allots sending time to each single node.

The way the next node is selected functions completely analogously to process scheduling mechanisms such as *first come first serve* (FCFS), *shortest remaining processing time* (SRPT), or *longest processing time* (LPT). But here the time of arrival of the request as well as the amount of data to be sent are used as parameters for the mechanisms. Correlating to the process scheduling, time scheduled mechanisms with static or dynamic priorities are also thinkable. As a result, a direct control of the communication of each node would be possible. E.g. nodes that act as bridging node could be preferred in order to support the in- and outbound communication traffic of a single domain.

In order to guarantee a high degree of fairness, *credit approaches* can be used. Each node obtains a predefined amount of credits that the node can set in order to send data. If it has used up its credits, it must wait until new credits are dispensed. In comparison to the token reservation mechanisms, the order in which the nodes come up is not predefined. The estimated average performance will then be higher than it would be when using token-based approaches. This is the result of a better usage of the communication channel. Additionally, single nodes can be preferred by dispensing a higher amount of credits to them. That way, such a node can send more data than others.

All these mechanisms have in common that they concern nodes which are in direct vicinity of each other. Concerning the BlueSpot system, the implementation of all resource reservation protocols takes place on the level of the network adaption layer and the network adapter interface. The only resource reservation mechanism currently provided by the BlueSpot system is a best-effort approach that works across the entire network. In order to make the local assignment of resources, the BlueSpot system uses an FCFS approach. The implementation of further approaches is planned for the future.

7.4.2. Network Control

For controlling the different presented mechanisms, an elaborate management must take place within the wireless network system. This management is named *network control*. In order to give a first direction for the management, a time scale is shown in figure 7.8, which illustrates exemplary events appearing in a possible life cycle of the BlueSpot system.

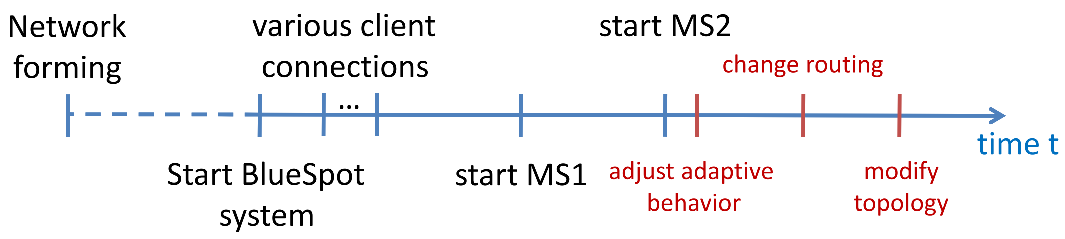


Figure 7.8.: Time scale with events occurring during the life cycle of the BlueSpot system

The solid line shows the time scale during runtime, whereas the dashed line indicates the initialization phase of the wireless network system. Within the time interval depicted with the dashed line, all nodes are connected in order to form the underlying infrastructure network. Here, a selected forming algorithm, described in section 7.1, takes place. The moment the forming is completed, the system is started. This event as well as other exogenous events, such as the connection of an client node or the start of an mobile service, are marked on the time scale with a blue vertical line.

Internal events that are the result of an adjustment action are marked with a red vertical line. As you can see in the figure, two more exogenous events take place after the system was started. These indicate the starting of two mobile services. After the second service has been started, the network control must undertake several adjustments in order to meet with the requirements of both services.

The order of the adjustments is predefined. Generally, the network control has three steps at hand. In order to meet with the requirements of the mobile services, resource reservation mechanisms are first executed. If the requirements for the mobile services cannot be met with by this, the network control adjusts the routing protocol in a second step, as described in section 7.3. The third step concerns the modification of the network topology as described in section 7.2. An alternative fourth step, that is not executed by the network control but rather must be executed manually, is the complete reformation of the network topology. For this, the entire system must be stopped. Afterwards, a new formation algorithm is selected and executed. In the end, the system is started again with the new network configuration that fits the requested mobile service.

The first step can be subdivided into four further steps. These are ordered depending on the network view that was introduced in section 6.2.1. First, the global resource reservation protocol is reconfigured or exchanged. In order to illustrate the changing behavior of the wireless network, the infrastructured network view is used. The second step concerns all nodes within a single domain. The used algorithms are based on the local assignment of resources. The methods applicable for these first two steps are described in the previous section.

The node with middleware view is the basis for the third step. Here, a single node is examined in order to adjust its provided hardware resources, such as the available memory. E.g. in case a video capturing unit is connected to the node - as is used by the CamCar Service described in section 8.2.3 - it is necessary to provide more memory to the mobile service that processes the data from the device. It must acquire a large amount of data as well as convert this into a video signal that can be sent over the network system. As a result, the service has increased requirements concerning the used memory space.

The last step bears a relation to the service view. In order to investigate a newly developed service before running it on top of the network system, it is run on a simulator. There is a simulator available for the BlueSpot system, which is described in section 9.1.2 in more detail. The simulator provides various standard network configurations. These are used to discover the values of its MIB's parameters as well as its estimated behavior during runtime. As a result, the mobile service can be certified for the BlueSpot system, and thus it is guaranteed to run properly in real life.

Summarizing, it can be seen that there is a large amount of possibilities to adjust the system in order to achieve a better performance. Each mechanism can be used for tuning the system in a different direction. The network control must be as decentralized as possible due to the lack of a central instance. Some approaches need a complete overview of the entire system, but most of them can be controlled by a single node and thus implemented fully self-sufficiently.

7.5. Mobility Aspects

Beside the adjustment of the system, the network control is also responsible for the support of client connections. Appearing clients need to be included into the network, and thus points of contact provided to them. Additionally, these clients are mostly mobile. As a result, they will move throughout the entire network. By this, their connection to the system will break up from time to time, which must be handled. The mechanism used to handle such events is named *handover*. In case a connection breaks up, the currently running session for a mobile service is kept alive. The system as well as the node immediately start to reestablish a connection.

After the node was successfully reintegrated into the system, the session is reconnected and the service that belongs to the session can be continued. This approach is named *hard handover*. All these actions must be completely transparent to the user currently working with the device. But as a result of the time consuming connection reestablishment, the users will notice a delay while using the service.

In order to evade such delays, a second kind of handover is used, named *soft handover*. Contrary to the hard handover, a node always tries to establish a second connection with the infrastructure. This connection is a backup connection and is switched inactive. The moment the primary active connection breaks up, the second connection is set active and all communication is detoured to it. Afterwards, the node immediately tries to establish a new backup connection, that is switched inactive.

The BlueSpot system provides both functionalities, and thus it is also able to provide a balanced support to mobile clients. In order to find a client the moment it appears as well as during a handover event, the network must have the ability to perform an inquiry for the client. This functionality is also part of the mobile behavior extension of the system and is implemented within the network device libraries that connect to the network interface. In case of Bluetooth, an inquiry request is sent out periodically. The moment a client answers, it is added to the `DeviceDetails` list (*see* apendix B.1) and thus advertised to the rest of the middleware. For TCP/IP the inquiry is implemented as a broadcast message that is sent into the entire subnetwork. The moment a client node has reestablished its connection to the network, it is able to receive and in the following answer the inquiry request.

7.6. Summary

The task of this chapter was to describe a general approach of including any kind of adaptive behavior into a wireless network, such as the BlueSpot system. The BlueSpot system was used exemplarily to explain how these mechanism can be classified and afterwards integrated step-by-step. Therefore, this chapter started with the description of various Bluetooth topologies, since the mainly used communication technology of the BlueSpot system is Bluetooth.

Next, the basics needed in order to construct an own network forming approach were introduced. This was done by use of geometric structures, such as the relative neighborhood graph, the Gabriel graph, or the Yao graph. In order to form a Bluetooth-based Scatternet, the most promising structure is the high-degree Yao graph approach. By use of it, the maximum size of a Piconet as well as the master-slave relation between the involved nodes can be taken into account, which is not possible with the others.

The BlueSpot system is a solution that is able to adjust itself to the demands mobile services make. Therefore, an elaborate network control mechanism was presented. Beginning with the individual categories of adjustments, all approaches were brought into a clearly defined order of application. Additionally, the approaches were classified concerning the network view presented in 6 as well as their position within the middleware stack. At the end of this chapter, the handover functionality of the BlueSpot system was introduced. This is needed in order to support the mobility of client nodes and thus to provide a network infrastructure comfortable for the users.

Chapter 8

Mobile Services

The main task of the BlueSpot system is to enable mobile services of any kind. Therefore, this chapter shall first introduce the main characteristics of mobile services concerning their properties. The architecture mobile services must have in order to run on the BlueSpot system shall be shown next. Here, the different kinds of mobile service implementation methods shall be described and in addition possible software concepts for the structure of such services discussed. This mobile service description is rounded up by the explanation of the service mobility needed to transport a mobile service from one node to another.

As described in section 6.2.2, each service is equipped with a meta information base (MIB). In this chapter, this MIB shall be investigated in more detail. Additionally, each node includes its own information database that contains the current status of its resources. The approaches used to compare the available resources with those that are needed by the service as well as the evaluation mechanisms shall be explained. Latter are used to decide whether a service can be started or not. The subsequent section concentrates on the life cycle of a mobile service. The process designed to develop and afterwards test a new service shall be explained. If the testings were successful, it can be run on the BlueSpot system.

Finally, the various example mobile services available for the BlueSpot system shall be described.

Contents

8.1. Characteristics of Mobile Services

In order to support mobile services, it is necessary to characterize them according to their properties. These properties are strictly linked to their requirements. A first distinction for mobile services is, whether their communication is based *on messages* or on *a continuous data stream.* While a simple chat service creates single messages in order to exchange information, an audio stream produces a constant data flow. As a result, the latter places much stricter demands to the system than a chat service does. In case the audio service is enhanced by an reverse audio channel, this service can be used as a telephony service.

A second characteristic is important to enable a proper service quality. This is the *latency time* that occurs during data exchange of an end-to-end communication path. Current telephony services such as VoIP services require a maximum latency time of 300 milliseconds. The moment the latency time is higher, the two persons using the service would interrupt each other. Generally, tests showed that for a high quality service, the latency time must remain below 150 milliseconds [James et al., 2004]. That way, these two persons can discourse comfortably.

The moment a video signal is additionally sent along with the audio stream, the available *bandwidth* must be kept in mind. The better the video signal is captured, the higher the occurring data rate will be. But especially with wireless networks, the throughput of the network is a limiting factor and thus must be watched incessantly.

Beside the limiting factors occurring due to the underlying network, the type of usage of the service can be taken to characterize it. In most cases a service is used in a *client-server* configuration. One side provides the service's point of contact, which is the server, the other side is responsible for establishing the connection to the server as in the role as client part.

Another configuration is a *peer-to-peer* based setup. In this case, each side can act as server as well as client. Any communication partner can communicate with any other one, with almost no restrictions [Cheriton and Mann, 1989].

Further, the result of the usage of a mobile service can change depending on the location at which it is used. Such services are named *location based service.* In order to run such a service properly, the user's position must be known. But positioning shall not be part of this thesis. Therefore, location based services play a marginal role here and are not discussed any further.

In summary, the following characteristics for mobile services are of interest:

- streaming vs. message based communication
- are the latency times critical?
- is the bandwidth critical?
- client-server communication vs. peer-to-peer communication

8.2. Mobile Services in the BlueSpot System

In general, mobile services are loaded and executed within the BlueSpot system by connecting them to the service interface. As a result, the service interface must provide all functionalities needed to run a service, such as finding a communication partner, or exchanging data with this partner.

Considering the definition of a service made by Dinkel et al. [Dinkel et al., 2006], a service consists of the following components: its *implementation*, a *service interface*, and *service meta information.* The implementation contains all functionalities the service provides, as expected. The service interface (not to be confused with the service interface of the BlueSpot middleware) contains all functionalities that are needed in order to access the service's implementation. Dinkel et al. describe these functionalities as a subset of a *text console*, a *GUI*, a *protocol*, and an *API.* Furthermore, they denote the service interface as *horizontal interface.* The meta information base contains information a service user must interpret in order to use the service in a semantically correct way. The service itself runs on a service platform. This interface is denoted by Dinkel et al. as *vertical interface.*

This definition fully applies to the mobile services of the BlueSpot system. Again, the implementation contains the functionalities of the service, as expected. But considering the service interface, the application of this definition is more interesting. Here, the possibilities of the provided functionalities depend completely dependent the underlying hardware platform. If a service is run on a Gumstix, a GUI is not available. Only a text console can be used, and this must be accessed via an additional protocol (ssh or telnet) that runs independently from the BlueSpot middleware. Considering a smartphone or a PDS, usually a text console is not available. Here, the support of a GUI is needed in order to enable user interaction. Functionalities such as a protocol or an API can be provided on all types of hardware platforms. These restrictions must be taken into account during the course of designing and implementing a mobile service.

In the application of the service meta information, a mobile service that runs on the BlueSpot middleware differs from the given definition. Here, the service meta information is amended by the *meta information base* (MIB, see section 8.2.1) required for the BlueSpot system. This MIB is provided to the underlying middleware via the vertical interface and used to characterize the service's requirements. A further specialty of our mobile services is that they can be transferred from one service platform to another. Therefore, the mobile service must be provided as an isolated software packet.

Usually, mobile services for the BlueSpot system can be provided in three ways: as binary, as Java MIDlet or as container service. If it is provided as binary, it is implemented in native C code that is compiled to a binary shared library. The library can be loaded into the process space by the middleware during runtime. Subsequently, the mobile service can be started and its services used by a client. If the service is provided as Java MIDlet it must be run on a JVM installed on the target device independently to BlueSpot middleware. The resulting issues that must be taken into account have already been discussed in section 6.2.2. The applied JVM versions have been described in section 5.2. All the used versions are MIDP 2.0 compatible, which is a profile for the Java 2 Micro Edition (J2ME). This profile was especially construed for mobile devices, such as smartphones or handhelds. It provides libraries for using the small displays and reading inputs from a telephone keypad and its softkeys. The implementation of a GUI as part of the service interface is immensely simplified due to this.

As a result of the various hardware platforms supported by the BlueSpot system, a native C service must be compiled separately for each hardware platform. In order to handle this problem, *container services* were added to the BlueSpot system. Such a service consists of a zip-compressed package. Within this package, a predefined folder structure exists that includes various binaries, one for each supported hardware. Additionally, it contains a folder named `Java` that includes the Java version of the service, if available. In order to distribute a mobile service throughout the BlueSpot system and to gain *service mobility* as defined in section 3.4.2, this type of service is best to be used. A single binary of native C service can also be sent from one node to another without the need of a container service. In this case, the underlying hardware platforms must correspond to each other in order to be able to run the service on the new platform. Concerning a Java service, the hardware platform does not play a crucial role, since Java was developed to enable a hardware platform independent code. But, as a result of the limited range of functions of MIDP 2.0, the implementation of such types of services is very limited. In most cases, a Java service would drain too much of the resources. Native C services can be optimized additionally, and thus function with better performance.

In some cases it is useful to divide a service into two parts. One part of the service is then responsible for user interaction, while the second enables the desired functionalities. The first part can be run on the client node. It is responsible for relaying the user commands to the second part. These can be run anywhere within the BlueSpot system. The Gumstix is the most suitable place for this due to its vast free resources in comparison with the other supported hardware platforms of the BlueSpot system.

8.2.1. Meta Information Base (MIB)

A mobile service always places demands to the system it shall be run on. These demands must somehow be captured in order to enable a prognosis whether it will function properly or not. In case of the BlueSpot system, this problem is solved by the use of the *meta information base* (MIB). In addition, knowledge about the status of the node the service shall be run on is necessary. This information is used as a counterpart for the comparison the prognosis is the result of. A distinction between the *node MIB* and the *service MIB* within the BlueSpot system is therefore made. The latter includes the demands the service makes, while the first displays the current status of the node.

In the following, the parameters of the service MIB shall be described in detail, and subsequently, the parameters of the node MIB. The selection process used to decide whether a service can be run or not will be described at the end of this section.

The parameters of the service MIB:

- *minimum bandwidth*: this parameter describes the minimum available bandwidth a service needs. Especially, streaming services demand a defined minimum of bandwidth in order to ensure their functionality.
- *maximum latency time*: the second parameter of interest for defining the qual-

ity of a network connection is the latency time. It describes the time span that elapses until data sent reaches its destination. Here, the maximum allowed time span is of interest, as the buffers of the receiver side would underrun if the latency time is exceeded, and thus the data flow would be interrupted.

- *minimum free memory space*: while the first two parameters are used to describe the minimum quality of the network connection, this parameter defines the minimum demands concerning the resources of the node. By usage of this parameter, the minimum needed working memory is specified. A definition of the minimum needed size of the persistent memory would not make any sense. Most devices that are used for the BlueSpot system such as the Gumstix do not provide this type of memory.

All these parameters define the minimum requirements of a mobile service. But in order to support and simplify the adaptive behavior mechanisms of the BlueSpot system, knowledge is also needed about the estimated maximum demands. Therefore, these parameters are also included into the meta information base, beside the preferred routing protocol and communication characteristics of the service. They shall be described in the following:

- *maximum bandwidth*: this parameter indicates the estimated maximum bandwidth the service will produce. In some cases, an estimation is not possible due to an unpredictable communication behavior of the service. In this particular case, the estimated bandwidth is set to the average throughput of the entire network system that was measured during the simulation runs. Setting it to the maximum value would result in the displacement of all other services that are running currently, otherwise the middleware would refuse the start of the service due to insufficient available resources.

- *maximum memory space usage*: this parameter defines the maximum needed working memory space.

- *preferred routing class*: depending on the communication behavior of the service, it can be useful to explicitly define a preferred routing class for a service, a routing class as it was introduced in section 4.2. The moment the service is started, a selection event is triggered. If there is more than one service running, the used routing algorithm is selected by the middleware globally. Therefore, it sends a request by use of a control message to all other nodes within the network in order to receive their preferred routing classes and their priority values. The returning values and the own value are used to calculate a mean value for each routing class spanning the entire network. The node with the highest mean value is determined and the predefined standard algorithm of this class is chosen as the next algorithm. If it is not the routing algorithm currently running, the middleware triggers an exchange event that is broadcasted to all other nodes.

