EXTENDING THE REACH OF PERSONAL AREA NETWORKS BY TRANSPORTING BLUETOOTH COMMUNICATIONS OVER IP NETWORKS

A thesis submitted in fulfilment of the

requirements for the degree of

Master of Science

of

Rhodes University

by

David Sean Mackie

December 2006

ABSTRACT

This thesis presents an investigation of how to extend the reach of a Bluetooth personal area network by introducing the concept of Bluetooth Hotspots. Currently two Bluetooth devices cannot communicate with each other unless they are within radio range, since Bluetooth is designed as a cable-replacement technology for wireless communications over short ranges. An investigation was done into the feasibility of creating Bluetooth hotspots that allow distant Bluetooth devices to communicate with each other by transporting their communications between these hotspots via an alternative network infrastructure such as an IP network. Two approaches were investigated, masquerading of remote devices by the local hotspot to allow seamless communications and proxying services on remote devices by providing them on a local hotspot using a distributed service discovery database. The latter approach was used to develop applications capable of transporting Bluetooth's RFCOMM and L2CAP protocols. Quantitative tests were performed to establish the throughput performance and latency of these transport applications. Furthermore, a number of selected Bluetooth services were tested which lead us to conclude that most data-based protocols can be transported by the system.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

My grateful thanks are due to:

- my supervisor, Professor Peter Clayton, who initially suggested this work and for his guidance and assistance over the last few years and giving direction when needed

- Professor George Wells, who so willingly stepped in and waded though my many grammatical and spelling atrocities to help put the final thesis together;

- Caro Watkins, for the no-nonsense counseling and advice she has given me during our many "progress chats";

- Barry Irwin, David Siebörger, Guy Halse, and Russell Cloran who freely offered their advice and technical expertise, and for those "ah ha" moments when it eventually made sense;

- Bev Moodie, David Siebörger, Derek Louw and Raymond Louw for their heroic efforts in proofreading various chapters on short notice;

- my family, especially my father, who have supported me through all the difficult and interesting decisions I have made;

- Center of Excellence in Distributed Multimedia and its sponsors, the National Research Foundation and the German Academic Exchange Service (DAAD), who provided equipment and funding.

Lastly and most importantly, I would like to thank my friends, Ingrid Brandt and Mici Halse, who badgered, cajoled, threatened and bribed me to work. To them I owe my deepest gratitude for without their unfailing support this thesis would not have been completed.

Chapter 1:

INTRODUCTION

We start with a basic introduction to Bluetooth and discuss the motivation for this project. We then pose a number of research questions and explain the research methodologies that are going to be followed to answer them. Lastly, we give a breakdown of the rest of the thesis.

1.1 Bluetooth

Bluetooth is a specification for a low-cost, low-power, short-range wireless communication technology. It was born out of a study undertaken by Ericsson Mobile Communication in 1995 and provides wireless connectivity between mobile devices such as cellphones, personal digital assistants (PDA) and portable computers.

Though the original idea was for a cable-replacement for point-to-point connections, Bluetooth was developed into a wireless ad-hoc network technology allowing users to create what have been called Personal Area Networks (PAN), allowing them to interact with information technology on a totally new level.

The technology is being driven by the Bluetooth Special Interest Group (SIG), a trade association of over 2000 different companies comprising the leaders in the telecommunication, networking and computing industries. The Bluetooth SIG was founded by Ericsson, IBM, Intel, Nokia and Toshiba in February 1998 and later joined by Microsoft, Lucent, Motorola and 3Com in December 1999. These companies are now called the SIG Promoters and are responsible for the administration of the SIG in relation to legal, marketing and qualification matters.

Bluetooth was envisioned as a global communication system and this has some bearing on the large amount of detail there is in the specification documents. The complete Bluetooth system is exhaustively defined in the specification from the lower physical radio layer up to and including software applications, thus preventing potential inconsistencies when implemented by various independent companies. All the Bluetooth specifications are available on the Bluetooth SIG's website. They are open for anyone to read and are patent and license free for SIG members, thus making it easier for the multitude of vendors to implement them without interoperability problems.

1.2 Project

This thesis presents ideas on how to extend the reach of Bluetooth by introducing the concept of Bluetooth hotspots. Currently two Bluetooth devices cannot communicate with each other unless they are within radio range. An investigation was undertaken into the feasibility of creating Bluetooth hotspots that allow distant Bluetooth devices to communicate with each other by transporting their communication between hotspots via an alternative network infrastructure such as an IP network.

1.3 Research questions

The main research question that this project set out to answer was "Can Bluetooth be seamlessly transported over IP networks?" This prompted a number of further questions that have guided this research.

1. **To what extent can this be achieved without changes being required to commercially active devices?**

Any system that was to be developed not only had to work with devices currently on the market but as much as possible with legacy Bluetooth devices. It was also hoped that a software solution could be found without requiring firmware modifications on existing devices.

2. **What are the security implications of extending the reach of Bluetooth devices?**

How does extending a Bluetooth network from a PAN to something more like a LAN affect the security paradigm of Bluetooth were the average Bluetooth device has a maximum range of only 10meters? Will the existing security options be able to be used or will the system have to implement it own security?

3. What are the efficiency tradeoffs compared to a direct connection?

How does increasing the distance between communicating devices and adding an extra layer of data traffic effect such factors as finding a Bluetooth device, finding a particular service on a Bluetooth device and using a service.

4. What are the limits of scope at the application level?

Will it be a complete solution that transports all Bluetooth communication, or will it have some limitations? How many of the services that the average Bluetooth device offers will be provided for. Will the boundaries of limitations be easily identifiable to end users? Will we be able to handle Bluetooth headsets seamlessly?

1.4 Research Aims and Goals

The primary goal of this project is to answer the question "can Bluetooth be seamlessly transported over IP networks"? However, of equal importance is how this could be done, which we have hypothesised is possible. This has resulted in us posing the number of secondary questions above, which also need to be addressed.

It was envisioned that the system that would accomplish this task would consist of a collection of devices connected together that would act as Bluetooth repeaters, picking up Bluetooth communications in one location and transporting them across a network to be broadcast in another location. The devices would be standard computers with Bluetooth modules (commonly referred to as “dongles”) connected to an IP network. These computers would run custom written software to perform management type tasks such connecting multiple repeaters together and transporting the Bluetooth communications. A proof-of-concept system would be designed and written so as to allow us to perform these tasks and allow various tests to be run. Real world testing would be performed and from this data the effectiveness of the system would be evaluated. The system will also be used to evaluate issues such as the security of the concept. Both the feasibility of such a system and the extent to which it could be done seamlessly and transparently to the user is to be investigated.

1.5 Research Method

The research method in this study was of the form of experimental science where both inductive and deductive reasoning was combined in an iterative process, where each step is re-evaluated against certain criteria and the process builds on previous steps. Initial investigation showed that there was a possibility of transporting Bluetooth data across an IP network but it was not known at what level this could be done, or the process that should be taken. Therefore simple applications were initially written at each step, on top of which features were added as needed.

Initially, basic proof-of-concept transporting applications were designed and written to demonstrate that it was indeed possible to transport Bluetooth communication across an IP-network. These were written to use different parts of the Bluetooth communication specification allowing us to simultaneously evaluate the best course of action for the final application. These were then further developed into stand-alone transporting programs by adding features and debugging issues that appeared. From these applications a larger application was developed that combined them into a single working prototype. This prototype was then used to evaluate the further issues/questions that were posed as well as a base-case for testing the effectiveness of the system.

1.6 Organisation of Thesis

Chapter 2: Supporting Theory

This chapter provides background information to the project. It starts off with an explanation of the Bluetooth protocol stack and its different layers, then moving on to Bluetooth's service discovery protocol. It then touches on how Bluetooth networks are formed and devices are located. It also covers the security aspects of Bluetooth. Finally Bluetooth services and service discovery is looked at.

Chapter 3: Proposal for Bluetooth Hotspots

This chapter describes the underlying architecture of the project. We present our proposal for a Bluetooth hotspot and examine two different implementation

approaches. BlueSpot, the software components of the proposed Bluetooth hotspot are introduced, as well as the connection framework between these components and other hotspots.

Chapter 4: Foundation and Design

We start off explaining the tools that are used in this project and the reason behind them. We explain why we decided to go with open source software, and the reasons we chose Linux as our operating system and BlueZ as a Bluetooth Stack. We explore the design of the different BlueSpot components and the decisions that went with them are discussed.

Chapter 5: Investigation of a Prototype Hotspot

In this chapter we investigate the implementation of the Transport Component and the Service Discovery Component of BlueSpot. The first part of this chapter deals with the Transport Component and the applications that were written to transport Bluetooth communications at different layers of the Bluetooth protocol stack. We introduce rcpipe and l2pipe, which can successfully transport RFCOMM and L2CAP communications successfully. A proof-of-concept application was then developed to offer the functions of the Service Database Component. We end the chapter with a discussion on some issue pertaining to lower layers that need to be considered.

Chapter 6: Results and Discussion

This chapter will look at our experiences we have had with BlueSpot and the applications that encompass it. We look at the effectiveness of the various programs under “real-world” conditions. We provide performance results on the communications via the pipes compared to communications directly between Bluetooth devices. We then investigate what protocols and services can be transported using BlueSpot. Finally we look at some scalability issues and possible solutions.

Chapter 7: Conclusion

In this chapter we discuss our success and limitations with BlueSpot. We re-examine the research questions posed in Chapter 1, where answers are given and discussed. We then make recommendations on future work that could be investigated.

The field of networking generally, and Bluetooth networking specifically, is marked by the extensive use of acronyms. A glossary of acronyms and technical terms is provided at the end of the thesis (p. 109), which expands and explains the terms used throughout this thesis.

Chapter 2:

SUPPORTING THEORY

This chapter provides background information to the project. It starts off with an explanation of the Bluetooth protocol stack and its different layers, then moving on to Bluetooth's service discovery protocol. It then touches on how Bluetooth networks are formed and devices are located. It also covers the security aspects of Bluetooth. Finally Bluetooth services and service discovery is looked at.

2.1 Bluetooth Background

2.1.1 Bluetooth Origins

Development of Bluetooth was started in 1994 when Ericsson started a study into alternative ways for cellular phones to be connected to accessories and other devices such as laptops and PDAs without the inconvenience of cables. The study looked into using radio signals, which are not directional and do not need line of sight (a problem with the infra-red links that were used at the time). Out of this study, Bluetooth was born. It was named after King Harold Bluetooth, a Danish Nordic King, who united and controlled Denmark and Norway in the 10th- century. The name was chosen as it was hoped that Bluetooth would become an ubiquitous communication standard that would unite the telecommunication and computing industries. [1]

2.1.2 Special Interest Group

The Bluetooth standard was later formalised by the Bluetooth SIG, a trade association of telecommunication, computing and network industry leaders which promotes and defines the specification. The SIG was founded in 1998 by the following companies who were known as core promoters:

- Ericsson Mobile Communications
- Intel Corp.
- IBM Corp.
- Toshiba Corp.
- Nokia Mobile Phones

Later that year the core promoters invited other companies to become SIG members, and they were joined by:

- Microsoft
- Agere Systems (formally a subsidiary of Lucent)
- Motorola

These promoter members and thousands of other Associate and Adopter member companies, all work together to develop Bluetooth wireless technology.

Any incorporated company willing to sign the membership agreement can join the SIG. This agreement commits the company to share key technologies needed to implement Bluetooth. In return they are given a free license to use the Bluetooth wireless technology as long as their products pass the Bluetooth qualification process. The license is important, as parts of the Bluetooth technology require the use of patented technologies so, without the license, a company would not be able to implement Bluetooth legally. Products that pass the qualification process and thus prove that they correctly follow the specification are allowed to use the Bluetooth brand and trademark. In this way it is guaranteed that any Bluetooth devices carrying the Bluetooth trademark will communicate with each other, even if from different manufacturers. [1]

2.1.3 Bluetooth's Aims

But why would a group as diverse as the SIG members cooperate with each other? Simply put, it is good for business. Customers are more willing to adopt a technology that they are able to buy from a number of different manufacturers rather than from a select few. A wider acceptance from customers means a larger market. Shipment of Bluetooth-enabled products in 2005 was forecast to be 316 million units, more than double the previous year, according to In-Stat, a high-tech market research firm in a report [2] released in September 2005. Companies can add Bluetooth support to their own core products making them more useful by allowing them to connect to other devices without cumbersome cables that can break and be lost. As a cable-replacement technology, Bluetooth needs to be as cheap as the cable it replaces, and as easy to use as simply plugging in a cable. This means it needs to be self-configurable, reliable and resistant to errors.

2.2 Bluetooth Protocol Stack

In order to achieve those goals, and since Bluetooth products are made by numerous manufacturers, the Bluetooth Specification is very in-depth and defines not just a radio layer but also software layers for applications to be built upon. This allows for manufacturers to design systems that interoperate with a wide range of devices, and in

fact devices must pass a stringent testing [3] and qualification [4] process to be able to carry the Bluetooth brand.

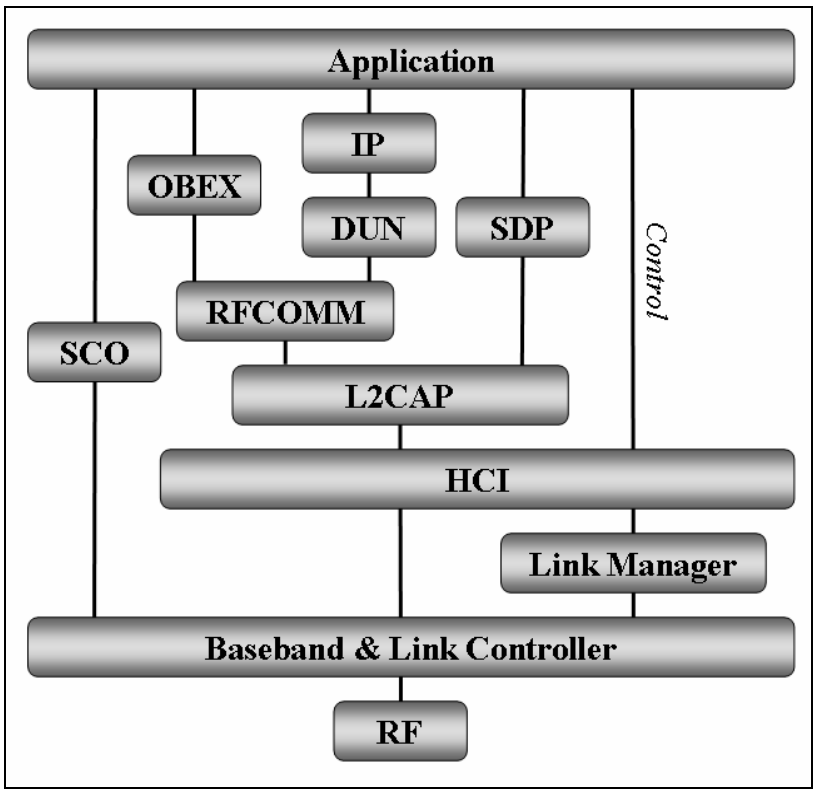

Figure 1: Bluetooth Protocol Stack

The Bluetooth stack is defined in a series of layers, though there are some services that span layers. Each block in Figure 1 represents a chapter in the Bluetooth specification.

2.2.1 Core Lower Layers

2.2.1.1 Radio

At the base of the Bluetooth protocol stack is the Radio Layer [5]. This layer deals with the conversion of data into radio frequency (RF) signals for transmission through the air. Bluetooth operates in the 2.4 GHz Industrial, Scientific and Medical (ISM) radio-frequency band, which is license-free for low-power transmissions in most of the world [6]. This band is shared with other Wi-FI networks and Microwaves making it quite noisy. Bluetooth employs Frequency Hopping Spread Spectrum (FHSS) which is a modulation scheme that uses a narrowband carrier that changes frequencies

in a pattern known to both the transmitter and the receiver. Data is broken up into very small packets and transmitted usually one packet per frequency jump or slot. The ISM band has been divided up into 79, 1 MHz spaced channels and Bluetooth devices hop between these channels at 1600 times per second in pseudo-random sequence. By hopping between the different channels the effect of a single (or a couple) of interfering channels is negated. Other physical aspects of the Bluetooth system are also controlled by the radio module such as signal modulation, power levels and timing.

2.2.1.2 Baseband / Link Controller

Above the radio layer are the Baseband Layer and Link Controller Layer [5]. These two layers are not clearly separated by the specification. The baseband layer's role is to properly format data for transmission to and from the radio module and perform basic error control. It also performs basic piconet management such as signal transmission timing and frequency hop selection. The link controller's role is to establish and maintain the links that are set up by the link manager.

2.2.1.3 Link Manager

The Link Manager [5] translates commands sent by the Host Controller Interface (HCI) layer and acts as a contact between the application and the link controller. Once two devices have set up a link with each other, the link managers on the devices can communicate with each other, probing for each other's communication characteristics using the Link Manager Protocol (LMP). It is responsible for establishing and configuring links.

2.2.2 Host Controller Interface (HCI)

Sitting between the lower layers and the upper layers is the HCI layer [5]. The HCI is defined by the Bluetooth specification as a standard to support Bluetooth systems that are implemented across two separate devices. For example, in a system where a USB Bluetooth dongle (Figure 2) is used, the lower layers of the stack (radio, baseband, link controller and link manager) are implemented in firmware on a small external device that plugs into the computers USB port and the upper layers of the stack are implemented in software on the computer.

Figure 2: Example of a USB Bluetooth dongle

(From http://www.dlink.com/)

These roles are known as the Bluetooth module and Bluetooth host respectively. In some devices such as headsets the module and host portions are combined for simplicity and size. In such a case, the HCI layer need not be implemented.

2.2.3 Core upper layers

2.2.3.1 Logical Link Control and Adaptation Protocol (L2CAP)

The Logical Link Control and Adaptation Protocol (L2CAP) [5] acts as the middle manager between applications and the Bluetooth Link Controller. Once a connection has been established (by the Link Manager) it handles the actual data communication between devices from the higher layers.

Its main functions are [1]:

- Establishing connections across links created by the link manager, or requesting links be established by the link manager.
- Protocol multiplexing between different higher layers allowing multiple different applications to use a single link between Bluetooth devices simultaneously.
- Segmentation and reassembly of packets to and from the lower layers.
- Quality of Service.
- Group Management.

2.2.3.2 Service Discovery Protocol (SDP)

The Service Discovery Protocol (SDP) [5] as defined by the Bluetooth specification is a very important layer in the Bluetooth protocol stack as it allows Bluetooth devices

to inquire what services Bluetooth devices offer or locate a device with a particular service. Services are defined by different profiles (also part of the Bluetooth specification) and represent a feature that is usable by a remote Bluetooth device. SDP consists of servers and clients components, where the requesting devices are a client and the requested device the server. A single Bluetooth device can perform both roles of a SDP-server and -client.

2.2.4 Non-core upper layers

2.2.4.1 RFCOMM

RFCOMM [7] is based on the GSM 07.10 Multiplexing Protocol standard [8], with a few minor differences. It provides a RS-232 serial port emulation which can be used by legacy applications. It also is used by several other Bluetooth profiles for their data transfer. Though RFCOMM is not defined as part of the core specification, it is included in almost all Bluetooth system, except for some embedded systems. RFCOMM's implemntation of the RS-232 serial port emulation is explained in section 2.6.3.1.

2.2.4.2 OBEX

OBEX ("OBject EXchange") [9] is a communication protocol that facilitates the exchange of binary objects between devices, and uses RFCOMM. It is mainly used for the exchange of business cards, calendar appointments and files. OBEX is discussed further in section 2.6.3.2.

2.2.4.3 Bluetooth Network Encapsulation Protocol (BNEP)

The Bluetooth Network Encapsulation Protocol (BNEP) sits on top of L2CAP and allows standard network protocols such as TCP, IPv4 and IPv6 to be transported across Bluetooth links. BNEP provides this encapsulation by replacing the Ethernet header, with a BNEP header and sends this header and the data across the L2CAP layer.

2.2.5 Audio

Audio information between Bluetooth devices such as headset to mobile phones are carried via Synchronous Connection-Oriented (SCO) channels. These channels bypass most of the Bluetooth protocol stack and connect via a direct PCM connection to the baseband layer. This direct connection avoids problems with flow-control across the HCI Layer. [1]

2.2.6 Comparison to OSI Reference Model

To get a better idea of the Bluetooth protocol stack we compare it in Figure 3 to the familiar Open Systems Interconnection (OSI) Reference Model [10] for protocol stacks. Though there is not an exact match between the two stacks, the OSI model is an idea reference model with well partitioned layers and it is useful to relate the two.

Physical Layer: The Physical Layer is responsible for the physical (electrical) interface to the communication medium be it a wire or radio waves. It manages the modulation of the signal and how bits are encoded onto the medium. This covers Bluetooth's Radio and a part of the Baseband as well. [6]

Data Link Layer: The Data Link Layer provides the functional means to transfer data between different entities, framing of data from higher layers and physical layer error control. This overlaps with the Link Controller and the control aspects of the Baseband. [6]

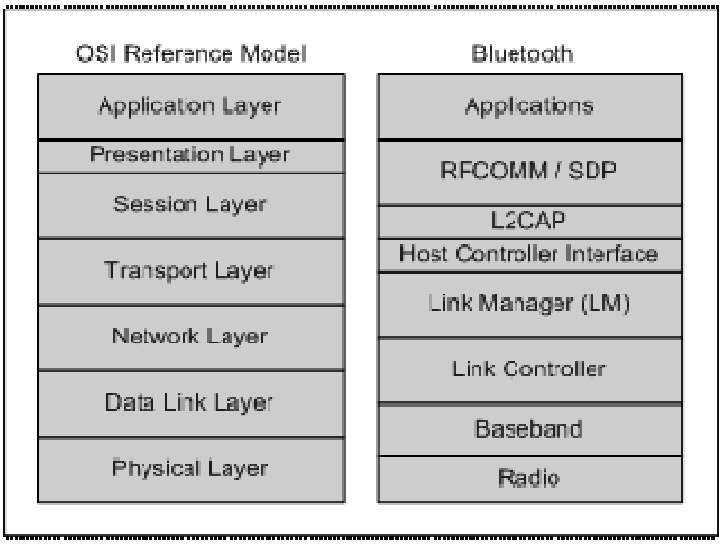

Figure 3: Comparison of Bluetooth stack to OSI Reference Model

From [6]

Network Layer: Actual data transfer across the network is handled by the Network Layer and is independent of the medium and the network's physical topology. This covers the setting up and maintenance of links by the upper part of the Link Controller. It also encompasses most of the Link Manager**.** [6]

Transport Layer: As the name says, the Transport Layer provides for the transport of the data reliably. This includes the higher layers of the Link Manager and covers the HCI Layer which provides the transport mechanism. [6]

Session Layer: The mechanism for managing connections and data flow is controlled in the Session Layer. This is covered by L2CAP and the lower parts of RFCOMM. [6]

Presentation Layer: The Presentation Layer is supposed to provide a common representation of data by resolving differences in data format between different entities. This is the main task of RFCOMM/SDP. [6]

Application Layer: Responsibility for managing communications between the actual host applications is done in the Application Layer. Applications running on Bluetooth hosts communicate with each other at this layer. [6]

2.3 Finding a Device

Before a device is able to make a connection to another device, it needs to discover that device. The process of discovering devices is managed by the Link Controller, which has a number of operational states defined to support the formation of piconets. These states are shown in Figure 4. For device discovery, the three states are inquiry, inquiry scan and inquiry response.

