Master Thesis MEE 01-28

Device and Service Discovery in Bluetooth Networks

Jimi Karlsson & Albin Persson

This thesis is presented as a part of the Master of Science degree in Electrical Engineering with emphasis on Telecommunication and Signal Processing

Blekinge Institute of Technology

June 4, 2002

Master of Science Programme in Electrical Engineering
Department of Telecommunications and Signal Processing
Blekinge Institute of Technology

Supervisors: Mr. Bruce Tunnicliffe and Prof. Hans Jürgen Zepernick, Australian Telecommunications Cooperative Research Centre, Curtin University of Technology, Perth, Australia.
Examiner: Dr. Markus Fiedler, Dept. of Telecommunications and Signal Processing, Blekinge Institute of Technology, Karlskrona, Sweden.

Abstract

In view of Bluetooth's growing popularity a lot of research is being done to improve the performance of Bluetooth. Most of the research being performed prerequisites that a connection has already been made. Due to the frequency hopping spread spectrum used by Bluetooth the connection establishment is not as easy as it seams. Two processes have been derived to bridge the frequency discrepancies between devices, *inquiry* and *page*.

In this thesis, we study the inquiry and page processes and the time it takes to complete a connection with no, one or two voice channels present at the initiating device. We have found the times to complete the processes to be lengthy at best, unreasonably long if there is voice traffic present. Through optimisation, we have decreased the mean time to connect. We propose some changes to the default values of a few time outs and variables, which yields a substantial improvement in performance, especially in the case when there is voice traffic present. The changes are in the software only, no changes in hardware are necessary.

After a connection is made a client may want to start using the services of the device it is connected to. This is another topic of this thesis. Bluetooth does not provide means of accessing a service, only discovering it. Thus, there is the need for higher-level protocols for accessing services. We have investigated some of the service discovery protocols currently available.

Contents

List of Figures

List of Tables

Acknowledgements

We would like to thank some of the people that have made our stay in Australia a pleasant one, and for guiding us in our project.

Mr Bruce Tunnicliffe and Professor Hans-Jürgen Zepernick, our supervisors at ATcrc.

Dr Markus Fiedler, our examiner at BTH, who came ”down under” to visit us.

Professor Sven Nordholm, Director of ATRI, who made it all possible in the first place.

And of course our friends in ”Team SWEDEN”.

Chapter 1

Introduction

This thesis is written as a part of the Master of Science degree in Electrical Engineering with emphasis on Telecommunications and Signal Processing at Blekinge Institute of Technology (BTH). The thesis work has been carried out at the Australian Telecommunications Cooperative Research Center (ATcrc) at Curtin University of Technology, Perth, Western Australia.

1.1 Background

Bluetooth has become the de facto standard for short range wireless communication. Over 2000 companies have registered their interest in the Bluetooth Special Interest Group (SIG). Although Bluetooth products are already being shipped, many aspects of Bluetooth are still uncharted territory. Among these are the timing of Bluetooth service discovery. Several companies are trying to adopt Bluetooth standard to a sort of mini cellular network to complement the existing 2G and 3G networks and even compete against them. These companies will have to consider several aspects of Bluetooth and the fact that the technology was not built with such systems in mind.

1.2 Task description

Bluetooth equipped hand held devices can be used to obtain access to services provided by a fixed Bluetooth network. As these devices are typically carried by a user, the user may not know what services are available within the network. Furthermore, the hand held device, being an ad hoc Mobile Terminal (MT), does not know what services are available and how to access them. Service Discovery is the process that enables the MT to access the services of the network. The Bluetooth specification specifies a minimal form of service discovery - i.e. between two Bluetooth devices. Due to the fact that a user can enter a network anytime from any physical position in the network, the timing of the connection setup and service discovery is of vital importance. In a cell with a radius of 10m a person will walk through it in approximately 15 – 20 seconds, during which a connection setup will have to be made and possibly a handover to

another cell and the service discovery procedure. This project aims to analyse the timing of a device entering the network and how to gain access to the services therein.

1.3 Structure of the thesis

This thesis is aimed at two topics closely related to each other, *device discovery* and *service discovery*. Chapter 1 provides an introduction and presentation of the thesis and some background about the Bluetooth standard. The 2nd chapter describes the Bluetooth standard and the lower levels of the protocol stack to give an overview and basic knowledge about Bluetooth.
Chapter 3 provides an in-depth description of the device discovery process used by Bluetooth to open a communication channel to remote devices. Furthermore, the impact of reserved bandwidth channels is discussed theoretically.
Chapter 4 deals with the second topic, service discovery, in providing a description of the Bluetooth Service Discovery Protocol and brief descriptions of other service discovery protocols that may work on top of the Bluetooth protocol stack. A discussion is made regarding the latter's advantages and disadvantages and their suitability in a distributed system.
In Chapter 5, the performance of the device discovery process is evaluated, with or without reserved bandwidth channels. An attempt is made to show that there is an increase in performance to be won simply by paying attention to some time outs and parameters.
Chapter 6 compiles the conclusions from the earlier chapters.
Appendix A is a manual to the simulator used and appendix B is a description of the problems encountered in the process of simulating the device discovery.

Chapter 2

Bluetooth technical overview

This chapter deals with the Bluetooth standard, it gives a brief overview of the Bluetooth technology and Bluetooth protocol stack. The first section describes the MediaCell network, to give the reader basic knowledge about the MediaCell system.

2.1 MediaCell system overview

The thesis is aimed at the MediaCell network (2.1), which is a network of Bluetooth Access Points working in conjunction with each other. The MediaCell network has a three tier topology with a centralized server called Gateway (GW) which serves a number of concentrators, Cluster Controllers (CC), these control up to six Access Points (AP).
A Mobile Terminal (MT) accesses the MediaCell network through the APs. The MediaCell network will support Bluetooth handover so a MT will only have to log on to the network once to gain access to the services therein.

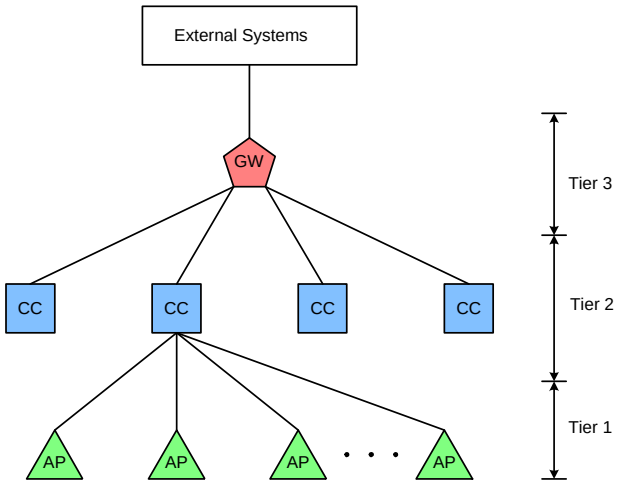

Figure 2.1: *The MediaCell network topology.*

2.2 Bluetooth overview

Bluetooth is a short-range radio technology designed to replace cables. It's origin is within the Ericsson concern as means to replace the cables to peripheral devices for Ericsson's mobile telephones.
It has since then grown to a short-range wireless ad hoc networking solution. The key features of Bluetooth are low cost, robustness and low power consumption. Bluetooth operates in the universally unlicensed Industrial, Scientific and Medical (ISM) band at 2.4 GHz. It uses a frequency hop spread spectrum, which changes the transmission frequency 1600 times per second in a pseudo random way. The range of the Bluetooth radio is approximately 10 m with a standard Bluetooth device. The range can be extended to 100 m by increasing the output power. When two or more Bluetooth devices form an ad hoc network it is called a piconet. Several piconets can be linked together and the resulting network is called a scatternet.

2.2.1 Bluetooth radio

The most common Bluetooth radio [1, pp 15–32] has an output power of 0 dBm. These devices are called class 3 devices. There are two more classes; class 2 which has a maximum output power of 4 dBm and class 1 which has a maximum output power of 20 dBm. Channel spacing is 1 MHz and a guard range is used in the top and bottom of the frequency band which makes for 79 channels. There are a 23 channel radio defined for countries with special radio frequency regulations. The modulation technique used is Gaussian Frequency Shift Keying (GFSK).

2.2.2 Baseband

The Baseband (BB) [1, pp 33–184] sets up connections, defines links and packet types and provides error correction. It sets up the hopping sequence of the piconet based on the master's unique device address (BD_ADDR). Two different connections are supported; Asynchronous Connection Less (ACL) and Synchronous Connection Oriented (SCO). Typically the ACL link transmits data and the SCO link is used only for voice.
The physical channel is divided into time slots of 625 μs in length, each time slot corresponds to a RF frequency which is determined by the masters BD_ADDR. A time division duplex TDD scheme is used where the master and slave transmits alternatively. Normally a BB packet covers one time slot but there are packets covering 3 and 5 time slots as well (Figure 2.2).

Each active member in a piconet is given a three bit address called the Active Member Address (AM_ADDR) which acts as a MAC address. There can only be seven active slaves participating in a piconet at the same time, AM_ADDR 000 is reserved for broadcast. There can however be a number of slaves in so-called parked mode, these slaves are not participating in the piconet traffic but remain synchronized with it.
The master starts its transmission in even-numbered time slots only, the slave starts its transmission in odd-numbered time slots only. The time slots are numbered according to the Bluetooth clock (CLKN) of the piconet master. In an ACL connection a slave can only transmit a packet if it has received a packet containing its AM_ADDR in the previous time slot. However, a slave

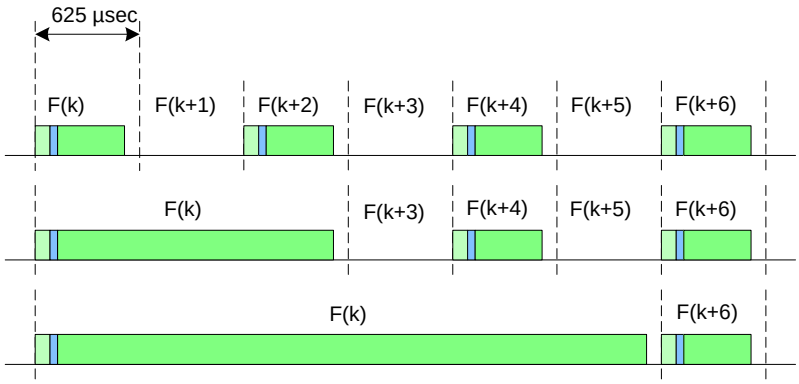

Figure 2.2: *Packet lengths supported by Baseband.*

with an SCO connection may transmit even though it has not received its AM_ADDR in the previous time slot.
Each packet carries an access code which is the first 72 or 68 bits, the latter if no packet header exists. The access code is used for synchronization, DC offset compensation and identification. There are three access codes defined within Bluetooth, these are as follows.

- Channel Access Code: identifies which piconet the packet belongs to. It is included in every packet transmitted on a piconet.
- Device Access Code: used during the device discovery process to page a device. It is derived from the BD_ADDR of the device.
- Inquiry Access Code (IAC): used in the device discovery process when a device inquires about which devices are in range. There are two kinds of Inquiry Access Codes, *General* IAC (GIAC) and *Dedicated* IAC (DIAC). GIAC is used in a general case when a device wishes to know all devices in range. DIAC is used when a device only wishes to know about devices with special characteristics.

The BB determines the hopping sequence of the Bluetooth device. There are five hopping sequences for a standard 79 channel device. These are described below:

- Channel Hopping Sequence: distributes the hopping frequencies pseudo randomly but equally over the 79 channels. It has a very long period so as not to repeat itself in the short term. It is calculated from the BD_ADDR of the master of the piconet.
- Inquiry Hopping Sequence: a sequence of 32 wakeup frequencies. It is calculated from the GIAC or the DIAC. The inquiry hopping sequence switches between 2 inquiry trains of 16 frequencies each. The phase is determined from the native clock (CLKN) of the unit.

- Inquiry Response Sequence: also uses two trains of 16 frequencies each, which are exactly the same as the 2*16 frequencies of the Inquiry Hopping Sequence.
- Page Hopping Sequence: a sequence of 32 frequencies calculated from the device being paged's BD_ADDR. The phase is determined from an estimate of the paged device's clock. The paging device will calculate the frequency at which the paged device will be listening on and start transmitting on that frequency to speed up the process. Two trains of 16 frequencies each are used.
- Page Response Sequence: uses two trains of 16 frequencies each, which are exactly the same as the 2*16 frequencies of the Page Hopping Sequence.

2.2.3 Link Manager Protocol

The Link Manager Protocol (LMP) [1, pp 185–254] provides means for setting up secure, power efficient links for both voice and data. It has the ability to, whenever necessary, update the link properties to ensure optimum performance. The Link Manager also terminates connections, either on higher layers request or because of various failures. The LMP also handles security and the different low-power modes *sniff, hold* and *park*:

- Active: a device participates in the piconet by listening (in the master-to-slave time slots) for packets containing its own AM_ADDR.
- Sniff: in sniff mode, a device acts similar to an active device. Upon entering sniff mode, the master and slave decide a sniff interval; the time between two time slots where the slave will listen for packets.
- Hold: a device that enters hold mode, does so for a specified time. During this time ACL packets are not supported but SCO packets can still be transmitted.
- Park: when a slave enters park mode, it gives up its AM_ADDR but remains synchronized to the piconet. It is given a Parked Member Address (PM_ADDR) that the master uses for un-parking the slave and an Access Request Address (AR_ADDR) that the slave can use to request to be un-parked.

2.2.4 Logical Link Control and Adaptation Protocol

The Logical Link Control and Adaptation Protocol (L2CAP) [1, pp 255–333] supports protocol multiplexing, packet segmentation and reassembly, the maintenance of Quality of Service and group management. L2CAP is placed above the Baseband Protocol and interfaces with higher protocols.

2.2.5 RFCOMM

RFCOMM [1, pp 396–427] is a simple transport protocol intended as a cable replacement protocol. The RFCOMM provides transparent data stream and control channels over the L2CAP channels.

2.2.6 Adapted protocols

There are a number of protocols that have been adapted to fit into the Bluetooth protocol stack [1, pp 446–539]. These include protocols such as TCP/IP, WAP and OBEX.

2.2.7 Service Discovery Protocol

The Service Discovery Protocol (SDP) [1, pp 334–395] enables a Bluetooth device to inquire what services are available in a piconet and how to access them. The SDP should be able to determine the properties of any service, future or present, of arbitrary complexity in any operating environment. However, it will not provide any access to the services themselves or negotiate service parameters.
SDP uses a server-client model where the server has a list of elements called Service Records. Each Service Record describes the characteristics of one service. There can only be one SDP server in a Bluetooth device, if a device has several services the one SDP server can act on behalf of all of them. Similarly multiple applications may use a single SDP client to query servers of Service Records. If a device only acts as an SDP client it does not need to have an SDP server. A device may operate as both a server and a client at the same time.
The server-client model used by SDP states that each transaction consists of one request protocol data unit (PDU) and one response PDU. To provide some flow control, a client must receive a response to each request before issuing another request on the same L2CAP connection.
The SDP database consists of a set of records describing the services of the device. A service is described using Service Attributes. A Service Attribute consists of a 16 bit Attribute ID and a variable length Attribute Value. The Attribute Value PDU has a header describing their type and length to the receiver. For more information about the SDP, see Section 4.1.

2.3 Profiles

A Profile defines how to use a set of base standards for a certain usage model and is the primary means of achieving interoperability [2]. Profiles in Bluetooth are horizontal segments of the Bluetooth protocol stack. All devices must adhere to at least one profile, the Generic Access Profile (GAP) [2, pp 15–62]. GAP contains the basic functionality all Bluetooth devices must have in order to function. Another important profile is the Service Discovery Application Profile (SDAP) [2, pp 63–95], it defines the procedures and protocols used by a Service Discovery Application on a device to locate services in other devices using SDP.
Several other profiles have been defined for different uses. This means that a Bluetooth device must not support the entire specification but rather a segment of the protocol stack.

Chapter 3

Device discovery

In mobile ad hoc environments, devices initially have no information about their surrounding environment or the devices that operate within their range. There is no centralized instance to query about the environment. Therefore, a protocol must exist that provides means for detecting devices and enables devices to set up a connection, Bluetooth uses the Baseband protocol for this task. Two procedures are used in the device discovery procedure; *inquiry* and *page*.
In this thesis we aim at evaluating the performance of the device discovery process. Especially the timing of the device discovery process is of vital importance. This section provides an in-depth description of the device discovery process.

3.1 Inquiry

In order to set up a connection, a device must detect what other devices are in range. This is the goal of the inquiry procedure (Figure 3.1). The process is initiated by the unit that wishes to collect device information or create a connection. To conserve power and coexist with other link activity, inquiry is always initiated by higher level control protocols [6, pp 72–75]. The inquiry procedure must overcome the initial frequency discrepancy between devices. Therefore, inquiry only uses 32 of the 79 hop frequencies. Typically a device enters inquiry mode periodically. Similarly, a device that wishes to be visible to inquiring units enters inquiry scan in certain intervals. In order to find each other, one device must be in Inquiry state and one (or more) device must be in Inquiry Scan substate simultaneously (Figure 3.2).

A device in Inquiry state broadcasts ID packets on the 32 frequencies of the Inquiry Hopping Sequence at twice the normal frequency hopping rate. I.e. it sends two ID packets every 625 μs and then listens for responses the following 625 μs. It does so for the duration of the inquiry window denoted $T_{\text{w inquiry}}$. The device can also be set to exit Inquiry state after a number, denoted $N_{\text{inq responses}}$, of devices has been found. In periodic inquiry mode, the time between two consecutive inquiries is determined by the inquiry interval, T_{inquiry}, (Figure 3.2).
An ID packet contains the Inquiry Access Code. As mentioned earlier in Section 2.2.2, a device can inquire about all units in range using GIAC or a specific type of device using DIAC. The Inquiry Hopping Sequence however, is always derived from the Lower Address Part (LAP) of

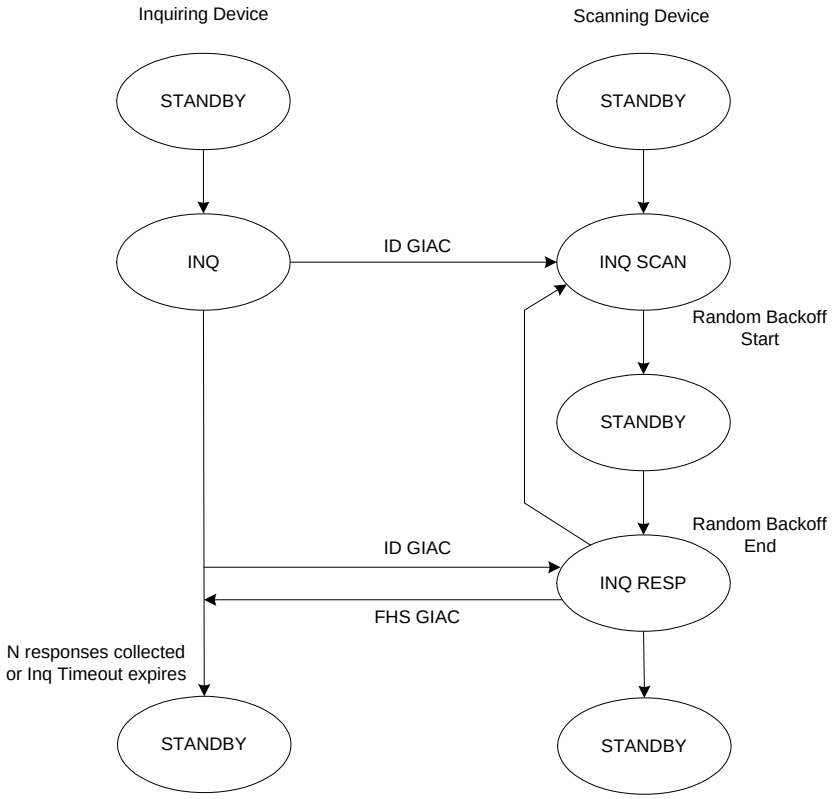

Figure 3.1: *State transitions during the inquiry process.*

the GIAC which is common for all devices and therefore it is the same for all devices. The 32 frequencies are divided into two trains, *A* and *B*, each train is 10 ms in length and according to the specification [1, pp 108–111] a train must be repeated at least 256 times before a new train is used ($N_{\text{inquiry}} \geq 256$). The specification also states [1, p 111] that in an error free environment at least three train switches must have taken place in order to ensure that all devices in range respond to the inquiry, resulting in $T_{\text{w inquiry}} = 10.24$ s. In an error prone environment more train switches may be nessecary to ensure responses from all devices in range.