- *type of communication*: this parameter is used to indicate if the service is a streaming or message-based service.
- *priority of the service*: the priority of a service indicates how the service must be handled in comparison to other services currently running on a node. The moment a particular service with a high priority shall be run and the requirements cannot be met with, another currently running service with a lower priority value is stopped. As a result, the resources of this service are freed and can be provided to the new service. This parameter is predefined manually.

It does not make sense to define a minimum or an average value for the latency time, since this value should always converge to zero. Additionally, the average value would give information about the latency times of the underlying network, but would not give any information about the service's requirements. Therefore, this parameter is excluded from the service MIB.

Additionally, the used CPU power could be of interest. But the ascertaining of this parameter is very difficult, especially on the very heterogeneous hardware platforms used in the BlueSpot system.

In order to make a prediction whether a service can be run or not, a counterpart is needed for the comparison of single values. As described above, the node MIB is used for this. Beside the information needed to perform the comparison with the service MIB, this MIB also contains the parameters that are provided to the adaptive behavior extensions. These can use the node MIB to determine own configuration issues.

The node MIB includes the following parameters:

- *modus of the node*: in case a S/S bridged network type is configured, the bridging nodes are not able to accept or to establish any connections. Therefore, all layers above the network adaption layer can be deactivated. Incoming messages are redirected to the other connection immediately. Possible values are `bridge` or `full service`.
- *current MTU size*: this parameter contains the maximum transmission unit (MTU) size of a data packet sent by the protocol layer. The protocol layer

itself contains a packet splitter that is responsible for data fragmentation. As a result, the maximum packet size can be bound to a static value, and the implementation of a routing protocol is simplified immensely. Additionally, this parameter can be used to calibrate the BlueSpot system to the underlying network technology. E.g. Bluetooth uses a different MTU size than IEEE 802.11. By adjusting this value to the underlying network technology, more performance can be gained.

- *inquiry interval size*: this parameter defines the time span between two inquiry processes. The smaller this interval is set, the more often the middleware searches for new or moved client nodes. But as a result of the smaller intervals, more command messages are produced. This requires bandwidth and thus should be configured with care.

- *service discover interval size*: on the other side of the middleware stack, the node polls periodically to obtain a list of all available mobile services within the entire network. The same rules apply for the adjustment of this parameter as for the previous parameter: the more often the middleware polls for the services list, the more current this list is, but also the more traffic is produced.

- *currently used routing algorithm*: the currently running routing algorithm is named by this parameter.

- *hardware platform*: this parameter is used to identify the underlying hardware platform. By its use, the middleware can find the correct version of a service within a container service.

- *available local resources*: this parameter contains a set of values that are composed of the available resources of the node. As a result, these are the parameters needed to perform the comparison with the service MIB. The set includes the current free memory, the network load, and a value for the current measured average latency time of the network. The latter parameter is gained by observing the in- and outbound traffic. The moment a mobile service starts its communication, the time until the awaited response returns is measured. The result is used for the average calculation. The rest of these parameters are read periodically from the system information base such as the `proc` directory of the Linux operating system.

The selection process used to decide whether a service can be run or not is part of the adaptive behavior extension module described in section 6.2.2. The decision is based on the application of the above parameters. Currently, the BlueSpot system uses a *first come first serve* (FCFS) approach, that additionally is aware of the priority of a service. This approach is part of the *local assignment of resources* mechanism of the resource reservation phase described in section 7.4.1. The moment a service shall be started, the MIBs are compared. If the needed resources are available, the service is started. Otherwise, the network control (see 7.4.2) is used to free the required resources. If the resources can still not be provided, the start of the new service is refused by the middleware.

Another method that forgoes the refusal of service starts can be applied by enabling the suspension of running services. Instead of refusing the start of a service, a service with a lower priority is suspended and its resources are revoked and assigned to the new service. This can be made for all types of resources, excepting the used memory. The most devices supported by the BlueSpot system do not have a persistent secondary memory storage, such as a hard drive or a memory card. The moment the used memory is revoked by the middleware, the service would loose its current state. A reactivation of the service would no longer be possible.

8.2.2. Code Life Cycle of Mobile Services

The life cycle of a mobile service shall now be described in more detail. Beginning with the development, every phase will be described until the service is run on the BlueSpot system.

Commencing with the *design and coding phase*, the functionalities of the new service are constituted. A template for both types - the native C version as well as the Java version - of services are provided and can be used. These templates are filled with the desired functionalities. Meanwhile, they are tested by use of the simulator and retested continuously. As a result, bugs can be found easily, and the operability of the service can be tested. In case of a native C typed service, the developer must decide whether he provides the binaries of the service for only one hardware platform or whether he additionally compiles the service for further ones. In latter case, the various versions must be collected within a container service.

After this phases is completed, the *evaluation phase* starts. During this phase, the parameters of the service MIB are specified and stored. This is done by use of the simulator and the available testing scenarios described in section 9.1.2.

The service is now ready for application and thus enters the *deployment phase*. Therefore, the binaries are placed at disposal by copying them to a designated node. In case the service shall be provided as container service, the binaries for the different hardware platforms are combined to a zip archive which afterwards is placed at disposal. Users wanting to use the new service must wait until the service is detected by the service discovery process run by the middleware of their own node. After detection, the service appears in a list the users can view. By selecting the service, it is automatically copied to its device. In case of a container service, the version of the service corresponding to the hardware is extracted into a defined directory. Finally, the service is started and can be used.

Sometimes when the node is in the deployment phase, it can be useful to readjust the MIB parameter. This can be the result of an exogenous change that affects the entire BlueSpot system. In this case, the service is rerun on the simulator, and the changed parameters are assigned to the service MIB. This phase is named *maintenance phase* within the BlueSpot system.

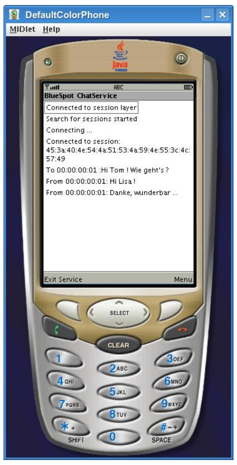
8.2.3. Example Mobile Service Implementations

In the course of our developments, we focused on the implementation of services that have very different requirements. Some need a high throughput rate, others depend on a maximum allowed latency time between the tow end nodes of the used communication path, whereas others yet again require additional functionalities provided by the BlueSpot system, such as handover, peer-to-peer functionality, or a streaming based communication. In the following, these services will be explained in detail and their main properties pointed out.

Chat Service

The Chat Service was the first mobile service we developed for the BlueSpot system. By its use, we investigated the behavior of the BlueSpot system concerning the service discovery mechanism as well as the different possible types of mobile services. Therefore, the chat service was implemented in all three types: as Java version, in various native C versions, and as container with the various native C binaries combined within it. The underlying communication model is a client-server configuration. For this reason, only two users can communicate with each other. Figure 8.1 shows a screenshot of the Chat Service in action.

As the name indicates, the service enables the chatting between two users. These two users can find each other by use of the service discovery mechanism. Every node that is currently running the Chat Service is depicted in the list of available services, presented to the user by the middleware.



```
Connected to session layer
Search for sessions started
Connecting ...
Connected to session:
45:3a:40:4e:54:4a:51:53:4a:59:4e:55:3c:4c:
57:49
To 00:00:00:01 :Hi Tom ! Wie geht's ?
From 00:00:00:01: Hi Lisa !
From 00:00:00:01: Danke, wunderbar ...
```

Figure 8.1.: Screenshot of the Java-based Chat Service

After a communication path to a selected destination is established, a session is created and the Chat Service started.

Car Service

By use of the Car Service the mobility aspects of the BlueSpot system can be investigated. Due to the high movement rate of one part of this service, the handover functionality as well as a high inquiry rate are required. In addition to this, the result of varying latency times can be demonstrated. A low latency time is required in order to run this service properly.

The Car Service is the result of modifying a remote controlled (RC) car in order to enable its integration into the BlueSpot system as a client node. This car is named Mini Mauler, as it is called by the manufacturer, and is depicted in figure 8.2.

The development of the hardware and the corresponding software drivers was performed by Schmidmeir and Schmidmeir [Schmidmeir and Schmidmeir, 2007]. They constructed a *printed circuit board* (PCB) that is connected to a Gumstix. The PCB board contains a microcontroller that communicates with the Gumstix via a UART interface. The steering servo as well as the electrical motor of a RC car are addressed by the microcontroller. In the left picture of figure 8.2, the completely assembled car can be seen. In the right picture, the car body was removed, so that the PCB board we developed can be seen. The Gumstix is placed on the bottom side of the board. The antenna is used to enhance the WLAN signal. The car

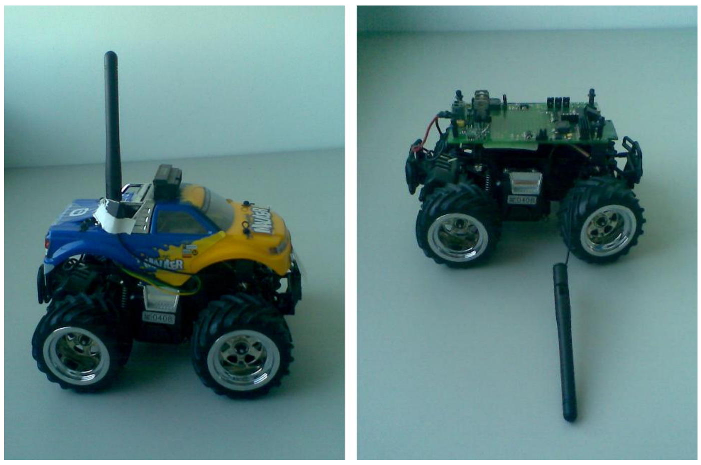


Figure 8.2.: The Mini Mauler: RC car with its own hardware that enables the steering of the car by aid of the BlueSpot system

additionally supports Bluetooth, due to the on-board Gumstix' Bluetooth chip.

The BlueSpot system is installed on the Gumstix. As a result, the car can be integrated into the BlueSpot system as a client node. One part of the Car Service, which is responsible for the steering, is run on the car's Gumstix, whereas the part that contains the graphical user interface was developed as a native C service for the Nokia smartphones. By use of these two parts, the car can be controlled with the aid of the smartphone's telephone keypad. On top of the car, we placed a distance sensor that is used to measure the distance to a obstacle. The measurement results are transferred to the destination node and are shown on the display of the smartphone.

pH/temp Service

In order to investigate the behavior of the BlueSpot system during communication that effects a large amount of nodes, the pH/temp service was developed. It produces a constant data rate that can be easily adjusted to own required values. As a result of this, this service can be used to analyze and demonstrate issues concerning the throughput rate of the underlying network over a large amount of hops as well as the behavior of current and new routing protocols within complex network topologies.

The basis of the service is a PCB board that was constructed by Schmidmeir and Schmidmeir [Schmidmeir and Schmidmeir, 2007], additionally to that of the Mini Mauler. The signal of a pH electrode and temperature sensor can be monitored.

The PCB board also contains a microcontroller that is responsible for the A/D conversion of the analogue signal of the two sensors. This board is connected to a Gumstix, which is equipped with the BlueSpot middleware. By use of the pH/temp Service, the measured data can be transferred throughout the BlueSpot system and viewed on the client screen which runs on the client part of the service. The service is available in the native C version for both the PDA and the Gumstix. By use of the service, the pH electrode can be calibrated as well. This is necessary each time the sensor board is restarted. See figure 8.3 for a screenshot.

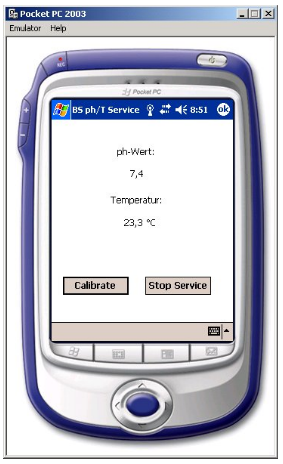


Figure 8.3.: Screenshot of the pH/temp Service

As you can see in the screenshot, the client side of the service displays the currently measured pH and temperature values. By pressing the `calibrate`-button, the calibration process for the pH electrode is initialized and executed. The needed steps for this process are displayed on the client node's screen.

CamCar Service

The CamCar Service is an extension of the Car Service. It can be used to demonstrate various adaptive behavior extensions and adjustments. On the one hand, the service supports different classes of quality. Due to a higher available bandwidth, a better class of quality can be selected, and vice versa. On the other hand, the capturing and encoding of a video signal requires hardware resources. As a result, available resource reservation mechanisms as part of the *node with middleware view* (see section 6.2.1) can be tested and demonstrated.

Our second RC car, the CamCar, had all the functionalities of the Mini Mauler, but additionally, a USB webcam was mounted onto its roof, as can be seen in figure 8.4.

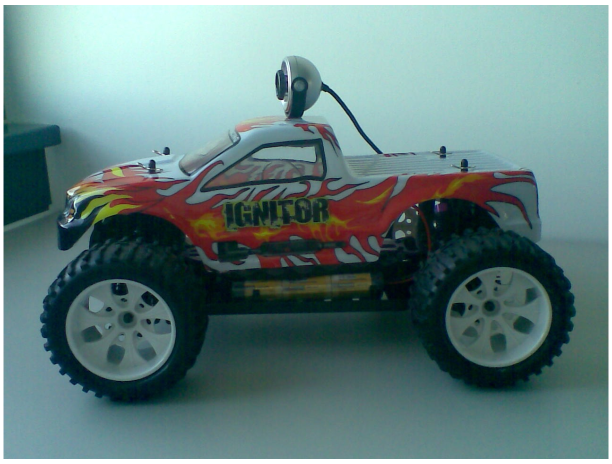


Figure 8.4.: The CamCar with the USB webcam on the roof

The video signal captured by the webcam can be transferred to the steering part of the CamCar Service and shown on the display. In order to reduce the needed bandwidth, the signal is compressed into a jpeg-based data stream, which is available in two different qualities. The standard signal has a resolution of 640 x 480 (VGA) pixels. By reducing this resolution to 320 x 240 (QVGA) pixels, the needed bandwidth is decreased by factor four and a second class of quality is defined for the service.

File Sharing Service

The File Sharing Service is used to construct a peer-to-peer network on top of the BlueSpot system. A defined amount of data can be sent across the network by selecting a file from another node that participates with the peer-to-peer network. The resulting data traffic can be used for measurement and analyzing proceedings. Additionally, the traffic stream can be used to demonstrate the maximum throughput of the system as well as the functionality of newly developed adaptive behavior extensions concerning the single domain and the infrastructure view. In addition to this, the File Sharing Service places high demands to the currently used routing protocol, as many communication paths are to be established simultaneously. The File Sharing Service is available in native C as well as in Java. Hence, it provides support for all hardware platforms within the BlueSpot system.

Audio Streaming Service

By aid of the Audio Streaming Service, a constant audio stream is sent throughout the BlueSpot system. The audio stream is compressed into the Ogg Voribs [Xiph community, 2008] format. At the client node, this stream is decompressed and played. The main task of the Audio Streaming service is to demonstrate a steady data stream throughout the network. It can be used for investigation proceedings concerning the latency time of the underlying network. This can be achieved by playing the audio stream on both end-points: on the client node as well as on the node the audio stream is fed into the BlueSpot system with. As a result, the latency time is presented by the delay between the two audio signals.

TCP/IP over BlueSpot Service

In order to support all TCP/IP based services through the BlueSpot system, the *TCP/IP over BlueSpot Service* can be used. When the service is run, a virtual network interface is provided to the operating system. This is achieved by use of the TUN/TAP virtual network kernel driver, which is usually used to connect to a VPN. As a result, the BlueSpot middleware can be run in the user space of the node. With the aid of the TUN/TAP driver, all requests are redirected to the kernel space and can be processed there by the kernel of the operating system. Another user space application, such as a ssh client program, can use the new virtual network device and thus function without any need of modification. This service is only available for Linux based nodes and is implemented in native C. A support of the smartphones and of the PDA is not possible due to the missing of this mechanism on such devices.

Usually, this service is applied by designating one node as gateway between the BlueSpot system and a network connected to the internet. As a result, all communication within the BlueSpot system will occur between the client nodes and the one gateway node. The network load in vicinity of this node will therefore be very high. New network forming approaches can be installed and tested in order to handle the locally unbalanced network by finding intelligent distributions of network connections and thus overcome possible bottlenecks.

Benchmark Service

The last service to be described is the Benchmark Service. It is used to pointedly measure the current bandwidth and the latency time of the underlying network. Execution of this service is divided into two steps. During the first step, ping packages are sent to the counterpart node and are answered by it. The time that elapses between the sending of a packet and the receiving of the answer is measured and divided by two in order to approximate the latency time of the network. The amount of ping packets is not predefinded. Therefore, the more pings are sent, the better the averaged result will be.

During the second step, packages in various pack sizes are transmitted to the counterpart node. The bandwidth of the network can be measured that way. The service starts to send packages with a size of 1 kilobyte. After 50 packages have been sent, the size of the packages is doubled and the procedure repeated. This is done until the packages have reached a size of 2048 kilobytes. The time needed to send 50 packages of the same size is measured and the average value gained. This is displayed on the screen of the client node. By dividing this value through the amount of data sent, the throughput can be calculated.

The service is available in native C code. A Java version does not exist. The controlling of the service is done via a text console. Therefore, is was designed to run on a Gumstix or within a terminal of a desktop computer. The Benchmark Service was the basis for our measurement proceedings presented in section 9.2. It is the easiest way to produce a defined amount of traffic on a communication path throughout the network. In order to establish a second communication path, the Benchmarking Service node must be run on two further nodes, which connect to each other by use of the service. In addition to this, this service can be used for testing proceedings of a new mobile service. The data transfer of this new service can be influenced by use of the Benchmarking Service, and thus its behavior during low data rates can be simulated.