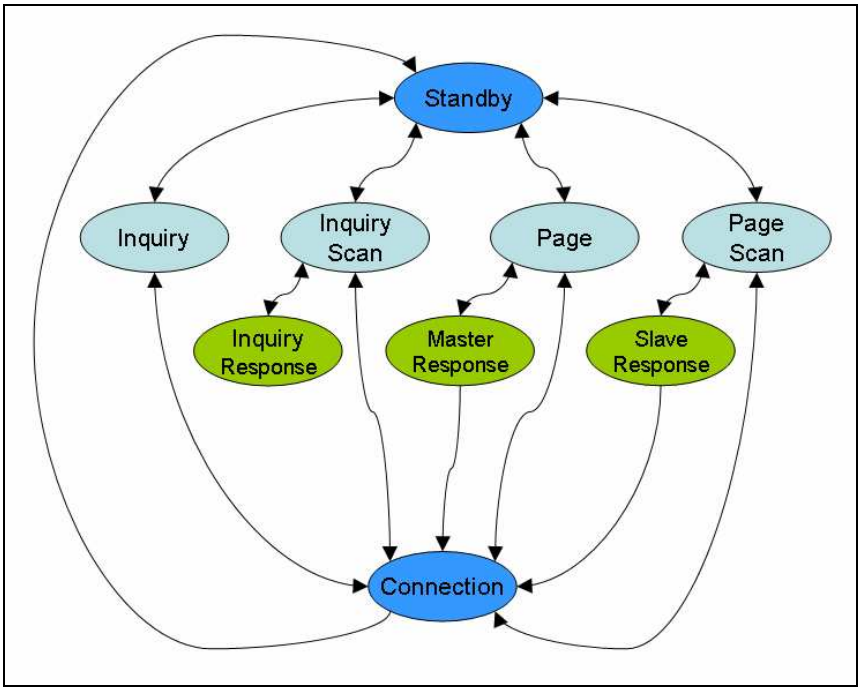

Figure 4: State Diagram of Link Controller

From [1]

2.3.1 Inquiry state

The inquiry state is entered when a device attempts to discover all other devices within range. In this state, the searching device repeatedly transmits an inquiry message on a set of different frequencies in a hopping sequence, whilst it listens for responses. When another device responds its information is added to an internal list of devices found, which is used if a connection is requested later. The inquiry state is continued until the Baseband has received enough responses, when a timeout has been reached or when the host issues a command to cancel the inquiry. [6]

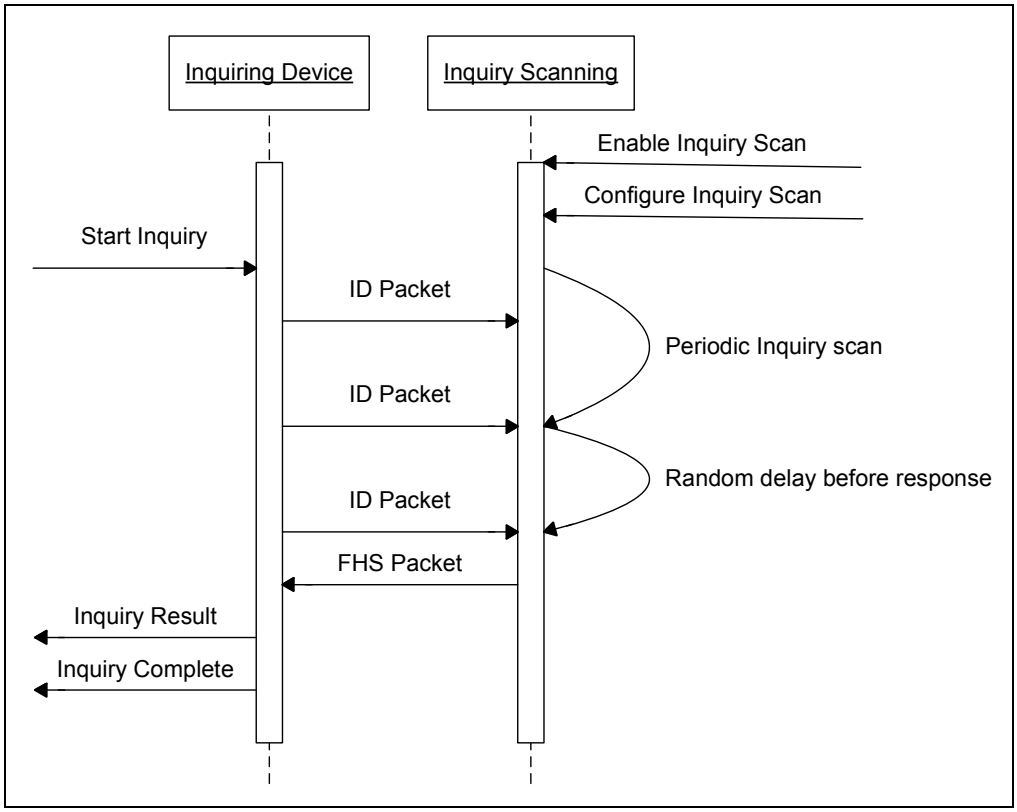

Figure 5: Sequence chart of Inquiry Process

From [1]

2.3.2 Inquiry Scan state

For a Bluetooth device to be discoverable, it has to answer inquiry messages from other devices. This is done by entering an optional inquiry scan state. A device which is discoverable does this periodically (at least every 2.56 seconds) and listens for an extended time compared to the inquiry state. [6] If a device does not want to be located it can be set to be non- discoverable and therefore will not enter the inquiry scan state.

2.3.3 Inquiry Response state

When a device receives a valid inquiry message it will then enter the inquire response state and respond with a frequency hopping synchronisation (FHS) packet. A FHS packet is a special control packet that contains the Bluetooth device's hardware address (called the BD_ADDR) and the Bluetooth clock of the responding device. [6] The Bluetooth clock is a 28-bit internal clock that ticks every 312.5 ms and is used for triggering critical events such as timing the hopping sequence [5]. The BD_ADDR will be used later by the searching device to address the discovered device. [6]

2.4 Connection Establishment

The paging procedure discussed below is used to establish a new connection between two devices. Only the Bluetooth address is required to setup a connection, but if the clock of the other device is known from an inquiry process or a previous connection, the setup procedure can be completed much quicker. The device that initiates the connection is designated the master of the connection. There are four states that devices go through to establish a connection; Page, Master response, Page Scan and Slave response, which are shown in Figure 4 in the previous section.

2.4.1 Page State

This state is used by a connecting device, known as the master, to find and connect to another device, known as the Slave. The master attempts to match the slave's scan hopping sequence by repeatedly transmitting a paging message to the slave in different hop channels. From the slave's BD_ADDR and clock, the master can work out the slave's hop sequence, but will not know at which point in the sequence the slave is as at at that time. To work out where in the sequence the slave is, the master uses an estimate of the clock, which it can base on a previous inquiry or connection. Given that clocks may drift, it is unlikely that the estimate will be entirely accurate so the master transmits the paging messages at twice the usual frequency to compensate. [6]

2.4.2 Page Scan State

As in the Inquiry Scan state, a device will periodically enter a Page Scan state where it will listen for page messages from other devices. It does this by following a particular hopping sequence and listening at each of the different frequencies for 1.28s [6].

2.4.3 Slave Response sub-state

On receipt of a page message from the master, the slave will then enter the Slave Response sub-state. It will then transmit a slave page response message to acknowledge the page message. This response is transmitted 625 μs after the page was first received and on the same frequency that it was received on. It will then wait

for the master to send a FHS packet. When the FHS packet is received, the slave device acknowledges it and will then adjust its hopping sequence and clock to match those of the master's, as given in the FHS packet. The slave will then be synchronised and connected to the master. [6]

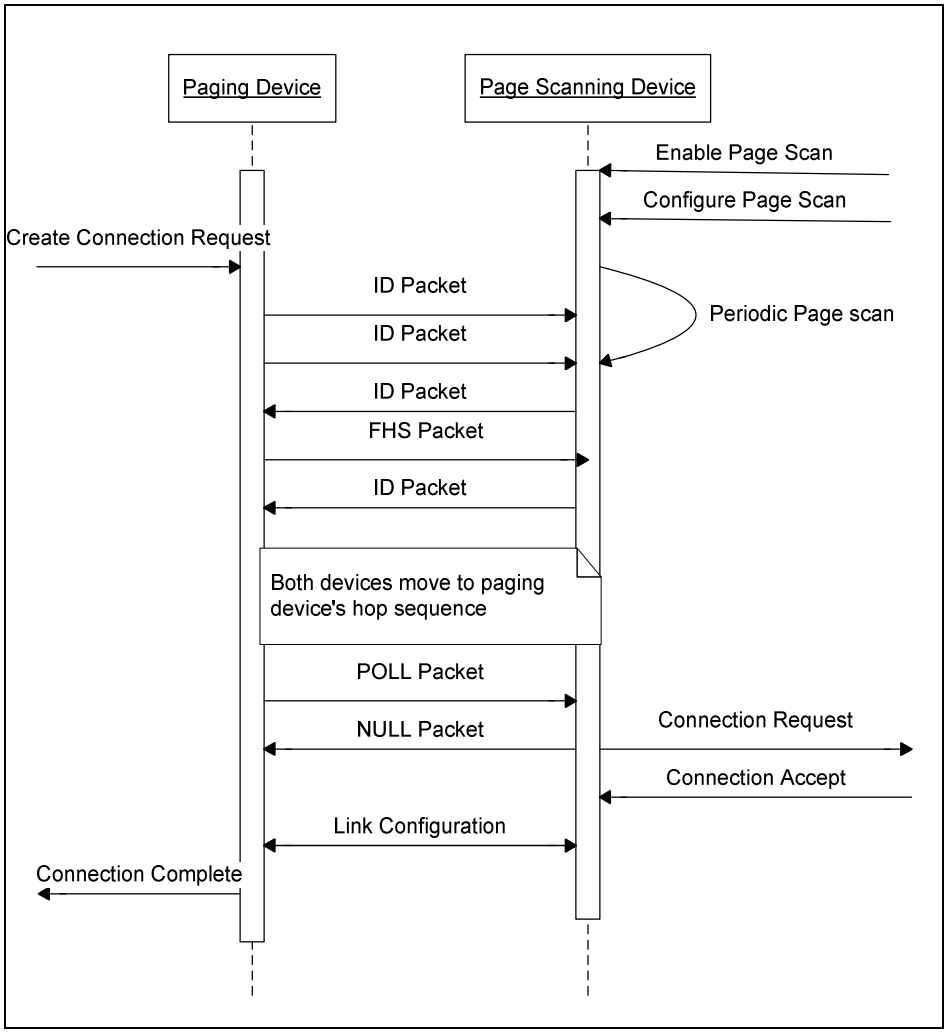

Figure 6: Sequence chart of Paging Process

From [1]

2.4.4 Master Response sub-state

When the master receives the acknowledgement of the page from the slave device it enters the Master Response sub-state and will reply with a FHS packet which contains its BD_ADDR and clock. The master will then wait for the second slave response message. If it does not receive one, it will send another FHS with an updated clock, and keep doing so in each time slot of the hopping sequence until the slave responds or a timeout is exceeded. If an acknowledgement is received, the master will revert to

using its own hopping sequence and enter the connection state. The first packet the master will send to the slave is a POLL packet. The slave may respond with any type of packet. If the slave does not receive a POLL packet, or the master a reply packet the master and slave will return to the Page and Page Scan states, respectively. [6]

2.5 Link Managment

2.5.1 Power-Saving Modes

As Bluetooth is designed to be highly mobile it is important to conserve power and thereby extend the battery life of the host device. Therefore a Bluetooth device may enter a number of power saving modes.

2.5.1.1 Connection Hold

In this state, the device ceases to support data traffic for a defined period of time. This is to relinquish bandwidth for other operations such as scanning, paging and inquiry or for the low power sleep mode. After the hold period expires the device re-synchronises to the master and listens for data again. [6]

2.5.1.2 Connection Sniff

A device is given a period time slot and periodicity to listen for traffic. The device will enter a low power mode deactivating the antenna and only activate it again during its slot. If it receives a packet during this slot it will carry on listening for further packets until no further packets are received and a time-out period has expired. It will then resume the low power mode and wait for its next sniff slot. [6]

2.5.1.3 Connection Park

A slave device can give up its membership of a piconet and only listen to traffic occasionally. The device can then enter a low power mode to conserve battery power, only needing to power up periodically to listen to a “beacon” signal to remain synchronised with the master. [6]

2.5.2 Role Switch

In some cases, a device may find that the master or slave roles it plays in the piconet it is in are not appropriate. This could be the case if a device wants to join an existing piconet by paging, since by definition, the device initiating the connection would becomes the master with the other device a slave to it. This scenario would happen if a device was offering the service as a network access point (NAP). It would need to remain the master so as to offer the service to multiple devices simultaneously. Another scenario is when a slave needs to set-up a new piconet with it as master and its previous master as its slave. In both these cases a role switch is required. Either a master or a slave is able to request a switch in roles with respect to the other device, though this request may be refused if necessary. [6]

An example from [6] is modified to show what happens when a device connects to a device that needs to stay a master. A role switch takes place in multiple stages once the request has been accepted. To avoid confusion the device that needs to stay a master is called Device A and the connecting device is called Device B.

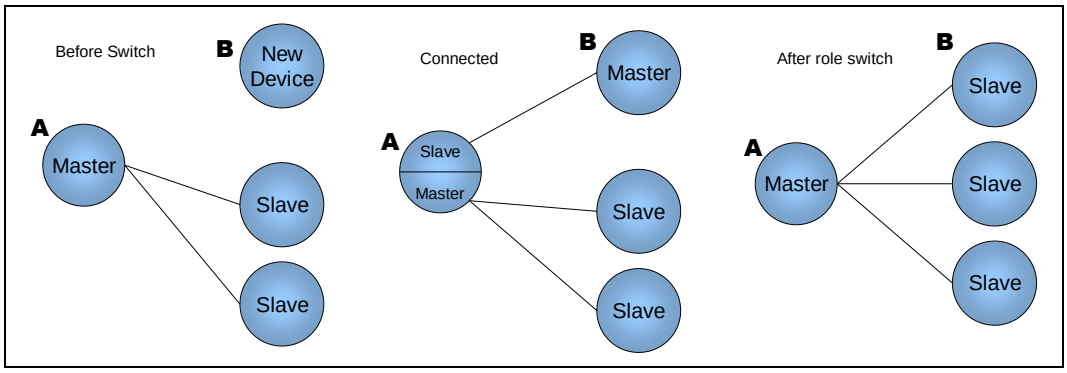

Figure 7: Role switch so hotspot stays master

1. Device B connects to Device A normally using a paging sequence.
2. Device A performs dual-roles, master to its existing piconet and slave to the connecting Device B, forming a scatternet.
3. Device A requests a role-switch from Device B.
4. Both devices switch their transmit and receive slots, but still use Device B's hopping sequence.

5. Device A sends Device B a FHS packet that allows Device B to synchronise to Device A's clock. This is similar to a page, except rather than happening on the page hopping sequence it takes place on Device B's hopping sequence.
6. Both devices switch to the hopping sequence defined by Device A's parameters.
7. Device A verifies the connection with a POLL message, just as done in a normal page.

2.6 Transferring Data and Audio Information

2.6.1 Physical Links

Once two Bluetooth devices have linked to each other, two different types of data packets can be exchanged at the lowest layer, Asynchronous Connection-Less (ACL) links and Synchronous Connection Oriented (SCO) links.

2.6.1.1 Asynchronous Connection-Less (ACL)

An ACL link exists as soon as a relationship has been established between a master and a slave. A master forms many ACL links with different slaves but only a single link with each slave. Data is transmitted sporadically as it is available from higher layers in the protocol stack in a manner comparable to a packet switched network. The master may choose the slave to transmit to and receive data from on a per-slot basis and can therefore offer both asynchronous and isochronous transmissions. This link is used to carry all data from the L2CAP and Link Manager layers. A slave device only responds in a slave-to-master slot if the master communicated with it in the preceeding master-to-slave slot. If it is not sure it was communicated with, it does not respond to avoid possible interfering with the slave that was communicated with. [1]

2.6.1.2 Synchronous Connection Oriented (SCO)

SCO links on the other hand are quite different as they provide a symmetrical link between the master and slave with reserved channel bandwidth and a set periodic exchange. It is an example of a circuit switched network and is used for time-bound data such as voice. A master can support up to three SCO links to one or more slaves

at any one time. The link managers on the master and slaves agree on and setup the timing information. The master then regularly transmits packets to the slave. All ACL packets are scheduled around SCO packets. Link manager packets take precedence over an SCO link, though, to enable the link manager to shut down a link when available bandwidth is exhausted. [1]

Specific slots are reserved for SCO communication to a particular slave. Reserved slots may be every slot-pair (a slot-pair being the master to slave communication and the slave's response), every second slot-pair or every third slot-pair, negotiated at link setup. Because the least infrequent option is every third slot-pair the maximum number of SCO links in a single piconet is three. Reserved SCO links will continue to transmit through scanning, paging and inquiry using up scanning time as the SCO slots interrupt the scans. Page scans are worst affected as they invovle the longest sequence of packets. To counteract this while SCO links are established, scanning frequency is increased in proportion to the number of established links. [1]

2.6.2 Logical Links

Above the ACL link is the L2CAP layer which multiplexes all user data from the applications above. The L2CAP layer abstracts the master/slave relationships within a piconet to make communication appear to be peer-to-peer. This layer is typically implemented in software by drivers running on the host computer. Since there is only one ACL link between any two devices, it needs to multiplex a number of different higher protocols. This is done by creating virtual channels between devices. These can be connection-oriented channels, such as between an application on one device to an application on another device, or connectionless channels, which are usually from a master to a number of slaves.

To distinguish between different channels each one is assigned a different Protocol / Service Multiplexer (PSM) number. PSM values come from two separate ranges. The first range is assigned by the Bluetooth SIG and is defined in the Bluetooth Assigned Numbers database [11] available on the Bluetooth website. The second range is dynamically allocated and is used with the SDP server. These could be used to support multiple implementations of a single protocol. [5]

Multiple channels can be set up between different host applications but there is at least one channel called the signalling channel which is setup by default when the ACL link is created. Though the terminology of master and slave is not used, all communication channels are between the master and a slave, as slaves cannot communicate directly.

2.6.3 Application/User Data

2.6.3.1 RFCOMM

RFCOMM emulates a serial port over an L2CAP connection. This is used as the data transport for many Bluetooth profiles and can be used by legacy applications. RFCOMM is based on the ETSI TS 07.10 [8] standard, though only a subset of that standard is used and there are adaptations that are specific to Bluetooth. ETSI TS 07.10 is an asymmetrical protocol used by GSM cellular phones to multiplex streams of serial data into a single signal. It is a simple transport protocol with provisions for emulating the nine circuits of RS-232, emulating the serial cable line settings and status signals.

RFCOMM provides multiple concurrent connections by relying on L2CAP to handle the multiplexing of packets over a single connection. The Baseband Layer provides reliable and in-sequence delivery of a stream of data so RFCOMM performs no error correction. RFCOMM can emulate up to 30 serial ports simultaneously. However the maximum of ports may be limited in each implementation. RFCOMM uses the concept of channels, each of which has a Data Link Connection Identifier (DLCI) associated with it, which identifies a specific connection between a client and a server application. A dedicated control channel is assigned a DLCI of 0 (zero). the control channel has been established, other channels must be setup before data can be transferred.

RFCOMM is symmetrical and communicates by sending TS 07.10 frames over L2CAP. These frames become the data payload in a L2CAP packet. Since RFCOMM frames are carried in a L2CAP packet, an L2CAP connection must exist before an

RFCOMM connection can be established. For this, RFCOMM has a reserved PSM value for L2CAP. This is defined as 0x0003 in the Bluetooth core specification and any L2CAP frame with this PSM will be sent to the RFCOMM entity for processing. Bluetooth may optionally have multiple emulated serial ports running on different endpoints. To do this the RFCOMM entity must be able to run multiple TS 07.10 multiplexer sessions. Each of these sessions would use its own L2CAP channel ID.

2.6.3.2 OBEX

OBEX is designed to allow the exchange of binary data (objects) between devices and could be described as a binary HTTP protocol. Objects may be many things such as files, photos, calendar entries (in the iCalendar format [12]) and business cards (in the vCard format [13]). OBEX is based on IrOBEX [14], a specification defined by the Infrared Data Association (IrDA). The protocol is specially optimised for ad-hoc networks where connections are short and unprompted.

The OBEX client/server architectures allows a client to pull data from a server and push data to a server. For example, a cellular phone may push a business card to an OBEX server running on another cellular phone, or a PDA might download a file from a laptop. More complicated scenarios of devices synchronising calendar events between each other are also provided for.

The OBEX specification includes an object model, which describes data objects and provides a standard format for transferring those objects, and a session protocol for transferring requests and responses between devices. [1]

For OBEX over RFCOMM, the following requirements must be satisfied:

- Each device must be able to act as either a clients or a server,
- a separate RFCOMM channel must be used for each simultaneously running server, and
- OBEX servers must be able to register their service records in the service discovery database, whose format is specified in the Bluetooth profiles.

OBEX packets are carried in RFCOMM frames. Since RFCOMM frames are undifferentiated byte streams, the receiving OBEX application will need to use the length field specified in the OBEX packet to determine packet boundaries.

Three separate application profiles that use OBEX are defined by the Bluetooth SIG including the main Generic Object Exchange Profile [15].

The Synchronisation Profile [16] specifies a method of comparing two collections of objects, determining the differences between them and then transferring the missing objects so as to make the collections identical. The profile says that the IrMC synchronisation specified by IrDA must be used.

The File Transfer Profile [17] allows for binary files, such as photos and word processor files, to be sent and received between Bluetooth devices.

The Object Push Profile [18] is a special case of the File Transfer Profile that allows for unidirectional transfer of objects. At a minimum an implementation should be able to transfer a business card but it need not be limited to that.

2.7 Disconnecting

2.7.1 L2CAP

There are two ways that a L2CAP channel can be disconnected; by a higher level protocol requesting that the connection be closed or through a time out.

2.7.1.1 Higher Level Protocol or Application

When a protocol or service has finished transferring data, it can request that the L2CAP layer disconnect the connection. L2CAP will then send a Disconnect Request packet across the L2CAP signalling channel to its peer L2CAP. Once a disconnection request is issued, L2CAP will stop sending and receiving data on that channel. All queues of outgoing data are emptied, and any new outgoing data or incoming data that is received will be discarded. The device receiving the Disconnection Request will discard all data that is queued to be sent since the device that sent the disconnect

request will already be discarding any data it receives on that channel. It then responds with a Disconnection Response message which confirms that it has received the disconnect request message. When the L2CAP that requested the disconnection receives the disconnection response it will inform the upper protocol layer that the disconnection was completed successfully. If the requesting L2CAP does not receive a response it will inform the upper protocol layer that the request timed out, whereupon the higher layer may elect to re-issue the disconnection request.

2.7.1.2 Time out

L2CAP channels can also be disconnected as a timer expires. A Response Timeout Expired (RTX) timer is started every time L2CAP sends a signalling request to a remote device. Though the length of the RTX timer is implementation-specific, it must initially be a value between 1 and 60 seconds, and can depend on how long the baseband would attempt to retransmit the packet. If the timer expires, a duplicate signalling request could be sent or the channel may be closed. If a duplicate request is sent then, the RTX value will be set to at least twice the previous timeout length. The number of retransmission attempts before a channel is disconnected is implementation-specific, but the maximum elapsed time between the start of the initial timer and the closing of the channel if no response is received is 60 seconds. For each outstanding signalling request there is one RTX timer.

If the remote endpoint indicates that it needs more time to process the request, the Extended Response Timeout Expired (ERTX) timer may be used instead of the RTX timer. The ERTX's minimum and maximum values are 60 and 300 seconds respectively but once again the actual value used is implementation-dependent. When the ERTX timer expires a duplicate request may be sent, or the channel may be disconnected. If a duplicate request is sent the process starts again with a new RTX timer and runs as described above.

2.7.2 RFCOMM

As with L2CAP, there are two ways of disconnecting a RFCOMM channel; by a higher layer protocol requesting that the connection be closed or though a time out.

2.7.2.1 Higher Level Protocol or Application

To shutdown an RFCOMM connection a Disconnect Command is sent. When the last data link has been closed a disconnect command is sent on the command channel to shut down the multiplexer. Which ever device shuts down the multiplexer is the responsible for disconnecting the L2CAP channel that was used.

2.7.2.2 Time out

RFCOMM also has a 60 second timer which is started whenever a command is sent. If there is no answer by the time the timer has expired the connection will be closed. This is different to the TS 07.10 specification, which will resend when the timer runs out. Since RFCOMM uses the Baseband layer; (which provides a reliable link) if the frame is not acknowledged the first time it is unlikely to be answered the second time. If RFCOMM times out and disconnects a connection it must send a disconnect command on the same channel that it was initially connected on in case the other peer comes back into range and still considers the connection established.

2.8 Securing communications

Security within Bluetooth encompasses three main areas: authentication, authorization and encryption. Authentication is the process by which members of a piconet are able to prove their identities to each other. Once a device has been authenticated, its identity can then be used for determining a client's authorization to access the various services on a server. Encryption is used to secure the communication between devices and prevent eavesdropping (even from other piconet members). These security processes are implemented at a number of levels in the Bluetooth specifications and consequently security is often termed a "cross-layer function". [6] Baseband uses a modified SAFER+ algorithm for device authentication The Link Manager configures link level security, such as encryption and authentication modes. HCI provides procedures reporting security-related events to the host, and responses from the host. The Generic Access Profile defines terminology and covers user-level procedures that all Bluetooth devices implementing profiles should follow. An optional security architecture is suggested by a white paper [19] on security provided by the Bluetooth SIG that provides a framework for implementing service level security. [1]

2.8.1 Security Modes

Three different security modes that may be implemented are defined in the General Access Profile [5].

2.8.1.1 Security Mode 1 – Non-secure

Devices will not attempt to authenticate or perform any other security procedures. Supporting authentication is optional by devices that only support Security Mode 1. [6]

2.8.1.2 Security Mode 2 – Service level enforced security

A non-secure ACL link may be created between two devices. Only when an L2CAP connection is requested are security procedures established. The service using the L2CAP channel decides what security procedures are needed. [6]

2.8.1.3 Security Mode 3 – Link level enforced security

Security procedures are initiated when an ACL link is created. A device in security Mode 3 may refuse a connection request based on a setting in the host. [6]

Apart from these specific security modes, there are other modes a device may be configured into to increase security. A device may be set to a non-discoverable mode, which means it will not enter an Inquiry Scan state and thus devices will not be able to find it while searching. Only devices that already know its BD_ADDR would be able to connect. For maximum security, a device could be placed into a non-connectable mode. Other devices would not be able to connect to it, as it would not enter the Page Scan state.