The phase within the trains is derived from the *CLKN* of the inquiring device and therefore unique to every device. The phase X_i is calculated as follows [1, p 137]:

$$X_i = [CLKN_{16-12} + k_{\text{offset}} + (CLKN_{4-2,0} - CLKN_{16-12}) \bmod 16] \bmod 32 \tag{3.1}$$

where $CLKN_{x-y,z}$ denotes bits x to y and bit z of the native clock. k_{offset} selects the train used [1, p 136]:

$$k_{\text{offset}} = \begin{cases} 24, & \text{train A} \\ 8, & \text{train B} \end{cases} \tag{3.2}$$

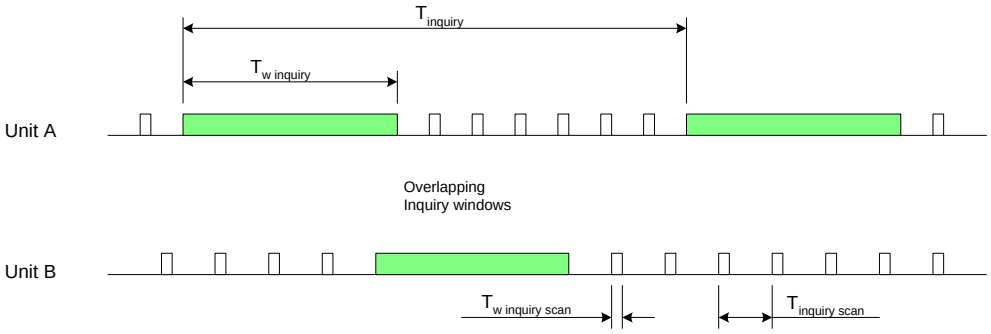

Figure 3.2: *Periodic inquiry and inquiry scan.*

A device wishing to be found by inquiring units periodically enters Inquiry Scan substate. The interval between two consecutive inquiry scans is determined by the inquiry scan interval, $T_{\text{inquiry scan}}$. The time a device stays in Inquiry Scan substate is determined by the inquiry scan window, $T_{\text{w inquiry scan}}$. During this time the device listens to a single frequency of the Inquiry Hopping Sequence. The phase is determined by its native clock [1, p 138]:

$$X_{ir} = [CLKN_{16-12} + N] \bmod 32 \tag{3.3}$$

The value of N is increased each time $CLKN_1$ is set to zero, which corresponds to the start of a master transmission slot.
According to the specification [1, p 107], $T_{\text{w inquiry scan}}$ must be greater than 10 ms in order to ensure that a frequency synchronization takes place, assuming that the listening frequency is in the train transmitted. It also states that $T_{\text{inquiry scan}} \leq 2.56$ s.
Upon reception of an ID packet a device in Inquiry Scan substate will leave Inquiry Scan substate for a random backoff time, uniformly distributed between 0 and 1023 time slots which corresponds to $[0, \ldots, 639.375\ ms]$. This is done to reduce the probability that devices will respond to the same ID packet simultaneously, thus colliding. After the random backoff the device will enter Inquiry Response state, in which it listens for a second ID packet. After receiving this, the device responds with a Frequency Hopping Selection (FHS) packet containing its device information, i.e. its BD_ADDR and its current clock (CLKN).
If the inquiring device receives a response FHS packet in the first half of the receive slot, it cannot receive a response from another device in the second half because the radio channel synthesizer does not permit that. Similarly if it receives a response in the second half of the receive slot, it cannot send the first ID packet in the following transmit slot.
In Inquiry state, a device cannot exchange ACL data with another device. However, SCO links are still supported in the Inquiry and Inquiry Scan substates. If there are SCO links active, the inquiring device must increase N_{inquiry} and $T_{\text{w inquiry}}$ to make up for the time slots lost to the SCO links, this will be discussed in more detail in Section 3.3.

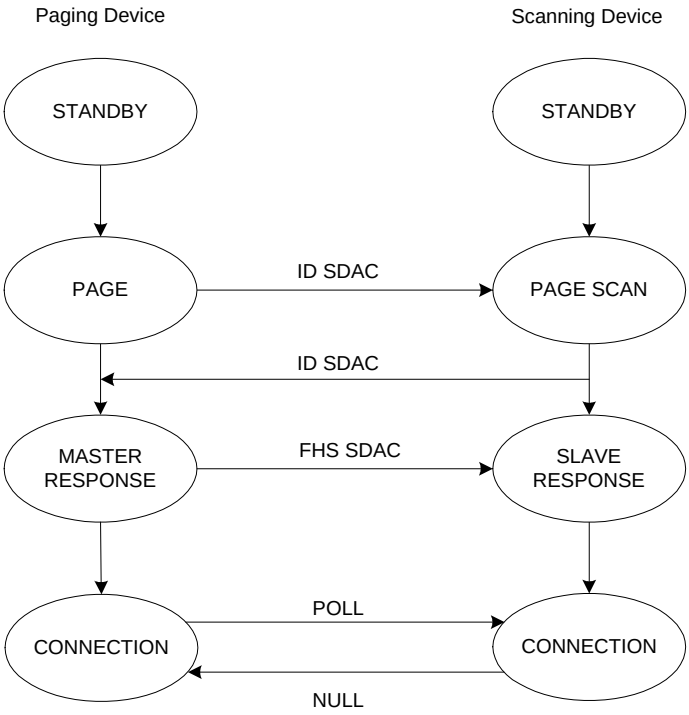

Figure 3.3: *State transitions during the page process.*

3.2 Page

In Bluetooth the connection establishment is handled by the page process (Figure 3.3). The page process requires knowledge of the BD_ADDR of the device with which the connection is to be established. Furthermore the device being paged must be in Page Scan substate, i.e. listening for page messages. At the end of the page process a connection has been set up, the paging device becomes the master and the paged device becomes the slave. As with inquiry a device typically enters Page state periodically and a device that wishes to be able to connect to paging units enters page scan in certain intervals.
In Page state a device transmits ID packets at twice the normal frequency hopping rate using the 32 frequencies of the Page Hopping Sequence. The Page Hopping Sequence is derived from the paged device's BD_ADDR and the phase is calculated as follows:

$$X_p = [CLKE_{16-12} + k_{\text{offset}} + (CLKE_{4-2,0} - CLKE_{16-12}) \bmod 16] \bmod 32 \qquad (3.4)$$

where $CLKE_{x-y,z}$ denotes bits x to y and bit z of the clock estimate of the paged device. The parameter k_{offset} selects the train used (3.2). Unlike inquiry the page process always start on train *A*. This is because the train *A* starts with the most probable listening frequency of the paged device, based on the *CLKE* received from e.g. an inquiry. It is of course possible to perform a page without the *CLKE* but it will consume more time.
Similar to the inquiry process, the time a device stays in Page state is determined by the page

window, $T_{\text{w page}}$. In periodic page mode, the time between two consecutive pages is determined by the page interval, T_{page}.
In periodic page scan mode the interval between two consecutive page scans is set by the page scan interval, $T_{\text{page scan}}$. The length of the scan is determined by the page scan window, $T_{\text{w page scan}}$. The scanning device listens to frequencies in the Page Hopping Sequence. The phase is determined by it's native clock:

$$X_p = CLKN_{16-12} \tag{3.5}$$

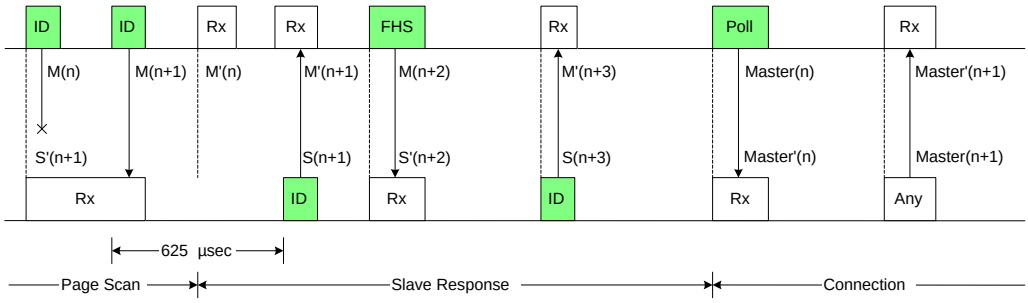

Figure 3.4: *Page procedure.*

There are three modes concerning the Page Scan substate, *R0*, *R1* and *R2*. These modes set the behavior of the scanning device (Table 3.1). Depending of the scanning device's scan mode the paging device must repeat the page train a certain amount N_{page} times, called the repetition number, or until a response is obtained. If the scanning device's scan interval corresponds to R1, then $N_{\text{page}} \geq 128$; if the slave scan interval corresponds to R2, then $N_{\text{page}} \geq 256$.

Mode	$T_{\text{page scan}}$	N_{page}
R0	continous	≥ 1
R1	≤ 1.28 sec	≥ 128
R2	≤ 2.56 sec	≥ 256
Reserved	-	-

Table 3.1: *Relationship between scan interval, train repetition, and paging modes [1, p 98].*

When a unit in Page Scan substate receives an ID packet containing its own Access Code it acknowledges the page message with an ID packet and enters Slave Response state. In this state it freezes the $CLKN_{16-12}$ bits at their current values, then using a value one larger than the preceding slot for determining the phase [1, p 136]:

$$X_{prs} = [CLKN^*_{16-12} + N] \text{mod} 32 \tag{3.6}$$

where $CLKN^*_{16-12}$ are the frozen clock bits and N is a counter set to zero in the slot the paged device acknowledges the page and incremented each master TX slot.
After receiving the ACK from the paged device, the paging device enters the Master Response substate. In this mode the phase is determined as follows.

$$X_{prm} = [CLKE^*_{16-12} + k^*_{\text{offset}} + (CLKE^*_{4-2,0} - CLKE^*_{16-12}) \bmod 16 + N] \bmod 32 \tag{3.7}$$

In Master Response substate the device sends a FHS packet containing its CLKN, which will be the piconet clock CLK, and the AM_ADDR the paged device shall use. After acknowledging the FHS packet the paged device, now called the slave, switches to the hopping sequence of the pager, now called the master. Now the two units are connected and they both switch to Connection state. To make sure the slave has changed to the master's hop sequence, it sends a Poll packet to the slave which in turn answers with an arbitrary packet. The entire page process [1, p 91] is shown in Figure 3.4. The Paging scheme described above is the normal one, all units support it. In addition to supporting this mandatory paging scheme, a unit may support one ore more optional paging schemes.

Optional paging schemes

To this date only one optional paging scheme has been defined, called optional scheme *I* [1, pp 1024–1028]. It uses the same 32 page frequencies as the mandatory paging scheme and they are split into two trains, A and B. In contrast to the mandatory scheme, the same 32 frequencies that are used for transmitting are also used for reception of page responses, called reception trials. The page train construction differs from the mandatory paging scheme in two ways; the page train consists of 10 slots, or 6.25 ms. The first 8 slots of the train are used to transmit the ID packets, the 9th slot is used to send a marker packet, and the 10th slot is used for the return of a slave response. The marker packet indicate that a response may be sent and it also contains information about what frequency to be used for the response.

3.3 Impact by existing SCO links

Due to the slot reservation nature of the SCO links, the performance of inquiry, page and the scan modes suffers greatly when a SCO link is in operation. The specification states that $T_{\text{w inquiry scan}}$ and $T_{\text{w page scan}}$ should be increased to increase the probability to respond to an inquiry or page message [1, pp 98 and 107]. If one SCO link is operating using HV3 packets and the SCO period $T_{\text{SCO}} = 6$ slots, a total scan window of at least 22.5 ms (36 slots) is recommended; if two SCO links are present using HV3 packets and $T_{\text{SCO}} = 6$ slots, a total scan window of at least 33.75 ms (54 slots) is recommended.
The construction of the inquiry or page trains is not affected by SCO channels other than the fact that every third master-slave transmit and receive slots are reserved for each SCO channel (Figure 3.5).

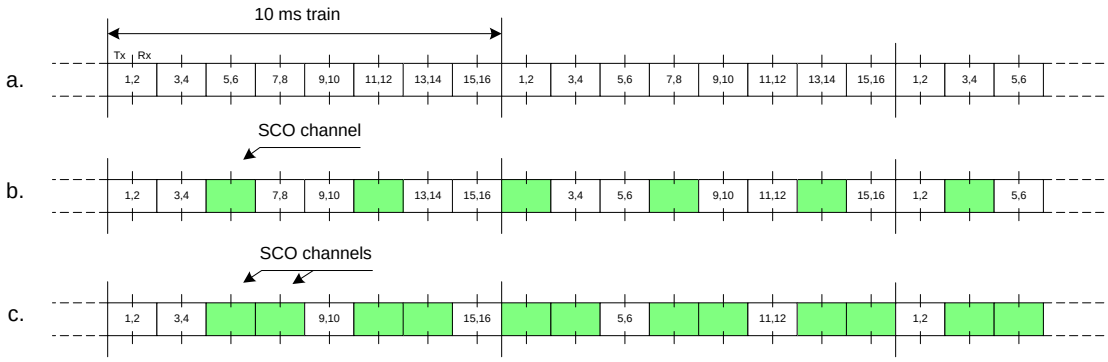

Figure 3.5: *Page trains. a. Normal page, b. Page with one SCO link, c. Page with two SCO links [1, p 101].*

3.3.1 Effects on inquiry

As mentioned earlier in Section 3.1, a device in Inquiry Scan substate must receive an ID packet and respond with an FHS packet 625 μs later. If it at the same time has an SCO connection the SCO connection takes priority over the inquiry scan, thus it must retain it by regularly Rx and Tx SCO packets. The latest possible Rx point of an ID packet after backoff without interfering with the reserved SCO slots is shown in Figure 3.6. Considerations must be taken to the FHS response packet as well as the radio settle time. Any ID packet received after this point must be ignored since there is no time to send the response FHS packet without violating the SCO link. The time loss is made up by the following [6, pp 80–81]:

- One slot for waiting to Tx the FHS packet 625 μs.
- FHS packet duration 366 μs.
- Radio channel synthesizer settling time $\leq$ 200 μs.
- Two slots for the SCO connection $2 * 625$ μs.

The total time loss is between 2214 and 2441 μs.

This means that approximately 60 % of the inquiry scanning time is lost [6] when a SCO link ($T_{SCO} = 6$) is up. When two SCO links are up the scan loss is about 90 %.
For the inquiry process, the SCO link also takes priority. The specification states [1, p 109] that $N_{inquiry}$ should be increased if SCO links are present. In the presence of one SCO link ($T_{SCO} = 6$), the repetition number should be doubled ($N_{inquiry} \geq 512$) and in the presence of two SCO links trebled ($N_{inquiry} \geq 768$).

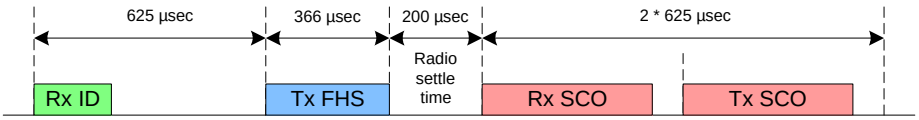

Figure 3.6: *Timing of inquiry scan with one SCO link present.*

3.3.2 Effects on page

Similar to the inquiry scan the page scan (Section 3.4), has a latest point of reception of the page ID packet (Figure 3.7). Considerations have to include; when the ID packet is received, 1st half or 2nd half of the time slot. The packets exchanged between the devices and the radio settle time. Because of the longer sequence of packet exchange, the impact on the page scanning process is greater then the inquiry scan process. The following indicates what factors contribute to the time loss:

- One slot for Rx of the page ID packet 625 μs.
- One or half a slot to Tx the response ID packet 625 or 312.5 μs.
- One slot to Rx the FHS packet 625 μs.
- Tx the FHS ACK ID packet 68 μs.
- Radio channel synthesizer settling time $\leq$ 200 μs.
- Two slots for the SCO connection $2 * 625$ μs.

The total time loss is between 3193 and 3393μs if the page packet is received in the 1st half of the time slot. If the page packet is received in the 2nd half of the time slot, the total time loss is between 2880.5 and 3080.5 μs.

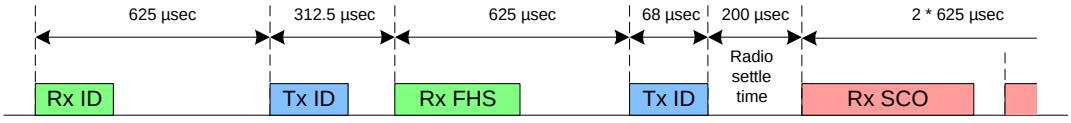

Figure 3.7: *Timing of page scan with one SCO link present, Rx ID packet in 2nd half slot.*

This corresponds to a 83 % and 76 % decrease in page scan time, respectively, for the two cases. In the presence of two SCO channels, paging will be very difficult. In Chapter 5, we attempt to simulate the impact on the inquiry and page procedures.
The SCO link also takes priority over the page process. The specification states [1, p 100–101] that N_{page} should be increased if SCO links are present. In the presence of one SCO link

($T_{SCO} = 6$), the repetition number should be doubled and in the presence of two SCO links trebled (Table 3.2).

Mode	No SCO link	1 SCO link	2 SCO links
R0	$N_{page} \geq 1$	$N_{page} \geq 2$	$N_{page} \geq 3$
R1	$N_{page} \geq 128$	$N_{page} \geq 256$	$N_{page} \geq 384$
R2	$N_{page} \geq 256$	$N_{page} \geq 512$	$N_{page} \geq 768$

Table 3.2: *Train repetition for the different and paging modes R0, R1 and R2 when SCO links are present [1, p 101].*

3.4 Master slave switch

In a network like the MediaCell network, all devices are allowed to perform inquiries and pages hence the role of master may fall upon a MT rather than an AP. This is not desirable, in such a case, a master slave switch should be performed to give the control of the piconet to the MediaCell network [1, pp 122–125 and 210–212]. A master slave switch means reversing the TDD timing of the master and slave and a redefinition of the piconet parameters. It can be initiated by either the master or the slave and further slaves can be transferred to the new master after the new piconet is set up.
A switch starts with the slave transmitting an LMP_slot_offset packet immediately followed by an LMP_switch_req, the master responds with LMP_accepted or LMP_not_accepted. In case the master accepts, both units perform a TDD switch, i.e. the former slave is now the master and transmits in the master time slots, and sets a timer at newconnectionTO. The new master sends an FHS packet containing the new AM_ADDR for the new slave which responds with an ID packet and the timers are reset to newconnectionTO. With the information from the FHS and the LMP_slot_offset packets the devices change the piconet parameters, i.e. the hopping sequence and piconet clock. Both are now running on the new parameters and roles which is confirmed by a POLL packet sent by the new master which in turn is acknowledged by a NULL packet and the timer is canceled. Should any of the timers timeout, both devices return to the old piconet parameters and roles and the switch will have to start over.
The master slave switch can theoretically take between 4.375 and 41.875 ms

Chapter 4

Service discovery and access

Bluetooth Service Discovery Protocol (SDP) provides means for discovering services within remote devices but it does not provide access to them, hence there is the need for other service discovery protocols that addresses this issue such as Jini, Salutation etc. These techniques will be described briefly here and a comparison of their features will be made to determine the technique best suited for the MediaCell network.

4.1 Bluetooth Service Discovery Protocol

The Service Discovery Protocol allows Bluetooth devices to discover what services other devices may offer. It permits service browsing and searching for specific services. But SDP does not provide access to the services themselves.
SDP relies on L2CAP links between the SDP server and client. Following the setup of a L2CAP link, a client may query the SDP server about services available (Figure 4.2) [6, p 216]. SDP uses a dedicated L2CAP channel known beforehand to all devices. This channel uses a Protocol Service Multiplexer reserved for SDP. After the SDP data has been received, a client must set up a new connection to access the service, the SDP channel cannot be used. Devices supporting the SDP application profile must support pairing and authentication.

4.1.1 Format

A SDP PDU is a packet of variable length depending on the parameters of the service record. Every SDP PDU consists of a header followed by PDU-specific parameters, the header consists of three fields [1, pp 354–355]:

- PDU ID (1 byte), identifies the type of PDU, i.e. its meaning and the specific parameters.
- Transaction ID (2 bytes), identifies request PDUs and is used to match responses to requests.
- Parameter Length (2 bytes), specifies the length of the parameters carried in the PDU.

As some requests may require responses that are longer than the PDU length, a PDU may contain a Continuation State parameter to indicate that not all parameters were sent in the PDU. Then the client repeats the request with the Continuation State copied from the response. The server then responds with the rest of the parameters.
Every request PDU has a corresponding response PDU. Should there be an error in the request or the server cannot respond to the request, the server will respond with an error PDU (*SDP_ErrorResponse*).