8.3. Summary

This chapter was divided into two sections. In the first section, the characteristics of mobile services that are important within the BlueSpot system were described. These characteristics were used beside other parameters to introduce the construction of the service meta information base as well as the node meta information base. In addition, two approaches were presented that can be used to influence the network control behavior of the BlueSpot system. These approaches are needed to enable the evaluation of the available resources in order to decide whether a service can be started or not. The decision process can be supported by use of priorities or by use of revoking resources that are already assigned to another service. That way, the decision process is the entry point for adaptive behavior mechanisms described in the previous chapter.

Subsequently, the life cycle of a mobile service within the BlueSpot system was introduced. This life cycle includes the designing and coding of a mobile service, as well as its evaluation concerning the parameter estimation for its service MIB. It also includes its deployment and the eventually needed reevaluation of the service MIB parameter the moment the service does not function properly after it was deployed. This can be the result of not-well estimated MIB parameters as well as of a configuration change concerning the entire BlueSpot system.

At the end of this chapter, the various example mobile services currently available for the BlueSpot system were presented. The main idea behind the selection to implement each service was given and possible scenarios for measurements of selected parameters and adaptive behavior mechanisms were shown up.

Chapter 9

Results and Benchmarking

An elaborate monitoring tool for the BlueSpot system shall now be presented. This tool can be used to view the current situation of the BlueSpot system focusing on each layer of the middleware individually. Furthermore, the environment used to simulate the BlueSpot system is introduced. By utilization of this environment, the services' MIB parameter can be stated and new adaptive behavior extensions tested before applying them in the real system. Next, scenarios shall be presented that are used as templates in order to test a mobile service in selected situations. Based on these templates, many common situations can be modeled and simulated.

Subsequent to this section, selected benchmarking results made by use of the BlueSpot system shall be presented. The used configuration of the system and the benchmarked parameters shall be described first. These configurations and parameters are used to explain the procedure for executing the benchmarking process. This is followed by the presentation of the results that we gained: the measurements of throughput and latency times of various Scatternet configurations. The influence of the size of data packets for communication by use of the BlueSpot system shall then be investigated. Additionally, the behavior of M/S bridged networks is compared to S/S bridged ones. This chapter will be rounded up by the investigation of the duration of establishing a Bluetooth connection. During a handover, this parameter is of great interest, as it is a result of the interruption of currently running communication processes.

Contents

9.1. Monitoring and Simulating the System

Further software applications and tools are needed in addition to the BlueSpot middleware. Beside various shell scripts, the two most important applications are the monitoring tool and the connection of the middleware to the NS2 simulator. These two applications shall be explained in more detail in the following.

9.1.1. Monitoring Tool

The monitoring tool was developed by Rieck [Rieck, 2007] as part of his Bachelor thesis. In order to collect information of the current status of the BlueSpot system, each node has the ability to continuously send monitoring messages to a predefined destination. This can be achieved in two ways: the first is to directly send messages to the monitoring tool by use of UDP packets. In figure 9.1 this is depicted by the red arrows from a node to the monitoring tool. Each node must be connected to an IP based network that can access the monitoring tool. The IP address of the host with the monitoring tool running on it is previously given to each node before the BlueSpot system is started. In order to keep the influence of the monitor as small as possible, datagram packages are used that do not need to be acknowledged. Due to the continuous monitoring process, the loss of a packet is not critical, thus the sending of datagrams is sufficient.

The second way is to use the BlueSpot system to relay monitoring messages. Some nodes shall not be connected to an IP based network, as this would influence the behavior of the node too much. Then, monitoring messages can be relayed to a node that is connected to an IP based network. This forwards the messages for the node to the monitor. This way can be seen in figure 9.1 on the top right side of the network. The client node uses the S/S bridge for relaying its messages to the monitoring tool.

As can be seen on the left side of figure 9.1, the slave node is not directly connected

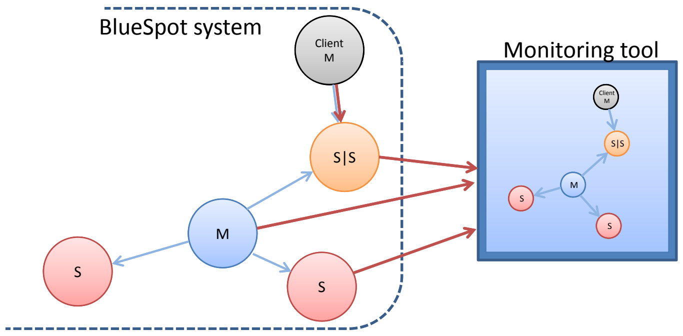


Figure 9.1.: Monitor message relaying

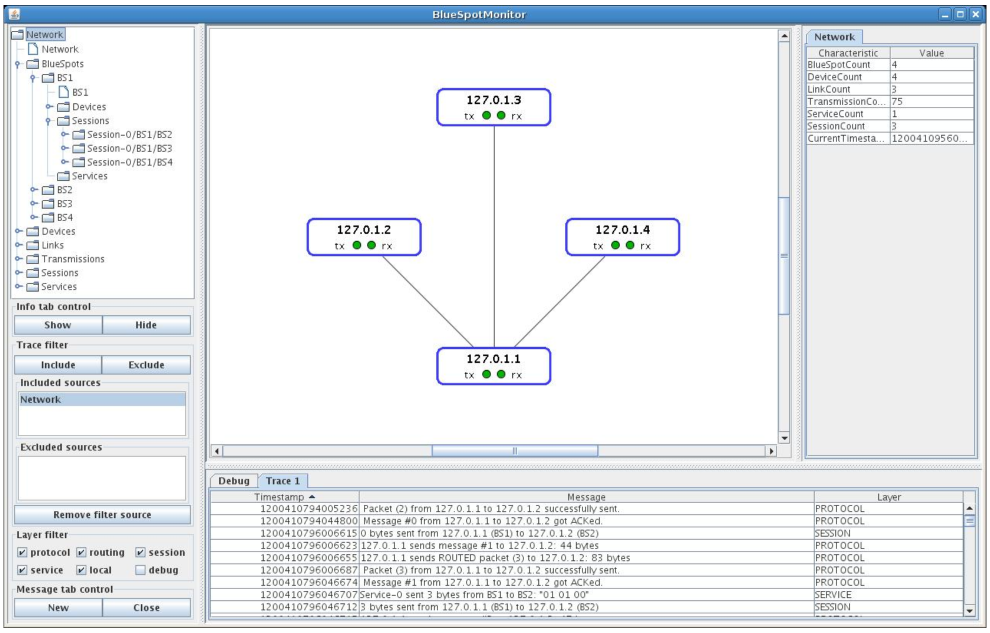
to the monitoring tool. Nevertheless, it is depicted on the monitoring tool due to the monitoring messages of its neighboring node. These contain that the node must exist as a result of occurring data messages that are sent to this node.

The monitor does not provide any possibility for taking influence on the BlueSpot system. In the course of development, this was one of the main tasks due to the need of keeping the disturbances originating from the activated monitoring mechanism as low as possible. E.g. by relaying monitoring information through the BlueSpot system, bandwidth is used. This in turn influences currently running services that are dependent on the available bandwidth. A screenshot of the monitor can be seen in figure 9.2.

In the middle of the application window, the current status of the network can be seen as an edge graph. This is surrounded by additional monitoring information that can be grouped, ordered, and filtered in various ways. In order to group information, the monitor supports *traces*. By use of such a trace, all messages that concern a selected work flow are collected as they appear. Afterwards, it is possible to reconstruct and follow the exact work flow as it happened before in the monitored system. The complete description of the monitoring tool can be found in Rieck [Rieck, 2007].

9.1.2. Simulation Environment

In order to provide a simulation environment for the BlueSpot system, the NS2 simulator is used [ns2webpage, 2008]. In his diploma thesis, Metzger [Metzger, 2007] implemented a new network device library named `BSSimDev`. By use of this device, all requests and answers of the BlueSpot middleware are redirected to the simulator



Network

Characteristic	Value
BlueSpotCount	4
DeviceCount	4
LinkCount	3
TransmissionCo...	75
ServiceCount	1
SessionCount	3
CurrentTimesta...	12004109560...

Timestamp	Message	Layer
1200410794005236	Packet (2) from 127.0.1.1 to 127.0.1.2 successfully sent.	PROTOCOL
1200410794044800	Message #0 from 127.0.1.1 to 127.0.1.2 got ACKed.	PROTOCOL
1200410796006615	0 bytes sent from 127.0.1.1 (BS1) to 127.0.1.2 (BS2)	SESSION
1200410796006623	127.0.1.1 sends message #1 to 127.0.1.2: 44 bytes	PROTOCOL
1200410796006655	127.0.1.1 sends ROUTED packet (3) to 127.0.1.2: 83 bytes	PROTOCOL
1200410796006687	Packet (3) from 127.0.1.1 to 127.0.1.2 successfully sent.	PROTOCOL
1200410796046674	Message #1 from 127.0.1.1 to 127.0.1.2 got ACKed.	PROTOCOL
1200410796046707	Service-0 sent 3 bytes from BS1 to BS2: "01 01 00"	SERVICE
1200410796046712	3 bytes sent from 127.0.1.1 (BS1) to 127.0.1.2 (BS2)	SESSION

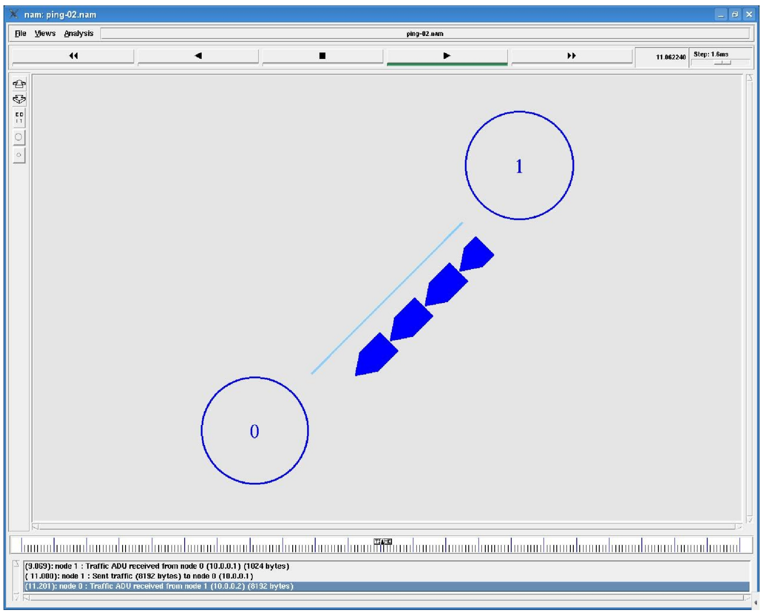
Figure 9.2.: Screenshot of the monitoring tool

and worked up there. For testing mobile services and for gaining the services' MIB parameters, various predefined scenarios are needed. These are provided by a set of predefined network formations, which are described in the next section. In order to simulate Bluetooth-based as well as TCP/IP-based networks, the simulator supports the simulation of both technologies. The desired technology is specified within the scenario description that is provided to the simulator.

All scenarios are defined in a separate network formation file. This file contains the configuration of available nodes as well as the topology the nodes will establish after the simulation is started. Ensuing the successful completion of the formation, the mobile service that shall be investigated is started. This and further exogenous events that shall appear during the simulation process are provided in a TCL script. This script must be provided by the tester, thus it must be adapted manually. In order to assist the tester in this procedure, various example scenarios are provided by the BlueSpot system. They already include the configuration parameters such as the available bandwidth of the underlying network or the mean time needed for a hop from one node to another. The detailed description for running such simulations is described in [Metzger, 2007].

9.1.3. Scenarios

In the following, predefined network formations are discussed. The shown screenshots are the result of illustrating the run simulations by aid of the `nam` tool, which is part of the NS2 simulator package. During a simulation run, all events are written into a simulation file. After the simulation is completed successfully, the `nam` tool is used to present the complete simulation process in a graphical illustration.



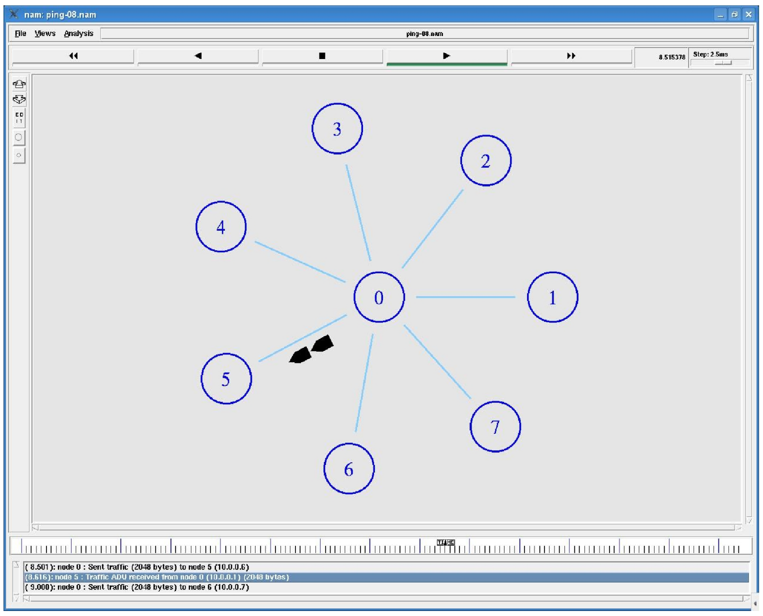
```
(9.069): node 1 : Traffic ADU received from node 0 (10.0.0.1) (1024 bytes)
( 11.000): node 1 : Sent traffic (8192 bytes) to node 0 (10.0.0.1)
(11.201): node 0 : Traffic ADU received from node 1 (10.0.0.2) (8192 bytes)
```

Figure 9.3.: Illustration of a simulation run with the `nam` tool

As can be seen in the screenshot of figure 9.3, two nodes appear in the simulation. Node 1 is currently sending four packets to node 0. Underneath the graphical illustration of the nodes, a time scale presenting the current simulation times as well as the detailed information of the currently appearing communication can be seen. These details concern occurring events such as the exchange of a data packet.

Full Piconet

The first scenario to be presented here is a Piconet with eight nodes. In figure 9.4 the master node with the number 0 is situated in the middle. The maximum amount of seven slave nodes possible for a Piconet is arranged around the master node.



```
( 8.501): node 0 : Sent traffic (2048 bytes) to node 5 (10.0.0.6)
(8.616): node 5 : Traffic ADU received from node 0 (10.0.0.1) (2048 bytes)
( 9.000): node 0 : Sent traffic (2048 bytes) to node 6 (10.0.0.7)
```

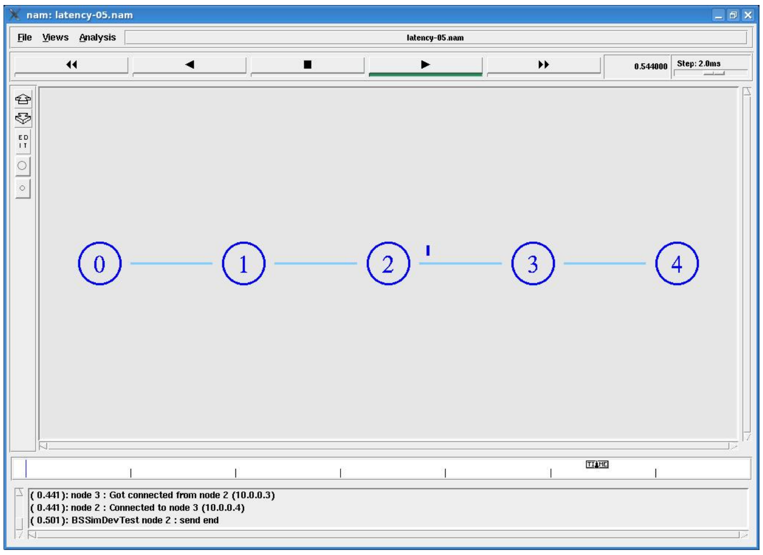
Figure 9.4.: Simulation of a full Piconet

All communication must be relayed by the master node. As a result, this simulation can be used to gain the maximum throughput the mobile service will need. This is achieved by starting a service on two slave nodes. The entire occurring communication of this service is measurable at the master node. In order to simulate a stress test, the Benchmarking Service can be started additionally on two further nodes. By varying the size of the packets sent by the Benchmarking Service, the behavior of the service to be investigated can be tested in case not enough bandwidth is available. Additionally, the hardware resource requirements of the service can be gained by regarding each node individually.

Linearly ordered network

By arranging several nodes in a linear order, this scenario allows the investigation of the latency time aspect in more detail. The scenario is depicted in figure 9.5.

Five nodes are lined up in a row. The master-slave allocation is usually turned off in this scenario, due to the desired interoperability of the BlueSpot system concerning the used communication technology. M/S and S/S bridges are the result of the Bluetooth specification. If this specialty would be added permanently to the



```
( 0.441 ): node 3 : Got connected from node 2 (10.0.0.3)
( 0.441 ): node 2 : Connected to node 3 (10.0.0.4)
( 0.501 ): BSSimDevTest node 2 : send end
```

Figure 9.5.: Simulation of a linearly ordered network

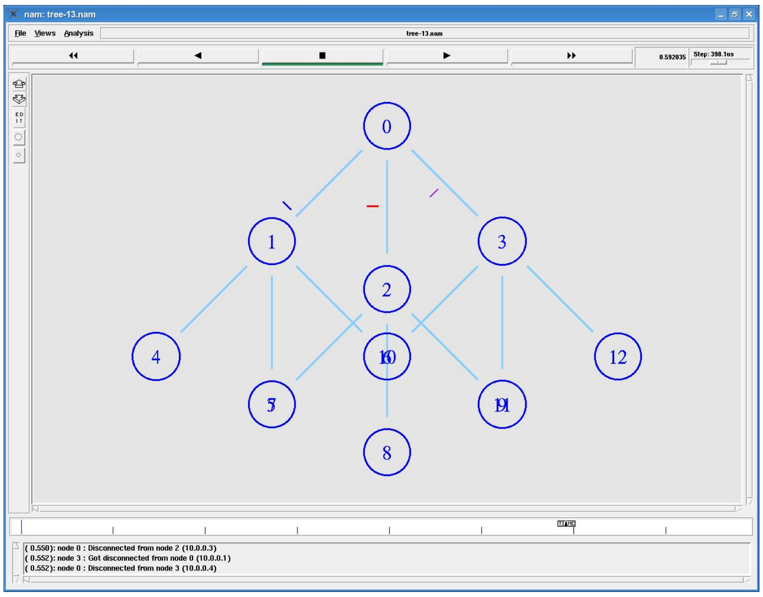
simulation scenario, it would lose its generality due to the lack of such restrictions in the standards of other network communication technologies. By turning the nodes' distribution of roles on, a Bluetooth-based linearly ordered network can be easily simulated.