2.8.2 Authentication

Bluetooth uses a challenge and response action for authentication. A variant of the SAFER+ (Secure and Fast Encryption Routine) [20] algorithm is defined in the Bluetooth specification. The cipher was designed by Cylink Corporation as a contender for the Advanced Encryption Standard (AES), a U.S. Government cryptography standard. The algorithm allows for a device, called the verifier, to compare a secret key that it has with that of another device that claims it has the same

key, called the claimant, without either actually transmitting the key itself. Once the encryption engine has been initialized, it takes four inputs, a number to be encrypted and decrypted that is passed in the clear, the master's BD_ADDR and the master's slot clock and the shared secret key. The verifier sends the claimant a randomly generated number which it then encrypts using the known information and its secret key. This is then sent back to the verifier that checks the response against it own calculations. If they match then both devices must share the same secret key. This process is shown in Figure 8. [1]

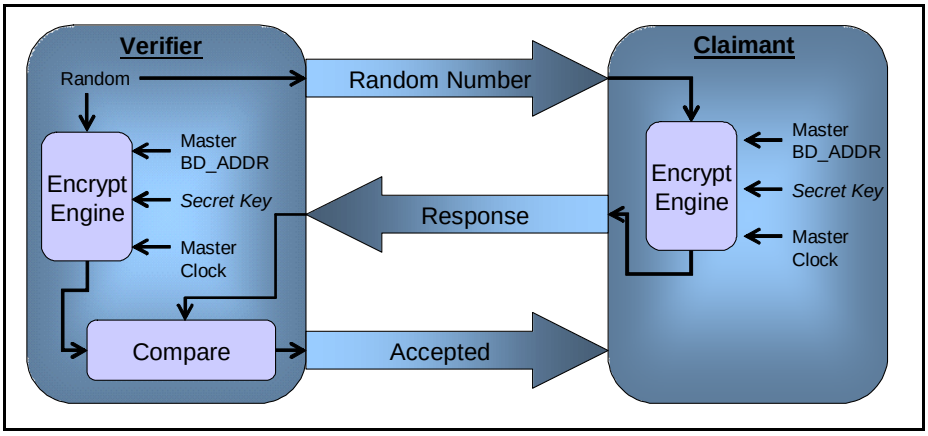

Figure 8: Authentication using the Bluetooth encryption engine

From [1]

2.8.3 Authorization

In a client/server scenario, a client connects to a server and requests the use of a particular service. Once a client has authenticated itself to the server, it can be granted access to the requested service based on a form of authorisation level. This authorisation could be automatically granted for all services, or only a subset of services. Devices can also require that the user gives authorisation for the use of a particular service. The Bluetooth security white paper proposes a security architecture that may be used to implement security mode 2 on a Bluetooth device. A database must be used to record devices that have been authorised and which access levels they have been granted for the different services. As services can be implemented at different layers in the stack, different protocols will need to be able to access this security information. For instance, OBEX would handle security related to file transfers and object synchronisation, RFCOMM would handle Dial-Up Networking (DUN) or L2CAP would handle cordless telephony. To enable consistent access to

the information by any layer, a middle-man, the security manager, resides in the host and handles the security queries from the different layers, as illustrated in Figure 9.

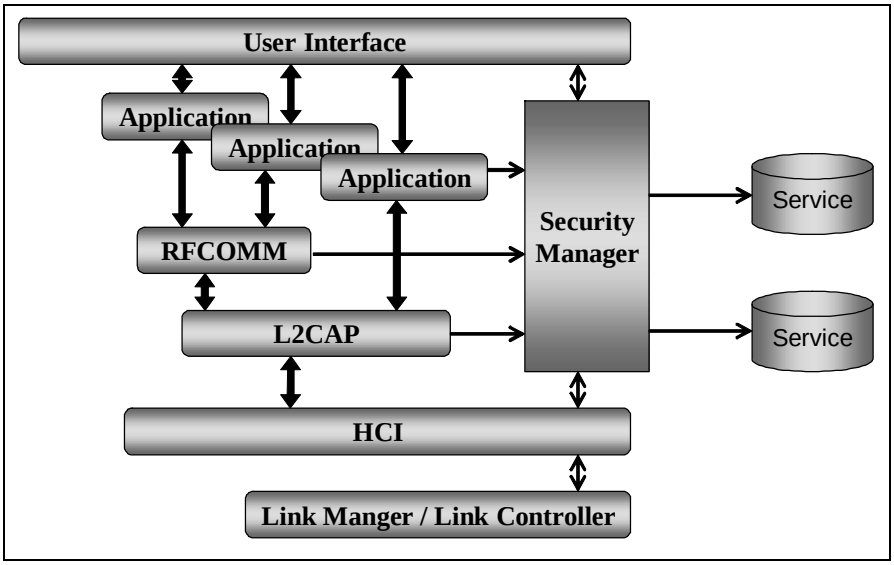

Figure 9: Security architecture with security manager

From [6]

A typical security scenario is suggested by Morrow [6]:

1. Another device requests an L2CAP connection.
2. The L2CAP layer queries the security manager.
3. The service manager looks up the requested service in the service database.
4. The security manager looks up the requesting device's BD_ADDR is in the device database to check access authorization.
5. The security manager starts the authentication and (if requested) encryption procedures within the link manager though the HCI.
6. The Link Manager gives a favourable response.
7. The L2CAP layer finishes the connection process.

2.8.4 Encryption

After two devices have authenticated themselves to each other and agreed on a link key, there are two more steps they need to takes before traffic can be encrypted; negotiating an encryption mode, and negotiating the key size. Thereafter, encryption can be started.

The encryption mode can be any one of the following: no encryption, both point-to-point and broadcast packets are encrypted, or only point-to-point packets are encrypted. When a device requires encryption, it will request that one of these encryption modes be used on the channel. If the other devices accept this mode, then they continue the process to negotiate a key size; if the mode is not accepted then the requesting device will request a different encryption mode.

Since strong encryption is regulated in some parts of the world, devices that are limited to use less than the full 128-bit key length need to be accommodated. Therefore both devices must agree on a key size before encryption can be enabled. The requesting device starts the negotiation by requesting the maximum key length it can use. If the key length is not within the capabilities of the other device it will not accept the key size and the requestor will try again with a shorter key. It will keep trying with ever shortening keys until the other device accepts.

Negotiating an encryption mode and key size does not automatically enable encryption on the link. Encryption has to be switched on for a link, and can be turned off and on again as needed by either device without re-negotiating parameters.

2.9 Profiles

In Bluetooth, services are defined by profiles, with each profile document defining how a particular service can be implemented, which parts of the Bluetooth stack are required and how the service will interact with those various parts. Profiles are organised into groups as shown in Figure 10, forming a hierarchy with profiles building upon and inheriting features from the profiles beneath them, with all profiles ultimately depended on the General Access Profile.

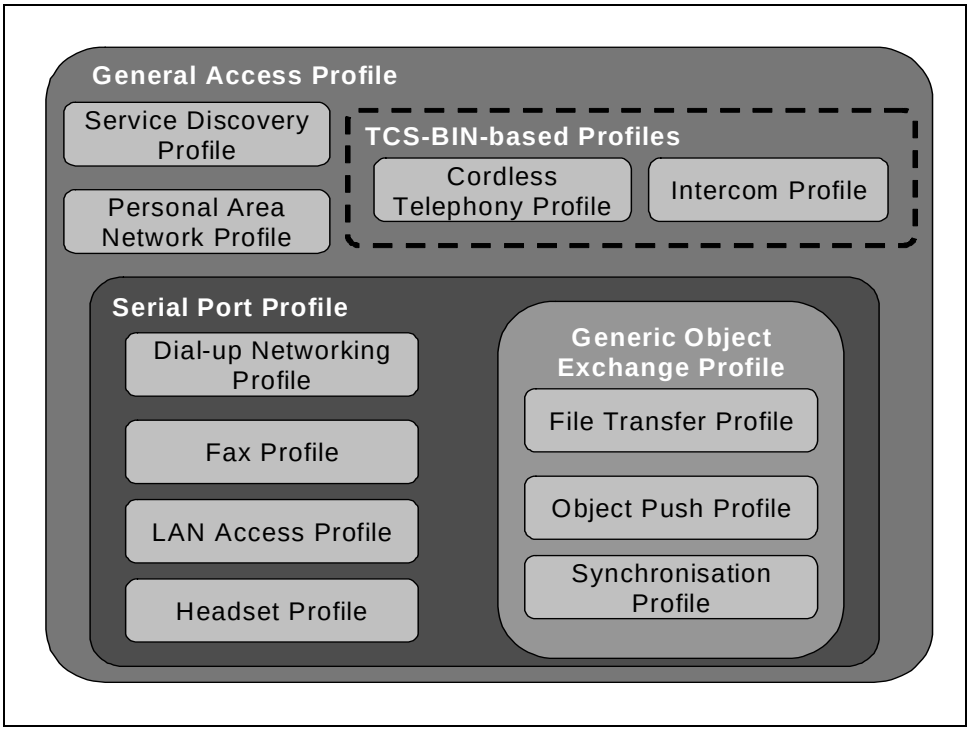

Figure 10: Dependencies of various service profiles

From [1]

For example, if we looked at the "File Transfer Profile" we can see that it relies on the "Generic Object Exchange (OBEX) Profile", which in turn relies on the "Serial Port Profile", which ultimately relies on, like all profiles, the "General Access Profile".

Each Bluetooth profile includes [1]:

- A short description of the purpose of this profile
- User scenario showing how the user will see the profile
- Indication on what decencies on other profiles it has
- Whether or not it is dependent on any other specifications, either by Bluetooth or another organisation
- Which parts of the Bluetooth protocol stack the profiles uses
- A description of how it makes uses of the stack, layer by layer
- A description of the Service Record, if required

Features can be defined as either as optional or mandatory by the profile documentation.

2.10 Service Discovery

As computing moves towards a more network-centric form, finding and making use of services, rather than of devices, becomes much more important. A service can include common examples such as printing or faxing, as well as other ways of information access such as teleconferencing or network bridging. A standard way for discovering these services is needed, and there are other issues that need to be taken account of; accessing the service (such as finding and obtaining the protocols), methods and drivers need to access the service, access control, advertising of the service, billing for usage of the service, amongst others. This problem is not unique to Bluetooth and a number of different solutions have been proposed, such as Service Location Protocol (SLP) [18], Jini [21] from Sun Microsystems, UPnP's Simple Service Discovery Protocol (SSDP) [11] and Salutation [22], each addressing differing aspects of service discovery and, with the exception of Salutation, designed with wired rather than wireless networks in mind [23].

A Bluetooth network is very different from a LAN or wired network as devices connect to each other rather than to a central network and these connections change quickly as devices move in and out of radio range. In a LAN network (even ones provided wirelessly via 802.11) a connection to a printer, once found and setup, is constant and stays in place for a long period of time. Bluetooth is designed so that one can instantly use a printer once it has been found without pre-configuring any settings and once one has finished using the printer, one would merely walk away and the printer would be forgotten. To achieve this, SDP was optimised for the dynamic nature of Bluetooth communications. It provides the means for devices to discover which services are available on other devices and determine the characteristics of those services.

2.10.1 Client/Server Model

SDP uses a client/server model, but one device can perform both roles. An SDP server is any Bluetooth device that offers any number of services to other Bluetooth devices. Each SDP server maintains its own database, where information about the services is

stored, as there is no centralised database. An SDP client is any device that accesses a SDP server and queries it about the services that the server device offers.

2.10.2 SDP Database

The SDP database is a collection of service records that describe all the services that a particular Bluetooth device can offer. SDP provides the procedures and functions for a SDP client to discover the existence of these services as well as their attributes. These attributes include information such as the class or type of the service and the protocol information needed to utilise the service.

2.10.2.1 Service Attributes

A service record consists of a list of service attributes, where each attribute describes a unique characteristic of the service. Each attribute consists of two components, a 16-bit identifier and a value. The attribute value is a variable length field which can contain text strings, Boolean values or integers. The attribute identifier determines the length and type and the service class of the service record. [1, 5]

2.10.2.2 Service Record

Each service is fully described by a single Service Record. This service record is made up of a list of service attributes. Each service record is identified by a service record handle, which is a 32-bit unsigned integer. The handle is unique within each SDP server and is independent of the service record it represents. Since SDP does not provide a mechanism for notifying clients when a service is added or removed, when a linked service record is added or deleted, the server must ensure that a record handle number is not reused by another service within the lifetime of any open L2CAP connections. In all SDP servers, the record with handle 0x0000000 represents the SDP server itself. This service record contains all the attributes of the server and the protocols it can use. Table 1 is an example of a service record for the OBEX Object Push service. This is explained below. [1, 5]

Table 1: Service Record for OBEX Object Push

ID	Name	Type	Value
0x0000	ServiceRecordHandle	UINT32	0x00010000
0x0001	ServiceClassIDList	UUID16	*OBEXObjectPush*
0x0004	ProtocolDescriptorList		
	Protocol0	UUID16	*L2CAP*
	Protocol1	UUID16	*RFCOMM*
	ProtocolSpecificParameter0	UINT8	Channel #
	Protocol2	UUID16	*OBEX*
0x0009	BluetoothProfileDescriptorList		
	Profile0	UUID16	*OBEXObjectPush*
	ProfileSpecificParameter0	UINT16	Version #
0x0303	SupportedFormatsList		
	Format0	UINT8	*vCard2.1*
	Format1	UINT8	*vCard 3.0*
	Format2	UINT8	*vCal 1.0*
	Format3	UINT8	*iCal 2.0*
	Format4	UINT8	*vNote*
	Format5	UINT8	*vMessage*
0x0000 + lang. offset			"OBEX Object Push"

The first item is the *ServiceRecordHandle*, a number that is assigned by the SDP server and is unique to the service that this record advertises. *ServiceClassIDList* which lists the other services that form part of this service. In this case there is only one, namely the *OBEXObjectPush*. The *ProtocolDescriptorList* lists the protocols needed by the service. OBEX Object Push needs RFCOMM, therefore L2CAP and OBEX itself too. Within the list is a *ProtocolSpecificParameter* (after RFCOMM) that indicates which RFCOMM channel the service is listening on.

The *BluetoothProfileDescriptorList* lists the profiles that this service supports. OBEX Object Push only supports the OBEX Object Push profile, which is the only profile listed here. It is also followed by a *ProfileSpecificParameter* which indicates which version of the profile is supported. Because there are many different binary data "objects", such as vCard, iCal and vNote, that different hosts could support, the OBEX Object Push's service record contains a *SupportedFormatsList* which lists all the different formats that this service supports. Finally a textual name for the service is given. A service record can provide the name in different languages by storing the value at a language-dependent offset.

2.10.2.3 Service Class

Each service record is an instance of one or more service classes. A service class defines all the attributes contained in a service record, the intended use of the attribute and the format of the attribute values. Each service class is defined by a unique number represented as a Universally Uniquely Identifier, or UUID (explained in section 2.10.3.1). The ServiceClassIDList attribute in the service record lists the service class(es) that the instance inherits. Typically each service class is a sub-class of another class and all are listed in the list. A service sub-class definition contains all the attributes from its super-class as well as additional attributes specific to the sub-class. [1, 5] The Bluetooth specification [5] has an example (shown in Figure 11) of what a service class list might look for a colour PostScript printer with duplex capabilities. The specification of course warns that this example is only illustrative and may not be a practical class hierarchy.

```
PrinterServiceClassID
      -> PostScriptPrinterServiceClassID
            -> ColourPostScriptPrinterServiceClassID
                  -> DuplexColourPostScriptPrinterServiceClassID
```

Figure 11: Example of a possible Service Class Hierarchy

2.10.3 Searching for a Service

When an SDP client knows the service record handle for a particular service it is easy for it to request values of specific attributes from that service record, but how does it find out the handle for a particular service in the first place? The Service Search transaction allows a client to request the service record handle(s) of service records whose attributes values match given parameters. The transaction can only search for attributes whose values are UUIDs, and cannot search for any given attribute value. [1, 5]

2.10.3.1 Universally Uniquely Identifier (UUID)

UUIDs [24, 25] are 128-bit numbers that are guaranteed to be unique across all time and space. They can be independently created in a distributed fashion and no central registry authority of assigned numbers is needed. The Bluetooth specification and

profile specfication define UUIDs for the attributes that are needed for the various Bluetooth protocols and profiles. But since UUIDs can be independently defined, manufacturers are able to allocate their own UUIDs when defining new services. [1, 5]

Since UUIDs that are defined by Bluetooth are expected to be frequently used, they can be shortened to 16- or 32-bit representations to conserve space storing them and resources transferring them. These shortened representations are possible as all pre-allocated Bluetooth UUIDs are based on the same Bluetooth Base UUID. The Bluetooth Assigned Numbers document [26] defines the Bluetooth Base UUID as 00000000-0000-1000-8000-00805F9B34FB. The 16-bit or 32-bit number is multiplied by 2^{96} and added to the Bluetooth Base UUID to give the full 128-bit UUID. A 16-bit UUID may be converted to a 32-bit UUID by simply zero-extending the 16-bit value to 32-bits. Only UUIDs of the same size may be directly compared. If the lengths of two UUIDs differ, the shorter UUID must be converted to the longer representation before comparison. [1, 5]

2.10.3.2 Service Search Pattern

When an SDP client searches for a particular service or browses for all services, it sends the SDP server a list of UUIDs called the service search pattern. The service search pattern matches a given service record if each UUID in the pattern is contained in the service record as one of the record's attribute values. These matches do not need to be in order, nor attached to any particular attribute. [1, 5]

2.10.4 Browsing SDP Records

SDP clients may also browse the service records on an SDP server, as opposed to searching for a particular service record by specifying characteristics of that service. Browsing allows an SDP client to discover a service record without any prior information about the service. To make it easier to browse the available services, the service records are organised into a hierarchical tree structure. The manufacturer can decide on the construction of this hierarchy and decide which services will be browseable. Figure 12 shows a hypothetical example of such a hierarchy. The top level of the browsing tree is called the Public Browse Root. In this example the

manufacturer has opted to separate the services into three different groups; Organiser, containing calendaring and business card services, Networking, listing data transfer services and the Audio group, where audio gateway services such as HeadSet are placed. [1, 5]

2.10.5 Service Records

2.10.5.1 RFCOMM Service Record

Bluetooth devices that offer services based on RFCOMM must have an entry in the SDP database which gives information on how to connect over RFCOMM. At a minimum, the RFCOMM channel numbers and service type must be included. RFCOMM channel numbers are dynamic as the channel number is assigned when an application opens an RFCOMM channel to listen on. These channels will be advertised in the service record within the *ProtocolDescriptorList* attribute. The service class is stored as part of the *ServiceClassList* to identify the type of service that is running on that channel. Many services also have additional parameters in that service's service record. For example, since RFCOMM is based on L2CAP; the L2CAP protocol must also be indicated in the service record wherever RFCOMM is used. [1, 5]

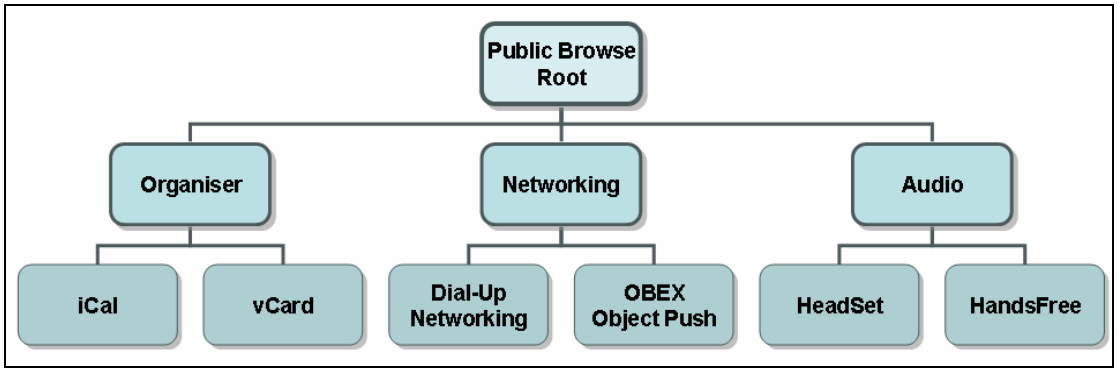

Figure 12: An example of a hypothetical SDP browsing hierarchy

2.11 Summary

Bluetooth is a short-range, low-power wireless communication protocol that was designed to facilitate the communication between mobile devices such as cellular phones and laptop computers. It is managed and marketed by the Bluetooth SIG, a

trade association consisting of the industry leaders in the technology. Bluetooth aims to be as simple to use as the cable it replaces and is therefore designed to be self-configurable, reliable and resilient to any errors. It uses FHSS to enable it to operate reliably within in the “noisy” 2.4 GHz IMS band, allowing it to operate even when there are high levels of interference. This pseudo randomly hopping sequence is generated based on the hardware address of the Bluetooth device which has been set during its manufacture. The hardware address of a Bluetooth device plays a very important part in the communication with the device also being used during authentication and encryption of communications.

The frequency hopping, which happens at 1600 times a second, also makes finding and connecting to devices a little more complex as there is not a single channel an initial connection can be made on. Bluetooth uses special hopping sequences called inquiry and page states to allow device to find and connect to each other respectively.

The different layers of the Bluetooth protocol can be divided into roughly three levels. The radio, ACL and SCO links are the lowest communication channels layers and implemented by the firmware on a Bluetooth chip. The middle layers, L2CAP and RFCOMM, are the transport layers, moving bits and bytes between devices. Application layers such as OBEX and SDP transport user data.

Since Bluetooth is a mobile ad-hoc network, a method of locating services is absolutely essential. Bluetooth uses a client/server-based service discovery protocol that allows any device to query another device to discover what services it can offer. It uses a service hierarchy and pre-assigned UUIDs to allow devices to quickly search for particular services or those with certain characteristics.

As Bluetooth is a wireless communication, eavesdropping is a great concern. Bluetooth provides three levels of security called Security Modes, ranging from open-to-all to only authorised devices may connect. Most layers provide means for authentication, authorisation and encryption.

Chapter 3:

PROPOSAL FOR BLUETOOTH HOTSPOTS

This chapter describes the underlying architecture of the project. We present our proposal for a Bluetooth hotspot and examine two different implementation approaches. BlueSpot, the software components of the proposed Bluetooth hotspot are introduced, as well as the connection framework between these components and other hotspots.

3.1 Introduction

As was introduced in Chapter 1, to allow Bluetooth devices that are out of radio range to communicate with each other we envision setting up a collection of devices that would be connected together by an IP network as shown in Figure 13 and would act as what has been called Bluetooth Hotspots. The reason for the name is that it is similar in concept to Wi-Fi hotspots where mobile devices can connect to each other through an access point connected to a wired network.

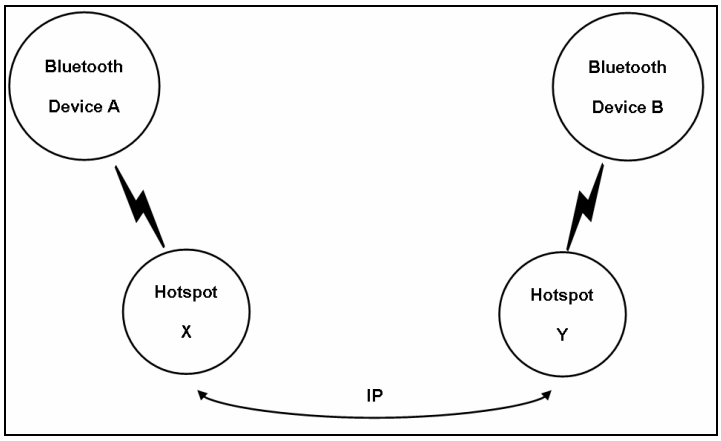

Figure 13: Bluetooth Hotspot

These Bluetooth Hotspots would allow Bluetooth devices that are distant from each other but within range of a hotspot to be able to communicate with each other via the hotspots. A possible usage scenario is demonstrated below in Figure 14, where a person who is in a boardroom meeting is able to access information on his/her computer back in the office. Even though the distance between the boardroom and the office is greater than the maximum transmission range of Bluetooth, both locations have Bluetooth Hotspots installed that connect these two locations and enabling the PDA to communicate with the computer as if he/she were back in their office.