4.1.2 Service Record

All information about a service is stored in a single Service Record within the SDP server [8]. A Service Record consists of a list of Service Attributes. There are 16 Service Attributes defined in the specification v 1.1 [1, pp 368–379]. A Service Record is identified by a Service Attribute called *ServiceRecordHandle*. Some Service Attributes are common to all service records, but there are reserved fields where vendors can define their own Service Attributes.
A Service Attribute consists of two components:

- Attribute ID (16 bits), distinguishes each Service Attribute within a Service Record. It also identifies the semantics of the Attribute Value.
- Attribute Value (variable length), a data element consisting of two fields, a header and data. The meaning of the Attribute Value is defined by the corresponding Attribute ID.

A typical Service Record is made up as follows: The first part is the ServiceRecordHandle followed by the ServiceClassIDList which lists the services offered by the device.
The ProtocolDescriptorList lists the protocols needed to use the service. There may also be protocol specific parameters that are needed to use the protocols listed, such as channel numbers etc.
The BluetoothProfileDescriptorList lists the profiles which the service supports described by their UUID and a version number. If a profile is dependent of another profile, only the top level profile is listed. GAP is never listed since it is mandatory for all services.
Finally there may be some extra attributes needed to use the service, e.g. ServiceName.

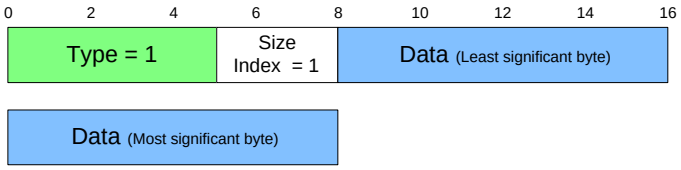

Figure 4.1: *Example of a Data Element, containing a 16 bit unsigned int.*

4.1.3 Representation of data

All data in the Attribute Values is represented as a data element. A data element is a typed data representation of variable length. It consists of a header and data field. The header contains a 5 bit Type Descriptor and a 3 bit Size Descriptor. There are nine types of data defined:

- 0, Null.
- 1, Unsigned int.
- 2, Signed two complement int.
- 3, Universally Unique Identifier (UUID).
- 4, Text string.
- 5, Boolean.
- 6, A sequence of data elements, all of which make up the data.
- 7, Data sequence alternative. A sequence of data elements, of which one must be chosen.
- 8, Uniform Resource Locator (URL).

The Size Descriptor is an index of the size using the three bits left in the byte from the Type Descriptor. Sometimes, however, Service Attributes can be too long to be described with the index, such as text strings, hence the length must be described in another way. Therefore, the Size Descriptor can also indicate where the actual data size can be read. The Size Descriptors and what they indicate are listed below:

- 0, 1 byte (or 0 if Null type).
- 1, 2 bytes.
- 2, 4 bytes.
- 3, 8 bytes.
- 4, 16 bytes.
- 5, Data size in next 1 byte.
- 6, Data size in next 2 bytes.
- 7, Data size in next 4 bytes.

This system saves bandwidth since most attributes fit in the 1–16 bytes of the Size Descriptor index part.

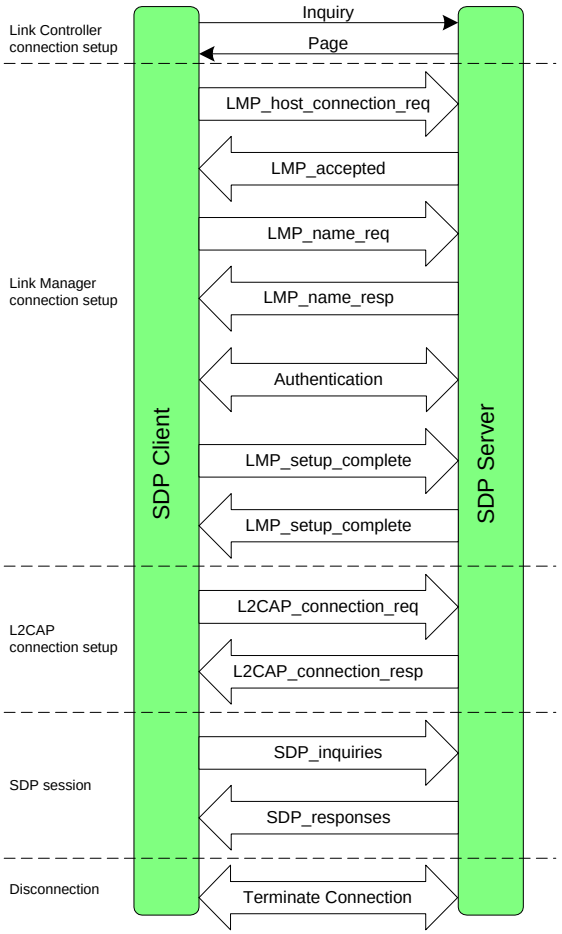

Figure 4.2: *SDP session messaging.*

4.1.4 Searching and browsing services

To make services easily searched and browsed, they are arranged in a tree structure. The hierarchy of the tree and which services will be browsable is up to the service providers to decide. The top of the tree is called PublicBrowseRoot. Service Classes are used to identify services, all services are members of a Service Class. A Service Class record defines all attributes contained in a Service Record of the fore-mentioned class. It always includes a BrowseGroupList attribute, which is a list of UUIDs of all browse groups associated with the service [6, pp 199–201][8].

A client can search for services with specific attributes contained in their Service Records provided that the attributes are of UUID type. Search for arbitrary values is not supported. In all searches SDP_ServiceSearchRequest is used, it includes a ServiceSearchPattern which is a list of UUIDs that are to be searched for by the server in its database. The server responds with a SDP_ServiceSearchResponse that includes the ServiceRecordHandle of the matching Service Records. The ServiceRecordHandles are then used to retrieve the attributes of the desired ser-

vices, done with SDP_ServiceAttributeRequest.
To speed up the process of retrieving Service Attributes a client can combine a service search and an attribute request using SDP_ServiceSearchAttributeRequest. The server then responds with the ServiceRecordHandles and attributes of the services matching the ServiceSearchPattern.
A client that wishes to browse through the services of a server creates a ServiceSearchPattern containing the UUID of the PublicBrowseRoot. All services that may be browsed from the top level are made members of the root group by including the PublicBrowseRoot UUID in the BrowseGroupList attribute.

4.2 Salutation

Salutation is being developed by the Salutation Consortium. It aims at solving the problem of service discovery and utilization among various types of devices and environments. Salutation is designed to be processor, operating system and communication protocol independent. It enables devices and services to announce, discover, search and access other services and devices.

4.2.1 Architecture overview

Salutation defines two major components: *Salutation Manager* and *Transport Manager* [3]. The Salutation Manager provides an interface to server and client-applications (SLM-API), containing service registration, service discovery and service access functions.
The Salutation Managers are the core of the architecture. Salutation Managers (SLM) handle the service brokerage. A service provider registers its capabilities with an SLM. Upon a query from a client to its local SLM the search is performed by coordination among Salutation Managers [9].
The SLM-to-SLM communications protocol is defined by the Salutation Architecture and called the Salutation Manager Protocol. This protocol uses Open Networking Computing Remote Procedure Call. This requires the underlying transport protocol to support multiple reliable bi-directional communications.
The Transport Manager(s) (TM) provides these reliable transport channels to the Salutation Manager(s) through the Salutation Manager Transport Interface (SLM-TI) (Figure 4.3).

4.2.2 Operations overview

The Salutation defines a Functional Unit as a logical subdivision of equipment that performs one meaningful task [3]. A service is subdivided into Functional Units, each representing some of the service's essential functions. A service record of a service consists of such Functional Units, e.g. a fax service may have Functional Units like [Print], [Scan] and [Fax Data Send].
Every Salutation Manager has a service registry which contains information about the services. Every client registers and un-registers itself, usually with the local SLM. A Salutation Manager can be set to periodically check the availability of functional units and report it to its clients or

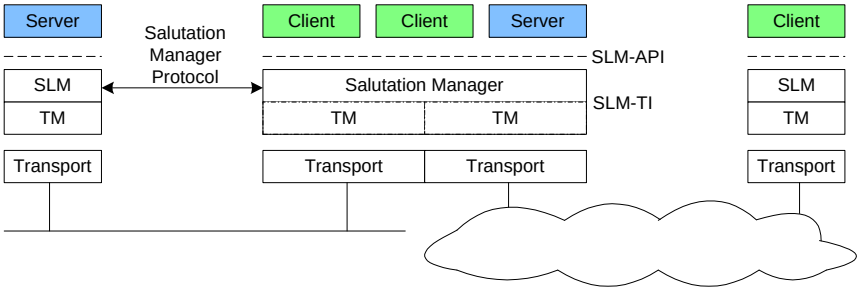

Figure 4.3: *Salutation system model [3].*

functional units.
The SLMs discovers other SLMs and exchanges the service records registered within them. Searching is done by matching type(s) and or set of attributes within the other SLMs of the network. The descriptions of the Functional Units are always stored at the local SLM but parts of the information can be stored at other SLMs for rapid response to queries from clients.
A session is established when a client wishes to access a Functional Unit (service) discovered through an SLM. The communication between a client and a Functional Unit can be of three modes; in the native mode they may use any protocol known to both and the SLM is not involved in the data transfer. In emulated mode, however, the SLM will manage the session and all data will be mediated through the SLM delivering them as messages to either instance. In salutation mode the SLM manages the session and carries the messages as well as defining the message format to be used in the session.

4.2.3 Suitability

Salutation does not address the issue of self-configuration, though with the coming IPv6 some of these features are addressed.
The main advantage of Salutation is its transport-independency. There is a possible solution given for mapping Salutation to Bluetooth SDP given in a white paper by the Bluetooth SIG [10].
Salutation seems to be a very well defined service discovery protocol.

4.3 Salutation Lite

Salutation Lite is not a freestanding Service Discovery Protocol as such, but a reduced version of Salutation mentioned above, see Section 4.2. It addresses some of the key issues of dealing with small hand held devices, such as limited screen size, limited memory, power consumption and bandwidth limitations. Thus, minimizing the footprint of the implementation is of vital importance [13].

Salutation Lite provides means for the server to determine the operating system, processor type, amount of free memory and input/output characteristics of the hand held device as well as screen size and if the screen supports color graphics. With this information it can tailor the interaction accordingly.

4.3.1 Architecture overview

Salutation Lite is not a full Salutation implementation, it does not contain a Salutation Client. The hand held device or Mobile Terminal (MT) never requests capabilities from other devices or applications, it only responds to these requests when received.

4.3.2 Operations overview

An MT enters the system and immediately establishes a communication with the server using Bluetooth's inquiry and page. The server automatically discovers requests the capabilities of the MT using Salutation Lite. The MT responds with a Service Description Record specifying its capabilities. The server parses the SDR and based on the MTs capabilities it can send and install an application designed for the environment. This application drives the user interface and captures any end user input for processing at the server. If there is an application present the server may choose to use that instead. When the MT exits the coverage of the local server or when the session is over or the device is powered off, any application data installed in the device is removed and the device is returned in the state is was before the connection was set up.
Figure 4.4 briefly describes the system model.

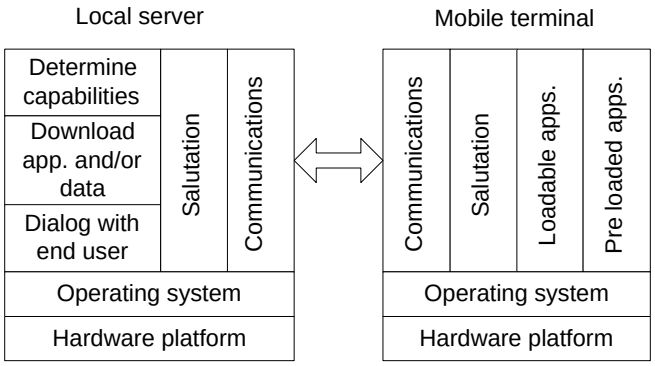

Figure 4.4: *Salutation Lite system model.*

Salutation Lite has been ported to WinCE and Palm OS, in the Windows version it accesses the communication technique (Bluetooth or IrDA) through WinSock calls. By using this higher level interface, Salutation Lite is easily ported to other operating systems [13].

4.3.3 Suitability

Salutation Lite was designed with mobile hand held devices in mind. Screen-, memory-, processor- and bandwidth limitations are all considered in the design.
It is royalty free and offered on an open source model for WinCE, Palm OS and Java. Salutation Lite is suitable for smaller services such as push advertisement and other info services. It requires nothing but Salutation Lite itself to be pre-installed in the MT.
Salutation Lite may be used as a reference model for beginning a full Salutation implementation.

4.4 Jini

Jini, developed by Sun Microsystems, uses Java to incorporate devices and software components into a single distributed system. The term federation is adopted as a collection of devices that may discover each other and cooperate. Jini provides means for service construction, lookup, communication and use in a distributed system [11].
Services are defined via an interface, and the implementation of a proxy supporting the interface that will be seen by the service client will be uploaded into the lookup service by the service provider. This implementation is then downloaded into the client as part of that client finding the service.

4.4.1 Architecture overview

The core of Jini is the three protocols *discovery*, *join* and *lookup* [4].
The major point of contact between the system and users of the system is the lookup service. Services are found and resolved by the lookup service which can be seen as a directory service. Objects in a lookup service may include other lookup services; this provides for hierarchical lookup.
The discovery and join protocols are used when a device enters the system. Discovery is used when a service is looking for a lookup service with which to register. Join is used when a service has located a lookup service and wishes to join it. Lookup occurs when a client or user needs to locate and invoke a service, described by its interface type, and possibly other attributes.

4.4.2 Operations overview

Upon entering the system, a device uses discover and join to locate an appropriate lookup service and join it. Every device must discover one or more lookup service before it can enter a federation. These services may be known beforehand or they may be discovered using a multicast request on the local network for any lookup services to identify themselves [12].
A device will then register itself with the lookup service, when registering it may upload service objects for its services. These service objects contains the Java programming language interface for the service including the methods that users and applications will invoke to execute the service, along with any other descriptive attributes. Jini uses Remote Method Invocation (RMI)

to allow a Java program to call a method in a remotely running Java program. If a device does not support RMI it is possible to use the lookup service to extract information to contact the service without RMI.
A device may then query the lookup service for information on other devices and services within the federation, the lookup service may pass queries on to other lookup services in a hierarchial way. A copy of the service object is then moved to the client and used to access the service using RMI.
In order to keep the lookup services updated with the current state of the network, i.e. which active devices are within it, devices must re-register in certain intervals called leases. Information about devices that exits the network without de-registering will be deleted from the databases after the lease time expires.

4.4.3 Suitability

Jini does not cover the area of self configuration. As in the case with Salutation, IPv6 partially covers this area.
Jini relies heavily on RMI calls but non-Java devices can participate in a federation using bridges and proxies. This solution is not attractive since it introduces a single point of failure in the system (the bridge/proxy).

4.5 Universal Plug and Play

Universal Plug and Play (UPnP), developed by Microsoft, is designed to enable simple ad hoc communication between distributed devices and services. UPnP uses the TCP/IP protocol stack which is virtually ubiquitous. It enables a device to dynamically join a network, obtain an IP address, convey its capabilities, and learn about the presence and capabilities of other devices [14].

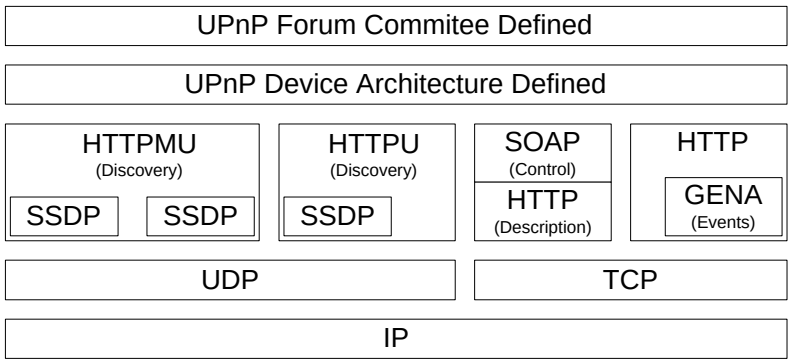

Figure 4.5: *The UPnP protocol stack [14].*

4.5.1 Architecture overview

A UPnP device, as mentioned earlier, uses the TCP/IP protocol suite as transport. It uses Simple Service Discovery Protocol (SSDP) for announcing its presence to other devices as well as discovering services. SSDP can function with or without a directory service. It uses HTTP over unicast and multicast UDP called HTTPU (unicast UDP) and HTTPMU (multicast UDP). The multicast is sent on a reserved channel to which all devices must listen. XML is used to describe devices and services. Control messages are also expressed in XML using Simple Object Access Protocol (SOAP). Event messages also use XML, the messages are formatted using General Event Notification (GENA). UPnP introduces the term control points as potential clients of services.

4.5.2 Operations overview

A device entering the system can use AutoIP to obtain an IP address without explicit administration. It then multicasts a number of discovery messages (ssdp:alive), each corresponding to one of its embedded devices and services to advertise its services to the control points in the network [5]. The messages should include an expiry time for the service or device, if the service or device is still available at the expiry time the advertisement should be resent.
A control point entering the network multicasts a discovery message searching for interesting devices, services, or both. All devices listen to the standard multicast address for these messages and must respond if any of their embedded devices or services match the search criteria in the discovery message. The discovery message may also be directed to a directory service if present. The fundamental exchange in both cases is a discovery message containing a few essential specifics about the device or one of its services e.g. its type, identifier and a pointer (URL) to more detailed information.
After a control point has discovered a device it will use the URL to retrieve more information about the device and/or how to interact with it. The description for a service is expressed in XML and includes a list of any embedded devices or services, as well as a URL for control, eventing and presentation.
A control point then can send actions to a device's service through control messages. The message is sent to the control URL of the device and the device in turn responds with any action-specific values.
A UPnP service description includes a list of actions the service responds to and a list of variables that model the state of the service at runtime. A service publishes updates when these variables change. A control point may subscribe to receive these updates through event messages. Event messages contain the name of one or more state variables and their values.
If a device or service has a URL for presentation; a control point may display this to the user and, depending on the capabilities of the page, allow the user control and or view of the device status. When a device and its services are to be removed from the network, the device multicasts a message (ssdp:byebye) corresponding to each of the discovery (ssdp:alive) messages it multicasted that have not already expired.

4.5.3 Suitability

UPnP addresses self configuration which is lacking in the other protocols.
The service invocation is not fully addressed in UPnP, a control point can obtain a description of a device or service, but it will still need a driver for that specific device since the description alone is not enough to access the service.
The Bluetooth SIG is working on an Extended Service Discovery Profile which uses UPnP over Bluetooth using Bluetooth SDP to discover units with UPnP capabilities. The profile defines how UPnP should run over L2CAP layer and/or over an IP stack using either the Personal Area Network profile or the LAN Access profile.

4.6 Other service discovery protocols

A number of service discovery protocols were not addressed in this chapter but we would like to point some of them out to show what else is being developed in this area.

IETF has released a protocol called Service Location Protocol (SLP) [16]. Developed by Sun Microsystems, it is aimed towards service discovery within an intranet or geographical site, e.g. a user may use SLP to determine whether or not there is a color printer on the same floor. It uses UDP/IP as transport protocols but may use TCP for larger messages. It is scalable to larger networks through its wide area extension (WASRV).

Ninja is the name of a project at University of California, Berkeley. Within this project, a protocol called Secure Service Discovery Service (SSDS) has been developed [15]. It is implemented in Java and uses RMI and XML for service location and description. A strong feature of SSDS is the mandatory security built into the protocol.

Lightweight Directory Access Protocol (LDAP) [17] is an IETF lookup service standard that provides a scalable hierarchy of name spaces, through which services can advertise their presence and clients can locate services. LDAP is an extensible protocol but it does not specify means for a spontaneous discovery, thus making it unsuitable for an application like MediaCell. Furthermore it is not very well suited for devices with limited memory and/or processing capabilities as it, despite its name, is a rather complex protocol.