By running one part of the mobile service on an endpoint of the row and the other part on varying nodes, experiments with the latency time can be made. The more hops are placed between the two service parts, the longer the resulting latency time will be. As a result, the maximum latency time as well as the average bandwidth can be approximated.

Spanning Tree

The third testing scenario defines a spanning tree formation. With this scenario, more complex simulations can be run due to the thirteen available nodes. As a result, different routing algorithms can be applied, and the one that fits best for a mobile service found. The complete formation can be seen in figure 9.6. Due to an automated presentation of the nodes done by the `nam` tool, some nodes are overlapped in this screenshot.

This scenario can also be used in order to test new adaptive behavior extensions.



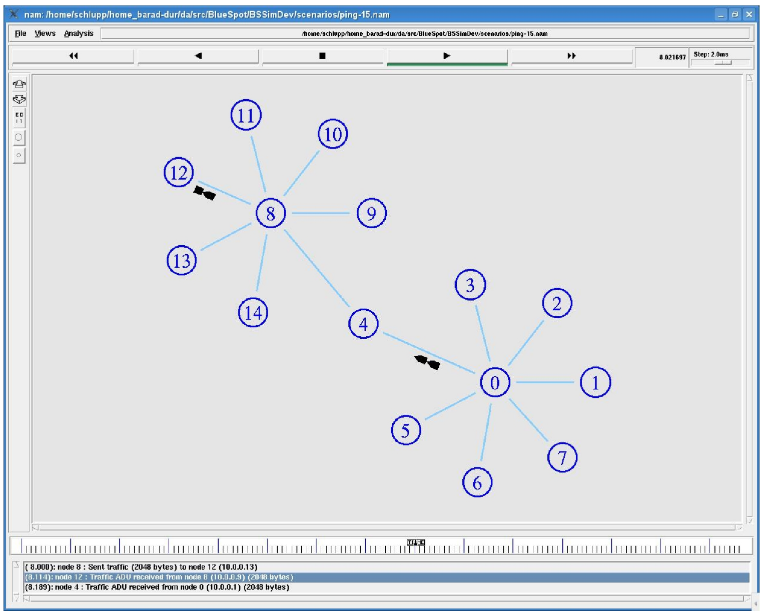
```
( 0.550): node 0 : Disconnected from node 2 (10.0.0.3)
( 0.552): node 3 : Got disconnected from node 0 (10.0.0.1)
( 0.552): node 0 : Disconnected from node 3 (10.0.0.4)
```

Figure 9.6.: Simulation of a network organized as spanning tree

A distinction between master, M/S bridges, and slave nodes is usually not made, but can be turned on if desired. E.g. a modification approach can be applied and tested. The characteristic of a spanning tree formation is that it contains only one connection path between two nodes. By use of this characteristic, the behavior of the algorithm can be tested respecting a connection loss and the resulting breaking apart of the network. The moment this happens, the network must reconnect, and the correctness of the applied algorithm can be verified.

Simple S/S Bridged Scatternet

Like the spanning tree scenario, this scenario is used for testing new adaptive behavior extensions. The network contains two Piconets that are connected by a S/S bridging node. This is the main difference to the spanning tree simulation. Here, the center nodes are handled as master nodes of each Piconet. The bridging node is a S/S bridge. All other nodes are handled as slaves. See figure 9.7 for the node configuration.


nam: /home/schlupp/home_barad-dur/da/src/BlueSpot/BSSimDev/scenarios/ping-15.nam
File
Views
Analysis
/home/schlupp/home_barad-dur/da/src/BlueSpot/BSSimDev/scenarios/ping-15.nam
8.021697
Step: 2.0ms
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
TIME

```
( 8.000): node 8 : Sent traffic (2048 bytes) to node 12 (10.0.0.13)
(8.114): node 12 : Traffic ADU received from node 8 (10.0.0.9) (2048 bytes)
(8.189): node 4 : Traffic ADU received from node 0 (10.0.0.1) (2048 bytes)
```

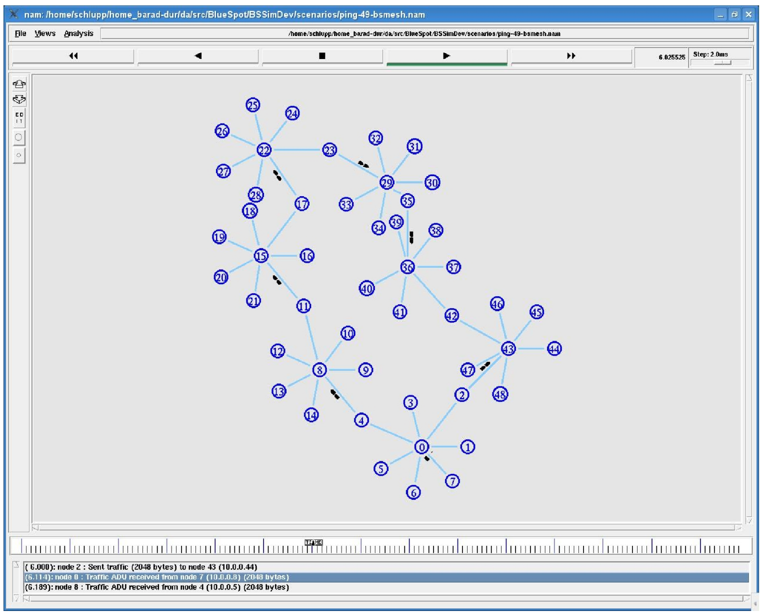
Figure 9.7.: Simulation of a simple S/S bridged Scatternet

Complex S/S Bridged Scatternet

The complex S/S bridged Scatternet simulation is the most demanding simulation available for the BlueSpot system., as it is the scenario that consists of the most nodes. The initial network is constructed as S/S bridged Scatternet with satellite nodes, and therefore each node is assigned a Bluetooth-based role. By applying a modification approach the topology can be restructured, and thus different types of topologies can be gained during runtime. The main field of application is to test extensions that apply especially to the Bluetooth based standard. But further extensions that fit to other fields of adaptiveness within the BlueSpot system can be also integrated and investigated. In figure 9.8 the entire formation can be seen.

As a result of the large amount of nodes, the simulation run needs a lot of computational power. Therefore, more complex simulations can take up to several hours. The aim of this scenario is to provide the possibility of testing new adaptive behavior extensions within complex environments. This scenario can be useful for testing modification approaches, new routing approaches, as well as resource reservation mechanisms.

All the simulation scenarios presented here are just a selection of possible network formations. Further formations can be constructed easily by editing the correspond-


nam: /home/schlupp/home_barad-dur/da/src/BlueSpot/BSSimDev/scenarios/ping-49-bsmesh.nam
File
Views
Analysis
/home/schlupp/home_barad-dur/da/src/BlueSpot/BSSimDev/scenarios/ping-49-bsmesh.nam
6.025525
Step: 2.0ms
TIME
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

```
( 6.000): node 2 : Sent traffic (2048 bytes) to node 43 (10.0.0.44)
(6.114): node 0 : Traffic ADU received from node 7 (10.0.0.8) (2048 bytes)
(6.189): node 8 : Traffic ADU received from node 4 (10.0.0.5) (2048 bytes)
```

Figure 9.8.: Simulation of a complex S/S bridged Scatternet

ing network formation file, and subsequently run in own simulations.

9.2. Benchmark Results

The BlueSpot system - in the form of its current implementation - is a proof of concept and it is not yet very stable. But with our present state of the implementation, it is already possible to demonstrate new approaches and mechanisms that concern the wireless networks technologies and configurations supported here.

In this section, various measurement results that concern Bluetooth-based networks will be presented. All of these measurements were made by use of the BlueSpot system. Some of these measurements were made by use of the simulator, but most of them were performed with real hardware components. Firstly, basic throughput measurements that take the underlying hardware into account shall be shown. Subsequently to this, the occurring latency times are investigated that are brought in correlation with the amount of hops. These fundamental measurements shall be rounded up by analyzing the dependency of the occurring data rate of the used packet size.

A more detailed investigation is made for the different behaviors of M/S and S/S bridged networks. In the course of the development of the BlueSpot middleware, we revealed great differences of these to network constellations concerning their performance. We used this revelation as an opportunity to analyze and afterwards gain the ability to explain this behavior.

At the end of this section we provide measurement results concerning the duration of a connection establishment. We discerned that a small amount of connection establishment approaches have a very long duration. Here, we can unfortunately only present the measurements results. An interpretation of these results we will present in the near future.

All results presented here must be considered as exemplary and constitutive. They show that the concept presented in this thesis is working and applicable. But due to the wide-reaching approach the BlueSpot system bases on, there are still many open questions that must be handled in the future. A selection of results of the benchmarking processes shall be shown next.

9.2.1. Benchmarking Procedures

In order to understand the following benchmarking approaches, it is necessary to explain the used benchmarking methods. These are described next in addition to their underlying BlueSpot middleware configurations.

Currently, we have three methods of performing measurements, each depending on the software configuration used for the benchmarking approach. The first bases on the measurement on the level of the protocol layer. The presented values in this chapter concern Bluetooth-based wireless networks. For this reason, initially, the used network library is the `BSBthDev` library. For the benchmarking process, we implemented an extra application that connects to the protocol layer of the middleware and uses its functionalities. In order to establish communication paths, the routing protocol DSDV was selected.

The second way of measurement connects to the BlueSpot system on the level of a mobile service. This is achieved by use of the Benchmarking Service that contains predefined testing scenarios which can be run in random order. This configuration enables the integration of any wanted set of functionalities of the BlueSpot system into the measurement process. Also, any available adaptive behavior extension can be easily integrated into the measurement process, and can thus be taken into account. Here, the standard configuration of the BlueSpot system is used, which runs without any additional adaptive behavior extensions and the standard DSDV routing algorithm.

By use of the simulator, results are gained in the mentioned third way. The simulator is connected to the BlueSpot middleware with the aid of the `BSSimDev` library, and thus the Benchmarking Service can be used to perform the measurement processes. Analogously to the second way of benchmarking, the standard configuration with DSDV routing and no other extensions is used here.

9.2.2. Throughput Measurements

The available bandwidth of a Bluetooth-based network shall be shown apriori. Therefore, the throughput of a single connection is investigated. Early experiments have shown that the real throughput of a connection depends highly on the underlying hardware that was used. The theoretical values of 723,2 kbit/s respectively 2Mbit/s defined in the Bluetooth standard were reached by none of our hardware configurations, but were near the mark (e.g. 241,331 kBytes/s = 1930,648 kbit/s). The testing results proved to depend immensely on the hardware configuration. In order to reduce the influence of the hardware, we permuted our available hardware and additionally often repeated the measurement process. Afterwards the measured values were averaged. All measurements were made on the level of the protocol layer. The results can be seen in table 9.1 and are illustrated as a diagram in figure 9.10.

	1	2	3	4	5	6	7	8	9
one hop	117,0759	128,604	241,331	138,433	145,318	232,442	85,594	89,866	85,672
two hops	34,183	37,761	65,065	67,559	83,892				
seven hops	5,242								

Table 9.1.: Values of throughput measurements based on the protocol layer in [kB/s]

All used network constellations are organized linearly. The installed bridging nodes are configured as M/S bridges (see figure 9.9 for an illustration). For the first set of benchmarking runs, the used topology consists of two nodes. The available hardware components were arranged randomly in order to construct different random hardware configurations. By chance, nine configurations were selected.

The seven-hop constellation consists of all types of hardware devices. These are the on-board chips of the Gumstix, and two types of USB Bluetooth dongles: one supporting the Bluetooth standards V.1.2, and one supporting the Bluetooth standard V2.0+EDR. The two-hops measurements are the result of connecting three Gumstix together - first two values - and combining two USB dongles Bluetooth V.1.2 and one USB dongle Bluetooth V.2.0+EDR - third and fourth value - and one USB dongle Bluetooth V1.2 connected to two USB dongles Bluetooth V.2.0+EDR - fifth value. Considering the nine one-hop configurations, the first two values are the result of connecting two USB dongles Bluetooth V.1.2, the third and the sixth values are the result of connecting two USB dongles Bluetooth V.2.0+EDR. The forth and the third value are the result of one USB dongle Bluetooth V.1.2 and one USB dongle Bluetooth V.2.0+EDR. The last three configurations were made exclusively with combinations of Gumstix.

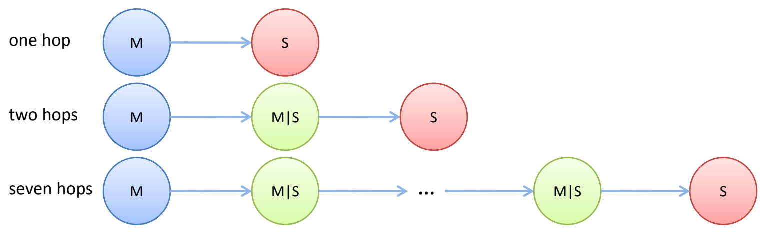


Figure 9.9.: Network constellations used for the throughput measurements

The second set of measurement processes depicted in the second row concerns topologies with tree nodes. Analogously to the first approach, different hardware components were used in a random order. Due to the few available hardware components and high connection break-up rates, only five measurement runs for this configuration could be achieved. Occurring connection break-ups can be explained by the overlapping of many Piconets. In this configuration the resulting topology contains four Piconets (one master, three M/S bridges and one slave) that disturb each other immensely. The moment the influence of other Piconets grows too high, some of the connections within a Piconet will break up. The reestablishment of such connections is tried by the BlueSpot system automatically, but cannot be guaranteed. That way, the BlueSpot system tries to minimize possible system failures and to enhance its stability. Unfortunately, this falsifies our measurement results, therefore, the best results were selected here.

Instable connections are even more relevant for the third measured network constellation. This one includes eight nodes. Due to the use of M/S bridges, the network consists of seven Piconets. The moment these Piconets overlap each other, steady connections cannot be assumed anymore. Due to this, only one successful measurement could be achieved that yields an acceptable outcome. The results of the three test assemblies are depicted in a diagram depicted in figure 9.10.

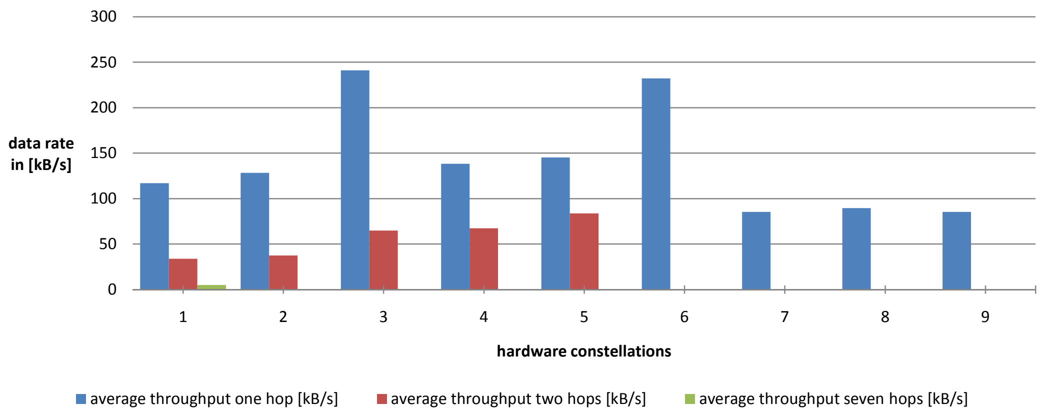


Figure 9.10.: Throughput measurements of the `BSBthDev` for one, two and seven hops

The results of the one-hop network are displayed by blue bars, while the results of the two-hops and the seven-hops measurements are marked as red and as green bars. It can be seen that the results of the one-hop configurations vary very much according to the throughput rate. Especially, during the usage of the hardware constellations three and six, the achieved data rate was nearly twice the data rate of other constellations. In the course of our experiments, it became apparent that the application of Bluetooth USB sticks brings a much better performance than the usage of the on-board Bluetooth chips of the Gumstix. The results of the Gumstix are depicted in constellations seven, eight and nine. In order to be able to explain this behavior, the implementations of the hardware and their connection to the hardware platform the BlueSpot system runs on must be investigated in detail in another place. This shall not be part of this thesis.

When comparing a one-hop scenario to a two-hop scenario, the throughput rate decreases dramatically. Furthermore, taking the seven-hop scenario into account, the throughput rate drops to approximately 5 kilobytes per second, which is very low. This high decrease can be explained by the bad scalability of M/S bridged network and is discussed in section 9.2.5 in more detail. As a result, the establishment of a Bluetooth-based Scatternet with seven or more nodes that is based on M/S bridges can be made but comes along with the restriction of a very low data rate. Therefore, long communication paths should be avoided.

9.2.3. Latency Times Measurements

This section presents measurements concerning the average latency times in comparison to the hop rate. The hardware dependency is put into perspective by averaging over a random selected set of hardware configurations and a high rate of iteration of the benchmarking processes. The used network topology also bases on M/S bridging nodes, where the nodes are ordered in a linear manner. All benchmarking processes were made by use of the protocol layer. The gained results are shown in table 9.2.