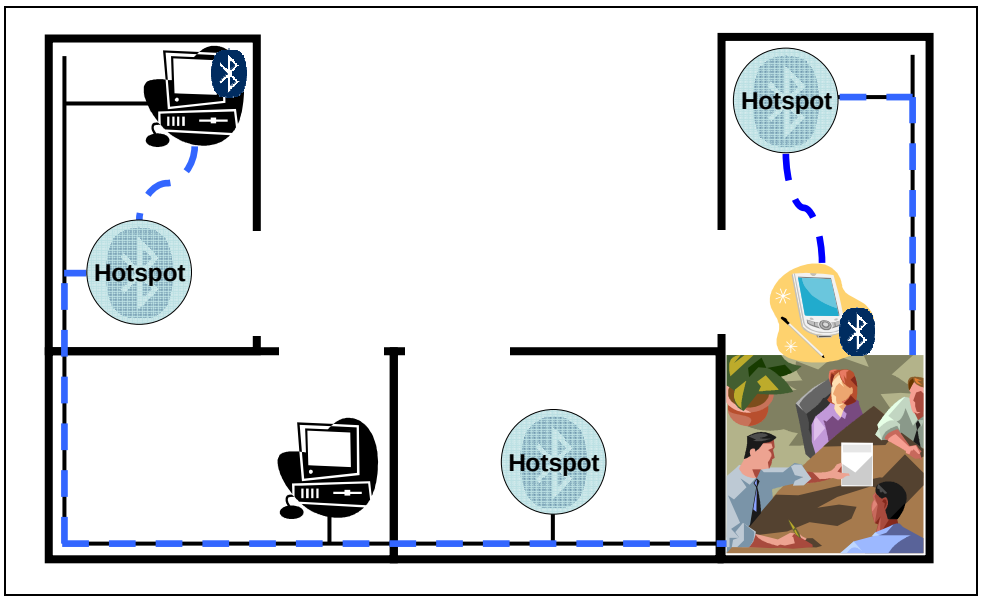

Figure 14: Possible usage scenario of a Bluetooth Hotspot

3.2 Different Approaches

The functionality of extending the reach of Bluetooth communication can be implemented at different layers in the Bluetooth protocol stack, each with its individual tradeoffs in efficiency, scope of service support and ease of implementation. These could be broken down into two primary implementation approaches:

- hotspots masquerade as other remote Bluetooth devices, and
- hotspots offer or otherwise proxy the services offered by other devices

These two processes are discussed in detail here, and how they have been implemented is expanded on in Chapters 4 and 5.

3.2.1 Masquerading

From a user's perspective the easiest means of using a Bluetooth hotspot would be to make it seamless and transparent while accessing the remote device. This requires designing a system where each hotspot is able to masquerade as other Bluetooth devices and therefore able to advertise itself as a device that is located at other hotspots. This is shown in Figure 15 where Bluetooth Device A is communicating with Bluetooth Device B via the hotspots. Each Bluetooth device thinks that it is

communicating directly with each other but in fact are communicating with the respective hotspot which is masquerading as the distant device. This would allow for very easy access to, and use of, the hotspots.

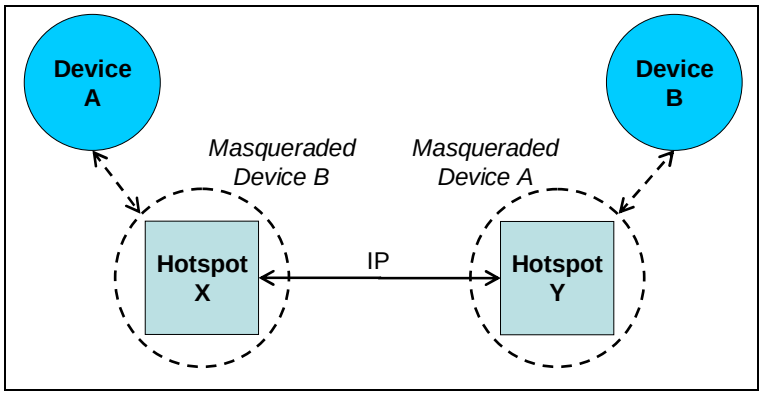

Figure 15: Using masquerading to create communications

The main obstacle foreseen for this approach is that a Bluetooth network does not use a single channel for all communications like a LANs physical cable or a Wi-Fi channel, but rather the communication is spread across 79 channels that are switched between quickly to combat interference. This makes it difficult for a Hotspot to "listen on" to all other communications.

3.2.2 Using Service Proxying

The second method was to examine Bluetooth's SDP and the feasibility of using SDP to provide facilities to discover remote devices and make possible the setting up of communication links between devices.

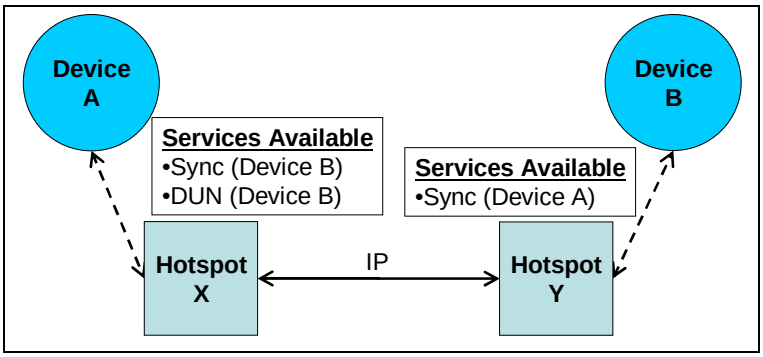

Figure 16: Using Service Discovery to create communications

The scenario that was envisioned is shown in Figure 16, where a user (Device A) within the range of a hotspot (Hotspot X) is able to query that hotspot for all the services that it offers. The hotspot will then return a list ("Services Available") of what other devices (Device B) are within range of other distant hotspots (Hotspot Y) and what services they can offer to allow for communication with a distant device. If the user then, for example, chooses to "Sync with Device B," a communication channel would then be set up between Device A and Device B by the hotspots that will transparently transport the communication over the IP network. Applications on either end will then seamlessly communicate with one another, unaware that the devices are not within radio range.

3.3 Bluetooth Hotspot

The Bluetooth hotspot is made up of various components. The hardware consists of a processing unit, a device to communicate over Bluetooth and a device to communicate to an IP network. The software consists of the Bluetooth protocol stack and a collection of custom written applications, which have been called BlueSpot, shown in Figure 17. The software is made up of three separate but interlinked components:

- **Management**: Manages the other software components that make up the hotspot as well as detects when Bluetooth devices come within range and leave.
- **Service Database**: Collects, stores and advertises services that are detected by all the Bluetooth hotspots within a network.
- **Transport**: This makes the connection between hotspot and Bluetooth devices and transports the Bluetooth data across the IP network.

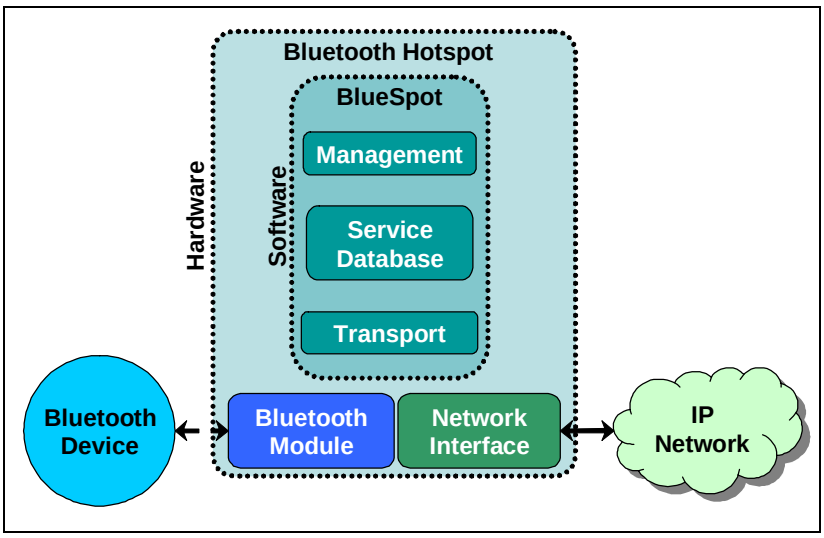

Figure 17: Components of the Bluetooth Hotspot

3.3.1 Management Component

The main role of the Management Component is to supervise the different applications that make up the Service Database and Transport Components, such as starting and stopping the applications when needed. One of the important tasks it would need to perform is to detect when Bluetooth devices come within and go out of range of the hotspot. It does this by periodically scanning for local Bluetooth devices that have entered range, as well as detecting when a previously detected device is no longer communicating. When a new Bluetooth device is detected it will be queried to find out its basic information such as its name and type of device. It will then inform the Service Database Component that there is a new device.

3.3.2 Service Database Component

The Service Database Component locates and advertises services from all devices within reach of the hotspots, and uses this information to manage the creation of the different connections using the transport component as needed.

When it is notified of a new Bluetooth device it will acquire a list of all the services that that device is advertising via its SDP server. These services are then checked to see if they have transport applications available. The services that can be transported are then added to an internal database and all the other remote databases are notified about the newly detected service. The local and remote databases will then start the

respective transport applications. Once the transport applications are running the remote service database will add the service to its local SDP server.

3.3.3 Transport Component

As their name implies the transport components in fact “transport” the Bluetooth communications between two Bluetooth devices via the Bluetooth hotspots over an IP network. In essence the transport component pairs are a proxy server in that they allow clients (Bluetooth devices) to make indirect (via IP network) network connections to other network services (i.e. services available on remote Bluetooth devices).

3.4 Connection Framework

The connection framework that enables the Bluetooth hotspots to communicate both internally and externally is explained below. The two jobs that a Bluetooth hotspot performs are the advertising of services from remote devices and the facilitating of communication between two devices.

3.4.1 Advertising a service

The process of advertising a service follows this sequence of steps shown in Figure 18:

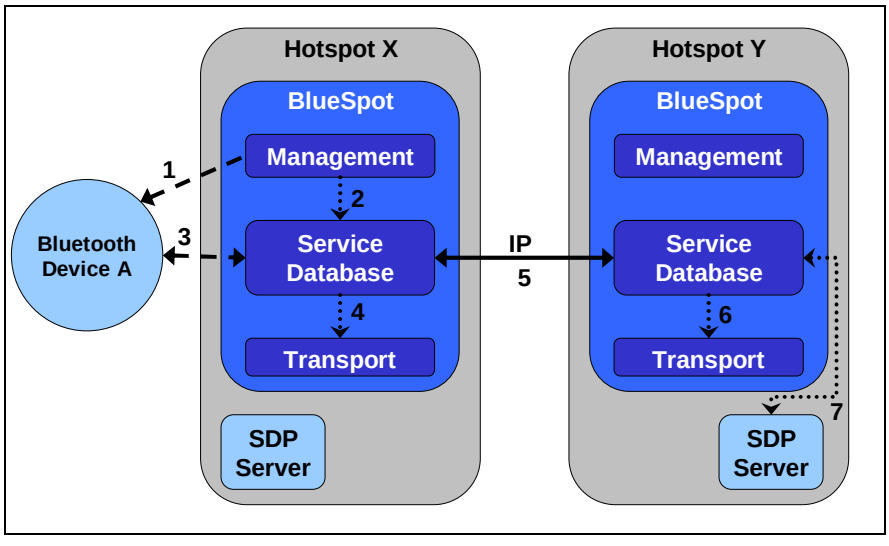

Figure 18: Communication Framework: Advertising a service

1. Bluetooth Hotspot X detects Bluetooth Device A
2. The Management Component informs the local Service Database Component (SDC) of the new device within range
3. The SDC queries the SDP Server on the Bluetooth device and finds out what services it is able to facilitate.
4. The SDC starts the Transport Components for the services above.
5. The SDC informs all other SDCs that have previously connected to it about the new services it has located
6. The remote Service Database starts its side of the Transport Components
7. The remote SDC adds the new services to the local SDP Server with the information needed to connect to the local Transport Components

3.4.2 Using a proxyed service

The process of using a proxyed service follows this sequence of steps shown in Figure 19:

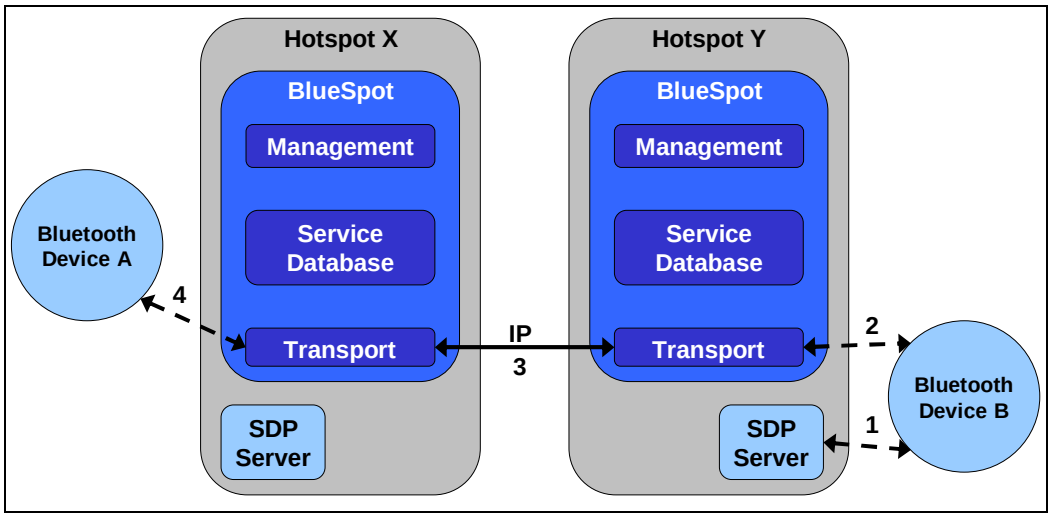

Figure 19: Connection Framework: Using a proxyed Service

1. A Bluetooth Device B browses the SDP Server on Hotspot Y
2. Using the information from above, it connects to the Transport Component
3. The Transport Component will then connect to its peer (on another hotspot) over the IP network and start transmitting the data it receives from Bluetooth Device B

4. The Transport Component on the remote side will then connect to Bluetooth Device A that initially advertised the service and forward the data on. Any data received will then be transported back to the other Transport Component on to Bluetooth Device B and, therefore, creating a two-way tunnel between the remote Bluetooth Devices A and B

3.5 Summary

In this chapter we introduced our proposal for what we called Bluetooth Hotspots as a means of increasing the range of use of standard Bluetooth devices. We started the chapter with a use case where such a system would be beneficial to the end user. Two possible solutions were put forward. Device masquerading is the most appealing as it would require minimal setup or change from the user but its implementation would be difficult. Service proxying is fundamentally simpler to implement though it requires knowledge on the user's part.

We then proposed a structure for the Bluetooth Hotspot and divided the software into three separate but interlinked components called BlueSpot collectively. The Management Component is the "overseer" of the software, managing the other components and also detecting when Bluetooth devices enter and leave the range of the hotspot. The Service Database Component collects, stores and advertises services that are detected within its range. This aspect is discussed further in the next chapter. The Transport Component facilitates the transfer of data between two Bluetooth devices which are out of range of each other. The conceptual framework for this structure is introduced in the following chapter and how it is implemented in Chapter 5.

Finally a proposed connection framework was discussed that showed how the different components of the system communicated and how these then interact with other hotspots to complete the transfer of Bluetooth data.

Chapter 4:

FOUNDATION AND DESIGN

We start off explaining the tools that are used in this project and the reason behind them. We explain why we decided to go with open source software, and the reasons we chose Linux as our operating system and BlueZ as a Bluetooth Stack. We explore the design of the different BlueSpot components and the decisions that went with them are discussed.

4.1 Design of a working prototype

First the various Bluetooth protocol stacks that are available are investigated and the criteria that were used to select the chosen stack are presented. This stack was further examined and its possible shortcomings discussed and how they could be overcome.

The second part of this chapter expands on our proposal by presenting our design for the various components that make up BlueSpot. The two approaches discussed in the previous chapter are applied to the Transport Component and possible designs are introduced and discussed. We also expand on the proposed Server Database Component and how the various service databases, both in the protocol stack and BlueSpot, would interact.

4.2 Foundation Software

One of the first important decisions that needed to be made before we could implement our proposal was what Bluetooth protocol stack we would use.

4.2.1 Available Bluetooth Protocol Stacks

There are a number of different Bluetooth protocol stacks implemented on different platforms and the decision would therefore have implications for the programming language and operating systems that could be used to implement the hotspot. Four of the most popular protocol stacks were investigated and are described next.

4.2.1.1 Bluetooth Stack for Windows – Broadcom/Widcomm

Broadcom [27], previously known as Widcomm, is the leading provider of Bluetooth software and networking solutions. Its Bluetooth for Windows was declared the world's first SIG-qualified Bluetooth against version 1.0b of the specification stack in 2001 [28]. A year later the certificate signifying compliance with the requirements of the Bluetooth specification version 1.1 [29] was obtained. This stack is the most widely used Bluetooth stack for Microsoft Window Platform and runs on all versions of Windows since Windows 98. The user interface includes shell extension, system

tray and control panel applications. They also provide software stacks for other devices such as embedded systems, mobile devices, smart phones and PDAs.

The Widcomm Development Kit must be bought from Widcomm. A Dynamic Link Library (DLL) is provided to link all your applications to and you are given a license to distribute this DLL with your application without licensing costs. The DLL contains a set of APIs to access the various protocol layers and profiles of the Bluetooth specification, Broadcom also supplies extensive documentation that exposes and explains the APIs [30]

4.2.1.2 Bluetooth Stack for Windows - Microsoft

Even though Microsoft is one of the promoting members of the SIG they only started providing their own Bluetooth stack for the Windows Desktop platform in the latter half of 2002 by including it in Windows XP Service Pack 2 [31]. Prior to this they only had a stack for their Windows CE platform.

Microsoft provides support for writing Bluetooth applications within its Platform Software Development Kit which can be downloaded free from their website. The SDK provides two approaches to programming Bluetooth on Windows, either using the Windows Socket interface [32] or by managing Bluetooth devices directly using the Bluetooth API [33]. The socket interface only exposes an API for RFCOMM at present. According to a presentation [34, 35] given by Microsoft at WinHEX 2005, socket support for L2CAP and SDP will be provided for Windows Vista (previously known by its codename “Longhorn”).

4.2.1.3 BlueZ – Official Linux Bluetooth protocol stack

The BlueZ Stack [36] was initially developed by Qualcomm [37] and was released by them under the GPL [38] in the beginning of 2001 [39]. A month later it was included in the official Linux kernel (v 2.4.6) by Linus Torvalds [39] and is now the “Official Linux Bluetooth protocol stack”[1]. [36]

[1] The first open-source Bluetooth stack for Linux was called OpenBT and was developed by Axis Communications. At the beginning of this project (2003) its development had already stagnated as BlueZ was being included in the Linux kernel and was officially discontinued on the 14th April 2005

Transport protocols L2CAP, RFCOMM and BNEP have been implemented in the kernel and can be compiled as separate modules. For development purposes the libraries and utilities can be downloaded from the BlueZ website if they are not already included within a Linux distribution. The different protocol layers have been implemented using the standard UNIX socket interface allowing for easy communication to any layer directly.

4.2.1.4 Bluetooth stack for FreeBSD (Netgraph implementation)

The Bluetooth stack for FreeBSD [40] is implemented using FreeBSD's Netgraph [41]. Netgraph is a framework where kernel objects that provide networking functions can be implemented in a uniform and modular approach. These kernel objects are known as nodes and each implement a different network protocol. For Bluetooth there are nodes that implement interfaces for HCI and L2CAP. Socket support is also provided by a node, which implements three different socket interfaces for HCI, L2CAP and RFCOMM. The core of Bluetooth, the Netgraph code, is included in the source tree of the kernel and the user space utilities are provided by FreeBSD's Ports Collection. [41]

4.2.2 Choosing a Bluetooth stack

When choosing which Bluetooth stack to use a number of criteria were considered, namely: access, limitations, integration, documentation and cost.

Access: How is the Bluetooth protocol stack accessed from the development environment and what APIs are provided? Also, is it possible to make modifications to the protocol stack itself?

Limitations: Are there any limitations in the protocol stack, or the API, that might hinder the project?

Availability: How is the protocol stack integrated into the operating system?

Documentation: What documentation resources are available and how accessiable is information on the protocols stack and the SDK?

Cost: What costs are involved in using and developing applications on the particular stack?

Table 2: Comparison between different Bluetooth Stacks

	Widcomm	Microsoft	BlueZ	FreeBSD (Netgraph)
OS	Windows	Windows	Linux	FreeBSD
How the stack is accessed	Function Calls	Winsock (Windows sockets) and Function Calls	Linux Sockets	Netgraph and BSD Sockets
Source Code	Closed	Closed	Open	Open
Availability and Integration into OS	Add-on DLLs	Windows XP Service Pack 2	Linux Kernel and BlueZ website	FreeBSD's source tree and ports collection
Documentation	Unable to evaluate without buying	Formal documentation on API available on MSDN[2]	Source code and mailing lists	Source code and mailing lists
Cost	Not-Free	Free	Free	Free
Limitation		No API exposed for L2CAP layer		

These four stacks can be divided into two groups, those for the Microsoft Windows platform and those for Open Source Software based platforms. The socket interfaces provided by most of the stacks were thought to be very beneficial in developing the transport applications as it would simplify linking the network communications to the Bluetooth communications. At this time Microsoft had not implemented a public socket interface for L2CAP, which was thought to be an important protocol layer to be able to transport. The other stack for the Microsoft platform was Broadcom which did come with a cost factor whereas all the other development kits were freely available. Though the cost was not a major issue, there seemed to be no technical

[2] Microsoft Developer Network – http://msdn.microsoft.com/

benefit in using Broadcom rather than the free alternatives, as they provide much the same level of completeness in access to the relevant Bluetooth stacks. Broadcom, as well as Microsoft, do not make the source code of their stacks freely available. This was an important consideration as, in the initial stages of development, it was thought that modifications might be needed to be made to the Bluetooth protocol stack itself to enable the interception of Bluetooth communications.

Both FreeBSD and BlueZ offered similar features when it came to development, available source code, socket support and access to all layers of the stack. They are also both based on the principles of Open Source Software and have all the benefits and advantages of open source systems, especially for the developer, as discussed in the next section. BlueZ was considered preferable to FreeBSD's implementation for two reasons. Firstly the BlueZ development community is far more active and larger then FreeBSD's as demonstrated by their respective mailing lists [42, 43]. And secondly parts of the FreeBSD code were already ported from BlueZ code [44].

4.2.3 Open Source Software

The choice of an Open Source Software (OSS) stack came with a number of additional advantages especially when developing new applications and having to interface with other protocols:

- OSS implies open standards since the source code of the software is available for viewing and the implementation of protocols can be read.
- OSS means that applications are able to be adapted to work with or in other applications easily, meaning that there is no vendor lock-in and greater support for the application
- OSS means the source code can be included within other applications as long as the license terms are obeyed. This means that code can be reused.
- OSS allows for the modification of the source code, allowing for debug messages to be placed in existing code.

4.2.4 BlueZ

BlueZ has been developed using the principle of modularity. Protocols have either been implemented as separate modules for the kernel or as user-space daemons. Communication with all the different layers of the stack provided by BlueZ is done using socket interfaces, which should be familiar to most network programmers. This makes it easier for a programmer to switch and offers the chance to utilise large amounts of already written networking code and thus shortening development time. Also by using standard sockets, developers coming from a networking background would not have a steep learning curve.

BlueZ consists of many different components that can be broken down into three main areas: Kernel, Libraries and Utilities, Testing and Analysis:

Kernel: This refers to the kernel modules that are included in versions 2.4 and 2.6 versions of the Linux Kernel. This consists of the Bluetooth subsystem core and HCI layer device drivers for USB, UART and PCMCIA cards/modules as well as a virtual Bluetooth device driver. L2CAP and SCO audio are implemented as kernel layers. RFCOMM, BNEP and HICD are also provided as kernel modules or compiled into the kernel.

Libraries and Utilities: These Kernel modules would be useless without the support of various utilities and libraries, with the SDP daemon that provides a SDP server being a very important one. There are also various tools to interact with the Bluetooth devices and manage the different protocols.

Testing and Analysis: Lastly, there are the testing and analysis tools that allow a user to test the various protocols such as L2CAP and RFCOMM. This includes hcidump, which intercepts and displays HCI messages been passed to and from a Bluetooth device, and allows for debugging and analysis.

One of the biggest short-comings of the BlueZ stack is the lack of formal documentation. As the source code is available for everyone to read the usual reply to requests for more info on the API for BlueZ was met with short response of “Read the

source". Though being able to read the source militates against the lack of formal API documentation, such a document would be of great benefit when it comes to developing for BlueZ. This hindrance was largely overcome by the abundance of testing applications for each protocol that come with BlueZ. The source code of these applications could be read and modified as basic starting points for further development. This was partially remedied by the release of a primer for programming with BlueZ [45], which formed part of a masters' thesis [46] of a MIT student on Bluetooth and location aware computing.

4.3 BlueSpot

4.3.1 Management Component

For our BlueSpot prototype a full management component was not implemented. For our investigation the other components of the Hotspot were interacted with through the command line and the applications directly. In a commercially viable system a management application would need to be designed and written. This application would act as an overseer of the other components and their applications. Its main role would be to locate new Bluetooth devices within the hotspot's vicinity and initiate the discovery of the available services.