4.7 Summary

All the fore mentioned protocols provide similar basic services, clients may find relevant services and obtain sufficient information to access the services. This is done through two basic processes, *advertisement* and/or *service request*. One or both of these processes are implemented in all protocols but they differ considerably compared to each other. It is desirable to have both processes implemented since this saves bandwidth. Analysis have shown that purely

advertisement or purely service request is not efficient since it will clog the network [18].
The higher level service discovery protocols share a feature that enables a kind of hierarchical or federated organization. This can be used to improve reliability and improve scalability.
Using a directory service keeps traffic from updating the network down to a minimum, but it may be desirable to have the option of not using a directory service for small networks of low complexity. Some protocols support both modes of operation.
Some of the higher level protocols have already been ported to work over Bluetooth and Bluetooth SDP, others may use the TCP/IP stack or other transport protocols supported by Bluetooth. But smaller devices may not have these protocols implemented, Bluetooth has this option. In that case it, may be necessary to provide a proxy or bridge to the service discovery protocol, which is very undesirable as it introduces overhead and may be a single point of failure for the system.
Security though is an area that very few of the protocols address. Service discovery in general faces a real challenge in security. On one hand the protocol must minimize network usage and user interaction, a service should just work without much of user interaction. On the other hand most services need to be secure. The only protocol that really addresses this topic is Ninja's SSDS.
With all these protocols existing today, the obvious conclusion is that there are far too many of them. Some of them are easily bridged or mapped but it should not be necessary to implement a number of service discovery protocols into a system or use bridges or proxies when the protocols themselves are logically similar. Perhaps a unified standard will emerge in the future much like the Ethernet standard. The top contenders for such a unified standard are Salutation, UPnP and Jini.
For a system like MediaCell, it would be suitable to aim at Salutation Lite since it was developed with small hand held devices in mind. Many manufacturers may incorporate it easily into their products in firmware or small software packets. If not, it is small enough to be downloaded by users. Furthermore it can be extended to a full Salutation implementation and still retain its original features towards devices of lesser complexity. Salutation may then be used throughout the system, making it easier to install new services and devices such as printers etc.

Chapter 5

Simulation of device discovery

In this chapter, we investigate the performance of the device discovery processes, *inquiry* and *page*.
The simulations has been an important part of this thesis, and it is on these simulation results that we have drawn our conclusions. Our results and conclusions have in some cases been confirmed and verified by practical experiments not performed by us, but they are referred to where mentioned.
The main goal with the simulations is to find the minimal average time it takes for a slave to get a connection to a master, while maintaining an acceptable connection loss ratio.

5.1 Simulator

There are two ways of approaching a simulation tool. Either build one, or use an existing one. There are advantages and disadvantages with both methods. The main reason for not building a simulator was the time aspect and the fact that there was a simulator with a bluetooth extension available for free that seemed to fulfill most of our requirements.
We had to debug and make some adjustments in the BlueHoc code, and also implement new features to be able to run the simulations required. We also had to randomize the start times of each device and their frequency hopping sequence. Furthermore SCO channels were implemented in BlueHoc, this was done by ”stealing” one third or two thirds of the packet transmission slots. It is important to keep in mind that the BlueHoc still is under development, and so running the program can be a bit unstable i.e. there are probably still bugs in the code that are not found and corrected yet, but this should not have any impact on our results. A more detailed description of problems encountered and bugs fixed can be found in Appendix B.

5.1.1 Network Simulator

The Network Simulator (NS) is an open-sourced, event-driven simulator targeted at networking research [19]. It simulates most of the network protocols existing, like TCP and UDP, and also supports simulation of routing and multicast protocols over wired and wireless networks.

5.1.2 Network animator

The Network animator (Nam) is a Tcl/Tk based animation tool for viewing network simulation traces and real world packet traces. It is very useful for result analysing and visualization of the simulated scenario. Nam supports topology layout, packet level animation, and various data inspection tools.

5.1.3 Tcl/Tk

Tool command language (Tcl) is an open-source, cross-platform scripting language. It works well with big projects that contains a lot of code. The Toolkit (Tk) is an addition to Tcl, which is a high level toolkit for building graphical applications. Tk works with other languages as well, but is best suited for Tcl.

5.1.4 OTcl

Object Tcl (OTcl) is an extension to Tcl/Tk for object-oriented programming.

5.1.5 System integration

NS is written in C++ and OTcl, this gives us two approaches for manipulating the simulator. One is by changing or adding the C++ code in the Simulator. Detailed simulations of protocols requires this to effectively be able to manipulate bytes, or write additional modules to the simulator. This means debugging, compiling and running. When the simulator is running, a lot of the research work involves running the same simulation under different scenarios, i.e. changing a set of parameters. This is done by using OTcl for writing the scripts, which is the other way of manipulating the simulator; instead of changing the actual C++ code every time a new scenario is simulated, only the parameters in the OTcl-script are changed.
The compiled C++ objects needs to be made available to the OTcl, this is done through an OTcl linkage called Tclcl. Tclcl creates a matching OTcl object for each C++ object. It also makes the control function so that OTcl controls the C++ objects. The Tclcl linking and matching of object is only done for those C++ objects that need to be controlled in the simulation, otherwise the linking is not done.

5.1.6 BlueHoc

BlueHoc Simulator provides a Bluetooth extension to the Network Simulator for simulating Bluetooth scenarios [20]. Like NS it is a C++ based simulator with OTcl interface for configurations, and it uses nam for displaying the simulations graphically. BlueHoc is used to simulate lower layers from the Bluetooth stack. The following Bluetooth layers have been simulated:

- Bluetooth radio.
- Bluetooth baseband.

- Link Manager Protocol.
- Logical Link Control and Adaptation Protocol (L2CAP).

The key issues addressed by BlueHoc are:

- Device Discovery performance of Bluetooth.
- Connection Establishment and QoS negotiation.
- Medium access control scheduling schemes.
- Radio characteristics of Bluetooth system.
- Statistical modeling of the indoor wireless channel.
- Performance of TCP/IP based applications over Bluetooth.

The BlueHoc simulates a Bluetooth piconet consisting of a master and one to seven slaves in connection mode, but it is possible to have more nodes doing inquiry and paging procedures.

5.2 The simulations

By manipulating different timeouts and parameters in the devices one can make the connection process faster or slower, it comes at a price though of an increased or decreased failed connection rate.
There are timeouts in all devices for the case the device is a master and for the case it is a slave. The device that initiates the connection always becomes the master. In the MediaCell system it is desirable to have the access point (AP) as the master to have control of the system. Hence, all the mobile terminals (MT) entering the system should become slaves. The master has no control of the slaves timeout settings when it enters the system, and cannot manipulate them. The only timeouts with which the master can manipulate the connection time are the ones controlled by the master itself, i.e. inquiry timeout and page timeout. These are the parameters we have manipulated to reduce the connection time.
The timeouts used by the slaves in the simulations have been set to their default values as defined by the Bluetooth specification [1].
The inquiry process and the page process have been simulated separately to easier find the best inquiry timeout, $T_{\text{w inquiry}}$, and page timeout, $T_{\text{w page}}$, values, and then the full connection procedure has been simulated.
The simulations for one value of $T_{\text{w inquiry}}$ or $T_{\text{w page}}$ are done for cases when one to seven slaves are being paged at the same time. They are also done for the cases there are no SCO channel present, when one SCO channel is present and when two SCO channels are present. In case there is one SCO channels present, only six additional slaves can get connected and in the case there are two SCO channels present, only five additional slaves can get connected.
When changing the number of SCO channels, the specification states [1, pp 101 and 109] that

the number of times the master sends the inquiry and page, called repetition number (N_{inquiry} and N_{page}), trains shall change with it. We have run the simulations, changing the repetition number according to the specification, but also trying other values of N_{inquiry} and N_{page}.
The inquiry and page simulations are both run until the confidence interval of the mean response time is less than 5 % of the value itself. Running the inquiry and page simulations give the foundation for running the mean time to connect (MTTC) simulations.
All simulations are made in an error free environment, i.e. all errors due to packet loss have been removed. Packet collisions can still occur.

5.3 Performance of the inquiry process

Since the inquiry process is a broadcast process, there is the need for a random backoff after receiving an inquiry message to minimize the probability of packet collision, see Section 3.1. Therefore, inquiry must achieve frequency synchronization twice, which together with a long random backoff can make inquiry a time consuming process. The time an inquiry takes is measured from the time the inquiring device starts the inquiry until the responding device has responded with its second ID packet.
$T_{\text{w inquiry}}$ can be set in increments of 1.28 s, hence the resolution of all graphs regarding the inquiry process is 1.28 seconds.
Simulations have been done with up to seven units in range of the inquiring device. The diagrams are all (unless specifically marked otherwise) averages over 1 – 7 devices in range. Note that the simulator has been configured to always run the length of the inquiry window regardless of whether the inquiring device has found all devices in range or not.

5.3.1 Percentage of slaves not responding to an inquiry

Due to the random nature of the inquiry process, some slaves may not respond to the inquiry. It is obvious that this is the case with shorter inquiry windows. The Bluetooth specification states that $T_{\text{w inquiry}} = 10.24$ s ensures at least one response from all devices in range of the inquirer [1, p 108] (no SCO links present, error free environment).

No SCO links present

The most common case is when there is no SCO link present (Figure 5.2). The inquiring device has with the default $T_{\text{w inquiry}} = 10.24$ s a loss probability of 0 % i.e. no loss has been measured, which is according to the specification. However, an acceptable loss ratio is obtained already with the 5.12 s window. The loss ratio is 1.2 % for $T_{\text{w inquiry}} = 5.12$ s. For the 7.78 s window, the loss ratio is as low as 0.3 %. The greatest performance improvement is made with an increase from $T_{\text{w inquiry}} = 2.56$ s to $T_{\text{w inquiry}} = 3.84$ s where the loss ratio drops from 51 % to 13 % (Figure 5.2). This sudden increase is due to the fact that after 2.56 seconds, the inquiring device will have had time to complete both the A and B trains once, thus covering all the inquiry frequencies. This is made visible in the histogram of the simulated inquiry responses. As shown in Figure 5.1, the histogram consists of two bell-shaped curves centered around 1.28 and 3.84

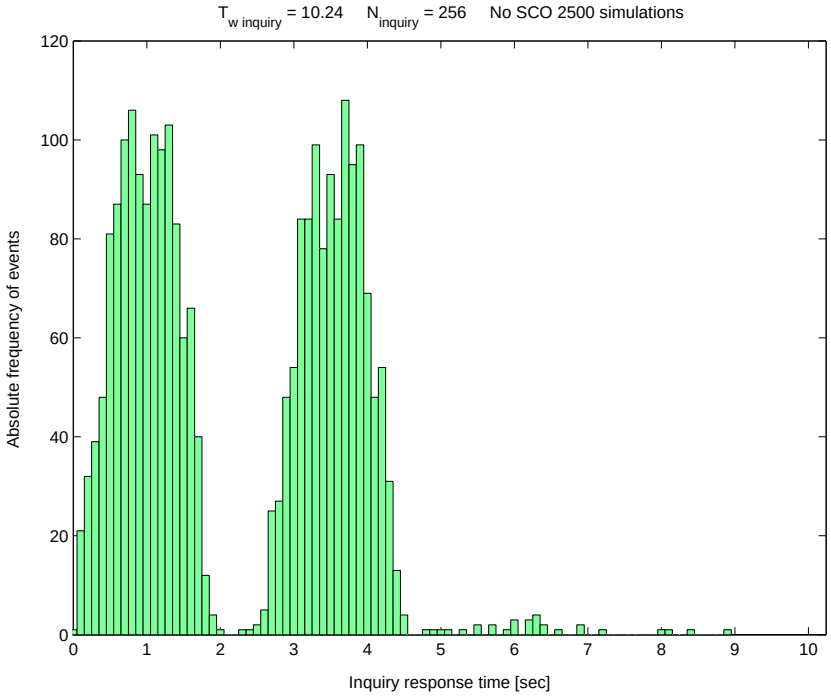

Figure 5.1: *Histogram for the average inquiry response time (first response).*

seconds. This is due to the repetition number N_{inquiry} set to 256. Since every train takes 10 ms to transmit and 256 trains of the same type are transmitted before changing to the next train, the trains changes every 2.56 seconds. This is discussed in greater detail in Section 5.3.3.

One SCO link present

In case there is one SCO link present at the inquiring device, the inquiry process will consume more time and the loss ratio will subsequently increase. The specification states [1, p 109] that the repetition number (N_{inquiry}) should be doubled to $N_{\text{inquiry}} = 512$ if one SCO link is present. In order to run through at least 3 train switches as stated by the specification, $T_{\text{w inquiry}}$ must also be doubled. If the default values are used, this would result in $T_{\text{w inquiry}} = 20.48$ s, the corresponding loss ratio will be 0.5 % (Figure 5.2), but the inquiry will be somewhat inefficient i.e. is too long.
Our studies have shown that with $N_{\text{inquiry}} = 512$ an acceptable loss ratio of 1.3 % is obtained with $T_{\text{w inquiry}} = 15.36$ s.
The loss ratio has its greatest performance improvement between $T_{\text{w inquiry}} = 7.68$ s and $T_{\text{w inquiry}} = 10.24$ s where the loss ratio drops from 51 % to 12 % (Figure 5.2). At $T_{\text{w inquiry}} = 10.24$ s, the inquiring device will have had time to complete both trains once, i.e one train switch, thus covering all inquiry frequencies.
The loss ratio increases with approximately 10 percentage points between the no SCO case and

one SCO case for 1.28 $s \leq T_{\text{w inquiry}} \leq 2.56$ s. The increase is bigger, between 40 – 50 percentage points for 3.84 $s \leq T_{\text{w inquiry}} \leq 7.68$ s. For 7.68 $s \leq T_{\text{w inquiry}} \leq 15.36$ s, the difference in loss ratio for the one SCO case drops from 50 to 1 % and for $T_{\text{w inquiry}} \geq 20.48$ s, it is almost 0. To have a loss ratio corresponding to the no SCO case, the inquiring device must increase $T_{\text{w inquiry}}$ with a factor ≈ 3, for $T_{\text{w inquiry}} \geq 20.48$ s there is no need for any increase. Note that the one SCO loss ratio curve has a flat section around 3.84 $s \leq T_{\text{w inquiry}} \leq 7.68$ s. This flattening of the curve results in a situation where a moderate increase of $T_{\text{w inquiry}}$ will not result in any improvement of performance.

Two SCO links present

In the case where two SCO links are present at the inquiring device, which is the maximum number of SCO links allowed while simultaneously performing inquiry (or page), it will result in a further deterioration of performance. The specification states that N_{inquiry} should be doubled compared to when there is one SCO link present or trebled compared to when there are no SCO links present [1, p 109]. If the default values are used, this would result in $T_{\text{w inquiry}} = 40.96$ s and in a loss ratio of approximately 3.4 % (Figure 5.2).
The loss ratio is 13 – 55 percentage points higher than in the one SCO case and 25 – 75 percentage points higher than in the no SCO case.
To have a loss ratio corresponding to the no SCO and one SCO cases, the inquiring device must increase $T_{\text{w inquiry}}$ with a factor ≈ 7 to 9 or ≈ 2 to 4 for the two cases, respectively. However, the two SCO loss ratio curve, as in the one SCO case, has a flat section around 10.24 s $\leq T_{\text{w inquiry}} \leq 15.36$ s. This flattening of the curve results in a situation where a moderate increase of $T_{\text{w inquiry}}$ will not result in any improvement of performance.

5.3.2 Inquiry response time

The average time to respond to an inquiry is only interesting in the case a device responds for the first time. During an inquiry process, devices will continue to respond after they have responded the first time because they will return to the previous state which is Inquiry Scan substate.

No SCO links present

The most common case is when there are no SCO links present. In the no SCO case the curve levels at 2.27 s for $T_{\text{w inquiry}} = 5.12$ s. An increase of the inquiry window beyond this point will result only in an efficiency loss; an inquiring unit may stay in Inquiry substate for a longer time than necessary, only collecting duplicate responses from units that have already responded. A small variation, of -0.0566 s to $+0.1055$ s, which corresponds to 2.5 and 4.6 % of the value, can be seen in the flat section of the curve (Figure 5.3).
Between $T_{\text{w inquiry}} = 2.56$ s and $T_{\text{w inquiry}} = 3.84$ s there is a very big increase in the mean response time ($\mathbf{E}[t_{\text{inq resp}}]$). $\mathbf{E}[t_{\text{inq resp}}]$ approximately doubles from $\mathbf{E}[t_{\text{inq resp}}] = 0.97$ s to $\mathbf{E}[t_{\text{inq resp}}] = 1.97$ s. This is due to the vast decrease of the loss ratio. When $T_{\text{w inquiry}} \geq \mathbf{E}[t_{\text{inq resp}}]$, a majority of the devices in range will reply, driving the mean response time higher. In the former

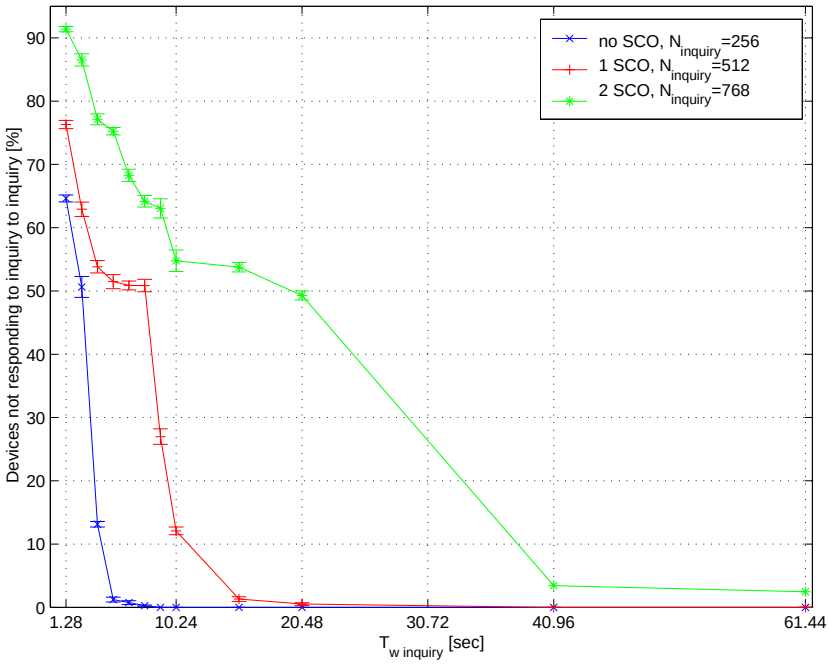

Figure 5.2: *Probability of a slave not responding with no, one or two SCO links present.*

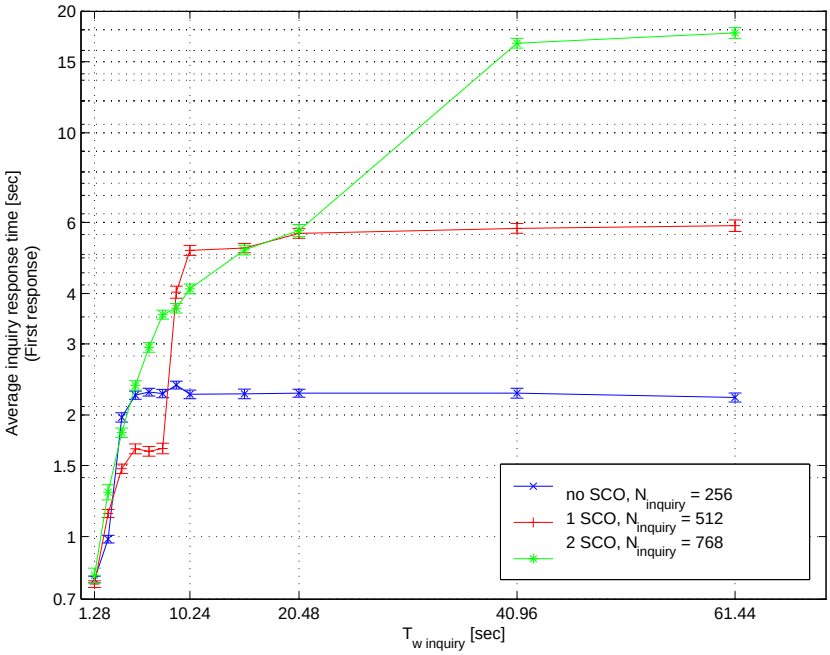

Figure 5.3: *Average inquiry response time (first response).*

cases, only the ”faster” units will have time to reply, hence $\mathbf{E}[t_{\text{inq resp}}]$ will be smaller but the loss ratio will be much bigger. This is because the inquiring device has had time to send both inquiry trains once, covering all inquiry frequencies.
When using the default settings of the inquiry process, 95 % of the devices should have responded in 4.16 s; in 5.39 s 99 % of the devices should have responded.