	$1hop$	$2hops$	$3hops$	$4hops$	$5hops$
average	22,345	58,806	62,145	91,785	144,341
maximum	45,356	108,459	137,2	185,67	265,563
minimum	20,943	49,491	59,873	83,833	138,457

Table 9.2.: Latency times measurements based on the protocol layer in [ms]

The table contains the average values as well as the corresponding maximum and minimum values for each performed measurement process. The results of applied network configurations with two to six nodes are depicted in figure 9.11.

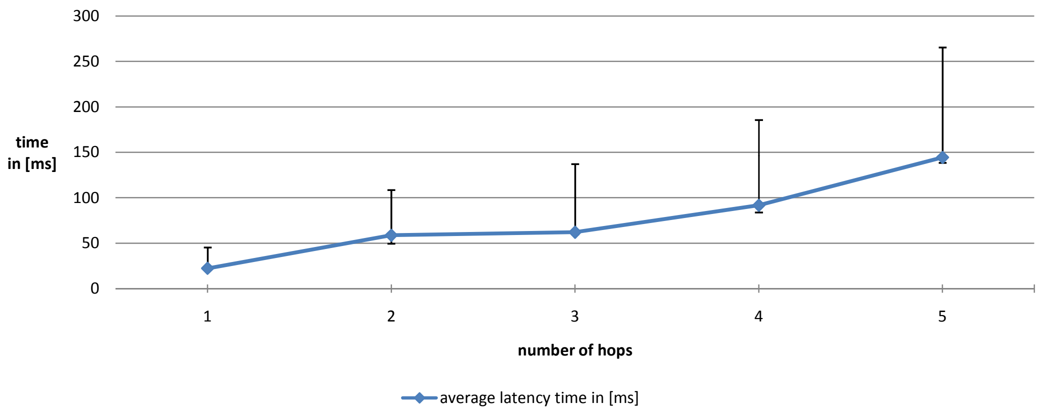


Figure 9.11.: Comparison of latency times in comparison to the number of hops

The results depicted in the graph show that the average latency time increases in relation to the number of hops, as expected. The interesting result in this measurement process is the large interval between the minimum and the maximum measured latency times. The more nodes are involved in a communication path, the larger the possible interval of latency time is. Especially in constellations with five or more nodes, the interval grows so large that it becomes quite difficult to make any statement of particulars describing the quality of the underlying communication path.

Mobile services with high demands according to their maximum allowed latency time can be impaired due to these results.

9.2.4. Packet Size Measurments

Throughput rates stand in direct correlation with the packet size used on the level of the protocol layer. In order to investigate this correlation, we performed measurements by use of the simulator. The results can be seen in table 9.3 and are presented as diagram in figure 9.12.

packet size	1	2	4	8	16	32
average	21,165	25,752	37,030	58,441	84,843	110,408
maximum	21,441	26,137	37,634	59,107	86,067	112,171
minimum	16,074	21,394	30,487	50,868	78,309	101,226
packet size	**64**	**128**	**256**	**512**	**1024**	**2048**
average	129,503	141,901	149,285	154,493	156,250	157,136
maximum	131,867	145,015	151,064	156,361	158,334	158,819
minimum	124,901	128,681	147,216	149,173	154,499	156,417

Table 9.3.: Values of packet size measurements based on the protocol layer in [kB/s]

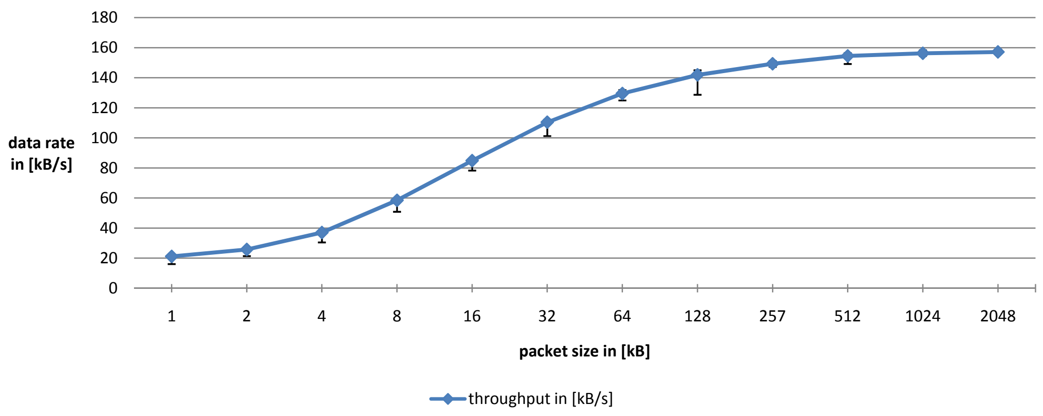


Figure 9.12.: Throughput measurements with the simulator for one hop with different packet sizes

The underlying network is a Piconet that consists of a master and a slave node. The resulting values for the twelve different packet sizes are averaged, but the corresponding minimum as well as the maximum value are also integrated into the graph.

The maximum values do not differ greatly from the average values, therefore, the points of the graphs lay very near each other. In fact, the maximum values can hardly be seen. The throughput rate increases very fast when small packet sizes are sent, as can be seen on the left side of the diagram. But, beginning with a packet size of 32 kilobytes, the slope of the curve starts to decrease again. At the maximum packet size of two megabytes, the throughput remains nearly static. It is important to keep in mind that the effective values of all throughput rates depend on the used hardware. The values presented here are an approximation.

In contrast to the throughput rate, it does not make sense to measure latency times in relation to the packet size (see section 8.2.1). The result would not be transferable to a real world configuration due to too many additional factors that would be needed to be taken into account. E.g. if only one packet is to be sent, the latency time does not depend on the packet size. But, the moment a second packet is in the queue, the latency times are influenced dramatically by the packet size. Consider a packet with the size of two megabytes and the resulting data rate of 150 kilobytes per second. The transmission of one packet would take at least 13 seconds. This is the time the queueing packet would have to wait for its forwarding in minimum.

However, in order to get realistic values during simulation runs, the worst case latency times of measurements with real hardware were taken into account. The simulator was adjusted accordingly, so that the resulting latency times during a simulation run approximate to the worst case value of these measurements. The comparison is depicted in figure 9.13.

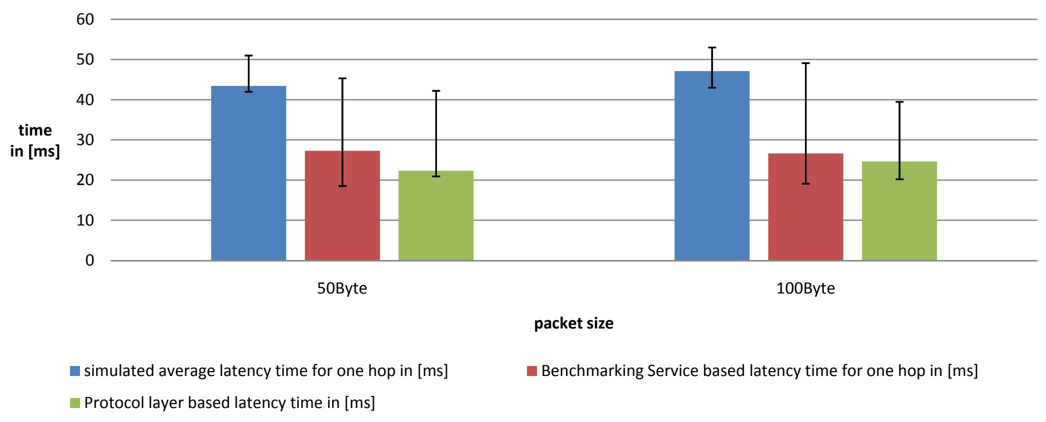


Figure 9.13.: Throughput measurements with the simulator for one hop with different packet sizes

The measurement process was made by the use of ping packets with a size of 50 bytes and 100 bytes. A small packet size was deliberately selected for keeping the influence of the packet size as small as possible. The measured round trip time was divided by two in order to get the one way latency time, and was averaged over various iteration runs afterwards.

The measured values of the simulator are shown by the blue bars. These are put into comparison with the latency times measured with real hardware (red bars) and measurements made by use of our benchmarking application that connects to the protocol layer (green bars). Obviously, the results of the simulator approximate the worst case values of the other two configurations, as assumed beforehand. By this, a good estimation of a mobile service's latency times can be made during the evaluation process of the service MIB parameters.

9.2.5. Comparison M/S vs. S/S Bridged Networks

Considering a Bluetooth-based Scatternet with nodes in a linear order, this Scatternet can be constructed in two different ways. On the one hand, it can be established by use of M/S bridging nodes, on the other hand, the occurring Piconets are bridged by S/S nodes. Both configurations have advantages and disadvantages; some were already described in section 5.1.1. Now these as well as further considerations shall be taken up again in order to give a complete understanding of the differences of these two types of network bridges.

In order to start the investigation, we made some measurements with the aid of the Benchmarking Service. Therefore, two M/S bridged and two S/S bridged topologies were used, each with two different hardware constellations to relativize the influence of the hardware. The gained throughput results can be seen in table 9.4. For a better understanding, the underlying topologies are depicted in figure 9.14.

	S/S 1	S/S 2	M/S 1	M/S 2
two hops	13,7	17,1	55,8	59,1
four hops	13,1	16,2	7,8	6,9

Table 9.4.: Throughput comparison of a S/S and a M/S bridged Scatternet in [kB/s]

The first topology depicted in the upper row of the table consists of three nodes. In the S/S bridged constellation, the nodes on the ends are masters, whereas the node in the middle is the S/S bridge. As a result, the network consists of two

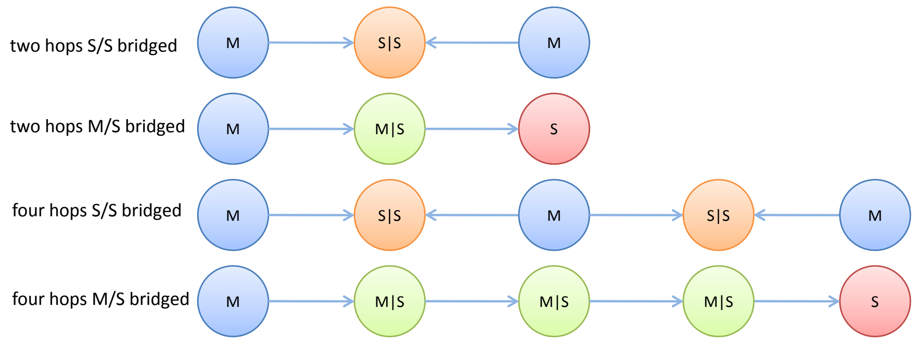


Figure 9.14.: Network constellations for the comparison of S/S bridged an M/S briged Scatternets with two and four hops

Piconets. The M/S configuration is constructed of a master, a M/S bridge in the middle, and a slave node, and thus also includes two Piconets.

The second row of the table shows the results of the four hop measurements. The S/S bridged networks consists of five nodes: three master nodes and two S/S bridges in alternating order. Therefore, this configuration contains three Piconets. For the M/S bridged network, one master, three M/S bridges, and a slave are used. In comparison to the S/S configuration, this constellation consists of four Piconets, which is one more than S/S bridged networks with the same amount of nodes have. For a better presentation, the values are illustrated in a diagram in figure 9.15.

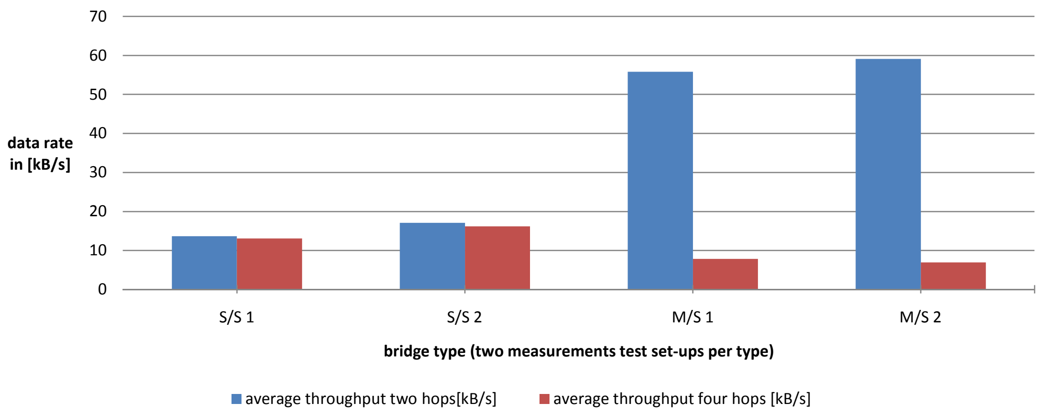


Figure 9.15.: Throughput comparison of S/S and M/S bridged Scatternets

The blue bars mark the measurements with two hops, whereas the red bars present the results of the four hop scenarios. Apparently, the measurement of M/S bridged

Scatternets with only two hops provide a much higher throughput than the corresponding S/S bridged constellations. But with the increasing amount of hops, the throughput rate drops rapidly. Contrary to this, the S/S bridged networks with few hops have a lower throughput, but the rate remains almost constant with an increasing number of hops. As can be seen in the diagram, the Scatternets with S/S bridges and four hops provide a higher data rate than those with M/S bridges.

In order to confirm this result, a further measurement approach was made. Four different network constellations were compared to each other. The formations are illustrated in figure 9.16.

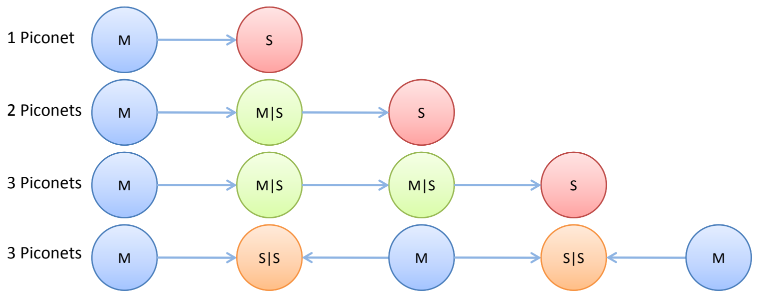


Figure 9.16.: The four different network topologies of this measurement

The first constellation is a simple Piconet. The measurement results for this Piconet are added in order to gain comparable values of a simple hop within a Piconet and hops over a bridge. The second and the third constellations are M/S bridged networks, where the first contains one bridging node and the second contains two. Hence, the first constellation consists of two Piconets and a two-hop path in length, whereas the second includes three Piconets and three hops. The fourth scenario consists of a S/S bridged network that is also constructed of three Piconets, but due to the S/S bridges, the communication path has a length of four hops.

The aim of this measurement is to compare the throughput and the latency times with the amount of Piconets and the used type of bridging node. In the course of this measurement, various testing runs were made with different packet sizes. Afterwards, the results were averaged to one value for each network constellation. By doing so, the dependency on the packet size is eliminated. All results are gained by use of the Benchmarking Service. The throughput-measurement results are presented in figure 9.17.

The diagram shows the throughput rates of the four different network constellations. The used packet sizes were 10kB, 50kB, 100kB, and 500kB, and they were averaged afterwards. The first entry on the left side, depicted as the blue bar,

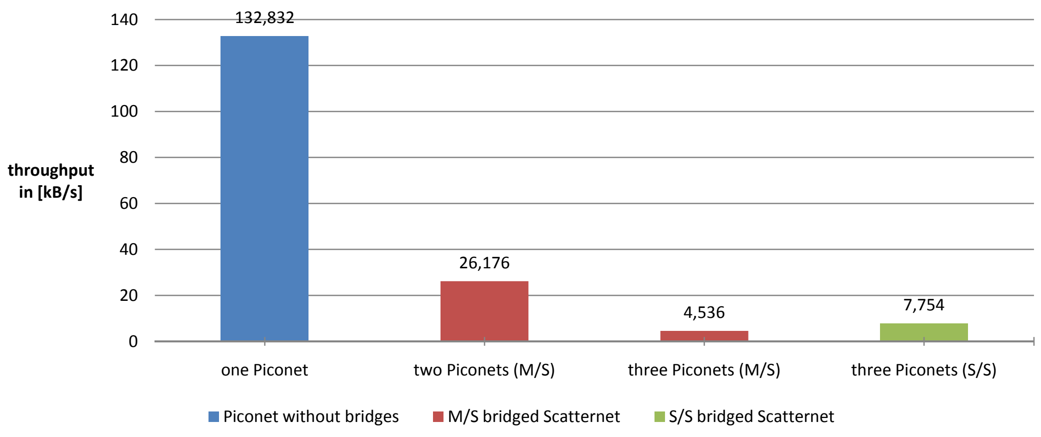


Figure 9.17.: Average throughput comparison of S/S and M/S bridged networks

presents the throughput rate of the single hop within a Piconet. The next two bars, in red, illustrate the two M/S bridged networks, whereas the green bar shows the measured throughput of the S/S bridged network with four hops.

When comparing the third and the fourth values, it can be seen that the S/S bridged network provides a higher throughput than the M/S bridged one, while they both consist of the same amount of Piconets. Furthermore, the measurement of the S/S bridged network was made over one more hop. In addition to this, the results of the corresponding latency time measurement are presented in figure 9.18.

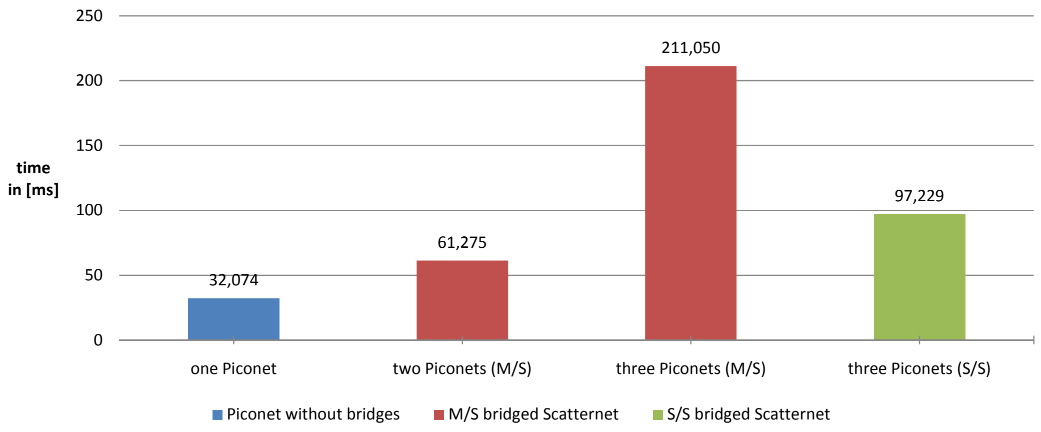


Figure 9.18.: Average latency time in comparison of S/S and M/S bridged networks

Analogously to the throughput results, the S/S bridged network behaves better than the M/S one. As a result, the latency time is lower even though the measured communication path is one more hop in length.