4.3.2 Transport Component

As their name implies the Transport Component is the mechanism that "transports" the Bluetooth communications between two Bluetooth devices via the Bluetooth hotspots over an IP network. In essence the transport application pairs are proxy servers in that they allow clients (Bluetooth devices) to make indirect (via IP network) network connections to other network services (services available on remote Bluetooth devices).

4.3.2.1 Masquerading

An obvious approach of providing a remote Bluetooth point of presence would be to use remote devices to impersonate the local device. However as discussed in Chapter 2, Bluetooth uses a system of FHSS where a Bluetooth device does not use a

single channel for communications, but jumps between different frequencies in a pseudo-random sequence to avoid interference with other devices. This sequence is determined by the clock and BD_ADDR of the master. Therefore, for a device to be able to impersonate another and be able to communicate as that device, it would need to be able to change its BD_ADDR to that of the device it is impersonating. Two possible approaches would be to change the BD_ADDR for each time slot and handling multiple connections at one time by multiplexing between them, or by changing the address for each connection and only handling a single connection at any one time. It is shown in the next chapter that these approaches proved to be not feasible as a result of our study and this avenue of investigation was abandoned in favour of service proxying.

4.3.2.2 Service Proxying

An alternative approach is for just the service to be provided remotely by having the Bluespots act as an agent between a device offering a service and a device wanting to use said service. In service proxying each transport application handles a single Bluetooth protocol such as L2CAP or RFCOMM. This allows for the development to focus on each protocol layer independently, creating a modular design.

A transport application is made up of two parts, a local part which runs on the hotspot that has a Bluetooth device advertising a service, and a remote part which runs on other hotspots where the service is being re-advertised. Each transport application only handles a single service, so there would usually be multiple transport applications running on a single hotspot. Since a hotspot would generally have both devices advertising services and those wanting to use the re-advertised service, there could be both local servers and remote clients running simultaneously. An example of these connections can be seen in Figure 20, where there are three Bluetooth hotspots X, Y and Z, each with a number of transport applications managing different services and therefore different protocols. The lines represent the communication channels that could be used when a connection is needed.

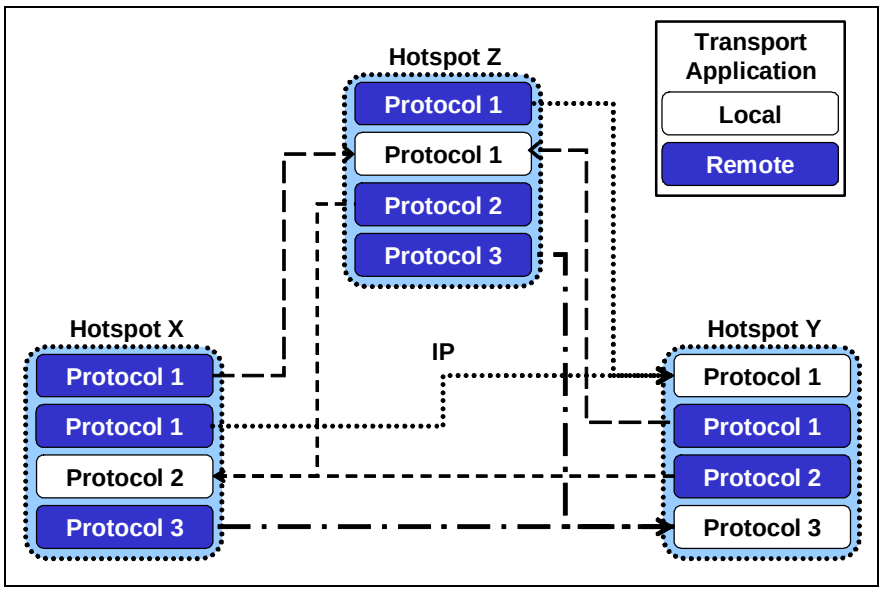

Figure 20: Transport Application Connection Example

The local server of the transport application facilitates the communication between the IP network and the Bluetooth device providing the service. It is started by a Service Database Component (SDC) when it detects the services and is added to its database. It will open a network port and listen for a connection from a remote client. When a remote client connects, it will then open a connection to the device providing the service. If the connection is successful it will then transfer data it receives from the remote client via the network to the device over Bluetooth, and vice-versa. When the data connection is finished it will disconnect from the Bluetooth device.

The remote client is started by the SDC when it is informed about a new service that is available by another hotspot. It facilitates the connection from a Bluetooth device wanting to use the service and the local server via the IP network. When the application is started it will open a port/channel on the Bluetooth network and wait for a connection. When a Bluetooth device connects it will attempt to open a connection to the corresponding local server. If the connection is successful it will then transfer any data it receives from the Bluetooth device to the local server via the network as well as forward any data it receives. It will close the network connection when the Bluetooth device disconnects.

4.3.3 Service Database Component

The SDC is an important part of the BlueSpot as this is what allows for Bluetooth devices to locate services on other Bluetooth devices located at remote Hotspots. Before a connection could be made by the Transport Component a Bluetooth device would first need to locate the service it wishes to use.

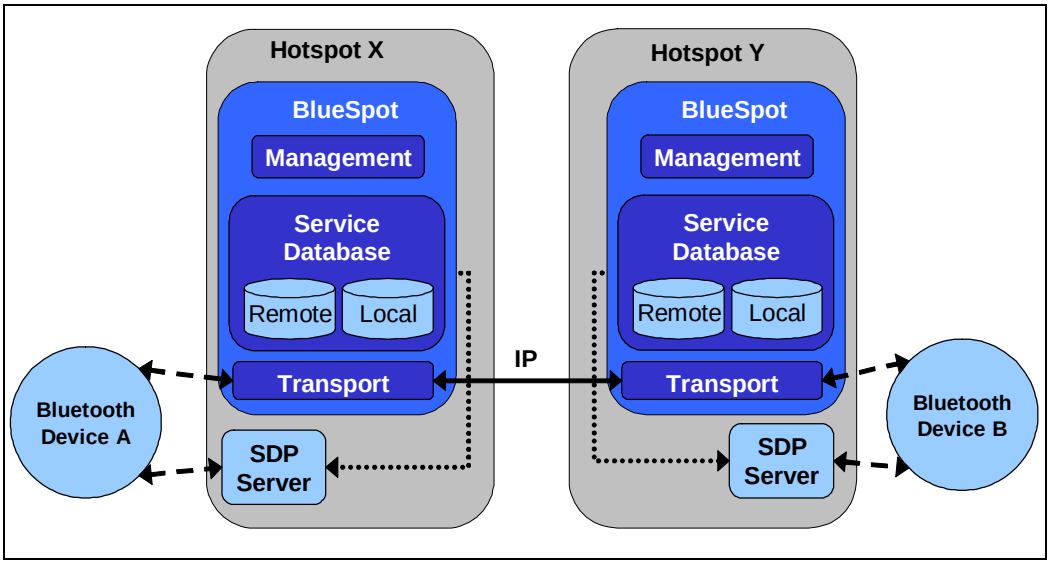

Figure 21: Service Database Component showing different databases

This component consists of two databases, a local service database and a remote service database as shown in Figure 21. The local service database keeps records of the services offered by Bluetooth devices within the vicinity of the Hotspot that have been detected by repeated scans. The remote service database keeps records of services offered by remote Bluetooth devices that are located around other Hotspots. Both these databases are separate from the service database used by the SDP Server which advertises the services to Bluetooth devices.

When a Bluetooth device is detected by BlueSpot it will have its service database interrogated by the SDC. Any services that the BlueSpot is able to transport will then be added to the local server database inside the SDC, and notification messages will be sent to any other BlueSpots connected to the network. They will then add the service to their remote service database within the SDC and also add it to the service database of the SDP Server of the Hotspot as well as open any necessary channels.

When a Bluetooth device requests information about what services are offered by the Hotspot, the service database of the SDP Server will be returned which will contain records from services originating both from the Hotspot as well as services from BlueSpot,, which originate from the remote service database of the SDC

4.4 Summary

Before we were able to design a system to implement our proposal for a Bluetooth hotspot a number of Bluetooth Protocol Stacks needed to be investigated. From the shortlist of four, the BlueZ stack was selected on the basis of access, limitations, integration, documentation and cost. The FreeBSD implementation came close, but BlueZ's large and active developer and user community made it the better choice.

BlueZ is the Official Linux Bluetooth Stack and has been included in the Linux kernel for a number of years. There are also a number of benefits that arise from using an OSS Bluetooth protocol stack and an OSS operating system, such as accessible source code and easy access to the operating system's kernel. This all contributes to making development of a proof-of-concept system easier.

The Management Component for our test system was not implemented as the command line was used to interact with the various applications. In a commercially viable system an application would be needed to be written to provide management functionality. Designs for the Service Database Component and the Transport Components were introduced and discussed. Solutions for these are implemented and investigated in the next chapter.

Chapter 5:

INVESTIGATION OF A PROTOTYPE HOTSPOT

In this chapter we investigate the implementation of the Transport Component and the Service Discovery Component of BlueSpot. The first part of this chapter deals with the Transport Component and the applications that were written to transport Bluetooth communications at different layers of the Bluetooth protocol stack. We introduce rcpipe and l2pipe, which can successfully transport RFCOMM and L2CAP communications successfully. A proof-of-concept application was then developed to offer the functions of the Service Database Component. We end the chapter with a discussion on some issue pertaining to lower layers that need to be considered.

5.1 Introduction

In Chapter 3, two approaches were proposed for BlueSpot, masquerading and service proxying. In Chapter 4, various designs for BlueSpot were introduced within a framework of different components. In this chapter these designs are implemented and a number of applications are written so as to investigate the feasibility of the proposals and designs.

The first part of this chapter looks at implementations for the Transport Component in a layered approach that matches the different layers of the Bluetooth protocol stack. Firstly an attempt is made to implement the masquerading approach by creating the Transport Component within the Baseband Layer. This would allow for the seamless spoofing of Bluetooth devices, making BlueSpot mostly transparent to the user. Next, an implementation is created within the Bluetooth protocol stack by creating virtual RFCOMM modules, each one emulating the RFCOMM module on another hotspot, and doing service proxying. From lessons learned in that attempt the next iteration was developed as a separate application to transport RFCOMM and later another application to transport L2CAP data.

Next, a proof-of-concept application is developed to implement the functions of the Service Database Component

The chapter concludes with a discussion of some details concerning the core lower layers.

5.2 Baseband Layer

Each Bluetooth device is allocated a unique 48-bit BD_ADDR at time of manufacture with the assumption being that this is fixed and unchanged for the life of the device (as in the MAC address for a network card). The Bluetooth specification does not provide a way for this address to be changed, but as with network cards, manufacturers have implemented their own vender-specific HCI commands for changing the BD_ADDR. For this change to take effect the module needs to be reset

in which case a loss in communication between the Bluetooth devices occurs as the clock used to synchronise the hopping sequence is reset as well. This rules out the option of changing the address for each packet as suggested in Chapter 4. As for the second option, a Bluetooth hotspot would not know what devices it should be impersonating so as to be able to accept a connection from another device.

When it was found that it was not feasible to change the physical address of a Bluetooth device to be able to spoof another device a different approach was looked at. Instead of spoofing the Bluetooth devices we would spoof or proxy the services they offered instead. This fits the network paradigm well as Bluetooth is a service-orientated network rather than a device-orientated network. Rather than attempting to have the Bluetooth hotspots masquerade as the service-provider, they would act as proxies and would re-advertise the services on the hotspot themselves and then, when necessary, transport the Bluetooth communication back to the original service-provider.

5.3 RFCOMM Layer – Modified Stack

As RFCOMM is a simple protocol compared to L2CAP by virtue of being a stream based protocol versus a packet-based protocol such as L2CAP, it was the layer that was first considered to develop a transport application. Though RFCOMM was designed to emulate RS-232 serial ports, the interface implemented by BlueZ provides a socket connection with the semantic type of SOCK_STREAM, which is the same type that is used by TCP. As a result, implementing an application transfer data from a RFCOMM socket across to a TCP socket and vice-versa seemed to be straightforward and why RFCOMM is the logical choice for an initial implementation.

The first approach investigated to provide this data re-direction was within the Bluetooth stack itself. The idea was to provide virtual RFCOMM protocols running on top of L2CAP, each being an agent for an RFCOMM protocol running on another device. Each virtual RFCOMM module would represent the RFCOMM protocol on the other device and be assigned its own L2CAP address. This address would then be

used in the service record for the proxyed service and be re-advertised on that device. Any data sent to a virtual RFCOMM module would be transported across the IP network to a modified RFCOMM module on the remote side where a connection to the application would be created. This concept is demonstrated in Figure 22. With respect to the address value the Bluetooth specification says that "[a] dynamically assigned value may be used to support multiple implementations of a particular protocol" [5] suggesting that such an approach is possible.

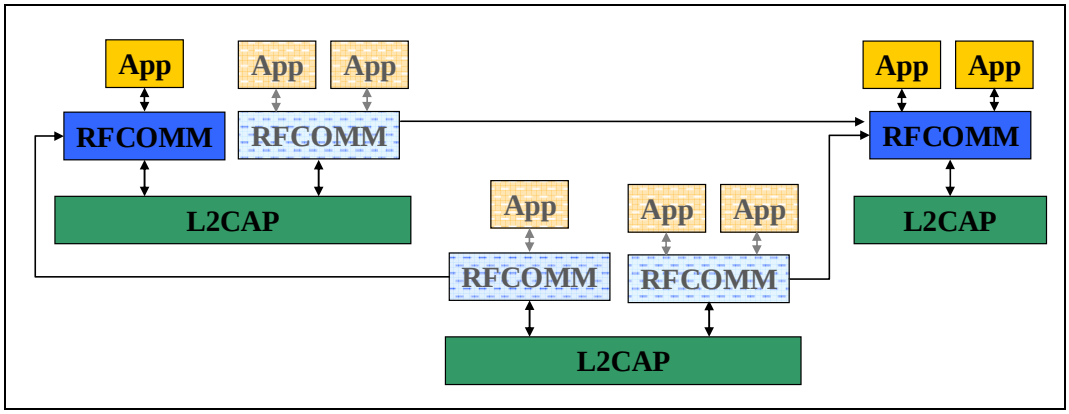

Figure 22: Virtual RFCOMM modules

It was during developing an application to test this hypothesis that it was found that though such a concept could be implemented, it would entail considerable modifications to the RFCOMM module within the kernel. Kernel level development is a slow process compared to user-space programming and is problematic to debug as it is running in kernel space and usually doesn't have terminal output. A less complicated approach would not have to modify the Bluetooth stack itself but use the RFCOMM protocol and develop a user space program to connect two RFCOMM ports on different devices together. This led to the development of rcpipe.

5.4 RFCOMM Layer – RCPIPE

Rcpipe was the first transport application written for this project that actually transports Bluetooth communication across an IP network. The initial design was influenced by the desire of getting an end-to-end test working quickly so as to demonstrate the concept of a transporting application was indeed possible. The name rcpipe is a combination of the names of the two programs that it is based on, **rc**test

and data**pipe**. Rctest is a testing utility provided by BlueZ which is used to test RFCOMM connections within a BlueZ setup. Datapipe is a simple TCP socket redirection application that can be found in various versions on the internet [47, 48]. It opens a listening TCP/IP port on the machine it's running on (intermediate) which in turn connects to a port on a remote machine (service) and forwards any data received on the local port to the remote port as shown in Figure 23 (a).

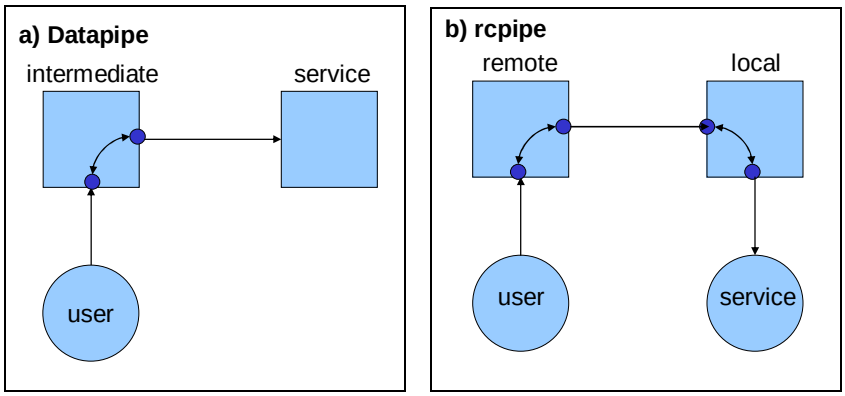

Figure 23: datapipe versus rcpipe

The core part of datapipe was a single select function call which performs "synchronous I/O multiplexing" [49] by watching the specified file descriptors for any change of state and then copying across the data from one socket to another thus forwarding the data. This is very similar in concept to what was required for *rcpipe* except that rather then there just being one intermediate node; there are two: a local and remote, as shown in Figure 23 (b).

5.4.1 Initial Investigation

To simplify the development of rcpipe, communication channels were initially stipulated on both ends and the two hotspots were configured with the IP address of the other hotspot and the BD_ADDR of the Bluetooth device that is near them. For example (shown in Figure 24) Hotspot X would be configured with Hotspot Y's IP address and the Bluetooth B address and Hotspot Y would be configured with Hotspot X's IP address and the BD_ADDR of Bluetooth Device A. This meant that a connection could be made from either of the hotspots. A connection made by Bluetooth Device A to a port on Hotspot X would be forwarded via Hotspot Y to a port on Bluetooth Device B and vice-versa.

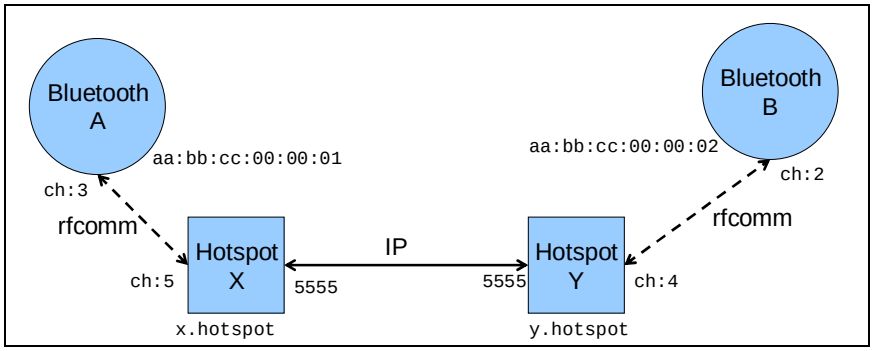

Figure 24: rcpipe communication framework (Initial Investigation)

The operation of rcpipe is shown in detail by the flowchart in Figure 25. The application could be started in either a client or server mode. The client/server terminology referred to the IP networking side of the application and only decided if the application would open a network port and wait for a connection from a client, or if it would make a connection to a server.

Once the network connection is made the applications no longer have a server or client role, but have equivalent roles. Both applications will then create a RFCOMM socket with the channel number given on the command-line and will listen on these channels and the open network connection. When a Bluetooth device connects to BlueSpot that hotspot will accept the connection and send a short message to the other hotspot via the network connection. This message indicates to the hotspot that there is incoming data and that it should make the outgoing RFCOMM connection to the Bluetooth device.

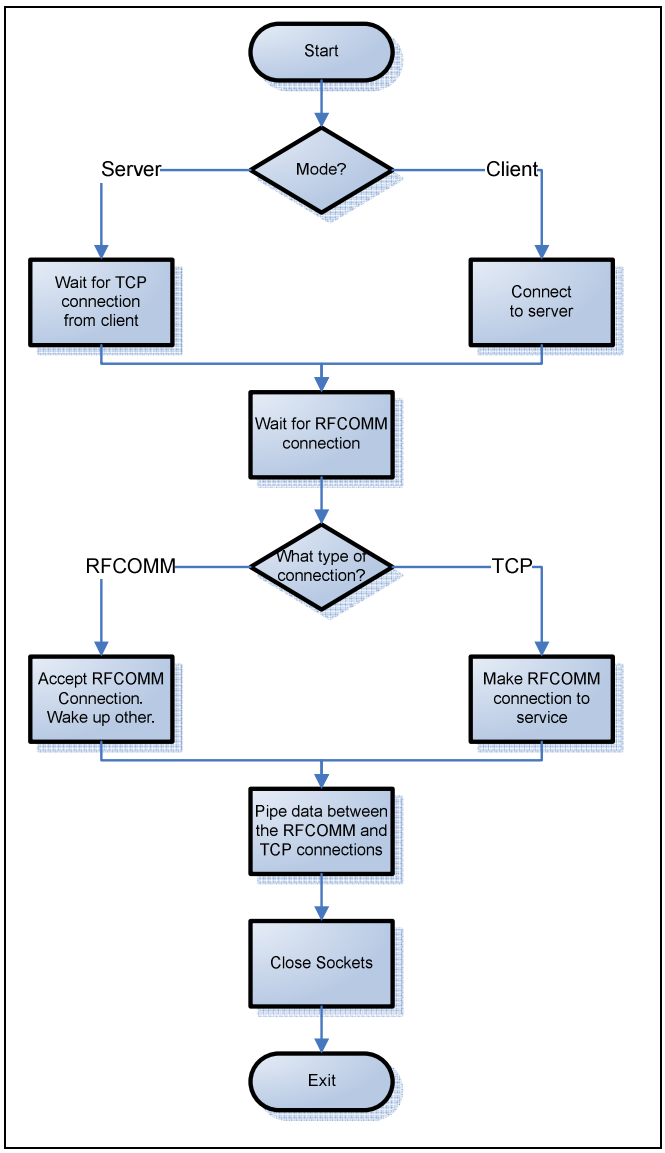

Figure 25: RCPIPE Application flowchart

Both hotspots will then enter a "piping" stage where both the RFCOMM and TCP sockets are monitored for incoming data, i.e. having data available to be read in the socket buffer. Data received from the TCP socket is piped to the RFCOMM socket and vice-versa. This will continue to be repeated until the RFCOMM connection is closed, which in turn closes the TCP socket and both applications exit.

5.4.2 Final Implementation

Though this approach initially seems acceptable, defining a bi-directional connection does not match the concept of a proxy for which we are aiming. A better approach is shown in Figure 26 below and was implemented. The figure is less cluttered than the

previous design shown in Figure 24. The reason for this that the connection can only be initiated in one direction and that is in the direction of the service-provider. In addition, you will notice that we now use the local/remote terminology introduced in the proposal for the Bluetooth hotspots in Chapter 3.

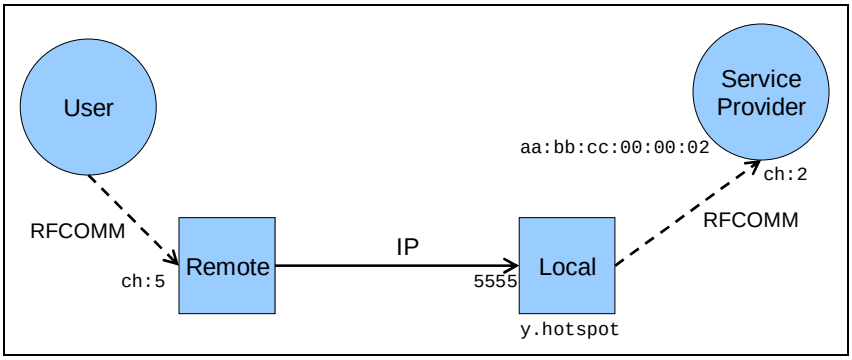

Figure 26: rcpipe communication framework (Final Implementation)

Depending on what hotspot rcpipe is running on, one of the two roles it performs needs to be chosen at start-up. On the hotspot nearby to the service provider rcpipe would be started with the `--local` switch giving the Bluetooth address of the service provider. It would also in most cases be necessary to specify the RCOMM channel that the service is running on by using the `--rc_channel` option, as the default is 10. In order that remote hotspots are able to connect to the local hotspot, the hostname or IP address of the local hotspot needs to be given when the `--remote` switch is used to select the remote mode. Once again the `--rc_channel` option is used to specify a RFCOMM channel though in this mode it is the channel that the remote would listen on for a connection and would advertise in the service record. The option `--inet_port` can also be used to specify the TCP port the local hotspot will be listening too. The RFCOMM and TCP ports would usually be chosen by the distributed service database and be unique for each service that is proxyed.