One SCO link present

Given that there is one SCO link present, the curve levels at 5.78 s. This corresponds to an increase of 154 % compared to the no SCO case. The curve flattens from $T_{\text{w inquiry}} = 10.24$ s after which any increase of $T_{\text{w inquiry}}$ only results in minor time variations or none at all (Figure 5.3). The time variations ranges between -0.1412 s and $+0.1170$ s, which are 2.4 and 2.0 % of the final value, respectively (Figure 5.3).
The biggest change of $\mathbf{E}[t_{\text{inq resp}}]$ occurs in the interval 7.68 s $\leq T_{\text{w inquiry}} \leq$ 10.24 s, where $\mathbf{E}[t_{\text{inq resp}}]$ increases from 1.65 s to $\mathbf{E}[t_{\text{inq resp}}] = 5.12$ s. As in the no SCO case this is due to the decrease in the loss ratio, caused by the train repetition number.

Two SCO links present

In the two SCO link case, the curve flattens at 17.18 s. This correspond to an increase of 657 % compared to the no SCO case or 154 % compared to the one SCO case. The curve flattens from $T_{\text{w inquiry}} = 15.36$ s after which any increase of $T_{\text{w inquiry}}$ results in no or very small time variations (Figure 5.3). The variation ranges between -0.50 s and $+0.50$ s, which corresponds to 2.9 % of the mean value for the flat part of the curve (Figure 5.3).
The two SCO case curve has a smoother slope than the other curves with no sudden leaps until it flattens at 40.96 seconds.

5.3.3 Improving the loss ratio and average time to respond

The 32 frequencies of the Inquiry Hopping Sequence is subdivided into two trains, A and B, containing 16 frequencies each. Every train of 16 frequencies takes 10 ms to transmit, using the fast frequency hopping scheme (3200 hops per second). The specification states that the repetition number, N_{inquiry}, shall be set to 256. Hence the time it takes before the inquiry frequencies change from those in one train to the other will be 2.56 seconds. If a device is in Inquiry Scan substate and listening to one of the frequencies in e.g. train A and the inquiring device is transmitting frequencies from train B, it will take a minimum of 2.56 seconds before frequency synchronization will occur. After the initial synchronization, the scanning device will backoff for a random backoff time of max 0.64 s and respond to the inquirer after a second frequency synchronization.
As shown in Figure 5.1 the distribution of the times to respond to an inquiry is roughly bell-shaped around the two values of 1.28 and 3.84 seconds. By reducing the repetition number, a device will need less time to cover all frequencies in the Inquiry Hopping Sequence. Thus the $T_{\text{w inquiry}}$ can be reduced without an increase in loss ratio. Also since more frequencies are covered in a shorter time, the average response time will decrease. Figure 5.4 shows a histogram of

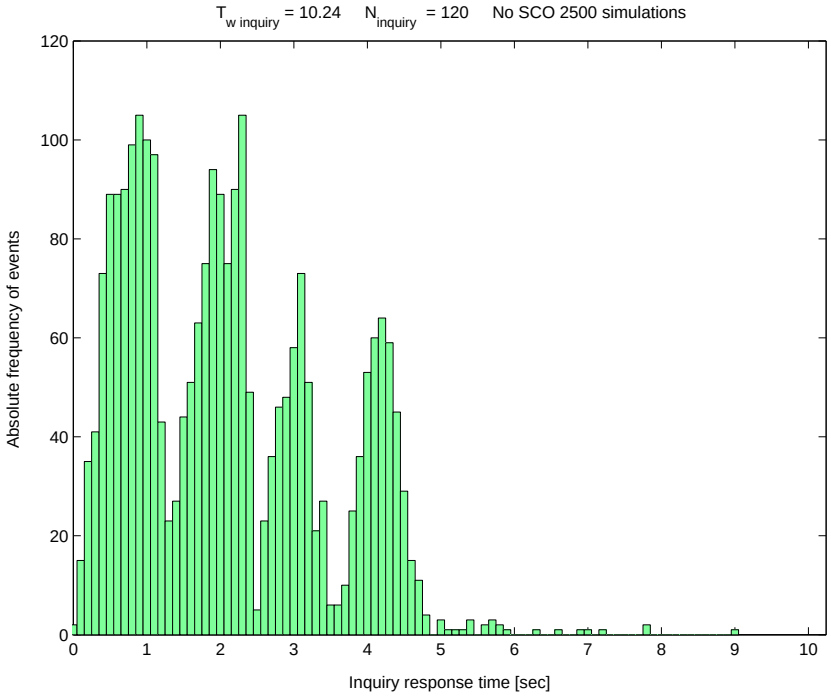

Figure 5.4: *Histogram for the average inquiry response time* $N_{\text{inquiry}} = 120$.

2500 simulated inquiries, note that due to the reduced N_{inquiry} there are four distinct bell-shapes separated by approximately 1.2 seconds which is the time it takes for the trains to switch when $N_{\text{inquiry}} = 120$.

No SCO links present

Simulations with N_{inquiry} less than the default value were performed on the no SCO case to see if the mean inquiry response time could be improved. There should be a slight improvement of the performance and indeed there was, see Figures 5.5 and 5.6.
A number of N_{inquiry} was tested and $N_{\text{inquiry}} = 120$ proved to be the optimal N_{inquiry} for the no SCO case. The mean response time for an inquiry is at its lowest value for this setting.
The mean response time for $N_{\text{inquiry}} = 120$ is $\mathbf{E}[t_{\text{inq resp}}] = 2.08$ s compared to 2.27 s for $N_{\text{inquiry}} = 256$, which is an improvement by 8 %.
The loss ratio should also improve. Our studies have shown that with $N_{\text{inquiry}} = 120$, an acceptable loss ratio will be obtained with $T_{\text{w inquiry}} = 5.12$ s; in that case the loss ratio will be 1.0 %. Should this be insufficient, $T_{\text{w inquiry}} = 7.68$ s will result in a loss ratio of 0.3 %.
When using the above mentioned settings of the inquiry process, 95 % of the devices should have responded in 4.36 s; in 4.88 s 99 % of the devices should have responded.

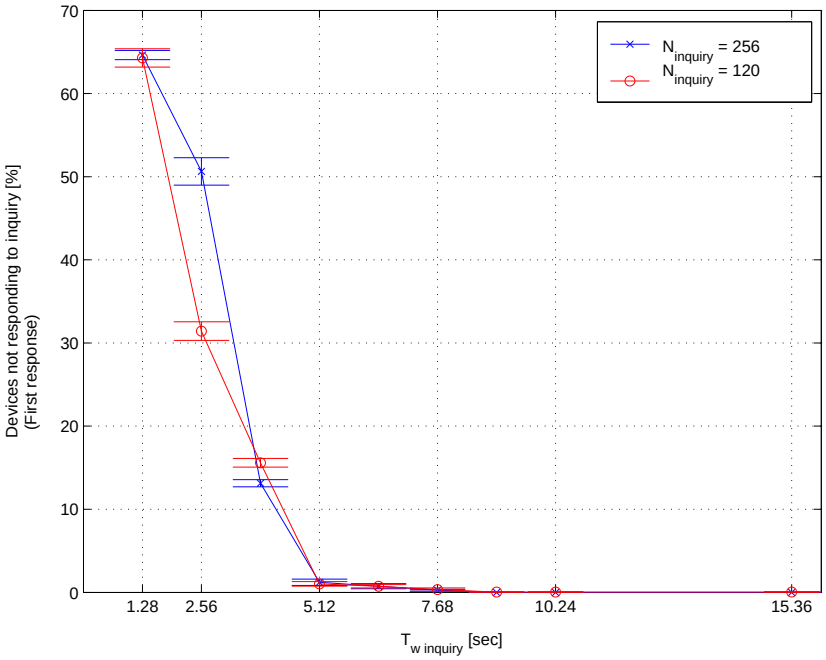

Figure 5.5: *Improvement in loss ratio, no SCO links present.*

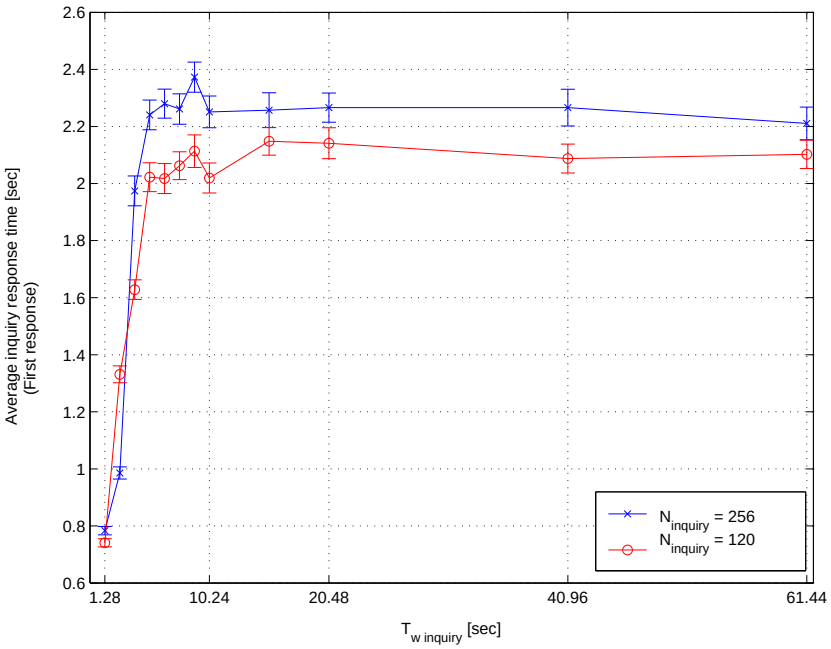

Figure 5.6: *Improvement in response time, no SCO links present.*

One SCO link present

Simulations performed with N_{inquiry} less than the default value were also performed on the one SCO channel case. As N_{inquiry} decreases, the number of devices not responding to an inquiry decreases, there is also a slight decrease of the mean time to respond to an inquiry (Figure 5.7 and 5.8). Interestingly, the optimal N_{inquiry} for the one SCO case is the same as for the no SCO case, $N_{\text{inquiry}} = 120$.
Our studies have shown that with $N_{\text{inquiry}} = 120$ an acceptable loss ratio of 2.7 % is obtained with $T_{\text{w inquiry}} = 8.96$ s. Should this be insufficient, $T_{\text{w inquiry}} = 15.36$ s will result in a loss ratio of approximately 0 %.
In the event of one SCO link present and $N_{\text{inquiry}} = 120$, the mean time to reply to an inquiry will be $\mathbf{E}[t_{\text{inq resp}}] = 3.33$, s which is an improvement of 42 % compared to the default setting of $N_{\text{inquiry}} = 512$.
The curve flattens from $T_{\text{w inquiry}} = 15.36$ s, after which any increase of $T_{\text{w inquiry}}$ only results in minor time variations or none at all (Figure 5.8).

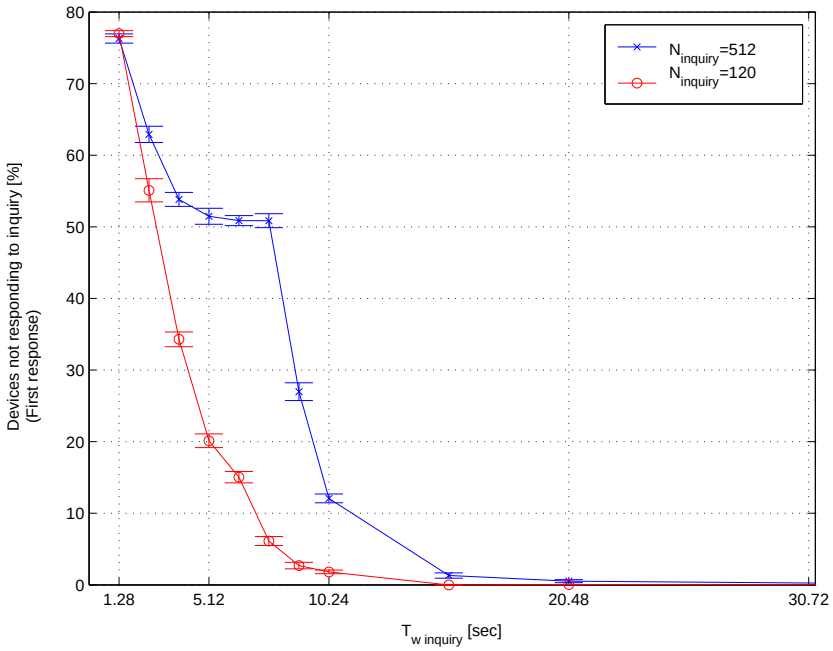

Figure 5.7: *Improvement in loss ratio, 1 SCO link present.*

Two SCO links present

When two SCO links are present, the N_{inquiry} should be somewhat bigger than the previous cases. We have found that $N_{\text{inquiry}} = 256$ is the optimal value for the two SCO case. An acceptable loss ratio will be obtained with $T_{\text{w inquiry}} = 40.96$s; the loss ratio will be 4.6 %. Should this be insufficient, $T_{\text{w inquiry}} = 61.44$ s will result in a loss ratio of 2.1 %.

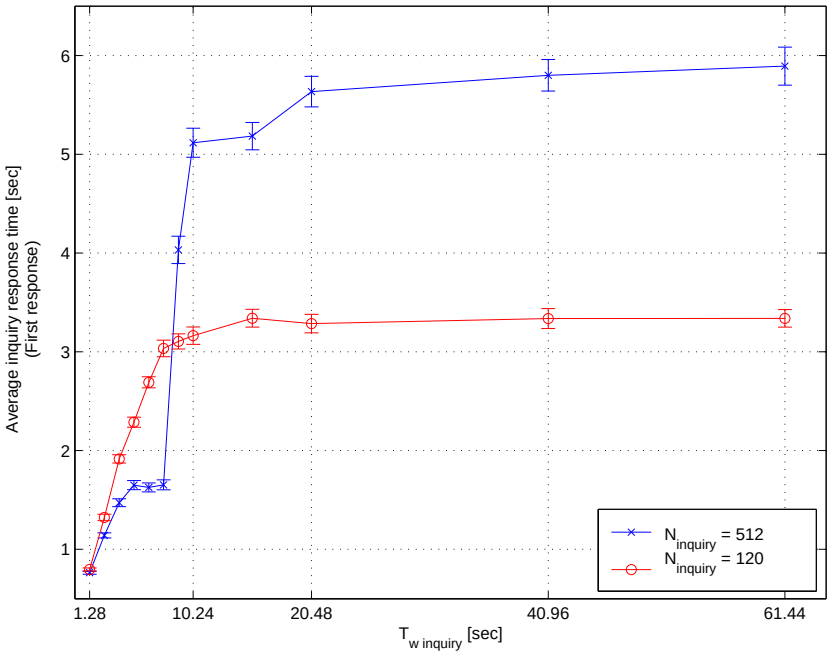

Figure 5.8: *Improvement in response time, 1 SCO link present.*

This is very similar to the default values, but the mean time to reply to an inquiry for $T_{\text{w inquiry}} = 40.96$ s will be $\mathbf{E}[t_{\text{inq reps}}] = 11.84$ s when $N_{\text{inquiry}} = 256$. This corresponds to an improvement of 31 % compared to the $N_{\text{inquiry}} = 768$ case .
An average response time of $\mathbf{E}[t_{\text{inq resp}}] = 11.84$ s is far too high though, it will mean an unacceptable loss of bandwidth for other users that might already have connected to the Access Point.

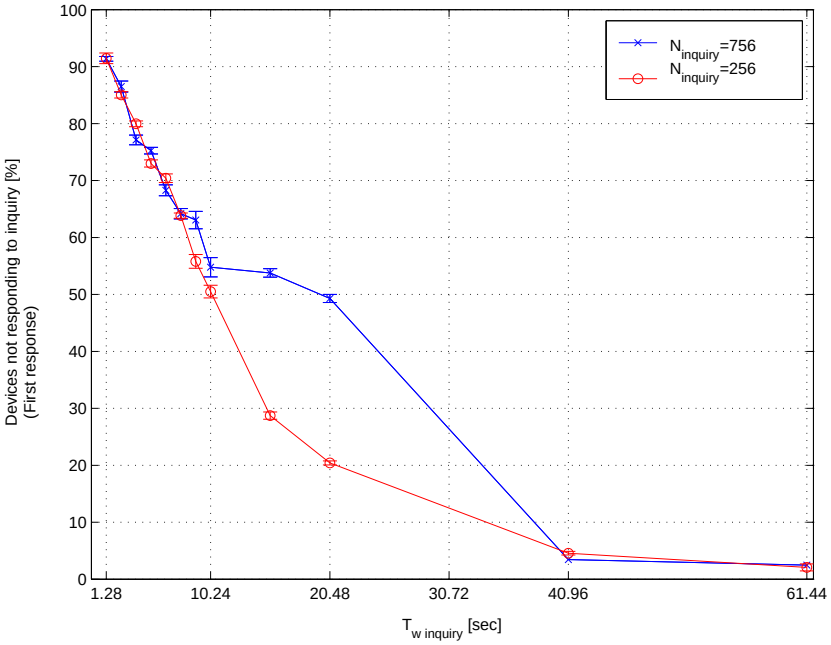

Figure 5.9: *Improvement in loss ratio, 2 SCO links present.*

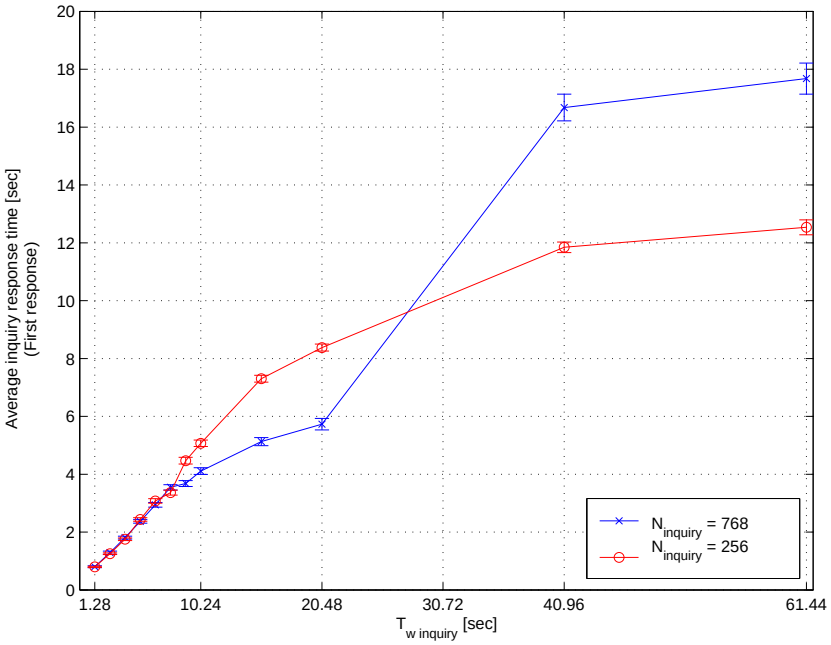

Figure 5.10: *Improvement in response time, 2 SCO links present.*

5.3.4 Discussion of results regarding the inquiry process

As we have shown there is time to be won in the inquiry process by paying attention to some simple timeouts and parameter settings of Bluetooth. All these changes can be made without altering the hardware or firmware, it is only in the software where changes need to be performed. The designers of Bluetooth have prioritized acquiring responses from all devices in range and not so much a short response time. Furthermore, devices will continue to respond to an inquiry after the initial response and thus making longer inquiry inefficient and producing more interference.

Because of the initial frequency and phase discrepancies between units there will, for shorter inquiry windows ($T_{\text{w inquiry}} \leq 10.24$ s), always be a probability of a device not responding to an inquiry. The specification states [1, p 108] that with $T_{\text{w inquiry}} \geq 10.24$ s all devices in range will have responded at least once. Our simulations show that this is indeed the case, in fact 97 % of the slaves will have responded already after 5.12 s. In case that the inquiring device has one SCO channel in use, $T_{\text{w inquiry}} \geq 15.36$ s ensures that all units in range will most probably respond. In the case of two SCO channels in use by the inquiring device, $T_{\text{w inquiry}} \geq 40.96$ s will ensure responses from all devices. The specification states that N_{inquiry} shall be doubled in the case there is a SCO channel present or tripled if there are two SCO channels present. This means that also $T_{\text{w inquiry}}$ should be doubled ($T_{\text{w inquiry}} \geq 20.48$ s) or tripled ($T_{\text{w inquiry}} \geq 61.44$ s) in the cases with one or two SCO links present.

In a system like MediaCell (Section 2.1), where a user can walk through an APs coverage in less than 15 – 20 seconds, the inquiry response time is of vital importance. In Figure 5.3, the mean inquiry response time for the first response is shown for increasing $T_{\text{w inquiry}}$. The curves level at 2.27, 5.78 and 17.18 seconds for no, one and two SCO channels present, respectively. This corresponds to a 154 % increase in response time with one SCO channel present at the inquiring device's side or a 657 % increase in the case with two SCO channels present. This shows the enormous impact of the SCO channels on the performance of the inquiry process. In the case of one and two SCO links present, a data traffic user will experience an unacceptable loss of bandwidth since the inquiry process takes priority over data traffic. All data traffic will be suspended for an average of 5.78 s in the one SCO case or as much as 17.18 s in the two SCO case. Then, there is the possibility of a subsequent page process which will suspend the data traffic further, see Section 5.4. A possible solution to this problem is to have a dedicated control Bluetooth chip in every Access Point to handle all inquiry and page as well as other control traffic.