In order to explain these results, various properties of the Bluetooth standard must be considered. To begin with, it is obvious that the throughput of a Scatternet with a small amount of Piconets provides a much higher throughput and much better latency times than a S/S bridged Scatternet of the same size. But with an increasing amount of nodes, the throughput rate of a M/S bridged network decreases rapidly in relation to the amount of nodes, whereas the throughput of a S/S bridged network remains near-constant, as seen in figure 9.15.

A bridging node can only be in one Piconet at a time, as described in section 5.1.1. In order to transfer data to the other Piconet, it must switch its mode to *hold* in the one Piconet and resynchronize to the other. In case this bridge is a M/S node, the resynchronization process happens very fast, as it is the master that provides the clock signal for this Piconet. In case of a S/S bridge, the node must wait until it has received the required synchronization signal from the master. But this is time consuming, and as a result, the latency time increases and the throughput decreases. The large difference concerning the throughput and the latency times of small Scatternets can be explained by this.

But, in order to explain the behavior of larger Scatternets, the amount of occurring Piconets must be taken into account. As seen in the previous measurement process, the amount of Piconets has a great influence on the quality of the network. By establishing a Scatternet on the basis of S/S bridges, a Piconet always contains two hops - from the S/S bridge to the master and furthermore to the next S/S bridge - whereas a M/S bridged Piconet contains only one hop - from one M/S bridge to the next (see figure 9.16). The latter results in networks with a larger amount of Piconets. Considering the previous measurement results of one hop within a Piconet, the throughput was at least six times as high as the rate of a M/S bridged network with two hops (first and second values of figure 9.17). By additionally taking into account that all S/S bridged Piconets contain two internal hops, the disadvantage of the worse performing S/S bridge is balanced out.

Another advantage of the S/S bridged network is the capability of this kind of network to maintain connections to other slave nodes, whereas in a M/S bridged network, a Piconet is closed to communication if the master node is switched to the other Piconet. If more than one node communicates via the master node, its communication paths can be kept upright, whereas in M/S bridged networks, the same paths would be interrupted.

Based on these facts, it is obvious that networks that run services which require short communication paths should be constructed on basis of M/S bridges. But if services are run on top of a network that requires communication paths that are longer than two hops, the underlying network should consist of S/S bridged Piconets.

9.2.6. Bluetooth Connection Establishment

A last testing scenario to be presented here concerns the establishment of a simple Bluetooth connection. In the course of the development of the BlueSpot system, we noticed that the establishment sometimes takes up to several seconds. Such a behavior dramatically influences the execution of mobile services the moment a handover of the underlying client node is in progress.

By use of a small benchmarking tool, eleven different hardware constellations were tested. The tool performed 100 connection establishments with a consecutive disconnection after each establishment. During this process, the duration of each connection establishment was measured. The results can be see in table 9.5.

	1	2	3	4	5	6
average	179,99	224,902	148,268	195,204	101,465	425,478
maximum	3070,764	4962,413	2883,666	3459,148	944,188	1289,163
minimum	100,089	87,886	83,048	86,105	70,861	387,931

	7	8	9	10	11
average	447,968	742,64	699,606	158,966	175,952
maximum	1098,537	1206,06	1161,202	3297,217	3991,811
minimum	400,58	639,32	654,384	97,969	97,88

Table 9.5.: Measurements for the establishment of a Bluetooth connection with different hardware constellations. All values are in [ms]

In the table, the average values are shown. Additionally, the maximum values as well as minimums are presented in the second and third row. The corresponding diagram to the table can be found in figure 9.19.

The calculated overall average duration of a connection establishment gives 318,22 milliseconds. The absolute maximum value measured in the complete process is 4962,413 milliseconds. Out of 100 testing cycles arises that between three and five connection establishments have an duration of above two seconds. A reason for this behavior could not be found.

Another interesting revelation can be seen in the diagram. In case of the hardware

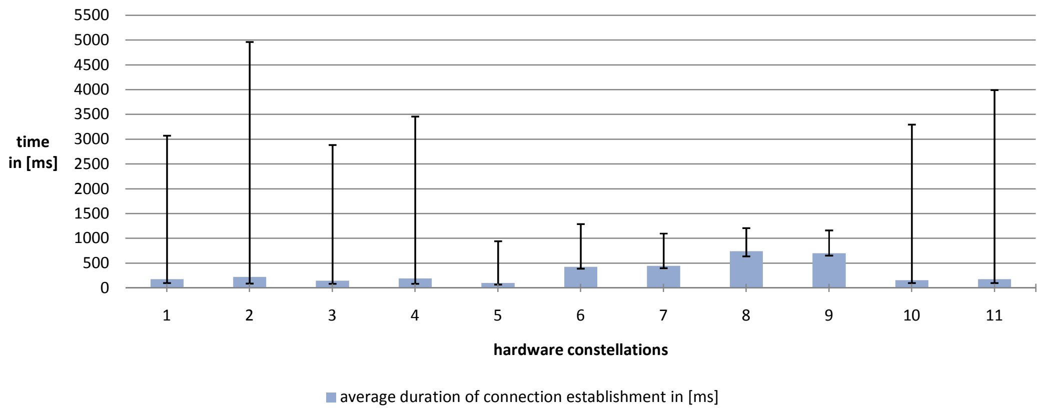


Figure 9.19.: Connection establishment times for different hardware constellations

constellations six, seven, eight, and nine, at least one Gumstix was involved. This led to a very poor average value of the connection establishment. But exactly these constellations have the lowest maximum values, which is very eye-catching. This result also cannot be explained.

9.3. Summary

The BlueSpot monitoring tool, an additional software tool available for the BlueSpot system, was introduced and explained in this chapter. A further important extension is the connection of the middleware to the NS2 simulator, which is implemented by use of the `BSSimDev` library. The application of this extension was presented by showing various preconfigured testing scenarios.

The second part of this chapter covers benchmarking results made by use of the BlueSpot system. These results were presented in great detail, and the possible conclusions discussed. Therefore, the throughput, the average latency times as well as possible data packet sizes were measured. Subsequently, the properties of M/S bridged networks were compared to those of S/S bridged ones. Finally, measurements of the duration of Bluetooth-based connection establishments were presented.

As a result, we were able to show that the concept of the BlueSpot system works. By its usage we easily showed the behavior of Bluetooth based networks concerning their performance. Furthermore, we were able to investigate the differences of M/S bridged and S/S bridged Scatternets in detail. An explanation of the measured

results that fits to our assumptions in the previous chapters could be given.

By use of the provided simulation scenarios, further aspects of wireless networks can be analyzed. Therefore, these scenarios can be modified in order to gain the wanted network constellations and configurations. These new scenarios can again be used to investigate adaptive behavior extensions, such as resource reservation mechanisms, wireless routing protocols, or network modification approaches.

Chapter 10

Conclusion

Contents

10.1. Aims of this Thesis

One of the aims of this thesis is to work out a viable system for providing a large spectrum of mobile services running on wireless networks. Our objective is that the underlying network technology, as well as the network constellation, should be insignificant for running a service. Into the system's development we had to add the formation and organization of the underlying network. Especially, the complexity of the formation process grows rapidly with an increasing amount of nodes.

The main group of target devices the BlueSpot system is dedicated to are mobile devices such as smartphones and PDAs. The usage of such heterogeneous hardware platforms is very challenging. They include different processor architectures, a highly diversified set of hard- and software, and a diverging set of operating systems. The built-in programming models for these devices are very limited, which is very challenging in the course of development.

In this thesis, the viable system's solution covers all their varying characteristics by providing a service platform that contains a standardized software interface on which mobile services can run. In order to gain service mobility, the transfer of mobile service packages between users of the network infrastructure created by the BlueSpot system was also approached. As a result, the BlueSpot system allows to automatically deliver mobile services to any point within the wireless network.

In the course of development, we enabled the resulting system to be enhanced by various adaptive behavior extensions. The software interfaces needed to integrate such extensions were designed to make as few restrictions as possible, thus providing the integration of a large potential spectrum of known and new network adaptivity approaches. By including such approaches, the underlying system can be pointedly adjusted to the evolving requirements made by mobile services. These requirements are manifold. Due to the wide-reaching spectrum of possible kinds of mobile services, this manifold is growing rapidly.

Due to the gained complexity of possible utilizations of adaptive behavior extensions, a concept was needed in order to organize the directed application of these extensions. Additionally, a point of entry for the adaptive behavior of the system was required. The moment a new mobile service is started, the network configuration must be adjusted to fit the service's requirements, specified by the meta information base (MIB) that is part of the service. At this point, the BlueSpot system must select the best fitting mechanisms provided by the adaptive behavior extensions.

10.2. Results

The before-mentioned aims were discussed in this thesis in great detail and demonstrated with the prototype of the BlueSpot system. A whole new middleware-based software system was developed in the course of our investigations. The BlueSpot system itself is seen as proof of concept and is used to verify the concepts and the solutions presented as pilot-experiments in this thesis.

Particular attention was devoted to the reuse of existing network technologies, network management and organization as well as currently state of the art software modeling approaches.

By combining these existing network technologies to a highly integrated and complex system, a broad software system was attained, that signifies by a high degree of generality, in contrast to many other approaches. This generality can be used to further develop, integrate, and test a wide spectrum of new procedures for wireless networks, and a range of new mobile services.

For a better classification of such procedures, we introduced the network view that contains different layers of abstraction in a software system. This network view as well as the middleware view that both include the points of contact - extension modules need to communicate with such a system - are used as basis for understanding and describing the adjustments that can be made to the entire system. Especially, current cross-layer approaches can be easily explained by use of these views and can subsequently become integrated into the system.

Enabled by the highly complex standard of the Bluetooth technology and the resulting Scatternet formations, we investigated possible configurations of networked nodes and analyzed their behavior in context of the BlueSpot system. For the formation of a network, an elaborate forming algorithm is needed. We modeled the most encouraging approaches defined within the field of the graph theory. As a result, we revealed the fundamentals of a variety of potential new network formation algorithms.

After a network is formed, a routing algorithm is needed to provide end-to-end communication. As part of the adaptive behavior extension, we can integrate almost any kind of routing algorithm for the BlueSpot system. The system provides the exchange of the routing protocol during runtime. In that way, the entire behavior of the software system can be modified and adapted to newly occurring or even changing demands of mobile services. For a better understanding of routing algorithms, we provided a classification of routing algorithms concerning their properties (from the point of view of the provider).

The utilization of a routing protocol exchange is part of the adaptive behavior of the BlueSpots system. This adaptive behavior can be seen as a new approach for covering adaptivity of wireless networks in general. It contains the initialization of wireless networks by explaining the main issues for finding the best fitting network formation as well as for selecting the corresponding forming algorithm. Additionally, it includes a concept for managing the underlying network during runtime. The moment when the demands made on the system changes, this concept describes various steps that are feasible ways to react to such events. By use of the previously described model of the system, available mechanisms can be assessed and ordered into these steps. Thus the utilization of these step-wise options can be optimized. This proof-of-concept-result may demarcate the academically most advanced result of this thesis - in terms of generality.

The network organization process was rounded up by aspects to be determined the very moment mobile clients appear. Here, the main tasks are the handover functionality and the integration of newly appearing nodes. With the help of the BlueSpot system's model we were able to integrate soft- as well as hard-handover functionalities into our proof-of-concept.

Also, in order to run mobile services on such a software system, the most important characteristics of such services were listed. In addition to this, these characteristics were used to provide a set of interesting parameters that are needed to describe a service's requirements. On the other hand, the BlueSpot system contains a set of system parameters that form the counterpart to those of the service. By use of these parameters, we provided a new approach that allows the determination whether a service can be run or not. This determination is in turn the entry point for the application of available adaptive mechanisms by use of the adaptive behavior concept previously introduced. After a successful readjustment of the underlying network, resources were freed and the requirements of the service can be met with.

The covering of mobile services was rounded up by describing various exemplarily developed mobile services for the BlueSpot system. During the selection process, when we asked ourselves which mobile services should be implemented, we focused on selecting services with widely varying requirements to the underlying network system. By their application, the scope of functionality of the BlueSpot system was demonstrated: it supported a whole variety of existing applications, and of emerging options.

One of these services is the Benchmarking Service. It was used to execute various measuring processes on top of the BlueSpot system. Selected parameters and phenomena dependent on an underling Bluetooth-based network were investigated. Measurements were made of the throughput rate, the average latency time, as well as of the used size of packets on the level of the protocol layer. While we elaborated the formation of different Scatternet structures, we noticed different behaviors of M/S bridged Piconets compared to S/S bridged ones. Additional investigations were made by use of the simulator. As a result, we were able to show that both types perform differently, owing to an increasing amount of nodes.

10.3. Outlook

The laboratory-version of the BlueSpot system provides a basis for advances. The full swing of experiments at the MVS-lab opened vistas at various further approaches concerning the rapid evolution in the field of wireless networks and mobile services. As a result of the many software interfaces, the BlueSpot system can be enhanced by a large amount of further adaptivity approaches. Especially, the investigation of security issues and energy consumption could be interesting. It was seen, but not applied as yet. The current state of development of the BlueSpot system is a proof of concept. Therefore, much "bug fixing" must be accomplished in order to enhance its stability and to provide a reliable basis for future approaches.

One way ahead is feasible by integrating different adaptive mechanisms. These can be compared to each other. E.g. a new routing approach can by applied to the system, and thus it can be compared to existing ones. Due to the standardized architecture for these proceedings, far-reaching mensuration processes can be executed and useful results gained in the future.

The actual version of the Bluetooth technology for the BlueSpot system entails various restrictions that need to be dealt with. Considerable expenditures must be expected to cover research and development costs. Especially, the distribution of roles, such as the master and the slave roles, as well as the two types of bridges, results in a highly demanding forming algorithm, if the resulting topology is to have a predefined configuration. Therefore, the forming of such networks will be a challenging topic in future elaborations.

Another way ahead regards the integration of further wireless communication technologies. The *ZigBee* standard [ZigBee Alliance, 2004] as well as the *wibree* [Bluetooth SIG, 2008] communication technology, which is part of the Bluetooth standard, provide an interesting possibility for the integration of further network constellation models. By use of the ZigBee standard, nodes are distinguished in three different roles: end device, router, and coordinator. The underlying networks are constructed in a star topology that contains one router and up to 240 end devices, which would represent a single domain within the BlueSpot system. Additionally, many routers can be combined to an infrastructured network, where one router is selected as coordinator. In respect of these characteristics, ZigBee can be easily integrated into the BlueSpot system. Thus, the functionality of the present system can be applied to future network technologies, as well.

Furthermore, in the course of this thesis, the Bluetooth standard v3.0 was announced. In contrast to the existing ones, this standard contains a completely reviewed baseband. As a result of this, the Bluetooth standard will be integrated into the *ultra wide band* (UWB) specification [Porcino and Hirt, 2003] in the near future. The used frequency spectrum will be between 3,1 GHz and 10,6 GHz. The main specification of UWB describes the underlying network as a star topology. Details concerning the standard that include the question, whether this can also be applied applied to the Bluetooth standard or if the master-slave distribution of roles will be kept upright, are currently unknown.