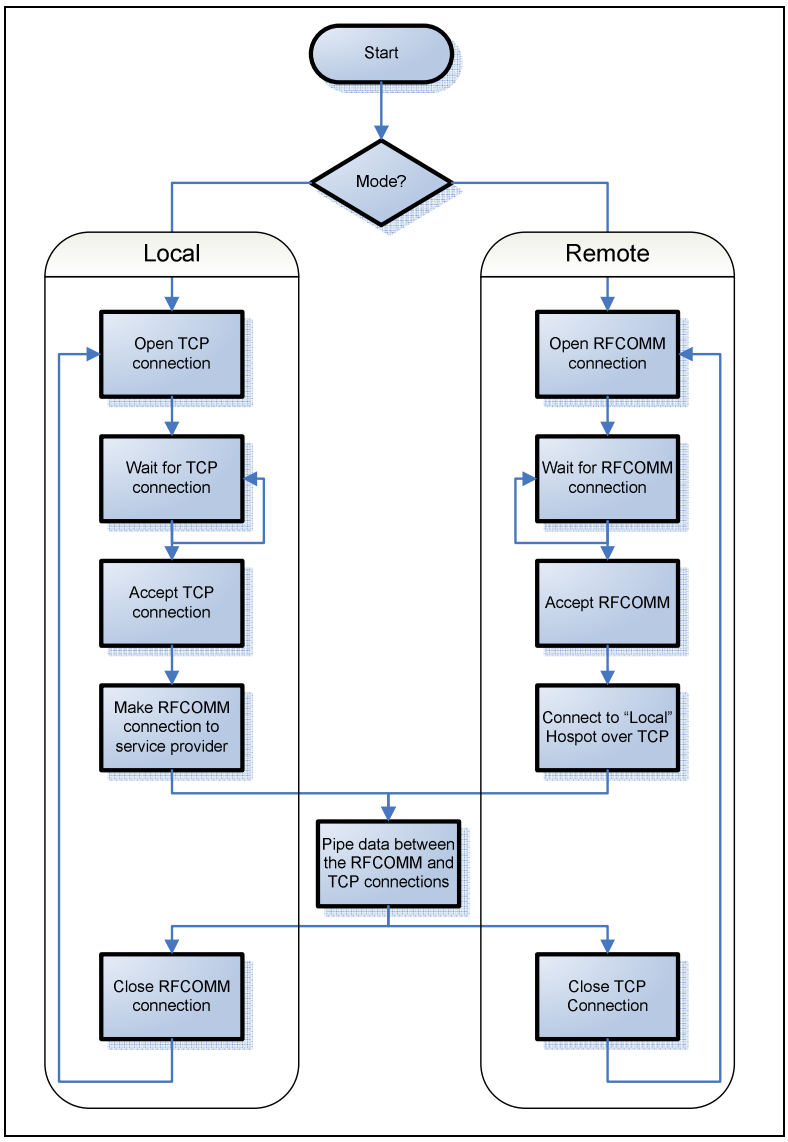

Figure 27: RCPIPE flowchart

When rcpipe is started all the parameters given are placed into a common data structure called pipe, which is shown below in Figure 28. Not all the variables are used in both modes with the Bluetooth address not being needed in remote mode and the IP host in the local mode. Once all the command line switches are dealt with the application either calls the *mode_local* or *mode_remote* functions depending on the role picked.

```
struct pipe_settings {
      char *bt_addr;                    // Bluetooth address
      uint8_t rc_channel;               // RFCOMM channel
      int inet_port;                    // TCP port
      struct hostent *server_inet;      // IP host
};
```

Figure 28: Structure for the different settings of the tunneling pipe

The function *mode_local* first calls *inet_listen* with the TCP port number. This function creates a TCP socket, binds it to the given port number, sets it to listen and returns a handle to this listening socket. The application then enters a continuous loop that starts with it blocking on the TCP socket waiting for another rcpipe to connect to it. The accept system call returns a handle to a new socket which is now the connected socket. The *rfcomm_connect* function is now called with the BD_ADDR and RFCOMM channel number of the service provider and services being advertised respectively. When a handle to the connected RFCOMM socket is returned by *rfcomm_connect* it and the newly created TCP socket are passed to the *rfcomm_pipe* function. This function performs the essential role of the application, that being transferring the data from one type of socket to another, and is expanded on further below. After the *rfcomm_pipe* function is finished the newly created TCP socket and RFCOMM socket are closed so that another device can therefore make a connection and once again the application blocks waiting for another TCP connection.

mode_remote function acts similarly to the *mode_local* function except that the TCP and RFCOMM sockets are swapped around. The function starts off with a call to *rfcomm_listen* function which returns a handle to a RFCOMM socket which is listening on the given channel number. It will then also enter a continuous loop where it waits for a Bluetooth device to connect to this RFCOMM channel. When a connection is made, it is accepted and a TCP connection is made to the local hotspot by the *inet_connect* function, which returns a handle to this socket. As before, the *rfcomm_pipe* function is called with these newly connected RFCOMM and TCP connections. Once the piping is finished for that connection the relevant sockets are closed and the application will then wait for another RFCOMM connection.

The *rfcomm_pipe* function is the central part of the rcpipe application, but is conceptually quite simple as demonstrated by a snippet of pseudo code in Figure 29. It performs the task of transferring the data from the RFCOMM socket to the TCP socket and visa-versa and thus transports the Bluetooth data across the network. It uses the *select* system call to await the arrival of data from either the RFCOMM or TCP socket. Data received on one socket is read into a temporary buffer and then written out to the other socket. This is repeated until either one of the sockets are closed, detected by a socket being marked readable but zero bytes being read, or by an error in the reading or writing of data, when this occurs the function returns to the function that called it, either *mode_local* or *mode_remote* functions.

```
while (not end loop) {
  select on RFCOMM and TCP sockets
    //blocks till either socket becomes readable
  check which socket is readable
  read from that socket to buffer
  write buffer to other socket
  if socket is closed then end loop
}
```

Figure 29: Pseudo code of *rfcomm_pipe* function from *rcpipe*

To set up the transport applications to match what is shown in Figure 26 the following commands would be run;

On the local hotspot (closest to service-provider):

```
rcpipe --local aa:bb:cc:00:00:02 –rc_channel 2 --port 5555
```

On the remote hotspot:

```
rcpipe –-remote y.hotspot --port 5555 –rc_channel 5
```

5.5 L2CAP Layer

The next step in the project after the success with *rcpipe* was to develop an application that was similar in function but transported L2CAP communication

instead. As L2CAP is a multiplexer of higher protocols, including RFCOMM, being able to transport communication at this level would be valuable contribution to the project. Thus a transporting application called l2pipe that handles L2CAP communication was developed.

Since BlueZ uses a consistent API design between it different protocol layers, by using sockets, l2pipe is based on rcpipe and the L2CAP testing application, l2test, provided by BlueZ. This meant that there was a lot of code re-use from rcpipe speeding up application development.

There is not much difference between the two transport applications l2pipe and rcpipe, and both operate very similarly to each other except that l2pipe uses the terminology PSM's rather than channel numbers as can be seen in Figure 30. l2pipe is started in the local mode on the hotspot closest to the service-provider and in the remote mode on other hotspots. When a Bluetooth device connects to a PSM that l2pipe is listening to on the remote hotspot a TCP connection is made to the local hotspot via the IP network. Any further data received on that PSM is then forwarded onto the local hotspot across the TCP connection. When the l2pipe on the local hotspot detects a TCP connection to it, it will make an L2CAP connection to the service-provider and then forward any data received on the TCP connection to the newly created L2CAP connection thus completing the route. One notable change from rcpipe is that L2CAP has an adjustable Maximum Transmission Unit (MTU). This MTU is the largest "packet" size L2CAP can transmit and can range from a minimum of 48 bytes to a maximum of 65,535 bytes. Some Bluetooth services define a required MTU. The MTU is set for incoming and outgoing connections separately.

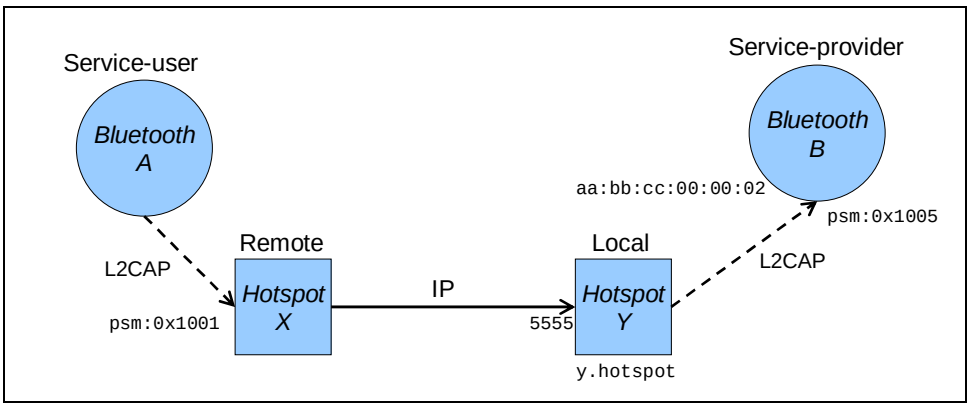

Figure 30: l2pipe communication framework

Since l2pipe functions very similarly to rcpipe this section should be read with the implementation section for rcpipe before. L2pipe is started the same way as rcpipe except rather than specifying an RFCOMM channel to listen or connect to, the PSM of the service is given by using the `--psm` option. The common data structure *pipe_settings* was extended for l2pipe. It was renamed *l2_pipe_settings* and is shown in Figure 31.

```
struct l2_pipe_settings {
      char *bt_addr;
      uint16_t psm;
      int inet_port;
      struct hostent *server_inet;
      struct l2_options options;
};
```

Figure 31: Data structure of settings for L2CAP pipe used by l2pipe

Besides the small change of renaming *rc_channel* to *psm*, a new structure (Figure 32) is included to store the different options that can be set on an L2CAP connection. Presently only the incoming and outgoing MTU's are stored, given by the `--imtu` and `--omtu` options respectively.

```
struct l2_options {
      int imtu;   // incoming MTU
      int omtu;   // outgoing MTU
};
```

Figure 32: Data structure for L2CAP connection options

The application will then operate in the same way as rcpipe except using L2CAP sockets rather than RFCOMM sockets.

5.6 Service Discovery Component

As a proof-of-concept to the design for the Service Discovery Component an application was developed and written called Distributed Service Database (DSDB) Daemon with the filename of dsdbd. It was called “distributed” as multiple DSDB’s would be able connect to each other in a peer-to-peer network and pass between them the different services that are detected by each other. The application was written as a daemon which runs in the background and is communicated with via a TCP socket.

DSDB can be started in either a server or client mode. As a server the application opens a TCP port and waits for connections whereas as a client the application would actively make a connection to another DSDB given its hostname on the command-line. Once two or more DSDBs are connected to each other the rest of the communication executed in a command and response, peer-to-peer manner. The DSDBs do not search for Bluetooth devices but are notified by the Management Component when a device is within range, and again when it no longer contactable. This is to simplify the design and to keep operational boundaries between the different BlueSpot components.

The DSDB was implemented so interacting with it would be simple and straight forward. By using a TCP socket for control allows for both applications running on the same machine, such as the Management Component, and applications running on other machines, such as other DSDB’s, a single point of access. Control is done by plain text commands each one handled by a separate function within the code allowing for the easy extension of new commands.

Once the DSDB are successfully connected to each other the connecting DSDB requests a list of available services to be transported by sending the *sendservice* command. When a DSDB receives this command it searches within it internal local database of services and for each one it will start the relevant transport application (in local mode) on the local machine. It will also return an "*addservice*" command to the requeting DSDB, which contains the service name or type, the hostname of the DSDB and the port that the transport application is running on.

When a DSDB receives the *addservice* command it will start the relevant transport application (in remote mode) using the hostname and port number given by the command. It will then add the service to the hotspot's SDP server At this point the service could be found by another Bluetooth device and used by connecting to the transport application that been started and is advertised in the service record.

At this point the concept has been proved, but further work would be needed to make it a complete system.

5.7 Lower Layer Considerations

5.7.1 Role Switch

As discussed in Chapter 2, a Bluetooth piconet has a master and a number (up to 7) of slave devices that are connected to the master. Communication at the physical layer in the piconet is controlled by the master device and all communication is between it and a slave device. L2CAP was defined so that it glosses over this master-slave relationship and logical connections are simply setup point-to-point between devices in a piconet. In order for the hotspot to be able to accept connections from multiple devices it needs to be a master, and to do this the transport application needs to perform a role switch when communication is initiated (see section 2.5.2).

To set the relevant link mode option on the Bluetooth socket before setting it to listen. Enabling the `RFCOMM_LM_MASTER` or `L2CAP_LM_MASTER` switchs on the link

mode of a RFCOMM or L2CAP socket respectively, means that the device will ask to become the master when a connection request comes in.

5.7.2 Security Limitations

Even though security was scoped out of this project, there are a couple of issues that should be pointed out. No security was implemented in the various applications developed for the Service Discovery Component and Transport Components and in fact, Bluetooth Security Mode 1 needed to be activated to prevent Bluetooth devices from actively seeking to authenticate. There are two areas that need to be considered to secure the communication between two Bluetooth devices using BlueSpot: first the communication between different hotspots and secondly the communication between a Bluetooth device and a hotspot as shown in Figure 33.

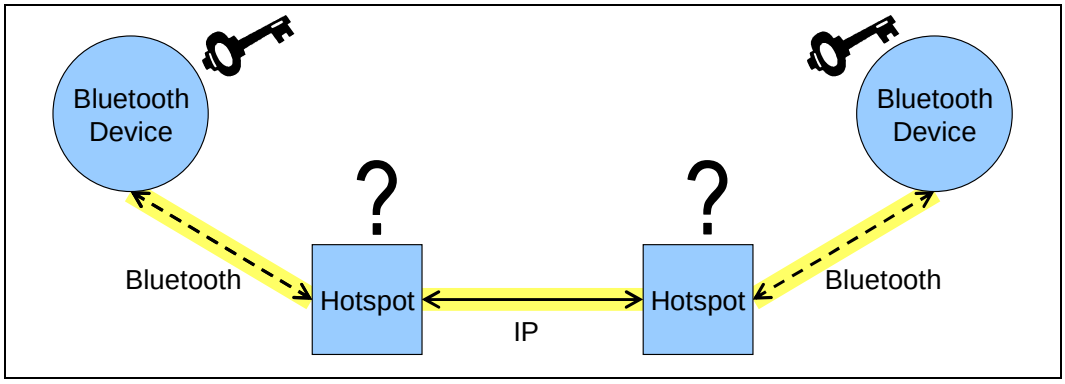

Figure 33: Securing communication channels

Securing the IP communications between different hotspots is trivial and a number of different solutions could be considered [44]:

- Placing all the hotspots in their own private network such as a Virtual Private Network (VPN) [50] using IPsec [51].
- Setting up secure tunnels between the hotspots using SSH [52] or TLS [53] without modifying the applications.
- Adding a cryptographic library, such as Cryptlib [54] or OpenSSL [55], to the applications to create secure communication channels. [56]

On the other hand, securing the communication from end-to-end is less trivial. This is due to the fact that the secret key is not exchanged during authentication but rather a

challenge-response with a random number is performed to calculate if both devices know the same secret key (see section 2.8.2). The keys in Figure 33 show how a hotspot does not know the secret key and therefore is unable to pass it on to another hotspot for it to make the outgoing connection. A solution for this would needed to be investigated with and study of man-in-the-middle and relay attacks such as [55] might lead to a possible solution

5.8 Summary

The first part of this chapter investigated the implementation of the Transport Component at different layers of the Bluetooth protocol stack. Initially an implementation at the Baseband Layer was considered following the masquerading approach introduced in Chapter 3. Because of the nature of Bluetooth communication and how the hopping sequence is reliant on both the BD_ADDR and clock of the Master device it proved not feasible to spoof other Bluetooth devices by simply changing the BD_ADDR of the hotspot. Due to this, the second approach of proxying the services only was looked at, which fits the service-orientated network paradigm of Bluetooth.

The RFCOMM protocol was the first protocol investigated to be transported due to it being a simple protocol and having the same socket connection type as TCP which it was to be transported over. The first approach was to provide virtual RFCOMM modules, each being an agent for an RFCOMM protocol running on another device, by attempting to modify the BlueZ Bluetooth protocol stack itself. During the difficult development of these kernel modifications it was found that a much simpler solution could be provided by writing a user-space application, which was done and called rcpipe.

Rcpipe uses code both from the rctest application provided by BlueZ and datapipe, a simple TCP socket redirection application. After an initial test development was done a final implementation was completed that allowed for the RFCOMM services to be transported across the IP network. RFCOMM data that was sent to a local rcpipe

application was buffered between the RCOMM and TCP sockets and finally an RFCOMM connection was made from a hotspot to the device offering the service.

After the success of the rcpipe application the investigation turned to the L2CAP protocol. Another application, called l2pipe, was developed to transport L2CAP communications. Since L2CAP is a multiplexer of higher protocols such as RFCOMM, being able to transport at this level would be a valuable contribution. Because of the consistent BlueZ API this was made easier as concepts and code could be reused from rcpipe with minor additions such as taking into account MTU sizes.

The second part of this chapter then looked at the implementation of the Distributed Service Database (DSDB) Daemon as a proof-of-concept for Service Discovery Component.

Finally, the chapter concluded with a discussion of issues with the core lower layers that needed to be considered, such as master/slave role switch and security of the Bluetooth communication.

Chapter 6:

RESULTS AND DISCUSSION

This chapter will look at our experiences we have had with BlueSpot and the applications that encompass it. We look at the effectiveness of the various programs under "real-world" conditions. We provide performance results on the communications via the pipes compared to communications directly between Bluetooth devices. We then investigate what protocols and services can be transported using BlueSpot. Finally we look at some scalability issues and possible solutions.

6.1 Introduction

In this chapter we discuss the results of a number of experiments that were performed to ascertain the effectiveness and feasibility of BlueSpot for extending the reach of a Bluetooth network. The chapter is divided up into three main sections, each considering a different layer of the system.

The first section deals with the bottom layer of the system, the network and data carrying layer of the Transport Component. A number of experiments were performed to test the impact of the throughput and latency on the Bluetooth communications that are carried by the Transport Component applications.

The second section discusses the protocols and application layers of the system. The different protocols, profiles and services that are available were investigated and a variety of test were performed to determine what services can be handled by BlueSpot.

The final section is an assessment of the scalability issues that affect BlueSpot. The limitation in RFCOMM channel numbers and how services can be advertised to users are discussed.

6.2 Throughput Testing

In order to ascertain if the transporting applications affect the bandwidth of the communication channels a number of throughput tests were performed. To test the throughput, the protocol testing applications provided by BlueZ were used.

6.2.1 Testing Applications

The BlueZ suite provides two applications, l2test and rctest to perform a number of tests on the L2CAP and RFCOMM protocols respectively. To perform a test the application is run on two different computers with Bluetooth connectivity. On one computer the application is placed into a listening mode and waits for a connection

from another device. On the other computer the application is told to connect to the listing computer, and then creates “chunks” of data that are then sent across the Bluetooth network to the listening application. The listening application then calculates throughput.

Before testing, the l2test application that came with BlueZ was modified slightly. The first few bytes of each L2CAP packet that l2test sent represented a sequence number and packet length. On the receiving l2test, each packet was checked for length and sequence. Since this test measured the bare throughput of data it was unnecessary for the application to perform this test on each packet and affect the results by increasing the duration time artificially. In the rctest application provided by BlueZ these checks were already commented out. Therefore checks were commented out of the l2test source code and the application was recompiled for these tests.

Each program sends a packet of data to the receiving application. In the case of l2test, these packets of data were made to fit inside a single L2CAP packet to maximise performance. In RFCOMM, since it is stream based, this was not an issue. Programs were set up to read a larger amount of data than was sent. This was to allow the data to fit into a TCP packet for best networking performance while having sufficient duration to give realistic results.

6.2.2 Device Setup

Four Bluetooth devices are required to test the BlueSpot system. All four “devices” are computers running BlueZ on a Linux system with USB Bluetooth dongles and are shown in Table 3. Two of the computers act as Bluetooth device A and B and are not running any BlueSpot software. The other two computers are running the BlueSpot applications and act as hotspots.

Table 3: Bluetooth devices used in the Throughput tests

Device	Operating System	Bluetooth Modules
Device A	Ubuntu 5.10	Gigabyte
Device B	Ubuntu 5.10	D-Link BT-120
Hotspot X	Gentoo	D-Link BT-120
Hotspot Y	Gentoo	D-Link BT-120

On all four devices Inquiry Scan was disabled on the Bluetooth module to prevent the module from entering the Inquiry Scan state and adversely affecting the throughput results. Throughput is less when Inquiry Scan is enabled since the module would stop sending and receiving data to enter the Inquiry Scan state. The modules were also set to stay as master devices and role-switch if necessary.

6.2.3 Tests Setup

To test the effect of the transport applications on the throughput of Bluetooth communications throughput on a direct connection between two Bluetooth devices, and a connection that uses the Bluetooth hotspots as shown in Figure 34, was comapred.

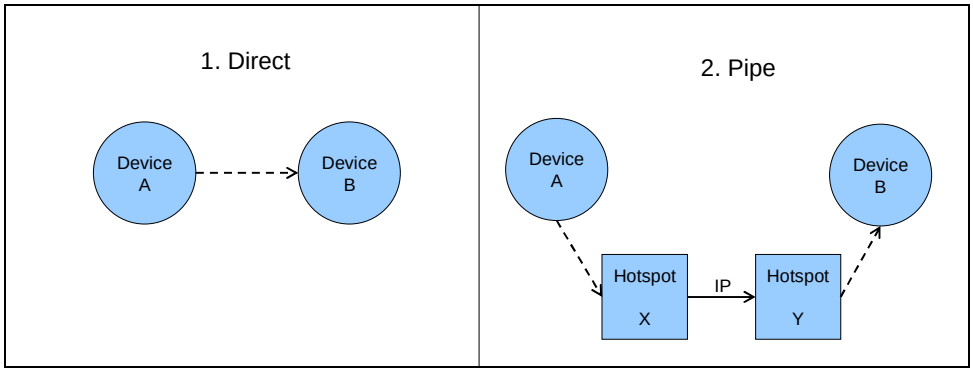

Figure 34: Throughput test setup

Direct: Device A connects directly to Device B using Bluetooth communication.

Pipe: Two Bluetooth hotspots are connected together by an IP network. Device A sends data to Hotspot X which transports it across the IP network to Hotspot Y that then sends the data on to Device B.

6.2.4 Theoretical Maximum Throughput

The theoretical maximum throughput of a Bluetooth link is dependent on what type and size ACL packet is used, which is chosen by the Baseband depending on the characteristics of the link it detects. The best possible performance is obtained when a DH5 packet is used in an asymmetric channel with a forward throughput of 723.2 kilobits/sec as seen in Table 4. This equates to Bluetooth having a theoretical maximum throughput of 90.4 kilobytes/sec when there is no interference. This does not take into account any control messages or protocol headers which would decrease the throughput in a real-world situation.

Table 4: ACL Packet throughput

The highlighted row shows the maximum Baseband throughput in a single direction [5]

Type	Payload Header (bytes)	User Payload (bytes)	FEC	CRC	Symmetric Max. Rate (Kbits/s)	Asymmetric Max. Rat (Kbits/s)	
						Forward	Reverse
DM1	1	0-17	2/3	yes	108.8	108.8	108.8
DH1	1	0-27	No	yes	172.8	172.8	172.8
DM3	2	0-121	2/3	yes	258.1	387.2	54.4
DH3	2	0-183	No	yes	390.4	585.6	86.4
DM5	2	0-224	2/3	yes	286.7	477.8	36.3
DH5	2	0-339	no	yes	433.9	723.2	57.6
AUX1	1	0-29	no	no	185.6	185.6	185.6

6.2.5 L2CAP

First the transporting of the L2CAP protocol was tested by using the written l2pipe and the supplied l2test applications.

6.2.5.1 Direct

Figure 35 shows the graph of five independent tests of the throughput measured when the devices communicated directly with the last bar showing the average of the five tests. As can be seen the results are extremely consistent with an average of 85.7 kilobytes/sec, which is 95% of the theoretical maximum.

6.2.5.2 Pipe

The results for the throughput test where the devices communicated via the hotspots are shown in Figure 36. The average for this test is the same as the average for a direct connection with the individual results showing very small deviations. The results are understandably more variable than the direct connection due to the increased complexities of completing the connection.