With the optimized N_{inquiry} values, the average time to respond to an inquiry is reduced, especially in the cases where SCO links are present.

The curves level at 2.08 s, 3.33 s and 11.84 s for no, one and two SCO channels present, respectively. This corresponds to an improvement of the mean inquiry response time, $\mathbf{E}[t_{\text{inq resp}}]$, with 8 %, 42 % and 31 % for the no SCO, one SCO and two SCO cases, respectively. Still, there is a 59 % increase in response time with one SCO channel present at the inquiring device's side or a 469 % increase in the case with two SCO channels present compared to the no SCO case. In the case of two SCO links present, a data traffic user will still experience an unacceptable loss of bandwidth. All data traffic will be suspended for an average of 11.84 s, which is insufficient

for most applications.
Oliver Kasten and Marc Langheinrich [7] have experimentally established the average inquiry response time to 2.221 s which is 2.2 % less compared with our simulated result of 2.27 s. It still falls within the confidence interval of our simulated results.

5.4 Performance of the page process

The page process has a slightly longer packet sequence, but needs only to performe one frequency synchronization. It also does not need to backoff since it is a specific device that is being paged. Furthermore, the page process can use the clock estimate (CLKE) of the paged device to further speed up the process. Hence, the page process is a much faster process than the inquiry process. Should though the CLKE not exist or be incorrect, the page process can be quite long or fail completely.
The timing for the page simulations are measured from when the master sends the first ID packet to the slave to when the first traffic packet is sent from the slave to the master.
The number of slaves trying to page are analyzed in two cases. In the first case, it is assumed that only one slave is being paged. In the other case, all the other slaves except the first one are looked upon as one big group. The response times for when two slaves are being paged, and when three slaves are being paged etc, are added together and a mean time is calculated. The same is done for the loss ratio. This is done because the one slave case the is most likely one to occur in the MediaCell system, and it may be easier to see the difference between one and multiple slaves when only comparing two cases rather than five to seven cases.

5.4.1 Percentage of slaves not responding

No SCO link present

Analyzing the most usual case, the one where only one slave is being paged and no SCO channels are present, there is a big improvement in successful connections between page time 0.64 s and 1.28 s, while the loss ratio sinks from 47 % down to 8 % in the latter case. Then there is a slight drop of 1 % down to 7 % for $T_{\text{w page}}$ = 2.56 s. In the default case, which is $T_{\text{w page}}$ = 5.12 s, the loss ratio is 3 % and it is not improving even if $T_{\text{w page}}$ is increased (Figure 5.11).
Having a look at the cases when more then one slave is being paged, there is a big improvement between $T_{\text{w page}}$ = 0.32 s and 0.64 s. The average drop in the average loss ratio is 25 %; however, the average of a page failing would still be 47 %, so the more important improvements are those between $T_{\text{w page}}$ = 1.28 s and 2.56 s: The loss ratio drops by 19 percentage points from 30 % down to 11 % (Figure 5.13).
Note that for $T_{\text{w page}}$ = 5.12 s, the average loss ratio is the lowest, the conclusion must be that there is no improvement when increasing the page window higher than 5.12 s.

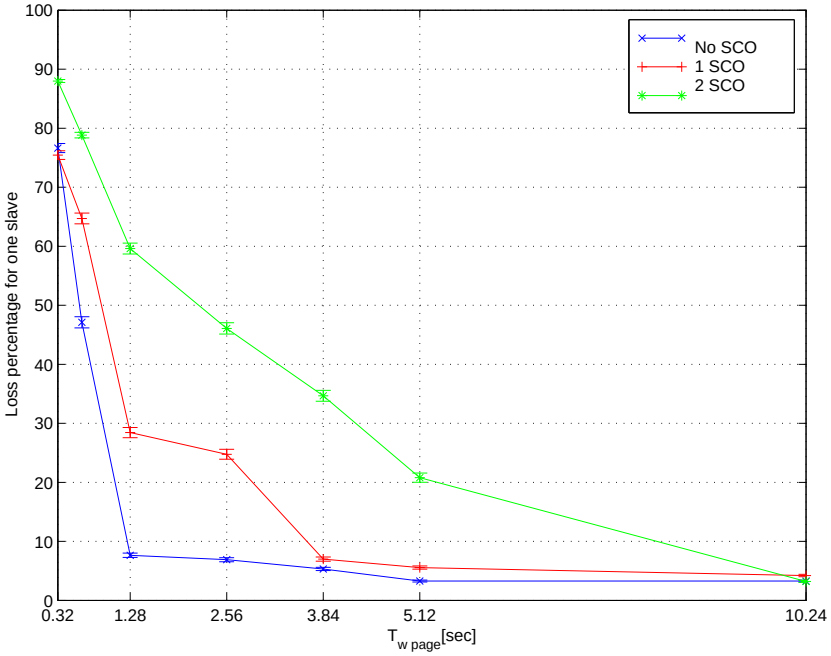

Figure 5.11: *Page loss ratio for one slave.*

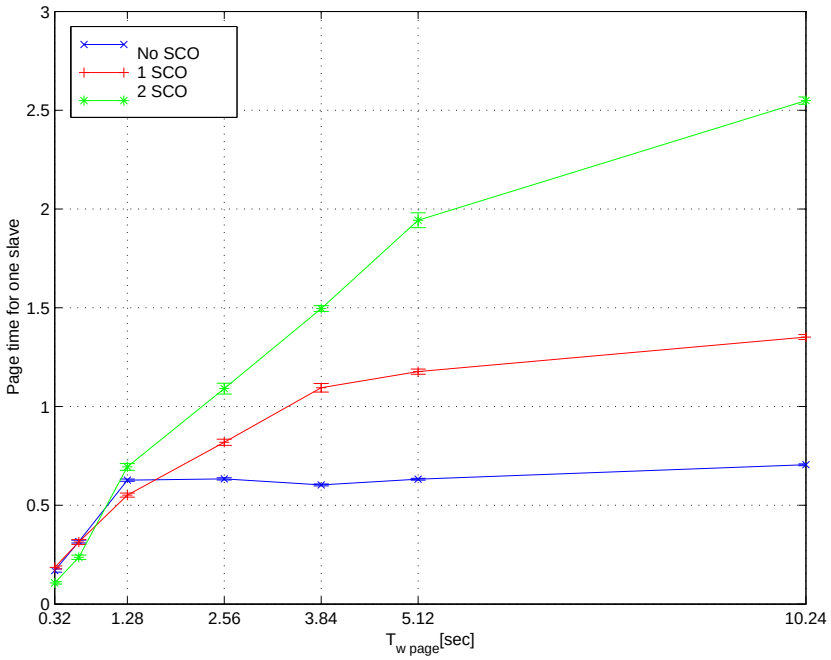

Figure 5.12: *Average page response time for one slave.*

One SCO link present

First consider the case in which only one slave is being paged with one SCO channel present. There is a big improvement of the loss ratio between 0.64 s and 1.28 s, namely a drop from 65 % down to 28 %. Looking at the percentage, this is the biggest improvement; however, the loss ratio is still quite high, so the most significant improvement is the one from 25 % for $T_{\text{w page}}$ = 2.56 s down to 6 % for $T_{\text{w page}}$ = 5.12 s (Figure 5.11). When the page window is 3.84 s the loss ratio is 8 %, and that may be enough for some systems.
For the cases when more than one slave is being paged, the big improvements of the loss ratio happens between the same page windows as in the one slave case: from 61 % when $T_{\text{w page}}$ = 0.64 s down to 37 % when $T_{\text{w page}}$ = 1.28 s, which is a drop of 24 percentage points. And from there to $T_{\text{w page}}$ = 5.12 s, the drop is only 7 percentage points, which gives an average loss ratio of 17 %. To get down to a reasonable loss ratio, the page window has to be at least 10.24 s, where the loss ratio is 9 % (Figure 5.12).

Two SCO links present

Looking at the results of the simulations when only one slave is being paged with two SCO channels present, it is notable that there are no big drops in the loss ratio between each $T_{\text{w page}}$, they are spread out with an almost consistent interval (Figure 5.11).
The loss ratio drops from $T_{\text{w page}}$ = 0.32 s down to $T_{\text{w page}}$ = 10.24 s are made with equals steps of around 12 percentage points.
The loss ratio does not start to reach acceptable values until around $T_{\text{w page}}$ = 8.96 s, where it is 5 %. It still drops 2 percentage points for $T_{\text{w page}}$ = 10.24 s, where it is 3 %.
When studying the cases when more than one slave is being paged with two SCO channels present, it shows that the average failed connection drop is quite consistent here as well. The biggest drop is 17 percentage points between $T_{\text{w page}}$ = 1.28 s and 2.56 s. The loss ratio for $T_{\text{w page}}$ = 1.28 s is 46 %. The lowest loss ratio of 14 % is to be found at $T_{\text{w page}}$ = 8.96 s, but there is hardly any change when increasing the page window to 10.24 s (Figure 5.12).

5.4.2 Page response time

No SCO links present

Analyzing the case where only one slave is being paged and no SCO channels are present, the only big difference in response time when increasing the page window occurs between $T_{\text{w page}}$ = 0.64 s and 1.28 s. The time nearly doubles from 0.32 s to 0.63 s. There is little increase in the response time between page windows up until $T_{\text{w page}}$ = 10.24 s, where it is 0.71 s (Figure 5.13).
In the cases where more than one slave is being paged, the increase in response time between the different page windows are fairly consistent up until $T_{\text{w page}}$ = 2.56 s. Then, after 2.56 s, the average response time is around 2.4 seconds (Figure 5.14).

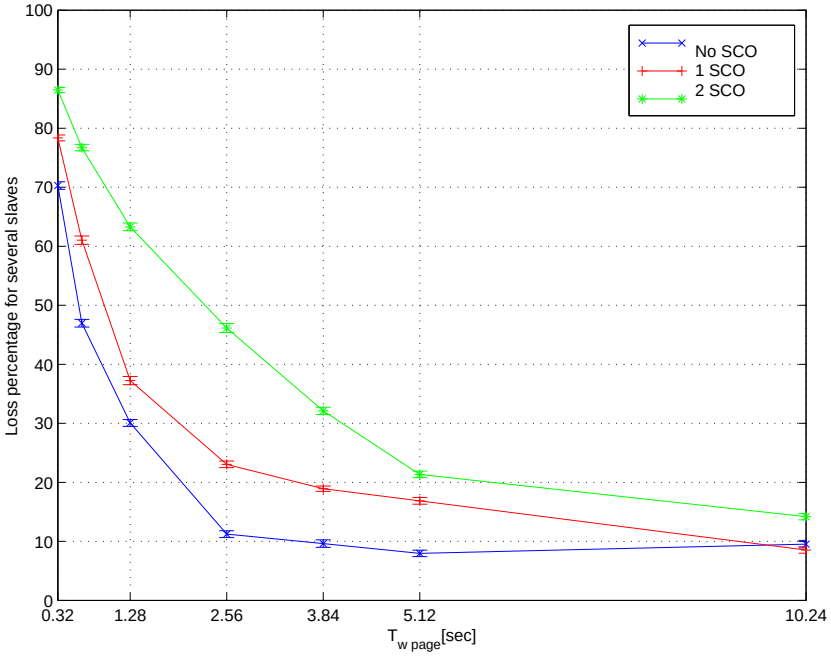

Figure 5.13: *Page loss ratio for multiple slaves.*

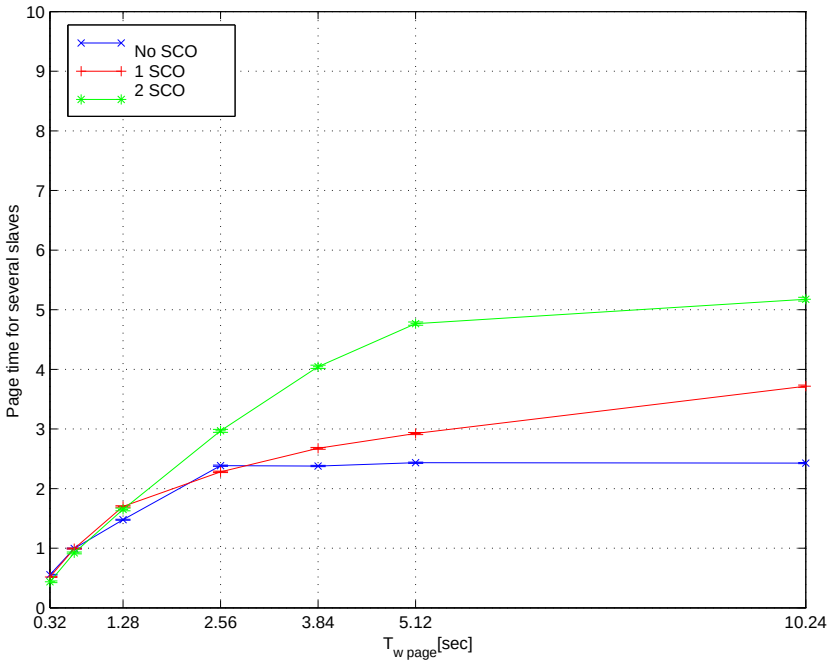

Figure 5.14: *Average page response time for multiple slaves.*

One SCO link present

Studying the results of the simulations when only one slave is being paged with one SCO channel, the fastest response times are 0.31 s for $T_{\text{w page}}$ = 0.32 s and 0.55 s for $T_{\text{w page}}$ = 0.64 s. The response time increases to 0.82 s for $T_{\text{w page}}$ = 2.56 s, and to 1.1 s for $T_{\text{w page}}$ = 3.84 s, and then it increases marginally when page window becomes even larger (Figure 5.13).
In the cases when more than one slave is being paged, there is a steady increase in the response time all the way from $T_{\text{w page}}$ = 0.32 s to 10.24 s. For $T_{\text{w page}}$ = 0.32 s, the response time is 0.52 s, and for $T_{\text{w page}}$ = 10.24 s, it is 3.72 s; in the MediaCell system, the latter value can be considered quite sufficient (Figure 5.14).

Two SCO links present

For the one slave case, the response time is below 1 second for all page windows up to 2.56 s, where it is 1.1 s. However, the loss ratio for these page windows is simply too high to be sufficient for any system. When $T_{\text{w page}}$ = 5.12 s, where the response time is 1.94 s, the loss ratio starts to get acceptable. For $T_{\text{w page}}$ = 8.96 s, the response time is 2.44 seconds, and it increases to 2.55 s when $T_{\text{w page}}$ = 10.24 s (Figure 5.13).
In the cases when more than one slave is being paged, an interesting thing to study is the response time for larger page windows. A big problem with two SCO channels and multiple slaves being paged is the high error rate, and on top of that, the page process will take quite a long time. For $T_{\text{w page}}$ = 3.84 s the response time is 4.04 s, while for 5.12 s it is 4.77 seconds. When increasing the page window to 10.24 s, the response time goes up by 0.51 s to 5.18 s (Figure 5.14). This is quite a long time for a page, and the loss ratio is above 14 %, and so this might not be quite sufficient enough. The two SCO multiple slave cases are extreme, and if it is likely that this occurs frequently, more AP's are needed to handle the traffic.

5.4.3 Discussion of results regarding the page process

Reflections shall be made to find the best page window for different kinds of performance levels, i.e. the gain and loss of increasing or decreasing the page window with regards to the percentage of failed pages and the page response time. Considerations shall be taken on how many SCO channels that are present, and how many slaves that are being paged. The most important case to consider is when only one slave is being paged. The default value for the page window is 5.12 s.
When looking at the results, it is noticeable to see that for $T_{\text{w page}} \leq 0.64$ s, the loss ratio is generally greater than 50 %. This should not be sufficient for any kind of system. The MediaCell system requires a successful connection rate of minimum 95 %, so all rates below that are not acceptable. This means that the loss ratio should be kept a fair bit below 5 % to allow some errors in the inquiry process. Also, the page process time should not be longer than 5 seconds in order to complete the connection within a reasonable time.
In the case of no SCO channels and only one slave being paged, an acceptable loss ratio is

obtained for page window 3.84 s. The loss rate is a bit lower for $T_{\text{w page}}$ = 5.12 s and there is no increase in response time. For the multiple slave case, the loss rate is marginally lower for $T_{\text{w page}}$ = 3.84 s than for $T_{\text{w page}}$ = 5.12 s, which suggests that the page window should be kept between 3.84 s $\leq T_{\text{w page}} \leq$ 5.12 s. The response times for the page process in all of these cases are 2.5 s or below, which is acceptable.
When one SCO channel is present, the page windows for when the loss ratio is acceptable for the default value, i.e. $T_{\text{w page}}$ = 5.12 s and above. However, for $T_{\text{w page}}$ = 5.12 s, it is just around 5 %, and that leaves room for failure in the inquiry procedure. Hence, it might be wiser to use $T_{\text{w page}}$ = 10.24 s. For the multiple slave case, the loss is above 8 % even for $T_{\text{w page}}$ = 10.24 s, which suggests that if the system mainly has to deal with multiple slaves being paged, the page window should be kept larger than 10.24 s.

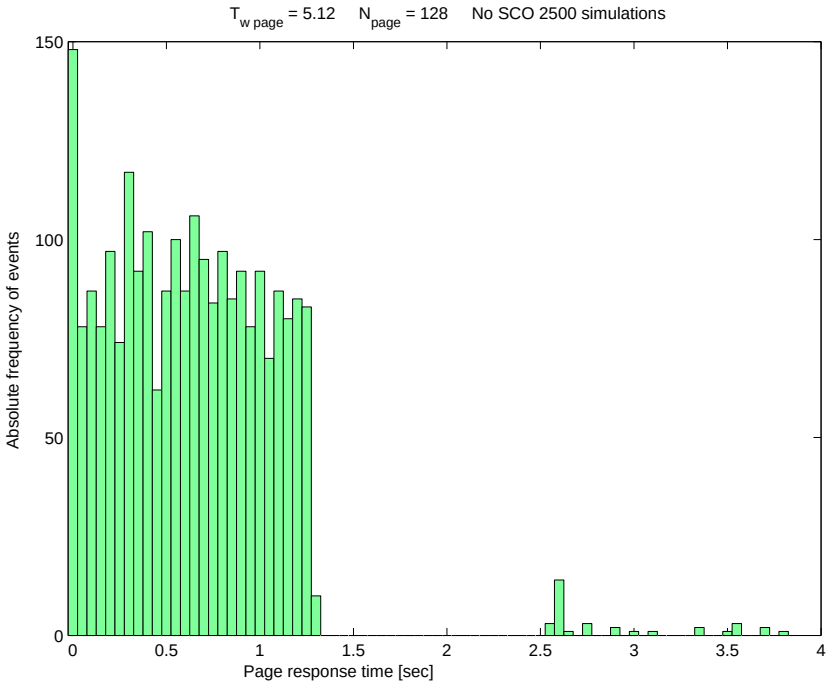

Figure 5.15: *Histogram for the average page response time.*

When one slave is being paged with two SCO channels present, the loss rate is not satisfying for $T_{\text{w page}}$ = 5.12 s, as for the no SCO and one SCO channel cases. The simulation results show that the page window should be kept above 8.96 s, a good value is $T_{\text{w page}}$ = 10.24 s. The time to complete the page in this case is 2.55 s, which is acceptable. $T_{\text{w page}}$ to should be 10.24 s in the multiple slave case as well, even though the loss ratio is 14 %. Even though the time to complete the page is 5.18 s, which is high, increasing the page window to decrease the loss ratio would make the time to complete the page unacceptably high.
The average time to complete a page is generally very quick, as shown in the histogram (Figure 5.15). Most responses happen instantly and the curve slopes a little bit down to the switching of trains. The probability that the device being paged is listening to a frequency in the A train

is greater than it listening to a frequency belonging to the B train. The remaining devices not responding when train A was sent the first time, is likely to respond the second time train A is sent. All depending on the page window, $T_{\text{w page}}$, and the number of times the trains are sent, N_{page}.
For the default values of N_{page} and $T_{\text{w page}}$, the mean response time was lower than 1.23 s in 95 % of the cases. After 2.58 s the page was completed in 99 % of the cases. The simulations shows that the default values for the number of times the trains are sent N_{page} as specified in the specification [1] are adequate. We have manipulated the $N_{\text{w page}}$ value for different simulations and cases and the results show that in most cases they are worse than the default values for N_{page}. However, in special cases, they can make a marginal improvement, for instance when one SCO channel is present and multiple slaves are being paged. Increasing the N_{page} to 384 will give a slightly better loss ratio in this case.