Bibliography

[Al-Karaki and Kamal, 2004] Al-Karaki, J. N. and Kamal, A. E. (2004). Routing techniques in wireless sensor networks: a survey. *Wireless Communications, IEEE*, 11(6):6–28. [cited at p. 28]

[Asada et al., 1998] Asada, G., Dong, M., Lin, T., Newberg, F., Pottie, G., Kaiser, W., and Marcy, H. (1998). Wireless integrated network sensors: Low power systems on a chip. *Solid-State Circuits Conference, 1998. ESSCIRC '98. Proceedings of the 24th European*, pages 9–16. [cited at p. 9]

[Barlow, 1968] Barlow, R. E. (1968). *Mathematics of the Decision Siences - Part 2*, volume 2 of *Lectures in Applied Mathematics*, chapter Reliability Theory, pages 255–279. American Mathematical Society, Providence, Rhode Island, USA. [cited at p. 10]

[Bernet et al., 2000] Bernet, Y., Ford, P., Yavatkar, R., Baker, F., Zhang, L., Speer, M., Braden, R., Davie, B., Wroclawski, J., and Felstaine, E. (2000). Rfc2998: A framework for integrated services operation over diffserv networks. [cited at p. 81]

[Blair et al., 1997] Blair, G., Coulson, G., Davies, N., Robin, P., and Fitzpatrick, T. (1997). Adaptive middleware for mobile multimedia applications. In *Proceedings of the IEEE 7th International Workshop on Network and Operating System Support for Digital Audio and Video, 1997.*, pages 245–254. IEEE, IEEE. [cited at p. 10]

[Bloch and Wagner, 2003] Bloch, C. and Wagner, A. (2003). *MIDP 2.0 Style Guide for the Java 2 Platform, Micro Edition.* Addison-Wesley. [cited at p. 48]

[Bluetooth SIG, 2004] Bluetooth SIG (2004). *Specification of the Bluetooth System - Bluetooth Specification Version 2.0 + EDR*, volume 0-3. Bluetooth SIG. [cited at p. 39]

[Bluetooth SIG, 2007] Bluetooth SIG (2007). The internet website of the Bluetooth SIG. http://www.bluetooth.org. [cited at p. 1]

[Bluetooth SIG, 2008] Bluetooth SIG (2008). Project webpage of the wibree standard. http://www.wibree.com/. [cited at p. 135]

[Broadcom, 2007] Broadcom (2007). Broadcom website. http://www.broadcom.com/. [cited at p. 42]

[Busybox, 2007] Busybox (2007). Busybox website. http://www.busybox.net. [cited at p. 47]

[Chen et al., 2002] Chen, B., Jamieson, K., Balakrishnan, H., and Morris, R. (2002). Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networks*, 8(5):481–494. [cited at p. 34]

[Cheriton and Mann, 1989] Cheriton, D. R. and Mann, T. P. (1989). Decentralizing a global naming service for improved performance and fault tolerance. *ACM Transactions on Computer Systems (TOCS)*, 7(2):147–183. [cited at p. 88]

[Clark et al., 1990] Clark, B. N., Colbourn, C. J., and Johnson, D. S. (1990). Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177. [cited at p. 75]

[Clark and Fang, 1998] Clark, D. D. and Fang, W. (1998). Explicit allocation of best-effort packet delivery service. *IEEE/ACM Trans. Netw.*, 6(4):362–373. [cited at p. 80]

[cognitas, 2005] cognitas (2005). *Pocket Loox 700 Series - Easy Guide.* Fujitsu-Siemens, http://support.fujitsu-siemens.de/de/support/manuals.html. [cited at p. 49]

[Cormen et al., 1990] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990). *Introduction to Algorithms.* The MIT Press, Cambridge, MA, USA. [cited at p. 30]

[Das et al., 2000] Das, S. R., Perkins, C. E., and Royer, E. M. (2000). Performance comparison of two on-demand routing protocols for ad hoc networks. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 3–12, Tel Aviv, Israel. IEEE. [cited at p. 31]

[Dijkstra, 1956] Dijkstra, E. W. (1956). *Numerische Mathematik*, volume 1, chapter A note on two problems in connexion with graphs, page 269–271. Springer Verlag. [cited at p. 31]

[Dinkel et al., 2006] Dinkel, M., Dümichen, U., and Baumgarten, U. (2006). Services and service platforms - a definition and classification. Handed in to IEEE Journal "Software". [cited at p. 24, 89]

[Dümichen and Baumgarten, 2007] Dümichen, U. and Baumgarten, U. (2007). Aspects for forming structured bluetooth based scatternets. In *Proceedings on WORLDCOMP PDPTA*, Las Vegas, Nevada, USA. [cited at p. 69]

[Dümichen and Baumgarten, 2008] Dümichen, U. and Baumgarten, U. (2008). *Advances in Communication Systems and Electrical Engineering*, volume 4 of *Lecture Notes Electrical Engineering*, chapter Including Adaptive Behavior in a Wireless Network System. Springer Verlag. [cited at p. 11, 80]

[Gabriel and Sokal, 1969] Gabriel, K. and Sokal, R. R. (1969). *Systematic Zoology*, volume 18, chapter A New Statistical Approach to Geographic Variation Analysis, pages 259–278. [cited at p. 76]

[Gerla and Tsai, 1995] Gerla, M. and Tsai, J. T.-C. (1995). Multicluster, mobile, multimedia radio networks. *Wireless Networks*, 1(3):255–265. [cited at p. 10]

[Günes et al., 2007] Günes, M., Spaniol, O., Wenig, M., Zimmermann, A., Makram, S. A., and Meis, U. (2007). Möglichkeiten und grenzen von drahtlosen mesh-netzwerken. *Praxis der Informationsverarbeitung und Kommunikation*, 3:170–176. [cited at p. 19]

[Gumstix, 2007] Gumstix (2007). Gumstix product website. http://www.gumstix.com/. [cited at p. 46]

[Haas, 1997] Haas, Z. J. (1997). A new routing protocol for the reconfigurable wireless networks. In *IEEE 6th International Conference on Universal Personal Communications Record*, volume 2, pages 562–566, San Diego, CA, USA. IEEE. [cited at p. 32]

[Haas and Pearlman, 2001] Haas, Z. J. and Pearlman, M. R. (2001). *Ad hoc networking*, chapter ZRP: a hybrid framework for routing in Ad Hoc networks, pages 221–253. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. [cited at p. 32]

[Hacker, 2006] Hacker, H. (2006). Evaluierung und realisierung von routing-protokollen für bluespot-netze. Master's thesis, TU München, Fakultät für Informatik. [cited at p. 64, 79]

[Halpern and Bradner, 1996] Halpern, J. and Bradner, S. (1996). Rfc1923: Ripv1 applicability statement for historic status. [cited at p. 30]

[Holtmann and Krasnyansky, 2007] Holtmann, M. and Krasnyansky, M. (2007). BlueZ - Official Linux Bluetooth protocol stack. http://www.bluez.org/. [cited at p. 42, 50]

[Hu et al., 2001] Hu, Y.-C., Johnson, D. B., and Maltz, D. A. (2001). Flow state in the dynamic source routing protocol. In Internet Draft draft-ietf-manet-dsrflow-00.txt. work in progress. [cited at p. 32]

[IBM WEME, 2007] IBM WEME (2007). Websphere everyplace micro environment. http://www-306.ibm.com/software/wireless/weme/. [cited at p. 50]

[IEEE 802.11, 2007] IEEE 802.11 (2007). IEEE 802.11 - Working Group for Wireless Local Area Networks. http://www.ieee802.org/11/. [cited at p. 1]

[J2ME, 2007] J2ME (2007). Sun Developer Network (SDN) - Java ME at a Glance. http://java.sun.com/javame/index.jsp. [cited at p. 47, 50]

[James et al., 2004] James, J., Bing, C., and Garrison, L. (2004). Implementing voip: a voice transmission performance progress report. *Communications Magazine, IEEE*, 42(7):36–41. [cited at p. 88]

[Johnson et al., 2001] Johnson, D. B., Maltz, D. A., and Broch, J. (2001). *Ad hoc networking*, chapter DSR: the dynamic source routing protocol for multihop wireless ad hoc networks, pages 139–172. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. [cited at p. 32]

[Josey et al., 2004] Josey, A., Cragun, D. W., Stoughton, N., Brown, M., and Hughes, C. (2004). *POSIX - IEEE Standard 1003.1, 2004 Edition.* IEEE Open Group, 6 edition. [cited at p. 47]

[Karp and Kung, 2005] Karp, B. and Kung, H. T. (2005). *GPSR: Greedy Perimeter Stateless Routing for Wireless Networks.* Ft. Belvoir Defense Technical Information Center, Harvard University, Boston, MA, USA. [cited at p. 34, 76]

[Langhammer, 2007] Langhammer, M. (2007). Beschreibung und Verwaltung von Netzwerkressourcen durch das BlueSpot-System und Integration adaptiven Verhaltens in die Middleware. Master's thesis, Lst I13, Fakultät für Informatik, TU München. [cited at p. 68]

[Langhammer and Metzger, 2007] Langhammer, M. and Metzger, C. (2007). System Entwicklungsprojekt: Portierung der BlueSpot middleware auf Symbian OS und WinCE. Technical report, TU München, Fakultät für Informatik. [cited at p. 64]

[Lee and Gerla, 2001] Lee, S.-J. and Gerla, M. (2001). Split multipath routing with maximally disjoint paths in ad hoc networks. In *IEEE International Conference on Communications*, volume 10, pages 3201–3205, Helsinki, Finland. IEEE, IEEE Press. [cited at p. 79]

[Li, 2004] Li, X.-Y. (2004). *Mobile ad hoc networking*, chapter Topology Control in wireless ad hoc networks, pages 175–203. IEEE Press, Piscataway, New Jersey, USA. [cited at p. 74, 76]

[Li et al., 2004] Li, X.-Y., Stojmenovic, I., and Wang, Y. (2004). Partial delaunay triangulation and degree limited localized bluetooth scatternet formation. *IEEE Transactions on Parallel and Distributed Systems*, 15(4):350–361. [cited at p. 77]

[Manoj et al., 2001] Manoj, B. S., R., A., and Murthy, C. S. R. (2001). Link life based routing protocol for ad hoc wireless networks. In *Tenth International Conference on Computer Communications and Networks*, pages 573–576, Scottsdale, AZ, USA. IEEE. [cited at p. 33]

[McKinley et al., 2004] McKinley, P.K., Sadjadi, S.M., Kasten, E.P., Cheng, and B.H.C. (2004). Composing adaptive software. *Computer*, 37(7):56–64. [cited at p. 11]

[Metzger, 2007] Metzger, C. (2007). Simulation eines bluespot netzwerks anhand ausgewählter szenarien. Master's thesis, Lst I13, Fakultät für Informatik, Tu München. [cited at p. 107, 108]

[Millar et al., 1998] Millar, I., Beale, M., Donoghue, B. J., and Lindstrom, K. (1998). *The Hewlett-Packard Journal*, chapter The IrDA Standards for High-Speed Infrared Communications. Hewlett-Packard Company, Palo Alto, USA. [cited at p. 42]

[Moore and Shannon, 1956] Moore, E. F. and Shannon, C. E. (1956). Reliable circuits using less reliable relays. *Journal of the Franklin Institute*, 262:191–208. [cited at p. 10]

[Nichols et al., 1998] Nichols, K., Blake, S., Baker, F., and Black, D. (1998). Rfc2998: Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers. [cited at p. 81]

[ns2webpage, 2008] ns2webpage (2008). The Network Simulator - ns2 - Project Webpage. http://nsnam.isi.edu/nsnam/index.php/User_Information. [cited at p. 107]

[Pahlavan and Levesque, 1995] Pahlavan, K. and Levesque, A. H. (1995). *Wireless information networks*. Wiley-Interscience, New York, NY, USA. [cited at p. 10]

[Peng and Lu, 2000] Peng, W. and Lu, X. (2000). Efficient broadcast in mobile ad hoc networks using connected dominating sets. In *Proceedings of ICPADS*, Iwate, Japan. [cited at p. 77]

[Perkins, 2001] Perkins, C. (2001). *Ad Hoc Networking*. Addison-Wesley. [cited at p. 19]

[Perkins and Bhagwat, 1994] Perkins, C. E. and Bhagwat, P. (1994). Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *SIGCOMM '94: Proceedings of the conference on Communications architectures, protocols and applications*, pages 234–244, New York, NY, USA. ACM. [cited at p. 30]

[Porcino and Hirt, 2003] Porcino, D. and Hirt, W. (2003). Ultra-wideband radio technology: Potential and challenges ahead. *IEEE Communications Magazine*, 41(7):66–74. [cited at p. 135]

[Rieck, 2007] Rieck, S. (2007). Bachelor thesis: Implementierung eines Monitoringtools zur Beobachtung des BlueSpot-Netzwerks. Technical report, Lst I13, Fakultät für Informatik, TU München. [cited at p. 106, 107]

[Salonidis et al., 2001] Salonidis, T., Bhagwat, P., Tassiulas, L., and LaMaire, R. O. (2001). Distributed topology construction of bluetooth personal area networks. In *IEEE INFOCOM 2001*, pages 1577–1586. [cited at p. 73]

[Schiller, 2003a] Schiller, J. (2003a). *Mobilkommunikation*, chapter 2.7.2 Frequency Hopping Spread Spectrum, pages 85–87. Pearson Education Deutschland GmbH, München, Deutschland. [cited at p. 39]

[Schiller, 2003b] Schiller, J. (2003b). *Mobilkommunikation*, chapter 7.5 Bluetooth, pages 318–345. Pearson Education Deutschland GmbH, München, Deutschland. [cited at p. 40, 163]

[Schiller, 2003c] Schiller, J. (2003c). *Mobilkommunikation*, chapter 7.3 IEEE 802.11, pages 248–285. Pearson Education Deutschland GmbH, München, Deutschland. [cited at p. 44]

[Schmidmeir and Schmidmeir, 2007] Schmidmeir, R. and Schmidmeir, M. (2007). Bachelor thesis and SEP: Systemanbindung einens Micontrolers. Technical report, Lehrstuhl I13 und Lehrstuhl für medizinische Elektronik, TU München. [cited at p. 97, 98]

[Schmidt, 2002] Schmidt, D. C. (2002). Middleware for real-time and embedded systems. *Commun. ACM*, 45(6):43–48. [cited at p. 11]

[Schupfner, 2007] Schupfner, C. (2007). Realisierung des Handover in BlueSpot-Netzen und Nutzung über ein Service Interface. Master's thesis, TU München, Fakultät für Informatik. [cited at p. 64]

[Steinle, 2007] Steinle, M. (2007). Erweiterung der Diensteschnittstelle und Integration adaptiver Unterstützung mobiler Dienste in das BlueSpot-System. Master's thesis, Lst I13, Fakultät für Informatik, Tu München. [cited at p. 65]

[Steinle and Eiband, 2006] Steinle, M. and Eiband, D. (2006). System Entwicklungsprojekt: Portierung des BlueSpot-Buetooth Adapters auf Linux-basierte Plattformen. Technical report, TU München, Fakultät für Informatik. [cited at p. 64]

[Sym, 2007] Sym (2007). The official website of Symbian OS. http://www.symbian.com/. [cited at p. 42, 47]

[Szewczyk et al., 2004] Szewczyk, R., Polastre, J., Mainwaring, A., and Culler, D. (2004). Lessons from a sensor network expedition. In *Wireless Sensor Networks, First European Workshop, EWSN 2004*, pages 307–322, Berlin, Germany. Springer Verlag. [cited at p. 9]

[Toussaint, 1980] Toussaint, G. T. (1980). *Pattern Recognition*, volume 12, chapter The relative Neighborhood Graph of a Finite Planar Set, pages 261–268. [cited at p. 75]

[uClibc, 2007] uClibc (2007). uClibc Website. http://www.uclibc.org. [cited at p. 47]

[Urrutia, 2002] Urrutia, J. (2002). *Handbook of wireless networks and mobile computing*, chapter Routing with guaranteed delivery in geometric and wireless networks, pages 393–406. John Wiley & Sons, Inc., New York, NY, USA. [cited at p. 62]

[Waitzman et al., 1988] Waitzman, D., Partridge, C., and Deering, S. E. (1988). Distance vector multicast routing protocol. Internet RFC. [cited at p. 34]

[White, 1997] White, J. E. (1997). *Software Agents*, chapter Mobile agents, pages 437–472. MIT Press, Cambridge, MA, USA. [cited at p. 25]

[Xiph community, 2008] Xiph community (2008). Website of the xiph open source community. http://www.vorbis.com. [cited at p. 101]

[Yau and Karim, 2004] Yau, S. S. and Karim, F. (2004). An adaptive middleware for context-sensitive communications for real-time applications in ubiquitous computing environments. *Real-Time Systems*, 26(1):29–61. [cited at p. 11]

[Zaruba et al., 2001] Zaruba, G. V., Basagni, S., and Chlamtac, I. (2001). Bluetreesscatternet formation to enable bluetooth-based ad hocnetworks. In *IEEE International Conference on Communications, 2001. ICC 2001.*, volume 1, pages 273–277. Erik Jonsson Sch. of Eng. & Comput. Sci., Texas Univ., Dallas, TX, IEEE. [cited at p. 71]

[ZigBee Alliance, 2004] ZigBee Alliance (2004). *ZigBee Specification v1.0.* ZigBee Alliance Board of Directors. [cited at p. 135]

[Zimmermann, 1980] Zimmermann, H. (1980). OSI Reference Model-The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communication*, com-28(4):425–432. [cited at p. 16]

Part III.

Appendix

Appendix A

Related Student Work

It is my desire to express my gratitude to the faculty of the TUM, especially to the chair, for the opportunity to cope with such a wide scope of topics by being granted many assisting students. More than a dozen parts of the BlueSpot system presented above are the results of a student works program. The themes and tests were guided by the viable system's objective implemented here. A list of these works including title and a short description are summarized in the following in chronological order, as they were finalized in terms of academic, empirical, and experimental output. Some works are still pending.

- *Personalisierte Informations- und Positionierungsdienste für mobile Endgeräte auf der Basis von Bluetooth und CE .Net (Ulrich Dümichen)*: in this diploma thesis, the initial developments for the BlueSpot system were made. At that juncture, all approaches were performed on basis of the Windows CE operating system. The underlying network topology bases on a network which at that time had a wired backbone. Multi-hop communication was not included. Mobile services were integrated into the system by the COM model. The necessary binaries were provided in the form of dynamic link libraries. A support for other hardware platforms that did not run with Windows CE was not available.
- *VoIP auf Nokia 6600 (Manfred Schreiber)*: the aim of this bachelor thesis was the implementation of a *voice over IP* (VoIP) client for the Nokia smartphone 6600. The approach was technically unsuccessful in the end, due to restrictions Nokia included into the smartphone. E.g. an audio signal received via a

Bluetooth connection could not be played by use of the internal speaker. This restriction was deliberately integrated into this type of device by Nokia in order to inhibit approaches in this direction. This, and further restrictions, was not documented by Nokia previous to the commencing of this work. As a result of the necessary investigations, the final work of Schreiber provided a great depth covering the thematic of VoIP.

- *Erweiterung des BlueSpot Systems um weitere Infrastrukturen (David Vidal-Rodriguez)*: in order to enable the usage of completely wireless networks without the need of a backbone mechanism, the multi hop capability was added to the BlueSpot middleware. Therefore, a simple routing protocol was implemented that bases on a flooding protocol. Additionally, simple topologies for the BlueSpot system were introduced, such as a spanning tree formation of a linearly ordered one. The establishment of the network was made manually, thus an automated forming algorithm was not included.

- *Portierung des BlueSpot-Bluetooth-Adapters auf Linux-basierte Plattformen (Steinle, Eiband)*: to begin with, it was apparent that the entire system needed to be ported to a Linux-based platform. This was the only possible way to support a larger spectrum of hardware platforms and to investigate the integration of further extension modules such as routing protocols. With this thesis, the porting proceedings were started and the `BSBthDev` library was implemented. Our first Bluetooth Scatternets were assembled, resulting in various mensurations of selected scenarios and parameters. This was the start of the designing of a wholistic system.