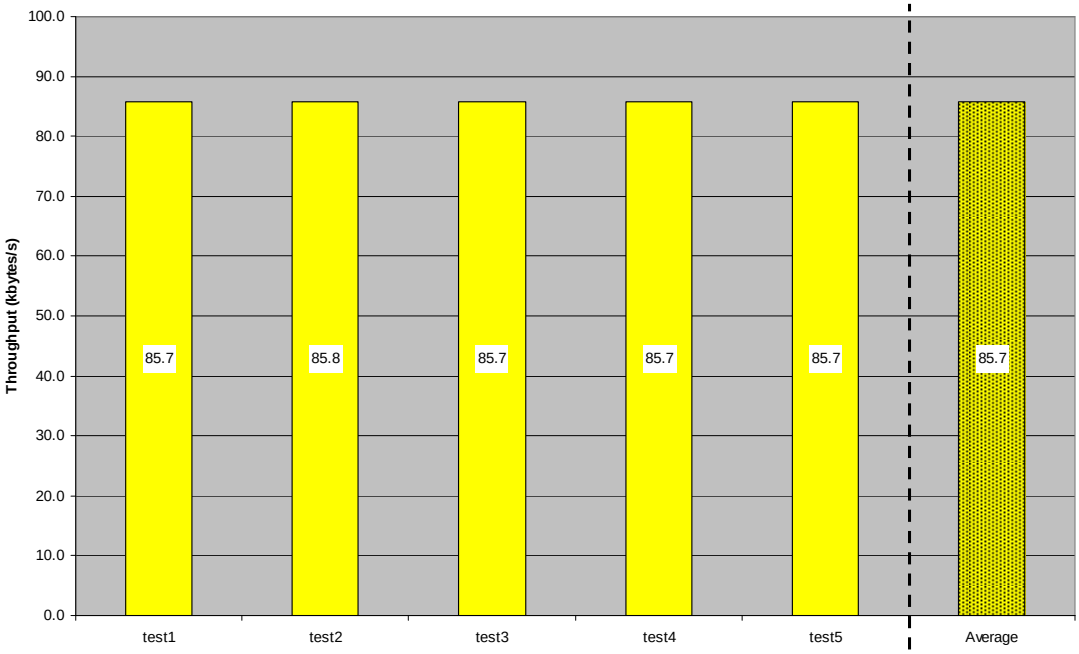

Figure 35: Throughput for direct L2CAP connection

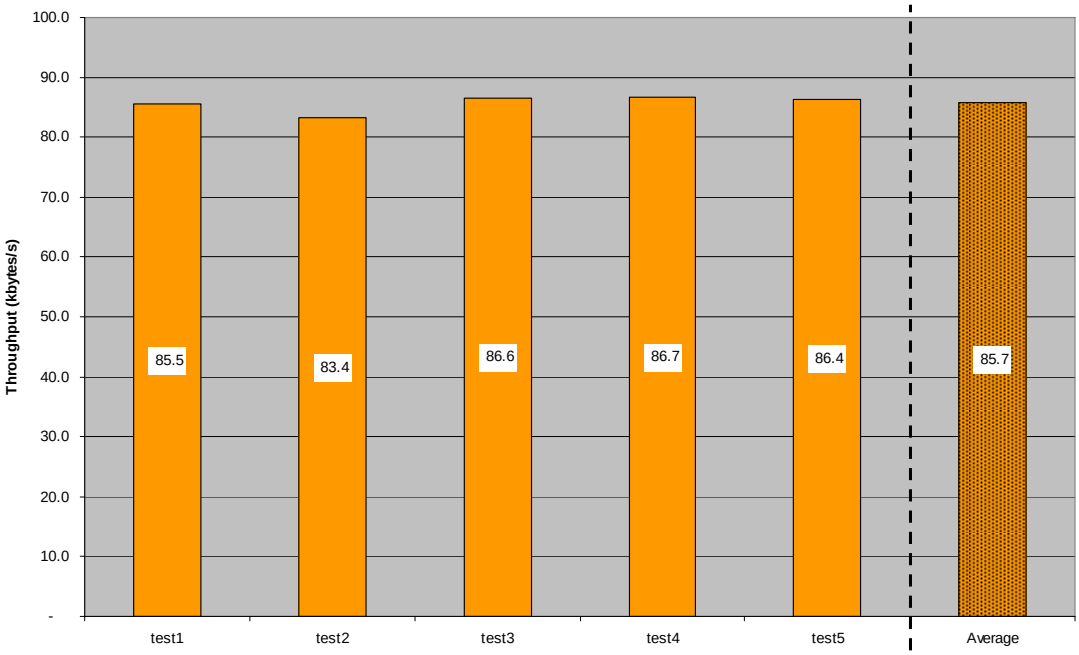

Figure 36: Throughput of transported L2CAP connection

6.2.6 RFCOMM

The tests were then repeated using rctest to test how rcpipe affected throughput.

6.2.6.1 Direct

First off the direct connection was tested and the results graphed in Figure 37. Five independent tests were performed and the average calculated. The average throughput is 78.1 kilobytes/sec, which is almost 9% less than the above tested L2CAP throughput.

6.2.6.2 Pipe

The test was repeated using the hotspots and the average throughput calculated. It was noted that there was more variance in results amongst the five tests as shown in Figure 38. This is most likely due to RFCOMM being a stream based protocol transported by TCP, a packet based protocol, and the chance that the end of a “chunk” of data is placed in a separate TCP packet and thus slightly delaying the completion of the “chunk” of data. The average was calculated to be 71.6 kilobytes/sec, which is 91.6% of a direct connection.

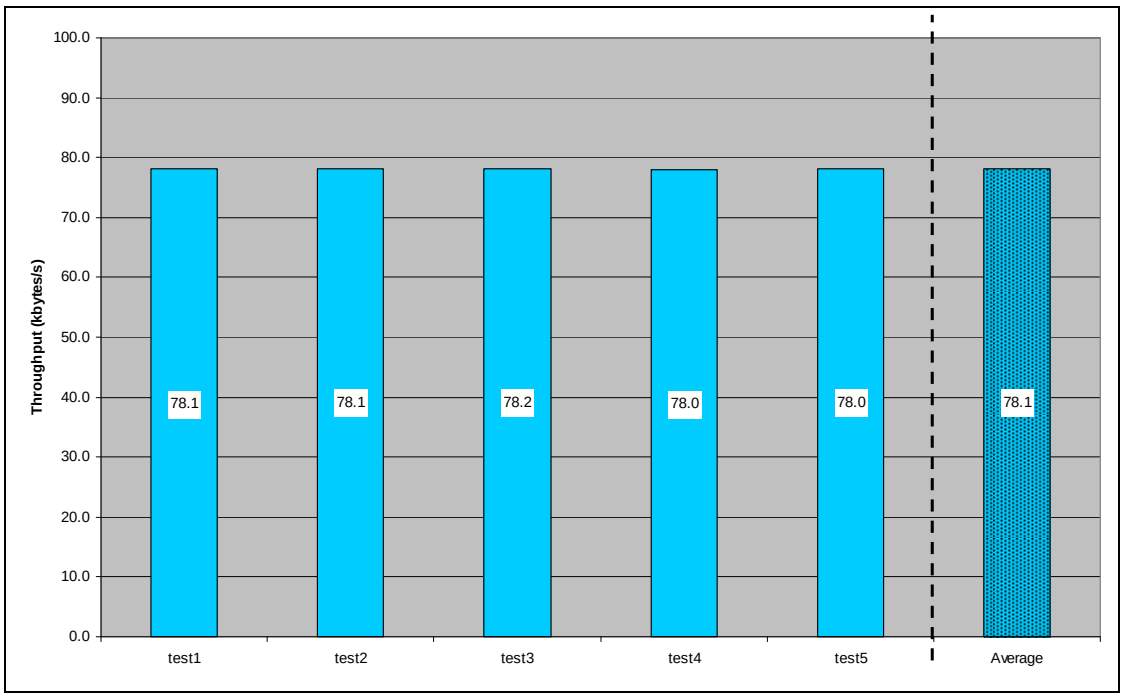

Figure 37: Throughput of a direct RFCOMM connection

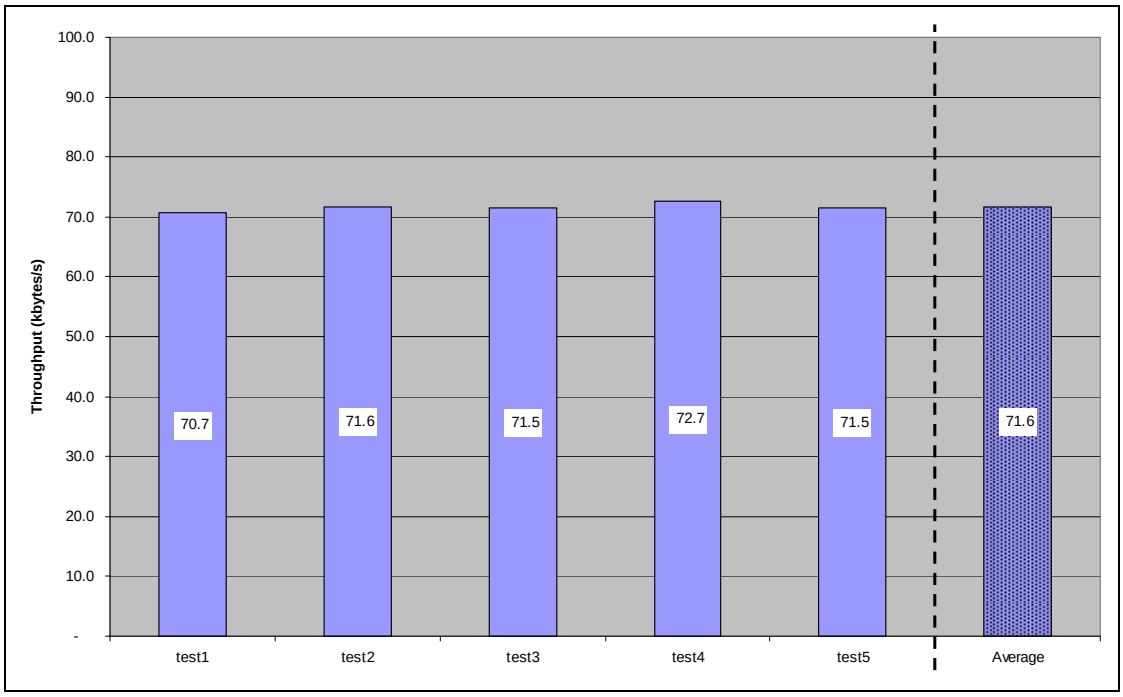

Figure 38: Throughput of a transported RFCOMM connection

6.2.7 Throughput Comparisons

In Figure 39 the averages of the throughput tests are graphed alongside the theoretical maximum throughput. As the tests have demonstrated, the decreases in throughput caused by transporting the Bluetooth communication across an IP network using the developed l2pipe and rcpipe is almost non-existent in the L2CAP case and within 10% for the RFCOMM case.

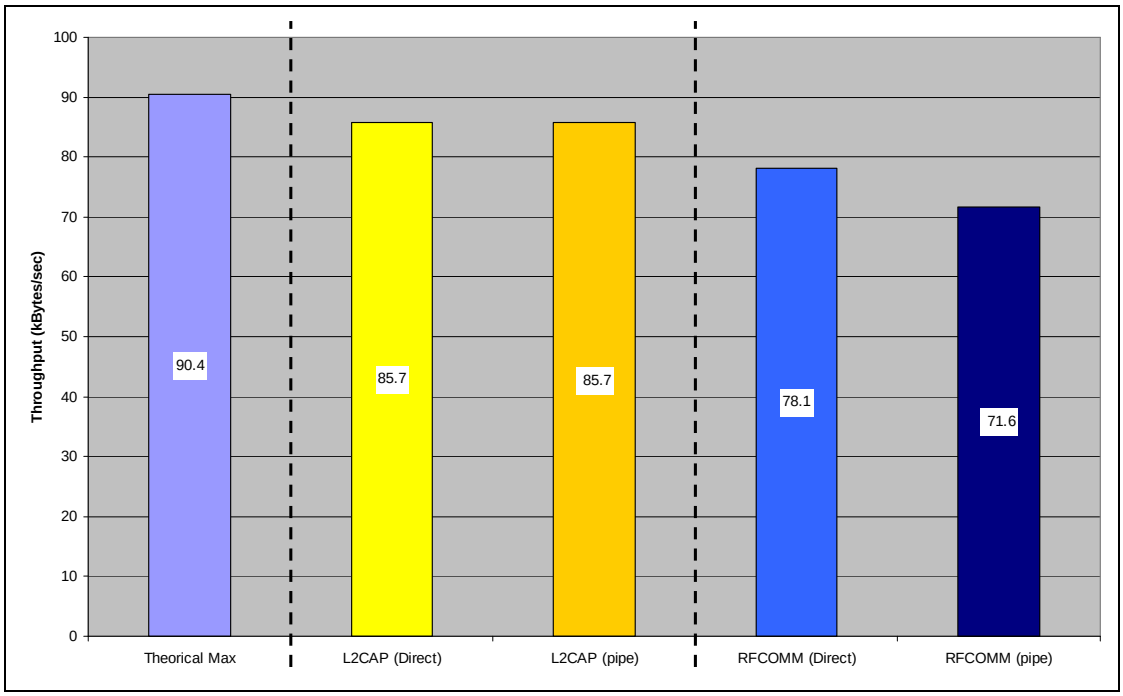

Figure 39: Comparison of throughput averages

6.3 Latency

While testing the throughput of the transport applications a noticeable delay was observed when communicating using the transport applications. This is caused by the lag between creating a Bluetooth connection and the creating of the TCP connection and to a lesser degree by the copying and buffering between the different sockets on either end. To measure latency the BlueZ testing applications, rctest and l2test, were used. The programs were set up to send a single packet. The time from starting the application until the data is received on the other side was measured using a stopwatch. The experiment was repeated five times and the averages are shown in

Table 5 and Table 6 with the differences between a direct connection and one using the relevant transport application.

Table 5: Latency test for l2pipe

	Delay (sec)	% increase
direct	3.724	
l2pipe	4.984	
Difference	1.26	33.8 %

Table 6: Latency test for r2pipe

	Delay (sec)	% increase
direct	3.690	
rcpipe	4.966	
Difference	1.276	34.6 %

The tests showed that the transport applications increase the delay by approximately 1 ¼ seconds (~34%). As most connection would be longer then 5 seconds, this delay would not be very noticeable to the average user.

6.4 Bluetooth Services

In the previous section we looked at the performance characteristics of the two transport applications and showed not only that connections could be made but that they were comparable to direct connections. Users are more interested in applications and what services are available, and therefore we need to consider what services are supported by BlueSpot.

Bluetooth services can use a variety of protocols for transport. In Figure 40 a hierarchy of some of the Bluetooth protocols are shown, with the protocols that have transport applications written for them highlighted, namely l2pipe and rcpipe. Because of the hierarchy of protocols, if we are able to transport one protocol, we should be able to transport any protocol that relies on it. Therefore, since we have transport applications for L2CAP and RFCOMM, we should be able to also transport

OBEX and BNEP respectively. By examining at the diagram it would seem that we should also be able to transport telephone control, but unfortunately this is not the case since telephone control also has a dependency on an SCO channel to carry the voice communication for which we do not have a transport application.

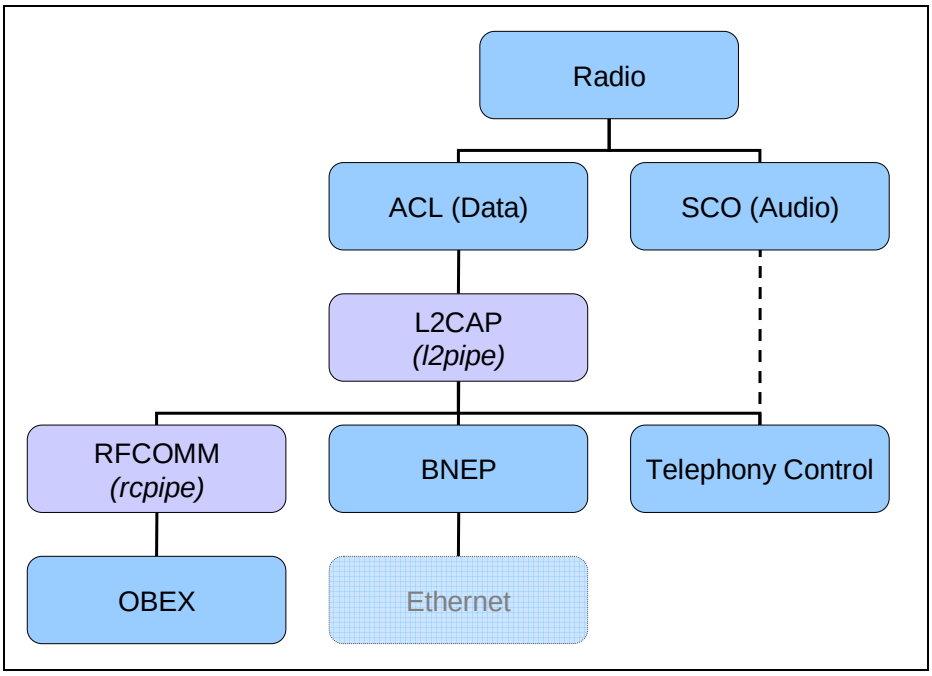

Figure 40: Bluetooth Protocol Hierarchy

Transport applications are shown for protocols that are transported

There are over 40 different Bluetooth profiles currently defined [57], some defining multiple services and to test all of them would take an unnecessarily long time. However, using the fact that the profiles are defined in a hierarchical manner and are further grouped together, a number of services could be picked which would give a representative sample. There are three Bluetooth protocols that can use the BlueSpot transport applications, BNEP, RFCOMM and OBEX, as seen in Figure 40. From each of these three protocols a service was chosen, shown in Table 7, to test if the protocol still works while being transported using the Bluetooth hotspots. An application to transport a simple string using the transport applications was first used to show that data could be passed by BlueSpot.

Table 7: Protocols tested and the service used to test them

Protocol tested	Transport Application	Service used to test
BNEP	l2pipe	PAN User
RFCOMM	rcpipe	Dialup Networking
OBEX	rcpipe	OBEX Object Push

An informal survey was taken amongst the Bluetooth devices available in the lab to see if the chosen services were a good representative of available services on actual devices. The three services are highlighted in Table 8 and as can be seen are well represented.

Table 8: Examples of services advertised

Device Type	UUID	Protocol	Nokia 6600	Nokia 6230i	iPAQ h4150	iPAQ h6340
Fax	0x1111	RFCOMM	*			
Generic Telephony	0x1204		*	*		
Dialup Networking	0x1103	RFCOMM	*	*	*	*
Generic Networking	0x1201		*	*		
Serial Port	0x1101	RFCOMM	*	*	*	*
OBEX File Transfer	0x1106	OBEX	*	*	*	*
OBEX Object Push	0x1105	OBEX	*	*	*	*
Handfree Audio Gateway	0x111f	RFCOMM	*	*		*
Generic Audio	0x1203	RFCOMM	*	*		*
PAN Group Network	0x1117	BNEP			*	*
PAN User	0x1115	BNEP			*	*
Headset Audio Gateway	0x1112	RFCOMM		*		*
SIM Access	0x112d	RFCOMM		*		
SyncML Client		RFCOMM		*		

6.4.1 Test platforms

The platform used to test the transport application at transporting various services was the same as that used to test the performance of BlueSpot. Two hotspots are connected together by an IP network as shown in Figure 41. Device A sends data to Hotspot X which then transports it across the IP network to Hotspot Y that then sends the data on to Device B.

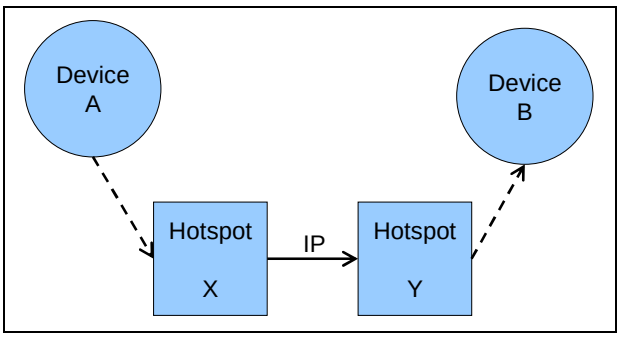

Figure 41: Test Platform

For some of the tests above an alternative testing platform was used that consisted of only two computers and two Bluetooth dongles, with each computer playing a dual-role of both a hotspot and a Bluetooth device shown in Figure 42.

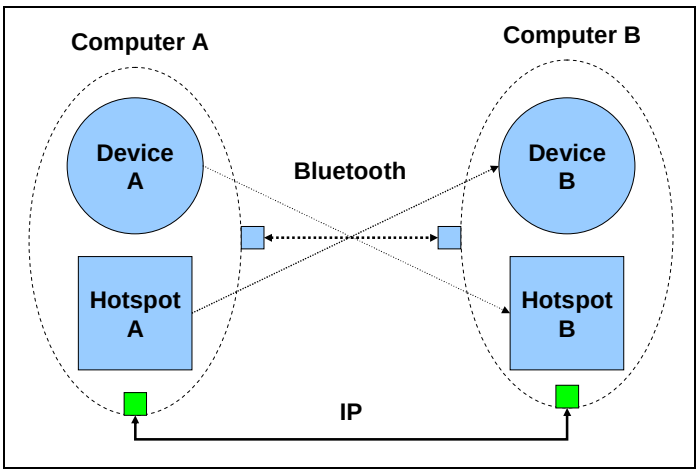

Figure 42: Alternative test platform

Device A connects to Hotspot B via Bluetooth, either via L2CAP or RFCOMM depending on the test. Hotspot B will then transport the Bluetooth data across the IP network to Hotspot A which will then connect to Device B to complete the connection. Notice that Device A and Hotspot A are the same computer, and share a single Bluetooth dongle. This is possible as there is a single baseband ACL connection between the two machines which is able to multiplex the two Bluetooth communications that are going in opposite directions. Note that this test only considers whether a service can be transported and used successfully and not at the performance characteristics since the module is shared by two applications and the throughput would be halved.

6.4.2 Protocols and Services Tested

6.4.2.1 L2CAP and RFCOMM "Hello, World!" examples

Though the performance tests above proved that it was possible to send data across the network, it needs to be checked that the data being passes is readable on receipt. Thus two simple server-client applications were written for testing this in L2CAP and RFCOMM. The server application opens the relevant Bluetooth socket and waits for a connection from the client, which when started connects to the server and passes a "Hello, world!" message which the server then displays. In both case the message "Hello, world!" was outputted by the server application demonstrating that the data was passed correctly between the hotspots for this simple case.

6.4.2.2 Bluetooth Network Encapsulation Protocol – PAN User

The PAN User (PANU) service was chosen to test how well a BNEP connection can be transported. PANU Service allows two PANU devices to communicate directly via a point-to-point connection. For this test the four devices are the same four computers used for the throughput tests shown in Table 3. Using pand, an application included with BlueZ it was possible to set up a BNEP connection between Device A and Device B. This created a virtual network driver on each device called bnep0 which was then assigned an IP address. Once the IP address was assigned it was possible for either device to ping the other. An SSH tunnel was then setup across this BNEP connection (which itself was being transported by l2pipe) over which websites were viewable.

6.4.2.3 RFCOMM Protocol - Dial-Up Networking

The Dial-Up Networking (DUN) [58] service allows a computing device (such as a PDA or laptop) to connect to a telephone network using the service of a communication device (such as a modem or cellular phone). DUN profile is part of the group of profiles that use the facilities provided by the Serial Profile, which in turn uses the RFCOMM protocol. Therefore the transport application that must be used is rcpipe. A service record is also added for the DUN service using sdptool so a Bluetooth device can establish which RFCOMM channel to connect too.

For this test the computing device was an iPAQ h4150 PDA (Device A) and the communication device was a Nokia 6600 phone (Device B) with a GPRS connection. The PDA was able to connect to the phone via the transport application and is able from there to connect to the Internet. This was tested by visiting a webpage which downloaded successfully.

6.4.2.4 OBEX Protocol– OBEX Object Push

OBEX is a binary protocol that allows for the exchange of data between an number of devices, simply and spontaneously, and was discussed in Chapter 2. To test the OBEX protocol the OBEX Object Push service was chosen. OBEX uses the RFCOMM protocol so once again the rcpipe transport application is used.

The device used in this test was a Nokia 6600 phone which would send a picture to an iPAQ h4150 PDA. The phone was able to successfully send pictures (JPEG files) to the PDA multiple times.

6.4.3 Protocols Not Tested

6.4.3.1 Audio-based profiles

No tests were done for services that required an audio channel. As was mentioned before, SCO audio is out of the scope of this project and there are no transport applications written to carry SCO/audio data. Thus services such as headset/hands-free as well as the Telephone Control Protocol based profiles were not tested. This proved to be a good decision since there is over a second latency added to the Bluetooth connections by the transport applications, The ITU-T[3] recommendations on one-way voice communication delays [59] states that anything over 400ms is unacceptable.

6.4.3.2 Human Interface Device

The Human Interface Device (HID) specification [60] allows peripherals such as keyboards, mice, game controllers, bar scanners and other interface devices to connect to a computing device wirelessly. Bluetooth HID uses the USB HID specification [61,

[3] ITU Telecommunication Standardization Sector

62] to leverage already existing support and drivers for devices and explains how to implement it over a Bluetooth connection. As extending the range of an HID device is impractical from a usability perspective, transporting the HID protocol was not considered.

6.4.3.3 Audio/Video Transport Protocols

The Audio/Video Control Transport Protocol (AVCTP) [63] and the Audio/Video Distribution Transport Protocol (AVDTP) [64] are a pair of protocols that manage and transport audio/video data. The AVCTP allows for the exchange of messages used to control multimedia device over a point-to-point connection whereas ACDTP facilitates the streaming of multimedia over Bluetooth and is based on the Real-time Transport Protocol (RTP) [65]. There was no hardware supporting these protocols available for tests and in fact the authors have not come across any devices that support either of these protocols at this time.

6.4.4 Analysis of Protocol Results

In the above tests the two transport applications were tested to see what protocols they were able to transport and therefore what services would be candidates for proxying across the network. First, two simple client-server applications that sent the text “Hello, World!” were used to show that data could be passed correctly and in the correct byte order as there was no garbage output on the other side.