5.5 Mean time to connect

The mean time to set up a connection (MTTC) is very important in a system like MediaCell as well as other possible Bluetooth networks. It should be as low as possible while retaining a high success probability.
Note that in simulating MTTC, we configure the potential master to search for a fixed number of devices ranging between 1 and 7 potential slaves. Since every SCO connection presupposes that an ACL link has been set up prior to it, a device with one SCO link present can only connect to 6 more devices, similarly a device with two SCO links can only connect to 5 additional devices. The loss ratio is the percentage of the potential connections that have failed.
The basis for all MTTC simulations are the previous simulations of the inquiry and page processes. From these simulations, we have decided the settings of the different parameters that are to be run as simulation cases.
The most common cases are when there is only one other device in rage, hence the following results are all simulations of one device initiating a connection with one other device in rage.

Case	$T_{\text{w inquiry}}$	N_{inquiry}	$T_{\text{w page}}$	N_{page}
1	5.12	120	5.12	128
2	5.12	120	3.84	128
3	7.68	120	5.12	128
4	7.68	120	3.84	128
5	10.24	256	5.12	128
6	10.24	256	3.84	128

Table 5.1: *Simulation cases, no SCO links present.*

5.5.1 No SCO channels present

The cases that were simulated are shown in Table 5.1. Notably the settings previously recommended in Sections 5.3.4 and 5.4.3 provide a performance that is fully acceptable. Case 2 will result in a mean time to connect of 2.75 s and a loss ratio of 0.8 %. Case 4 however, seams to result in a better trade-off with a mean time to connect of 2.78 s and a loss ratio of only 0.1 % (Figure 5.16).
The default settings have been simulated in case 6, where the response time is 3.01 s, so the improvement in response time from the default settings to the settings of case 4 is 8 %, at the cost of hardly any increase in the loss ratio at all.
Notable is also the fact that none of the cases simulated has a loss ratio greater than 1.3 %. A histogram of case 4 (Figure 5.17) shows that in 95 % of all connections made, the mean time to connect was lower than 4.81 s. After 6.05 s, a connection should have been established in 99 % of the cases. Note that the bell shapes produced by the N_{inquiry} (Figure 5.4) are still visible in the MTTC histogram.

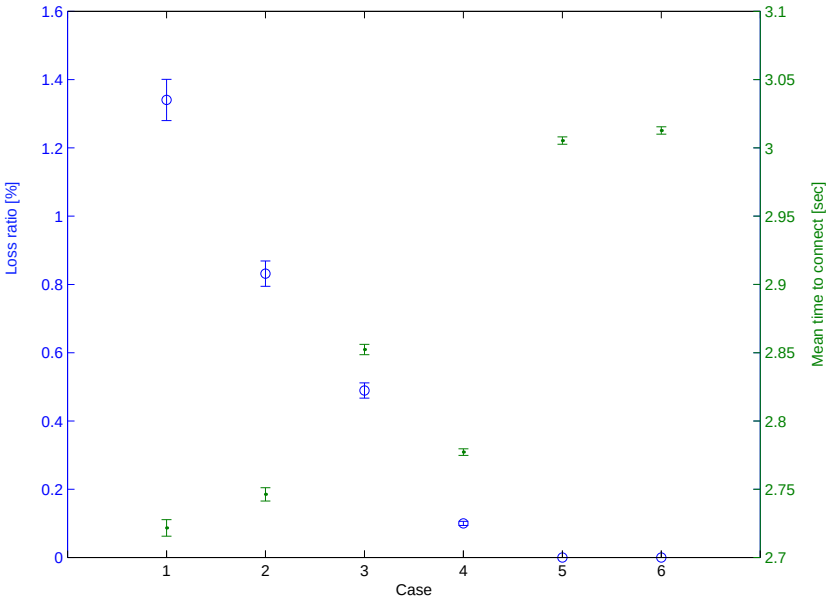

Figure 5.16: *Loss ratio, blue (o), and mean time to connect, green (·) for cases 1 to 6, no SCO channels present, (Table 5.2).*

5.5.2 One SCO channel present

When there are SCO channels present, the improvements will be more obvious than without SCO channels. The cases simulated for when one SCO channels is present are shown in Table 5.2. The recommended values of the repetition numbers, inquiry- and page-windows, denoted as case 5, yield a loss ratio of 2.9 % and a mean time to connect of 4.47 s. But as in the

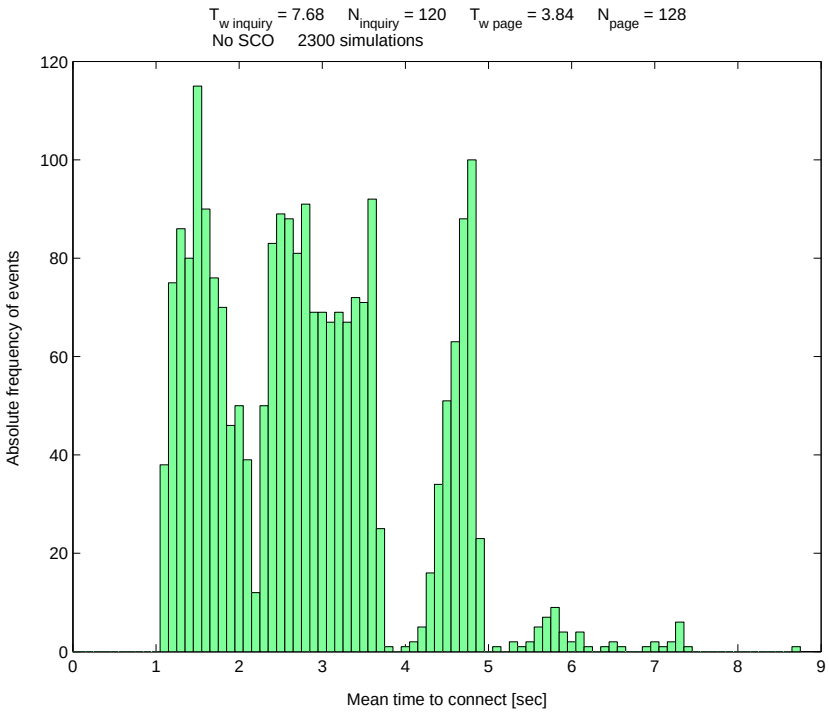

Figure 5.17: *Histogram of the mean connection time with no SCO channels present.*

Case	$T_{w\ inquiry}$	$N_{inquiry}$	$T_{w\ page}$	N_{page}
1	8.96	120	3.84	128
2	8.96	120	5.12	256
3	8.96	120	5.12	384
4	8.96	120	3.84	384
5	10.24	120	3.84	384
6	15.36	120	3.84	256
7	15.36	120	3.84	384
8	15.36	120	5.12	256
9	15.36	120	5.12	384
10	20.48	512	5.12	384
11	20.48	512	3.84	384
12	20.48	512	5.12	256

Table 5.2: *Simulation cases, one SCO link present.*

case when no SCO channels were present, there seems to exist a better trade-off namely case 8 which will result in a loss ratio of 0.5 % and a mean time to connect of 4.65 s. This should be compared to the default values shown in case 12 which has a loss ratio of 0.8 % but a mean time to connect of as much as 7.35 s. Case 8 gives an improvement in the connection time by 38 %, and an improvement of the loss ratio by 38 %.

A histogram of case 8 still shows the distinct bell curves of the inquiry process (Figure 5.4). In 95 % of all connections made, the mean time to connect was lower than 8.00 s. After 9.49 s, a connection should have been established in 99 % of the cases.

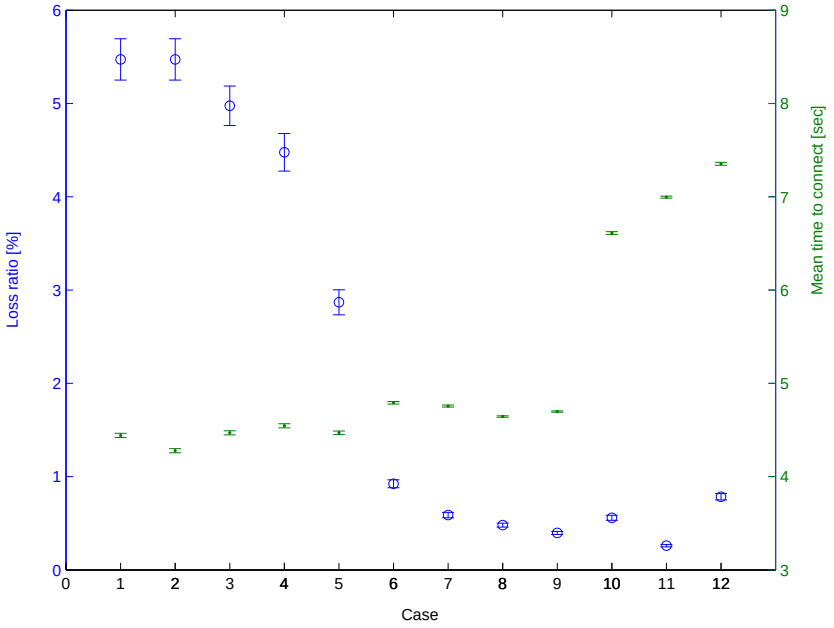

Figure 5.18: *Loss ratio, blue (o), and mean time to connect, green (·), for cases 1 to 12, one SCO channel present, (Table 5.2).*

5.5.3 Two SCO channels present

When there are two SCO channels in use, obviously the mean time to connect will be much greater than when there are no SCO channels in use. As shown in Figure 5.19, the default values, case 9 in Table 5.3, produces an acceptable, albeit quite high, loss ratio of 5.9 %. But the mean time to connect at 19.0 s is unacceptably high. A better tradeoff can be obtained by using case 12 which yields a much lower mean time to connect of 15.0 s while maintaining a loss ratio of 5.5 %. Using case 12 will result in an improvement in connection time of 21 % while retaining an acceptable loss ratio.

A histogram of case 12 does not show the distinct bell curves of the inquiry process (Figure 5.20). In 95 % of all connections made, the mean time to connect was lower than 33.47 s. After 40.10 s, a connection should have been established in 99 % of the cases.

Case	$T_{w\ inquiry}$	$N_{inquiry}$	$T_{w\ page}$	N_{page}
1	20.48	256	5.12	384
2	20.48	256	7.04	384
3	20.48	256	8.96	384
4	20.48	256	10.24	384
5	40.96	256	5.12	384
6	40.96	256	5.12	384
7	40.96	768	7.04	384
8	40.96	768	10.24	384
9	40.96	768	8.96	384
10	40.96	256	7.04	384
11	40.96	256	8.96	384
12	40.96	256	10.24	384

Table 5.3: *Simulation cases, two SCO links present.*

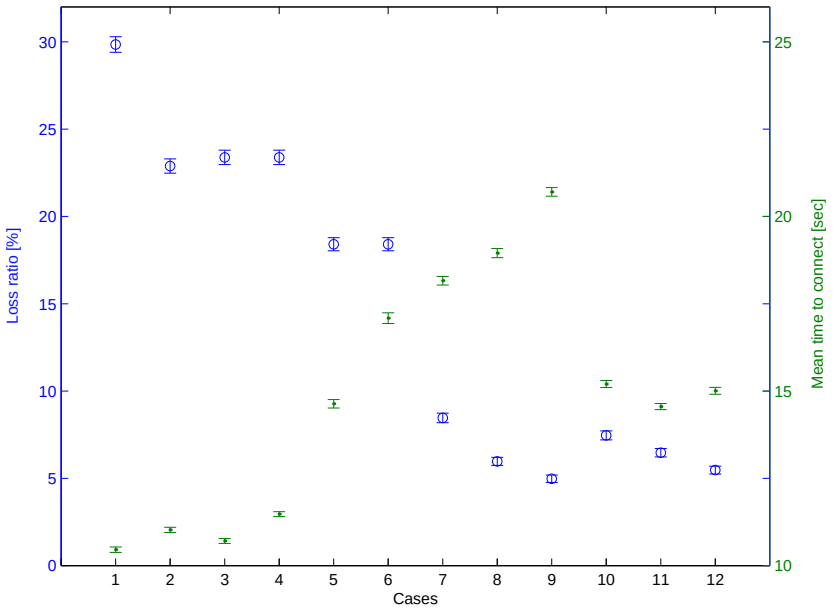

Figure 5.19: *Loss ratio, blue (o), and mean time to connect, green (·), for cases 1 to 12, two SCO channels present, (Table 5.2).*

5.5.4 Multiple devices in range

In the case there are more than one device in range of the device trying to establish connections, the loss ratio and mean time to connect will of course increase. Simulations have been performed with two to seven devices in rage for the no SCO case, for two to six devices in rage for the one SCO case and for two to five devices in rage for the two SCO case. As stated previously a device with one or two SCO channels already in use can not connect to more than six or five

other devices respectively.

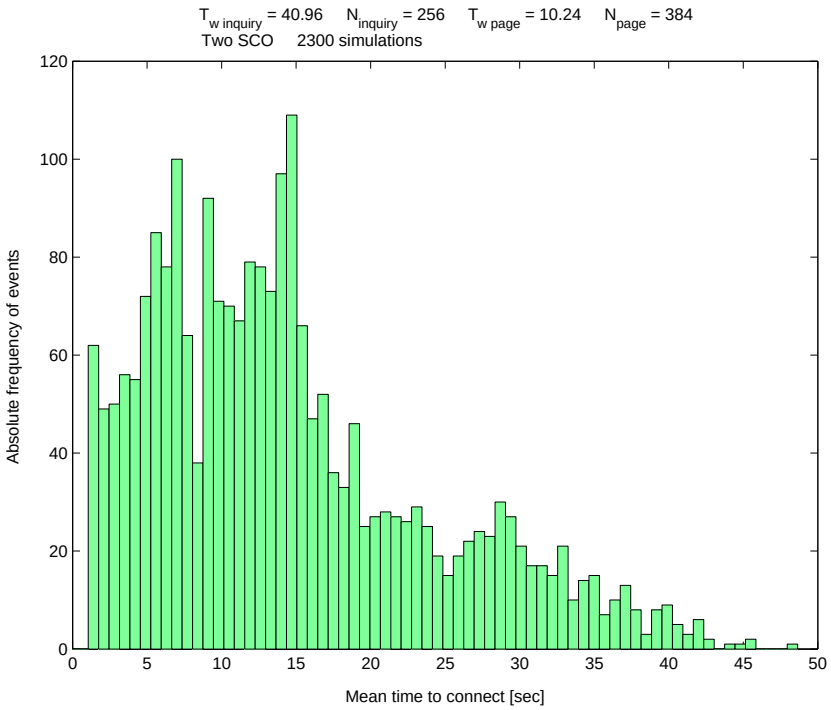

Figure 5.20: *Histogram of the mean connection time with two SCO channels present.*

No SCO channels present

In case there are no SCO channels present, the mean time to connect will increase as the number of devices in range increases. In Figure 5.21, the mean connection time for 1 to 7 devices in range is shown Note that the optimized values gives slightly better performance if the number of devices is lower or equal to four. Should there be more than four devices in range, we recommend using the default values.

One or two SCO channels present

Should there be SCO channels present, the increase in mean connection time is more obvious. The optimized values gives a performance gain in the mean connection time by 30–40 % (Figure 5.22).

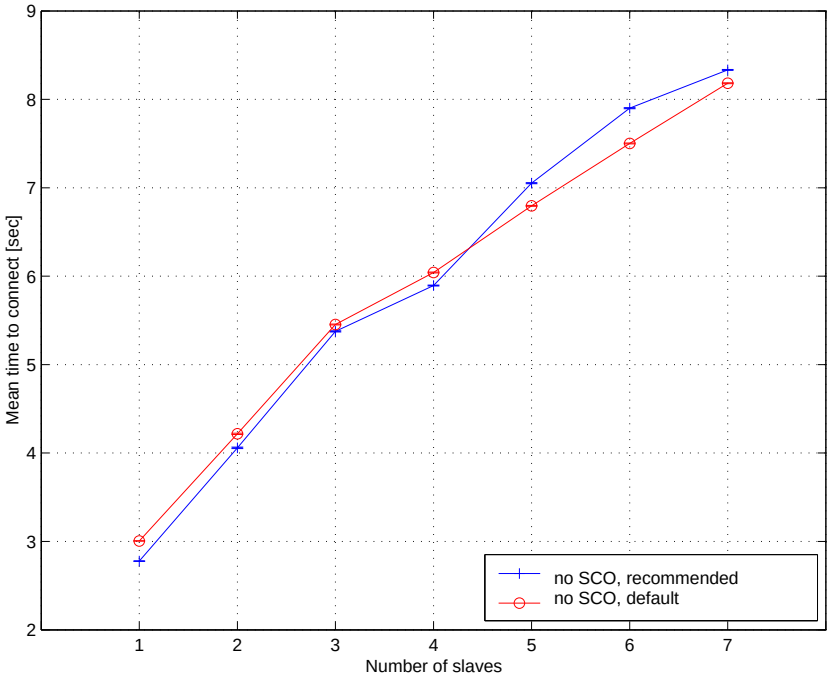

Figure 5.21: *Mean time to connect for multiple slaves, no SCO channels present.*

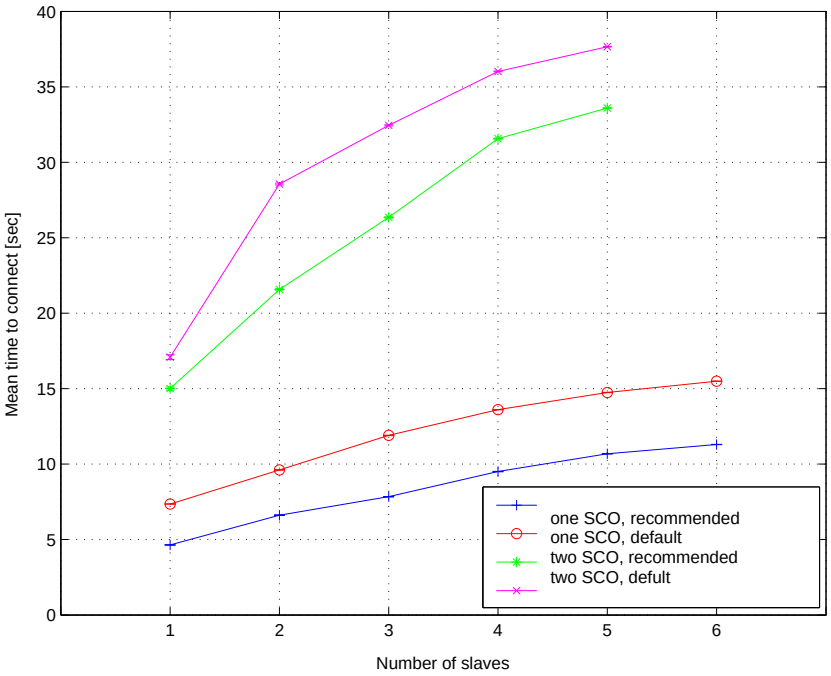

Figure 5.22: *Mean time to connect for multiple slaves, one and two SCO channels present.*

Chapter 6

Conclusions

6.1 Device discovery

The device discovery process in Bluetooth is, as we have shown, a lengthy process. There are, however, some simple ways of reducing the time to complete the connection process. The inquiry process is the part that consumes most time since it not only has to achieve frequency synchronization twice, but also has a random backoff delay which is relatively long. The parameters for reducing the inquiry process are the inquiry window $T_{\text{w inquiry}}$ and repetition number N_{inquiry}. By selecting the values for these parameters carefully, the time to complete the inquiry process can be reduced. Hence, the time to complete the whole device discovery process, or the mean time to connect, will be reduced.
We have reduced the mean inquiry response time by 8 % for no SCO channels present. In case there is one SCO channel present, we have reduced the mean inquiry response time with 42 %, and for the case of two SCO channels, we have reduced the mean inquiry response time with 31 %. The improvements have been achieved without any increase of the loss ratio.

The page process is faster than the inquiry process since the paging device may have knowledge of the paged device's clock estimate. Furthermore, frequency synchronization only has to be obtained once. If one device is being paged, our simulations indicate that not much time or loss ratio can be gained by changing the page window from the default values for no or one SCO channel present. When two SCO channels are present, the page window should be increased in order to achieve an acceptable loss ratio.
Changing the repetition number from the default values does not improve the page response time.