- *Evaluierung und Realisierung von Routing Protokollen für BlueSpot Netzwerke (Johann Hacker)*: as a next step of the porting of the BlueSpot middleware from Windows CE to Linux, the lower layers of the middleware were completely revised. In order to achieve a better performance of the underlying network, two new routing protocols were implemented and integrated into the system. A mechanism was defined for exchanging the currently running routing protocol with another one without needing to completely restart of the system. The product of this diploma thesis, therefore, was basic work on the ground layers of the BlueSpot middleware with proof of subsystem.

- *Portierung der BlueSpot Middleware auf Symbian OS und Pocket PC 2003 (Langhammer, Metzger*: in order to enlarge the coverage of supported devices,in the course of this work, the middleware was ported to two further software platforms . By doing so, the support of smartphones such as the Nokia 6600 and the Nokia 6630 as well as the PDA FSC Pocket Loox was enabled. As a result, these devices could be used as client nodes within the

BlueSpot system, and thus a more powerful presentation of the entire software system was possible.

- *Realisierung des Handover in BlueSpot-Netzen und Nutzung von Mobilen Diensten über ein Service Interface (Christian Schupfner)*: by use of this diploma thesis, a session layer was integrated into the BlueSpot system. For a better support of mobile services, the service interface was worked out and implemented. As a result, mobile services could be implemented as Java services as well as native C ones. In order to overcome the gap of supporting more than one hardware platform by one service, the container service was additionally introduced. To keep a session upright even when a node moves to another point of contact, the handover mechanism was integrated additionally as part of this thesis. In the end of this thesis, a large spectrum of viability of the BlueSpot system was gained.

- *Systemanbindung eines Mikrocontrollers zur Sensorik und Gerätesteuerung (Schmidmeir, Schmidmeir)*: after the service interface and the general structure of mobile services for the BlueSpot system were defined, a demonstrator was needed in order to show the functionalities of the BlueSpot system. The result was the ability to steer the Mini Mauler RC car over the entire underlying network system. In addition to this, the pH/temp sensor board was developed and simple API for its usage implemented. Concluding their work, the Schmidmeir brothers demonstrated the capability of the BlueSpot system to support mobile services as required by the fundamental concept.

- *Implementierung eines Monitoringtools zur Beobachtung des BlueSpot Netzwerks (Simon Rieck)*: this bachelor thesis describes the monitoring tool for the BlueSpot system. In the course of the development of the system, it became apparent that a view of the current situation of a running BlueSpot network would provide a large amount of useful information. The resulting tool allows to search for selected events in all received monitoring messages. For a better overview, messages that belong together are grouped, and an easy way for surveying the behavior of the system is given.

- *Erweiterung der Dienstschnittstelle und Integration adaptiver Unterstützung mobiler Dineste in das BlueSpot-System (Michael Steinle)*: the diploma thesis of Langhammer and of Steinle accompanied. In order to integrate adaptive behavior into a large spectrum, Steinle in the course of his diploma thesis integrated the adaptivity extension module into the middleware. The resulting extensibility of the BlueSpot system through various network management mechanisms formed the basis for the interoperability of the system, as it works today.

- *Beschreibung und Verwaltung von Netzwerkressourcen durch das BlueSpot-System und Integration adaptiven Verhaltens in die Middleware (Matthias Langhammer)*: the diploma thesis of Langhammer and of Steinle accompanied. Where Steinle developed the integration of the service MIB and the node MIB, Langhammer achieved the integration of adaptive behavior on the lower layers of the middleware. He enabled the modification mechanisms during runtime and reviewed the mechanism needed for the integration of further routing protocols. Therefore, a broadcast mechanism was integrated into the middleware that allows to send control messages to all nodes of an established network.

- *Simulation eines BlueSpot Netzwerks anhand ausgewählter Szenarien (Christoph Metzger)*: as a result of the limited hardware available during our research proceedings, we decided to provide a connection to a simulator. This connection was investigated and implemented in the course of the diploma thesis of Metzger. For a solution as realistic as possible, various available Bluetooth extensions for the NS2 simulator were investigated. Surprisingly, none of them showed to be feasible for the BlueSpot system. The implementation of the `BSSimDev` library was the result. It enables the connection to a generally formed wireless network simulated by the NS2. The parameters needed in order to make a realistic simulation possible were directly integrated into the simulation environment. For the first time in the course of the research and development of the BlueSpot system, the existence of a new relationship between state of the art in technology and generalizability of the viable system was demonstrated.

Appendix B

Overview of Selected Software Interfaces

In the following, the `DeviceDetails` structure as well as the main software interfaces shall be described. The here presented code must be seen as an extract of the entire software design. The intension is to provide the reader with a better understanding of selected functionalities described in this thesis. For a complete description of the software design, we refer to the available BlueSpot code documentation.

B.1. DeviceDetails

The `DeviceDetails` structure is used within the software stack any where and any time a communication partner must be addressed. The structure contains information such as the used communication technology needed to contact the device, its address (a union that includes an IP address, a Bluetooth address, or the address of an IrDA based network device), and a representative name. In order to enable a connection to this device, further information is needed: a flag that indicates whether the BlueSpot system is installed on that device, and a listing of the parameters needed to establish a connection. The latter consists of the RFComm channel in case a Bluetooth based channel shall be used, or of the TCP port number, if the underlying network is IP-based.

The structure itself is constructed as a linked list. Therefore, each entry of the

DeviceDetails list contains a pointer to the next entry. The last element contains a null pointer to signalize the end of the list. The following code fragment shows the basis of the DeviceDetails structure:

```
struct BSLAYERINDEP_DLL_API DeviceDetails
{
/** Type of device.
 *  The type of the device (LAN_DEVICE, BTH_DEVICE oder IR_DEVICE).
 *  Currently support is available for Bluetooth and WLAN.
 */
DeviceType devType;

/// Bluetooth-, LAN- or IRDA-Address of the device.
DeviceHWAddress addr;

/** Name of device.
 *  The (Bluetooth-)name of the device. Ethernet uses the host name.
 *  The name is a zero terminated C-String.
 */
BD_NAME devName;

/** is the BlueSpot middleware available at the device.
 *  BlueSpot support available.
 *  Boolean value that indicates BlueSpot support.
 *  This value is gained by executing a discovery process
 *  on the other device.
 */
bool hasService;

/// Middleware parameter (RFCOMM channel, TCP Port number, etc.).
DeviceService service;

/** Next entry in list.
 *  Pointer to the next entry in the DeviceDetails list.
 *  If the entry is the last in the list, this parameter is NULL.
 */
DeviceDetails *pNext;
};
```

B.2. Service Interface

The service interface provides the needed structure to connect mobile services to the BlueSpot middleware. It is divided into two parts, the `ServiceInterface` class and the `CServiceBase` class.

The following code fragment contains the main methods and functions needed to enable the communication from a mobile service to the middleware. A mobile service must include this class in order to enable the middleware to load the service's shared library object. The code presented here is part of a native C service. In order to implement a Java-based service, a Java class with similar functions and methods is available. The internal socket communication to the middleware is already included in this class, and thus does not need to be provided by the mobile service.

```
/** Interface to connect mobile services.
 *  These methods can be called by a service in order to
 *  communicate with the middleware stack.
 */
class BSSESSION_DLL_API ServiceInterface
{
public:
/// destructor.
virtual ~ServiceInterface() {};

/// connect the service.
virtual int Connect(uint16_t service,
      const std::string& RemoteSession) = 0;

/** disconnect service.
 *  @param connectionhandle this parameter contains the
 *                          handle for the connection.

 *  @return                 true indicates success, else false.
 */
virtual bool Disconnect_nothrow(int connectionhandle) = 0;

/** enabling the service to send data.
 *  by usage of this method, services are able to send data via
 *  the middleware.
 *  @return true indicates success, else false
 */
virtual bool SendData_nothrow(int connectionhandle,
      const void* data, int size) = 0;

/** receive list of possible communication partners.
 *  The list contains the virtual addresses of all devices that
 *  are currently running the service.
 *  @param sesslist handle of the list.
 *  @param service  ID of the service the list contains
 *                  communication partners of.
 */
virtual void GetVirtualAddressList(std::vector<std::string>
      &sesslist, uint16_t service) = 0;

// additional adaptivity functions
virtual void *AllocateRessources (uint16_t ServiceID,
      int connectionhandle, CAdaptivityModule::BS_ADM_Request Type,
      int Value) = 0;
virtual bool FreeRessources (uint16_t ServiceID,
      int connectionhandle, CAdaptivityModule::BS_ADM_Request Type,
      int Value) = 0;
virtual bool RequestFeature (CAdaptivityModule::BS_ADM_Feature
      Feature, int Value) = 0;
virtual int GetCurrentFeatures () = 0;
};
```

For enabling the communication in the other direction (from the middleware to the mobile service) the `CserviceBase` class is needed. By including this class, the service must implement the callback functions of the class.

```
/** Base for a mobile service.
 *  This class contains the callback functions needed to be
 *  implemented by the service.
 */
class CServiceBase {
public:
/// destructor.
virtual ~CServiceBase() {};

/// returns the ID of the service.
virtual uint16_t GetServiceId() = 0;

/// returns the revision of the service.
virtual uint16_t GetServiceRevision() = 0;

/// returns a representative name string.
virtual std::string GetServiceString() = 0;

/// starts the service.
virtual void start() = 0;

/// ends the service.
virtual void stop() = 0;

/** connection callback is triggered the moment a connection
*   was established.
*   @return 0 indicates a successful establishment.
*   all other values represent an error situation.
*/
virtual bool OnConnect_nothrow(int newconnectionhandle,
      uint16_t service, const std::string& RemoteSession) = 0;

/// callback that indicates the disconnection from the partner
virtual void OnDisconnected(int connectionhandle) = 0;

/** Incoming data.
 *  This method is triggered the moment data is received for
 *  the service.
 */
virtual void OnDataReceived(int connectionhandle,
      const void* data, int size) = 0;
};
```

B.3. Adaptivity Module

The integration of adaptive behavior extensions is implemented by the use of the `CAdaptivityModule` class. This class must be included into a new developed extension in order to enable the communication in direction of the middleware as well as in the opposite direction. Due to the complexity of this class, the most important functions and methods are extracted and presented here.

```
/** Central class of the Adaptivity module.
 *  This class provides support for the integration of adaptive
 *  extension modules.
 */
class CAdaptivityModule
{
public:
/** Constructor of the class CAdaptivityModule
 *  During instantiation of this class pointers to all layers of
 *  the middleware are needed. That way, the interaction with
 *  the various layers is enabled.
 *  \param Si  pointer to the CBSServiceLayer object
 *  \param Sl  pointer to the Session object
 *  \param Pl  pointer to the CBSProtocolLayer object
 *  \param Nal pointer to the CBSCommunicationLayer object
 */
IMPORT_C CAdaptivityModule(CBSServiceLayer *Si, Session *Sl,
CBSProtocolLayer *Pl, CBSCommunicationLayer *Nal);

  /// destructor
  IMPORT_C ~CAdaptivityModule();


/** provide resource reservation to a service.
 *  By use of this method, requests for specific resources are
 *  provided to an extension module. A distinction is made by
 *  use of the parameter 'Type' that defines the class of the
 *  needed resource, whereas the 'Value' parameter provides
 *  the amount of the resource needed for the service.
 *
 *  \param ServiceID the ID of the demanding service
 *  \param AdmID     the ID of the connection. If the call is
 *                   not connection specific, this value is 0
 *  \param Type      type of the required resource
 *  \param Value     amount of the resource specified by the
 *                   type value
 *
 *  \return the return value depends on the value of the type
 *          parameter:
 *  \return - ADM_REQUEST_BANDWIDTH/ADM_REQUEST_LATENCY:
 *                 0 indicates that an error occurred, every
 *                 value non-equal indicates success.
 *  \return - ADM_REQUEST_MEMORY: null indicates an error,
 *                 otherwise a pointer to the allocated
 *                 memory is returned.
 */
void *AdmSiAllocateRessources (uint16_t ServiceID, uint32_t AdmID,
      CAdaptivityModule::BS_ADM_Request Type, int Value);

/// frees the provided resource.
/// Parameters are equal to previous method.
void AdmSiFreeRessources (uint16_t ServiceID, uint32_t AdmID,
      CAdaptivityModule::BS_ADM_Request Type, int Value);

/** Requests a specified feature a service requires.
 *  This method tries to enable a requested feature specified in
 *  the parameter 'Feature'.
 *
 *  \param Feature BS_ADM_Feature based value that specifies one
 *                 or more requested features combined by
 *                 disjunction.
 *  \param Value   in case 'Feature' indicates a routing
 *                 algorithm, the ID of the algorithm. Otherwise
 *                 this value is ignored.
 *
 * \return true if the providing was successful, otherwise false.
 */
bool AdmSiRequestFeature (CAdaptivityModule::BS_ADM_Feature Feature,
      int Value);

/// returns a list of activated features
int AdmSiGetCurrentFeatures ();
```

```
/** A new service shall be run.
 *
 *  This method compares the minimal requirements of the service
 *  to the available ones. That way, a decision is made whether
 *  the service can be started or not.
 *
 * \param ServiceID        ID of the service to be started.
 * \param ServiceContainer in case of a container service, a
 *                         handle to this container.
 *
 * \return ADM_ERROR_SUCCESS if the service can be started.
 * \return every other value indicates an error occurred.
 */
unsigned int AdmSiStartService(uint16_t ServiceID,
      BlueSpot::System::CBSZipFile& ServiceContainer);

/// stops the service and frees its allocated resources.
void AdmSiStopService(uint16_t ServiceID);

/// callback that is triggered the moment a service connects to
/// another one.
unsigned int AdmSiConnect(const std::string& Session,
      uint16_t ServiceID);

/// callback that is triggered the moment another service connects
/// to a service running on this node.
bool AdmSiOnConnect (const std::string& Session, uint16_t ServiceID,
      uint32_t AdmID);

/// callback that is triggered when a connection was closed on the
  /// level of the service interface.
void AdmSiDisconnect(uint32_t AdmID);

/// callback that is triggered when data was received on level of
/// the service interface.
bool AdmSiRecvData (uint32_t AdmID, const std::string& Session,
      uint16_t ServiceID, const void *data, uint32_t size);

/// callback that is triggered when data is sent on level of the
/// service interface.
bool AdmSiSendData (uint32_t AdmID, const std::string& Session,
      uint16_t ServiceID, const void *data, uint32_t size);

/// callback that is triggered when data was received on level of
/// the session layer.
bool AdmSlRecvData (uint32_t AdmID, const std::string& Session,
      const void *data, uint32_t size);

/// callback that is triggered when data is sent on level of the
/// session layer.
bool AdmSlSendData (uint32_t AdmID, const std::string& Session,
      const void *data, uint32_t size);

/// callback that is triggered when data was received on level of
/// the protocol layer.
void AdmPlRecvData (uint32_t AdmID, const void *data, uint32_t size);

/// callback that is triggered when data is sent on level of the
/// protocol layer.
unsigned int AdmPlSendData (uint32_t AdmID,
      const DeviceDetails *Device, const void *data, uint32_t size);

/// callback that is triggered when data was received on level of
/// the network adaption layer.
IMPORT_C void AdmNalRecvData (const DeviceDetails *SrcDevice,
      const void *Data, uint32_t Size);

/// callback that is triggered when data is sent on level of the
/// network adaption layer.
IMPORT_C unsigned int AdmNalSendData(uint32_t AdmID,
      const DeviceDetails& Device, const void *Data, uint32_t Size);
};
```

List of Symbols and Abbreviations

Abbreviation	Description	Usage
ACL	Asynchronous Connectionless Link	41
AMA	Active Member Address	40
AP	Access Point	44
BGM	Backbone Mesh Gateway	19
BNEP	Bluetooth Network Encapsulation Protocol	42
BTCP	Bluetooth Topology Construction Protocol	73
DSDV	Destination-Sequenced Distance-Vector Routing	30
DSR	Dynamic Source Routing	31
DSRFLOW	Dynamic Source Routing with flow extension	32
DVMR	Distance Vector Multicast Routing Protocol	34
EDR	Enhanced Data Rate	41
FCFS	First Come First Serve	81
FHSS	Frequency Hopping Spread Spectrum	39
GG	Gabriel Graph	76
GPSR	Greedy Perimeter Stateless Routing	34
HCI	Host Controller Interface	50
IBSS	Independent Basic Service Set	44
IEEE	Institute of Electrical and Electronics Engineers	1
ISO-OSI	International Standardization Organization introduced the Open System Standard	16
ISM	Industrial, Scientific, and Medical Band	39
JNI	Java Native Interface	64
JVM	Java Virtual Machine	47
L2CAP	Logic Link Control and Adaptation Protocol	42
LBR	Link life Based Routing	33
LPT	Longest Processing Time	81
MANET	Mobile Ad-hoc Network	19
MIB	Meta Informatin Base	91
MIDP	Mobile Information Device Profile	48

Abbreviation	Description	Usage
MTU	Maximum Transmission Unit	79
NS2	Network Simulator 2	6
OBEX	Object Exchange protocol	42
PCB	Printed Circuit Board	97
PMA	Parked Member Address	40
POSIX	Portable Operating System Interface	47
RFCOMM	Radio Frequency Communication profile	42
RNG	Relative Neighborhood Graph	75
RSSI	Received Signal Strength Indicator	61
SCO	Synchronous Connection Oriented link	41
SDP	Service Discovery Protocol	43
SIG	Bluetooth Special Interest Group	38
SMR	Split Multi-path Routing	79
SRPT	Shortest Remaining Processing Time	81
UDG	Unit Disk Graph	75
UWB	Ultra Wide Band	135
WINS	Wireless Integrated Network Sensors	9
WIS	Wireless Information Systems	10
WMN	Wireless Mesh Network	19
WSN	Wireless Sensor Networks	9
YG	Yao Graph	76
ZRP	Zone Routing Protocol	32

List of Figures

List of Tables

Index