We have improved the mean time to connect by 8 % and only marginally increased the loss ratio for no SCO channels present. The gain is more evident if SCO channels are present. The improvement in mean time to connect is 38 % and 35 % in case there are one or two SCO channels present, respecively.

It is important to note that the changes we propose does not involve any changes in the Bluetooth hardware, they are purely changes in software (parameter settings).

6.2 Service discovery

The Bluetooth Service Discovery Protocol does not provide access to the services discovered through it. It will in many cases be necessary to use a higher level service discovery protocol to provides the access.
There are too many service discovery protocols in the market today. Some of them are easily bridged or mapped to each other, but it should not be necessary to implement a number of service discovery protocols into a system or use bridges or proxies when the protocols themselves are logically similar. Due to market forces, perhaps a unified standard will emerge in the future. The protocols that are most likely to compete for such a unified standard are Salutation, UPnP and Jini.
Some of the higher level protocols have already been ported to work over Bluetooth and Bluetooth SDP, others may use the TCP/IP stack or other transport protocols supported by Bluetooth. Smaller devices, however, may not have these protocols implemented, making it necessary to provide a proxy or bridge to the service discovery protocol, which is undesirable.
For a system like MediaCell, it would be suitable to implement Salutation Lite. It was developed for small hand held devices with limited screen sizes, memory and processing capabilities. Salutation will most likely be one of the big contenders for a unified service discovery standard and Salutation Lite can be extended to a full Salutation implementation for the entire system but still retain its original features towards devices of lower complexity, thus making it a very suitable choice for a system like MediaCell.

Bibliography

[1] Bluetooth SIG, *"Specification of the Bluetooth system v1.1, Core"*, December 2000.

[2] Bluetooth SIG, *"Specification of the Bluetooth system v1.1, Profiles"*, December 2000.

[3] Salutation Consortium, *"Salutation Architecture Specification v2.1, Part 1"*, 1999.

[4] Sun Microsystems, *"Jini Architecture Specification v1.2"*, December 2001.

[5] Microsoft Corporation, *"Universal Plug and Play Device Architecture v1.0"*, June 2000.

[6] Jennifer Bray and Charles F Sturman, *"Bluetooth, Connect without cables"*, Prentice Hall, 2001.

[7] Oliver Kasten and Marc Langheinrich, *"First Experiences with Bluetooth in the Smart-Its Distributed Sensor Network"*, Swiss Federal Institute of Technology, October 2001.

[8] Eugene A Gryazin, *"Service Discovery in Bluetooth"*, Helsinki University of Technology, 2000

[9] Choonhwa Lee and Sumi Helal, *"Protocols for Service Discovery in dynamic and mobile networks"*, University of Florida, September 2000.

[10] Bluetooth SIG, *"Mapping Salutation Architecture APIs to Bluetooth Service Discovery Layer"*, White paper, July 1999.

[11] Sun Microsystems, *"Jini Architectural overview "*, White paper, January 1999.

[12] John Rakesh, *"UPnP,Jini and Salutation - A look at some popular coordination frameworks for future networked devices"*, California Software Laboratories, June 1999.

[13] Salutation Consortium, *"Salutation Lite, Find-And-Bind Technologies for mobile devices"*, White paper, June 1999.

[14] Microsoft Corporation, *"Understanding Universal Plug and Play"*, White paper, June 2000.

[15] Steven Czerwinski, Ben Zhau, Todd Hodes, Anthony Joseph and Randy Katz, *"An Architecture for a Secure Service Discovery Service"*, University of California, Berkeley, 1999.

[16] E. Guttman, C. Perkins, J. Veizades and M. Day, *"Service Location Protocol, Version 2"* IETF, June 1999.

[17] M. Wahl, T. Howes, S. Kille, *"Lightweight Directory Access Protocol (v3)"* IETF, December 1997.

[18] Erik Guttman, *"Service Location Protocol: Automatic Discovery of IP Network Services"* IEEE Internet Computing, vol 3, no.4, pp 71–80, August 1999.

[19] Network Simulator web page, available 8/2/2002. http://www.isi.edu/nsnam/ns/

[20] BlueHoc web page, available 8/2/2002. http://oss.software.ibm.com/bluehoc/

Glossary

A

ACL Asynchronous Connection Less.

AM_ADDR Active Member Address.

AP Access Point.

B

BB Baseband.

BD_ADDR Bluetooth Device Address.

C

CC Cluster Controller.

CLKN Native clock.

D

DIAC Dedicated Inquiry Access Code.

F

FHS Frequency Hopping Selection (packet).

G

GAP General Access Profile.

GFSK Gaussian Frequency Shift Keying.

GIAC General Inquiry Access Code.

GW Gateway.

I

IAC Inquiry Access Code.

IETF Internet Engineering Task Force.

ISM Industrial, Scientific and Medical.

L

L2CAP Logical Link Control and Adaptation Protocol.

LAP Lower Address Part.

LDAP Lightweight Directory Access Protocol.

LMP Link Manager Protocol.

M

MT Mobile Terminal.

MTTC Mean Time To Connect.

N

Nam Network Animator.

NS Network Simulator.

O

OBEX Object Exchange Protocol.

OTcl Object Tool command language.

P

PDU Protocol Data Unit.

PM_ADDR Parked Member Address.

R

RFCOMM Simple cable replacement protocol.

RMI Remote Method Invocation.

S

SCO Synchronous Connection Oriented.

SDAP Service Discovery Application Profile.

SDP Service Discovery Protocol.

SIG Special Interest Group.

SLM Salutation Manager.

SLM-TI Salutation Manager Transport Interface.

SLP Service Location Protocol.

SSDP Simple Service Discovery Protocol.

SSDS Secure Service Discovery Service.

T

Tcl Tool command language.

TDD Time Division Duplex.

Tk Tool kit.

TM Transport Manager.

U

UPnP Universal Plug and Play.

URL Uniform Resource Locator.

UUID Universally Unique Identifier.

Appendix A

The Simulations Manual

To run these simulations, the network simulator (NS) [19] must be installed. BlueHoc [20] must also be installed on top of NS.
To run a simulation, certain parameters must be set, this is done in the tcl scripts. To run the same tcl script many times we have written a awk script. The awk script runs the same tcl script, but with different seeds, until the conditions, i.e. the confidence intervals and the minimun number of simulations, in the awk script is fulfilled. Then it prints the result of the simulations to an output file. Here follows a description of what scripts are used for the different simulations and what they do.

A.1 The Tcl scripts

The tcl scripts is the last interface towards the user, it is where most of the parameters regarding the simulation are set. When executing the tcl script, the script links to other scripts, and ultimately they run the simulation.
Seven tcl scripts are used in these simulations, from 01.tcl to 07.tcl, one script for each number of slaves that are in range. We have simulated an error free environment, hence the distance between the master and the slave has been set to 0.

```
set Sim(xpos_) [list 0 0]
set Sim(ypos_) [list 0 0]
```

Some time-outs and parameters are set in the the tcl scripts, for instance the Inqiry timeout.

```
set InqTimeout 32768
```

The parameter value is the number of half time slots that shall pass before the time out is activated. A half time slot is 312.5 s, so in this example, the time out is set to be 10.24 seconds.

A.2 The Mean Time To Connect (MTTC)

The mttc.awk script takes a tcl script as an input parameter, and executes the script.
It grabs the ”POLL_ACK” packets from the output and places it in a temp file. The script will then select the time for when the ”POLL_ACK” packets are sent and calculate the average connection time and confidence interval as well as the loss ratio and its confidence interval. If the confidence level is lower than 95 % of the mean value, the script will run another simulation with the same tcl script. When the condition of the confidence interval is satisfied, mttc.awk will calculate the percentage of slaves that did not get connected, the average connection time of the slaves that did get a connection and the confidence interval, and print it to a file named ”mtresults”.
Since there are 7 tcl -scripts, one for each number of slaves trying to connect, there is a script called mttcscript that runs all of them (Figure A.1). After the simulations are done, the content of the ”mtresult” file looks something like this:

```
1  68.6667  +-0.775114  4.103322  +-0.00688751
2  64.5349  +-0.629907  5.279962  +-0.0135417
3  67.3203  +-0.463261  6.448494  +-0.0192325
4  62.2549  +-0.419416  7.535986  +-0.0120505
5  59.2308  +-0.431862  8.725084  +-0.010134
6  58.8235  +-0.431234  9.741399  +-0.00919069
7  57.423   +-0.341618  9.845272  +-0.00627726
```

The columns are, in order, ”number of slaves trying to connect”, ”percentage of slaves that did not get connection”, ”the confidence interval of the loss ratio”, ”average connection time of the slaves that got connection” and ”the confidence interval of the time”.
Printing the output in this form makes it easier to import the values into Excel or MATLAB, to draw graphs or make further calculations.

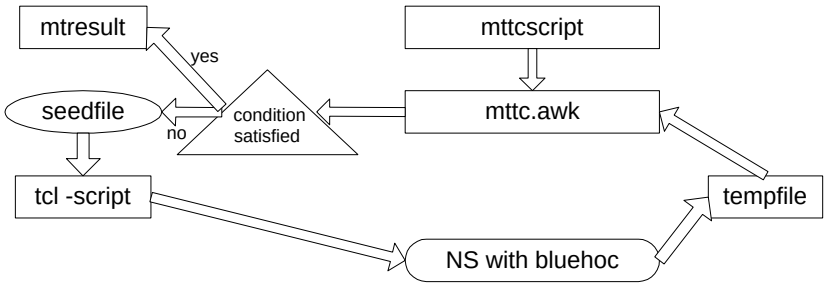

Figure A.1: *Dependencies between the scripts and Network Simulator.*

A.3 The inquiry scripts

These simulations are run to determine the average time of an inquiry and to investigate the performance of the inquiry procedure.
The inq.awk script takes a tcl script as an in parameter, and executes the script.
It grabs the ”INQ_MSG_AFTER_BO” and calculates the mean time for an inquiry response and the mean time of every device’s first response. The script will continue to execute simulations until the confidence interval is less than 5 % of the mean value. The script will always run at least 100 simulations.
A side from the mean inquiry response times the script will calculate the percentage of devices that did not respond to the inquiry. All data is printed to a file called inresults, the format of the data is as follows:

```
83  0.96401   +-  0.018491    1.32362  +-  0.00874712  48.1928
50  1.00858   +-  0.0147705   1.36325  +-  0.0076409   50
50  0.992985  +-  0.010913    1.34838  +-  0.00500559  49.3333
50  0.980084  +-  0.00764606  1.34344  +-  0.00382831  49
50  0.994012  +-  0.00656937  1.32754  +-  0.00312406  50.4
50  0.968834  +-  0.00534645  1.31169  +-  0.00251982  50
50  0.96783   +-  0.00453896  1.31089  +-  0.00211928  49.7143
```

The columns are, in order, ”number of simulation runs”, , ”average response time (first response)”, ”+- confidence interval”, ”average response time (total)”, ”+- confidence interval” and ”percentage of slaves that did not respond”. Printing the output in this form makes it easier to import the values into Excel or MATLAB, to draw graphs or make further calculations.
For the inquiry simulations, the interesting parameter to change is the inquiry time out (InqTO). The focus lies on finding out how the error percentage and average inquiry response time vary with the increase or decrease of this time out. Note that the number of responses is always set to a number higher than the number of slaves to ensure that the inquiry always time outs.

A.4 The page scripts

These simulations are run to determine the average page time and performance of the page procedure, so we have to isolate the page phase of the connection procedure.
The page.awk script takes a tcl script as an input parameter and executes the script.
The output from the simulation is placed in a temporary file. Then the script steps through the tempfile and grabs the ”INQ_MSG_AFTER_BO” packets from the output. This is a check to see how many slaves have completed the inquiry phase. If not all the slaves have completed the inquiry procedure, that simulation is discarded. Since the inquiry phase of this simulation is not important in this simulation except that all the slaves must pass it, the inquiry time out is set to 20.48 seconds. The inquiry time out means that if the master have not received a ”INQ_MSG_AFTER_BO” packet from all the slaves in 20.48 seconds, it will stop the sending inquiry and start the page procedure. However, should the master receive all the responses

before 20.48 seconds it will start the page procedure earlier. To prevent that from happening, the ”number of reposes” parameter in the tcl scripts are set to be at least equal to the ”Sim(NumDevices)” parameter.
The high inquiry time out means that all the slaves will have finished the inquiry procedure in almost all cases.
When this is done, the script runs the same tempfile again but this time it grabs the ”POLL_ACK” packets, there is one ”POLL_ACK” packet sent for each successful page. The time for when the ”POLL_ACK” packet is sent is subtracted by 20.48 in order to get the actual time it took to do a page.
Then calculations of the average page time and confidence interval are made. If the level of confidence is lower than 95 %, the script will run another simulation with the same tcl script. When the condition of the confidence interval is satisfied, page.awk will calculate the percentage of slaves that did not get complete the page procedure, the average page time of the slaves that completed page and the confidence interval, and print it to a file named ”paresults”.
Since there are 7 tcl scripts, one for each number of slaves trying to page, there is a script called pagescript that runs all of them. After the simulations are done, the content of the ”paresult” file looks something like this:

```
1  5.32544  +-0.260414  0.603172  +-0.0050665
2  4.26829  +-0.212835  1.06081   +-0.00593009
3  9.52381  +-0.475922  1.5367    +-0.00899664
4  13.6139  +-0.548393  2.20265   +-0.00953441
5  5.32374  +-0.26545   2.75913   +-0.00608369
6  9.40594  +-0.446902  2.95607   +-0.00805164
7  15.5587  +-0.637569  3.7567    +-0.00896331
```

The columns are, in order, ”number of slaves trying to page”, ”percentage of slaves that did not get connection”, ”the confidence interval of the loss rate”, ”average connection time of the slaves that got connection” and ”the confidence interval of the time”.
Printing the output in this form makes it easier to import the values into Excel or MATLAB, to draw graphs or make further calculations.
For the page simulations, the interesting parameter to change is the page time out (PageTO). The focus lies on finding out how the error percentage and average page time vary with the increase or decrease of this time out. The pageTO is changed in ”globals.h”. For the changes to come through a ”make” has to be done.

A.5 The SCO channels

All the above simulations can be simulated without SCO channels or with one or two SCO channels present. The change of the number of SCO channels is done in baseband.cc under the method `CLKNHandler::handle (Event* e)`. The line regulating the number of SCO channels are:

```
if (lm_->counterSCO_ != 1) { // with SCO !=1 one, ==1 two SCO, comment away for no SCO!
```

This is an example with one SCO channel. To get two SCO channels, just change the condition from !=1 to ==1 in the parenthesis. To simulate without SCO channel, just comment away the whole line.
When simulating page, it can be useful not to introduce the SCO channel until the inquiry is over and the page begins. This is also done in baseband.cc under the method `CLKNHandler::handle (Event* e)`. The two lines regulating this are:

```
//else if (lm_->state_ == INQUIRY && SCO) {
else if (lm_->state_ == INQUIRY) {
```

The example is valid for page simulations. If simulating inquiry och mttc, just uncomment the first line and comment the second line.

A.6 Seed

There are two types of seeds to choose between, heuristic seed and raw seed. The heuristic seed uses the system clock to initiate the random number generator. In this case, it is not possible to recreate exactly the same simulation. The choice between using raw seed and heuristic seed is done in baseband.cc. The lines regulating it are in the method `Baseband::start()`.

```
//rngAJ_->set_seed(RNG::RAW_SEED_SOURCE, seed_); { // sets seed explicitly
rngAJ_->set_seed(RNG::HEURISTIC_SEED_SOURCE); { // sets seed with the system clock
```

In this example the simulations are done using heuristic seed. In order to use raw seed, just uncomment the first line, and comment away the second line.
Using the raw seed, the user can recreate exactly the same simulation again. The user sets in the tcl script what seed the random number generator shall be initiated with. The seed needs to be updated after each simulation not to get the exact same result each time. There is a file called ”seedfile” which contains the seeds, and the tcl script will import the seeds from this file. The awk script updates the seedfile with a new seed before every simulation.
There is an advantage in the it is possibility to recreate the same simulation when using the raw seed, specially when debugging.

Appendix B

Problems regarding the simulations

All through the project, we have been running into problems and difficulties. This is a top down description of the mistakes we, and others have done, and how we solved them or got around them. This may not be very interesting from a scientific popint of view, but maybe it can give an idea on what can happen during a project.

B.1 Installation problems

First we started by searching for a simulator that supported Bluetooth traffic. There was a well-recognized simulator called Network simulator (NS) [19], and a Bluetooth extension called BlueHoc [20] available. Both are open source programs.

We had access to two PCs with Windows 2000 operating system and we wanted to install NS on them. There are a lot of patches to run and make changes in to make this work, and we did not succeed. After consulting experienced programmers and the replies we received from the BlueHoc mailing list, it became clear that nobody had succeeded in installing it on Windows 2000. We then decided to use another operating system.

We found out that there was a Linux server used by the department of electrical engineering that already had NS installed on it. The system administrator was kind enough to install BlueHoc on it as well and give us access to it. However, when we ran simulations on one computer, and another computer started a simulation, the first simulation crashed. So we settled to run the simulations on just a PC. This seemed to work just fine, but after a couple of weeks, strange errors started to occur that had never occurred before. Possibly there was a sharing violation when all the other users of the NS ran their simulations.

So we installed NS on a Sun Solaris7 server, this took a fair bit of time since our system administrator has just begun working for ATRI and had never used NS before. She did a good job though and finally got it installed on the Sun machine. We executed a simulation, but there was no output. Somehow, the results did not go to the standard output, and no matter where we looked, we could not find them.

Then we came across a program called VM-Ware. This is a virtual machine, which allows the user to install an operating system on top of the existing one. We installed Linux on the

virtual machine, and then NS and BlueHoc on top of that. It has been running without problems most of the time, but there is some instability, which led to a reinstallation when the virtual machine crashed once.

This was the trouble we had before we even were able to have a look at the actual software and what it could do for us.

B.2 Bugs and problems running BlueHoc and NS

BlueHoc is under development, which means that there are quite a lot of bugs in the program, and that some things we needed for our simulations, were not implemented.
The randomizing of the slaves that is trying to connect was not implemented, so they all entered the system at exactly the same time, which is not very likely in a real scenario.
The native clock of the slaves were not randomized either, and since the frequency that the slave is listening to is determined by the native clock, all the slaves listened on the same frequency, which is not realistic.
So we created an object in the rng class to randomize both the time when the slaves enter the system and what their native clock is set to at that time.

The random backoff time, the time a slave waits when it has received an inquiry until it send its inquiry response, was not randomized so we had to randomize it. The backoff is uniformly distributed between 0 and 1023 time slots, but we discovered that it never waited all the time slots. After a lot of debugging, we found the error in the BlueHoc code. The inquiry response timeout was set to its value according to the spec, but it should have been the random backoff plus the inquiry response timeout, hence the inquiry response timeout was in many cases activated before the random backoff was complete. This was only a small error in the code, but it took some time to find it.

We wanted random simulations, so that we did not run the same one all the time, and yet we wanted to have some control over them. We wanted to be able to recreate the exact same simulation if necessary, i.e. initialize the random number generator with a seed. This was a bit tricky, because we had to import the seed from the tcl scripts in order to have control of it. This worked fine and it is very useful to run the exact same simulation when debugging. However, when we had finished our debugging and ran many simulations, there seemed to be some kind of vague periodic pattern in our results. To make sure that our seed algorithm did not cause it, we then used the system clock as the seed generator. Using the system clock does not necessarily mean that it is completely randomized and that you won't get any periodic pattern, but it is generally accepted to use the system clock for randomization.

We found that many of the parameters and timeouts in the BlueHoc were not set to the value given by the Bluetooth specification. We do not know if this was done accidentally or deliberately. It is a simple task to solve once the problem is discovered.

The task of implementing SCO channels was an interesting one. We used a method of simply ”steeling” the bandwidth that should have been allocated for the SCO channels. This means that if there is one SCO channel present, every third packet is discarded, and if two SCO channels are present, two out of three packets are discarded.

While using BlueHoc, we have corrected a number of bugs in the program. Either we have found them ourselves or got them pointed out to us from the BlueHoc mailing list.
Bugs that we have corrected with help from the BlueHoc mailing list are in the method PERM and FH-kernel in baseband.cc. In PERM, the numbers in the different cases was wrong compared to what the Bluetooth specification says [1, p 130]. In the method FH-kernel, the case of page hopping and master response, the value for X can be negative. If that happens, a number of 16 must be added to the result. This was forgotten, but was corrected. It is very hard to find these bugs, and we are thankful to the people that have reported them on the mailing list.

One cannot be absolutely sure that there are no bugs left that would affect our simulation results, since this software is still under heavy development. However, the simulation results seem to be OK and are in some cases verified with experiments done by others, so the results should be valid.