Since our own protocols worked, attention was then turned to Bluetooth’s own protocols. If we consider Bluetooth’s profiles in Figure 43 we can see that there are three layers of profiles each on building on the one below. As noted in the introduction, it would take a very long time to test each and every profile and all its service combinations so a service from each profile group was chosen. Since they all have similar requirements (by requiring the same base profile), if one works, the others should work as well. The three services chosen were the PANU service from the Personal Area Network Profile which uses the BNEP protocol, the DUN service from the Dial-Up Networking Profile which uses the RFCOMM protocol and OPUSH Service of the Object Push Profile which uses the OBEX Protocol. PANU tested the

transport of L2CAP communications by l2pipe and DUN and OPUSH tested the transporting of RFCOMM communications by rcpipe.

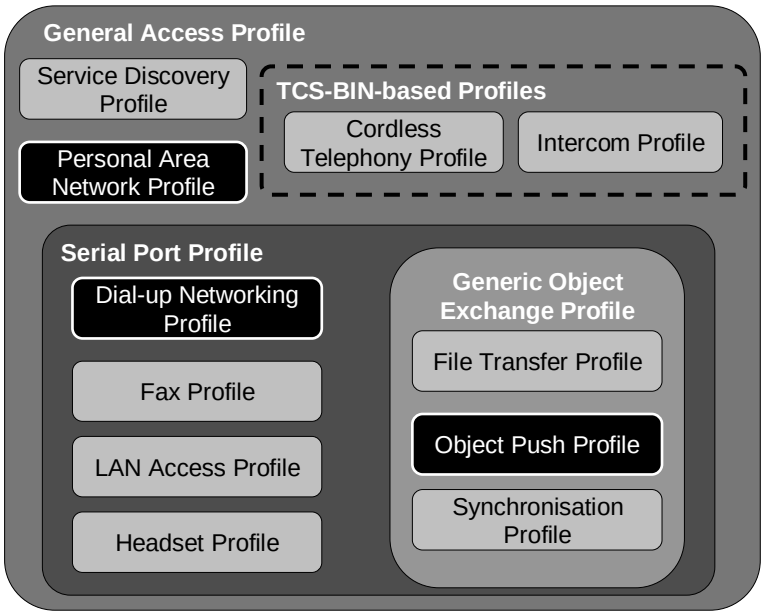

Figure 43: Bluetooth Profiles hierarchy showing services tested

Through our tests we have shown that a number of services are capable of being transported across the network, specifically PANU, DUN and OPUSH, on a broader scale, most services that use RFCOMM for data transport should be transportable, with the exception of the few services that use SCO connections for audio transport. With L2CAP communications it was shown that a string of characters could be transported, as well as the BNEP protocol.

6.5 Scalability Assessment

Apart from the networking and services testing done in the previsions two sections, issue of scalability also needs to be looked at and possible complications assessed.

6.5.1 Channel availability

The availability of channels or PSM numbers available on each hotspot to be able to re-advertise all their services that are detected by BlueSpot is a concern. Figure 44 shows a hypothetical situation with three Bluetooth hotspots and a total of six

Bluetooth devices. The numbers inside the circles represent how many services which device is advertising that can be transported by the hotspot. The numbers inside the hotspots represent the number of services it detects locally (near) and how many it has to be able to re-advertise from other hotspots (far).

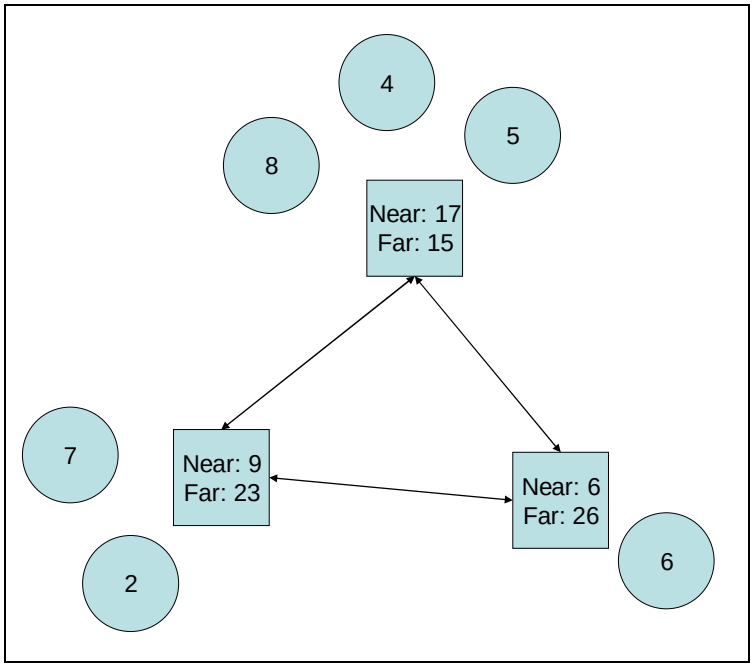

Figure 44: Channel scarcity

L2CAP uses PSMs to separate different channels. There are 32 768 possible PSM values with only approximately 20 having already been defined. Therefore there is address space for a number of L2CAP channels.

RFCOMM has a limit of 30 channels by specification. This will be an issue if there are many hotspots that are connected together. If we consider the survey of services offered done in section 6.4 listed in Table 8 we see that one device has over ten services, while the device with the lowest numbers of services has six. In addition, if we examine Figure 44 we can see that one hotspot is already proxying 26 services. That means that all channels could be used up if another device comes within range of a remote hotspot.

One possible way of dealing with this limitation is to be very selective of which services are picked up by the Service Discovery Component. All services that have

requirements that are not met by the hotspot such as requiring SCO transport should by default not be re-advertised. If there are services that perform similar functions then only one should be advertised. For example, if both OPUSH and OFTP are offered, then only OPUSH needs to be advertised as it is able to transfer files as well as other objects.

6.5.2 Human Interface and Service Identification

There are two main scalability issues when it comes to users utilising the Bluetooth hotspots. Due to a hotspot advertising the services of multiple devices there should be a way for a user to be able to identify the original device offering the service. One means for making the different services distinguishable is by including the name of the device that is offering the service in the "Service Name" field of the SDP record. Figure 45 shows a screenshot from a PDA running Microsoft Windows Pocket PC showing how this would look. Two DUN services are listed, one for a device called "skew" and the other for a device called "twisted". The device "crooked" is also offering an OFTP service.

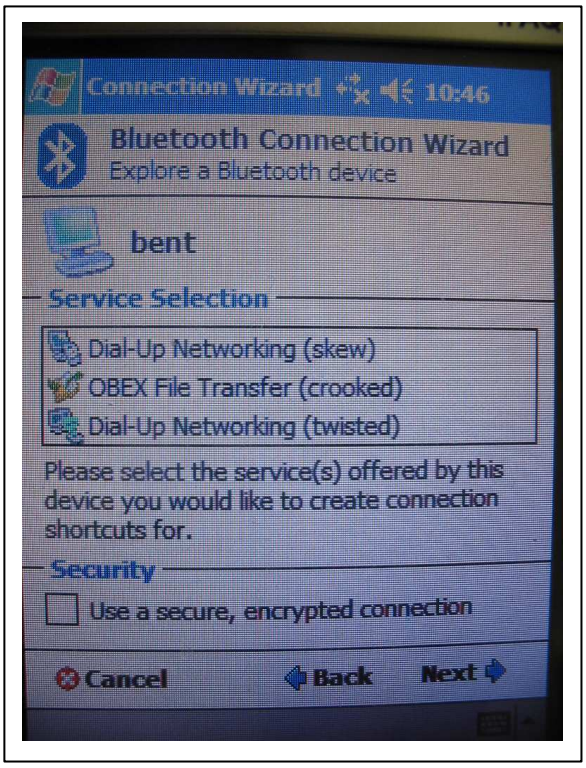

Figure 45: Screenshot of Service Selection from a Pocket PC

Figure 45 shows the second scalability issue. Mobile devices such as this PDA are limited in screen size and this limits the number of services that can be shown on the screen at a single time. This device can only show three services at a time and if there are many more than that it might take the user a while to scroll through all the others until they get to the service they are looking for. In larger Bluetooth hotspot networks where hundreds of services from multiple devices could be advertised it would prove impracticable for a user to find the service they are looking for. A number of possible solutions have been thought about such as the Service Database Component filtering out services that the requesting device does not support and not sending them. Dividing up the services into different groups and placing those groups into a hierarchical tree as discussed in section 2.10.3.2 and Figure 12 in Chapter 2 is another possible solution. Possible groupings are all the services from a single device, or all the common services from different devices

6.6 Summary

Throughput testing was performed by using testing applications provided by the BlueZ suite. A comparison was performed between a direct Bluetooth connection and a connection that used BlueSpot. The experiments show that there is almost no difference between throughput when transporting L2CAP communication using BlueSpot and a direct L2CAP connection between two Bluetooth devices. With RFCOMM, there is a less than 10% decrease in throughput. Therefore BlueSpot does not affect the throughput in an unacceptable way. Latency on the other hand is noticeably affected by as much as 1 ¼ seconds due to BlueSpot having to create two extra TCP connections to transport the data.

Through the tests on various services it was shown that a number of protocols could be transported by the rcpipe and l2pipe applications. It was also determined that most services that use the RFCOMM protocol would be able to be transported as well as some services that use the L2CAP protocol.

A number of scalability issues were discussed due to the limited number of RFCOMM channels and possible solutions to this were suggested. L2CAP does not have such a limited number of PSM numbers with over 32000 available. Issues with human interface due to limited screen size of portable devices were also considered and possible solutions discussed

Chapter 7:

CONCLUSION

In this chapter we discuss our success and limitations with BlueSpot. We re-examine the research questions posed in Chapter 1 and answers are given and discussed. We then make recommendations on future work that could be investigated.

7.1 Introduction

This project initially set out to design a system that would allow Bluetooth devices to take part in the same Piconet even though they were out of communication range by providing hotspots and transporting the Bluetooth communication over an IP network between these hotspots. This was to be done transparently and seamlessly to the user, with no difference in user experience whether they were within range of the device or using a hotspot. Early in the project it was established that such an idea was not feasible due to the way Bluetooth devices communicated using Frequency Hopping Spread Spectrum and how this pseudo random hopping sequence was dependent on the device's MAC address and a timing counter which was reset when its address was changed. A hotspot would not be able to listen in to multiple devices at a single time.

Bluetooth is a service-based communication system where connections are usually made to a service offered by a Bluetooth device rather than just connecting to the device. Therefore, instead of masquerading devices transparently, an investigation was done into how to proxy the services offered by a device across the network.

To allow Bluetooth devices to locate the services advertised by remote devices a distributed service database was written. This peer-to-peer based application locates services on devices within range of itself and advertises them on remote hotspots, while at the same time advertising services from remote hotspots. It also uses the transporting applications to set up the pipes between hotspots allowing for the remote connections.

7.2 Discussion of successes and limitations

7.2.1 Successes

7.2.1.1 Transport Component

The major success of the project was the design and implementation of the Transport Component applications rcpipe and l2pipe. These applications allow for the transfer

of the two most commonly used Bluetooth protocols, RFCOMM and L2CAP respectively, across an IP network and are used by BlueSpot to transport Bluetooth communication between hotspots.

7.2.1.2 Bluetooth services

A representative selection of different services were chosen to be tested, the PAN User service from the Personal Area Network Profile which uses the BNEP protocols, the Dial-up Networking service from the Dial-Up Networking Profile which uses the RFCOMM protocol and the Object Push service of the Object Push Profile which uses the OBEX Protocol. By selectively testing these different services, each of which fall within a different layer of Bluetooth's profiles, it was determined that not only these particular services are capable of being transported but that others using the same protocols should also be supported as well. Most services that use RFCOMM for data transport should be transportable, with the exception of the few services that also use SCO connections for audio transport. With L2CAP communications it was shown that our own L2CAP application was transportable, as well as the BNEP protocol.

7.2.2 Limitations

7.2.2.1 Service Discovery Component

Only a proof-of-concept was developed for the Service Discovery Component. For this project to be rolled-out and be used in a real world situation a number of features would still need to be implemented in the distributed service database daemon. Also a Management Component would also need to be written to manage the BlueSpot system.

7.2.2.2 Security aspects

Even though security was scoped out of this project, a number of security aspects were discussed that would needed to be considered to make BlueSpot as secure as using a Bluetooth device normally. A number of suggestions were made on how this could be tackled and should be considered for future research.

7.2.2.3 Audio

One of the shortcomings of this project is that it did not investigate the transport of audio. This was scoped out early in the project due to not being able to transport at a low enough layer of the Bluetooth protocol stack. Later, during testing, it became apparent that this was a correct decision due to latency and delay issues making it infeasible.

7.2.2.4 Scalability

Scalability is an issue due to the limited number of 30 RFCOMM channels available per Bluetooth device. This is not a problem with L2CAP, which has over 32000 available PSM values. The other issue was the difficulty a user would have to select the service they want when there is a large number of services being advertised by a single hotspot. A number of ways for dealing with this were discussed in section 6.5.

7.3 Research Questions Revisited

In section 1.3 we posed the main research question which this project set out to answer.

Can Bluetooth be seamlessly transported over an IP networks?

Yes, with some limitations. This project developed applications that are able to transport RFCOMM and L2CAP communication across an IP network.

The main research goal prompted a number of further, more detailed questions that have guided this research:

1. **To what extent can this be achieved without changes being required to commercially active devices?**

No modifications were needed to be made to any Bluetooth devices to be able to use the hotspots. A number of custom applications were written to offer transport and service discovery services that used an unmodified Bluetooth protocol stack. These

are transparent to the users of the system, who simply see the required service being provided locally.

2. What are the security implications of extending the reach of Bluetooth devices?

By placing hotspots in-between the communication between two Bluetooth devices it is difficult to secure the Bluetooth communication links between the devices and the hotspots. Some suggestions on how to address this are made in section 5.7.2.

3. What are the efficiency tradeoffs compared to a direct connection?

Throughput performance of these transport applications was very pleasing with no noticeable effect on throughput for L2CAP and no more than 10% decrease for RFCOMM when compared to direct connections. Latency on the other hand was noticeably affected by as much as 1¼ seconds, an increase of about 34%, due to BlueSpot having to create two extra TCP connections to transport the data.

4. What are the limits of scope at the application level?

The developed BlueSpot system is able to transport both RFCOMM and L2CAP communications and the services that use these two protocols. This means that the vast majority of data services offered by current Bluetooth devices are able to be transported. Voice uses a separate data channel called SCO that falls below L2CAP and out of scope of this project.

7.4 Future Work

7.4.1 Implementing Transport Components within the protocol stack

The initial investigation into developing the transport applications considered implementing them directly within the Bluetooth protocol stack by providing alternative kernel modules. This line of investigation was pre-empted when it was

discovered that a much easier way would be to implement them as user-space applications instead, as this would not entail kernel level development. Possible future work could consider the original approach of providing virtual RFCOMM protocol modules, one for each hotspot in the network, particularly as a means of overcoming the limited number of RFCOMM channels.

7.4.2 Security Investigation

Further investigation should be done into the security limitations of the BlueSpot system and how these can be solved. Though securing IP communications between the different hotspots is a relatively trivial problem there are a number of solutions that are available and the best solution would needed to be investigated. However the non-trivial problem of securing the Bluetooth communication between the Bluetooth devices that are using the hotspots is more interesting. Finding a solution that would allow two Bluetooth devices to authenticate each other though a third party without modification could be quite critical for a commercially viable system.

7.5 Use Case

In Chapter 3 a possible usage scenario involving a person in a meeting wanting to access information back in their office on their computer was introduced. BlueSpot now solves this problem. By means of hotspots in the boardroom and their office, linked using BlueSpot, they would now be able to use their Bluetooth enabled PDA to access their Bluetooth enabled computer to access the information they need. Thus, while a commercially viable system would require some further development, the project has successfully achieved its main goals.

GLOSSARY

ACL	Asynchronous Connection-Less
AES	Advanced Encryption Standard
API	Application Program Interface
AVCTP	Audio/Video Control Transport Protocol
AVDTP	Audio/Video Distribution Transport Protocol
BD_ADDR	Bluetooth Device Address
BNEP	Bluetooth Network Encapsulation Protocol
DLCI	Data Link Connection Identifier
DLL	Dynamic Link Library
DSDB	Distributed Service DataBase
DUN	Dial-Up Network
ERTX	Extended Response Timeout Expired (timer)
ETSI	European Telecommunications Standard Institute.
FHS	Frequency Hopping Synchronization (packet)
FHSS	Frequency Hopping Spread Spectrum
GSM	Global System for Mobile communication
HCI	Host Controller Interface
HID	Human Interface Device
IrDA	Infrared Data Association
ISO	International Organization for Standardisation
ITU	International Telecommunications Union
ITU-T	ITU Telecommunication Standardization Sector
L2CAP	Logical Link Control and Adaptation Protocol
LAN	Local Area Network
LMP	Link Manger Protocol

MC	Management Component
MTU	Maximum Transmission Unit
NAP	Network Access Point
OBEX	OBject EXchange
OFTP	OBEX File Transfer Protocol (service)
OPUSH	OBEX Push (service)
OSI	Open Systems Interconnection
OSS	Open Source Software
PAN	Personal Area Network
PANU	PAN User (service)
PCM	Pulse Code Modulation
PDA	Personal Digital Assistant
Piconet	A network of Bluetooth devices that is connected, and in sync with a single master.
PSM	Protocol / Service Multiplexer
RFCOMM	Bluetooth protocol providing emulated RS232 serial ports
RTP	Real-time Transport Protocol
RTX	Response Timeout Expired (timer)
SAFER	Secure and Fast Encryption Routine (algorithm)
Scatternet	A set of piconets connected through devices in multiple piconets
SCO	Synchronous Connection-Orientated
SDC	Service Database Component
SDP	Service Discovery Protocol
SIG	Special Interest Group – The Bluetooth SIG is the body that controls and manages the Bluetooth specification
SLP	Service Location Protocol
SSDP	Simple Service Discovery Protocol from UPnP
TC	Transport Component
UUID	Universally Uniquely Identifier

LIST OF REFERENCES

[1] J. Bray and C. Sturman , *Bluetooth: Connect without Cables.* Prentice Hall, 2001,

[2] InStat. *Bluetooth 2005: The future is here*. InStat, Sept 2005. [Online]. Available: http://www.instat.com/catalog/Ccatalogue.asp?id=161#IN0501837MI

[3] Bluetooth SIG. *Bluetooth Test Specifications*. Oct 2006. [Online]. Available: https://www.bluetooth.org/testspecs/

[4] Bluetooth SIG. *Bluetooth Qualification Program Reference Document.* (2.0 ed.) April 2005. [Online]. Available: https://www.bluetooth.org/qualification/

[5] Bluetooth SIG. *Specification of the Bluetooth System Version 1.2.* Nov 2004. [See attached CD]. Available: \bluetooth_specs\BT_Core_v1_2.pdf

[6] R. Morrow, *Bluetooth Operation and use.* McGraw-Hill, 2002,

[7] Bluetooth SIG. *Part F:1 RFCOMM with TS 07.10.* June 2003. [See attached CD]. Available: \bluetooth_specs\rfcomm.pdf

[8] ETSI. TS 07.10, version 6.3.0.

[9] Bluetooth SIG. *IrDA interoperability.* Feb 2001. [See attached CD]. Available: \bluetooth_specs\OBEX.pdf

[10] ISO, *ISO 7498-1:1994*. ISO/IEC, 1994,

[11] Y. Goland, T. Cai, P. Leach, and Y. Gu, *Simple Service Discovery Protocol -- Version 1.0*, internet draft, Microsoft, April 2000. [Online]. Available: http://quimby.gnus.org/internet-drafts/draft-cai-ssdp-v1-03.txt

[12] F. Dawson and D. Stenerson. *Internet Calendaring and Scheduling Core Object Specification (iCalendar*). RFC 2445. Nov 1998. [Online]. Available: http://tools.ietf.org/html/rfc2445

[13] Internet Mail Consortium. *VCard - the Electronic Business Card Version 2.1.* Sept 1996. [Online]. Available: http://www.imc.org/pdi/vcard-21.txt

[14] Infrared Data Association. *IrDA Object Exchange (OBEX) Protocol*. [Online]. Available: http://irda.affiniscape.com/displaycommon.cfm?an=1&subarticlenbr=7

[15] Bluetooth SIG. *Part K:10 Generic Object Exchange Profile*. Feb 2001. [See attached CD]. Available: \bluetooth_specs\10_goep.pdf

[16] Bluetooth SIG. *Part K:13 Synchronization Profile*. Feb 2001. [See attached CD]. Available: \bluetooth_specs\13_syp.pdf

[17] Bluetooth SIG. *Part K:12 File Transfer Profile* Feb 2001. [See attached CD]. Available: \bluetooth_specs\12_file_transfer.pdf

[18] E. Guttman, C. Perkins, J. Veizades and M. Day. *Service Location Protocol, Version 2*. RFC 2608. June 1999

[19] Bluetooth SIG. *Bluetooth Security White Paper.* May 2002. [Online]. Available: http://www.bluetooth.com/Bluetooth/Apply/Technology/Research/Bluetooth_Security_White_Paper.htm

[20] J. L. Massey, G. H. Khachatrian and M. K. Kuregian. “SAFER+ cylink corporation's submission for the advanced encryption standard.” Presented at *Proc. 1st Advanced Encryption Standard Candidate Conf.* Aug 1998. Available: http://csrc.nist.gov/CryptoToolkit/aes/round1/conf1/saferpls-slides.pdf

[21] Sun Microsystems. *Jini*. [Online]. Available: http://www.sun.com/software/jini/

[22] The Salutation Consortium Inc. *Salutation architecture specification*. 1999 [Online]. Available: http://www.salutation.org/

[23] N. A. Nordbotten, T. Skeie and N. D. Aakvaag. Methods for service discovery in bluetooth scatternets. *Computer Communications 27(11),* pp. 1087-1096. Sept 2004.

[24] P. Leach, M. Mealling and R. Salz. *A Universally Unique IDentifier (UUID) URN Namespace.* RFC 4122. July 2005.

[25] ITU, *Information Technology - Open Systems Interconnection - Procedures for the Operation of OSI Registration Authorities: Generation and Registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier Components, X.667.* International Telecommunication Union, 2004.

[26] Bluetooth SIG. *Bluetooth Assigned Numbers Document*. Dec 2006. [Online]. Available: http://www.bluetooth.org/assigned-numbers/

[27] Broadcom Corporation. *Boardcom website*. [Online] Available: http://www.broadcom.com/

[28] Bluetooth Qualification Program: *Qualified Product: BlueStack (widcomm).* June 2000. *[Online]* Available: http://qualweb.bluetooth.org/Template2.cfm?LinkQualified=QualifiedProducts&Details=Yes&ProductID=8

[29] Bluetooth Qualification Program. *Qualified Product: BTW bluetooth communication software for windows (widcomm).* Nov 2003. [Online]. Available: http://qualweb.bluetooth.org/Template2.cfm?LinkQualified=QualifiedProducts&Details=Yes&ProductID=250

[30] Broadcom Corp. *Bluetooth RF silicon and software solutions*. [Online]. Available: http://www.broadcom.com/products/Bluetooth/Bluetooth-RF-Silicon-and-Software-Solutions

[31] Bluetooth Qualification Program. *Qualified Product: Bluetooth support for Windows XP SP2; Bluetooth support for Windows Vista (Microsoft).* Aug 2004 Available: http://qualweb.bluetooth.org/Template2.cfm?LinkQualified=QualifiedProducts&Details=Yes&ProductID=956

[32] Microsoft Corp. *Windows sockets support for Bluetooth*. [Online]. Available: http://msdn.microsoft.com/library/en-us/bluetooth/bluetooth/windows_sockets_support_for_bluetooth.asp

[33] Microsoft Corp. *Bluetooth reference*. [Online]. Available: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/bluetooth/bluetooth/bluetooth_reference.asp

[34] V. Bhardwaj and R. Heydon. “Bluetooth on the PC.” Presented at *Microsoft WinHEX 2005*. March 2005. [Online]. Available: http://download.microsoft.com/download/9/8/f/98f3fe47-dfc3-4e74-92a3-088782200fe7/TWMO05002_WinHEC05.ppt

[35] R. Heydon. “Bluetooth driver development interfaces overview.” Presented at *Microsoft WinHEX 2005*. March 2005. [Online]. Available: http://download.microsoft.com/download/9/8/f/98f3fe47-dfc3-4e74-92a3-088782200fe7/TWMO05002_WinHEC05.ppt

[36] BlueZ Project. *BlueZ: Offical Linux Bluetooth Stack*. [Online]. Available: http://www.bluez.org/

[37] Qualcomm Inc. *Qualcomm website*. [Online]. Available: http://www.qualcomm.com/

[38] Free Software Foundation. *GNU General Public License*. Cambridge, Massachusetts, 1991. [Online] Available: http://www.fsf.org/licenses/gpl.html

[39] BlueZ Project. *BlueZ: History*. [Online]. Available: http://www.bluez.org/history.html

[40] P. Lucistnik, “Bluetooth.” in *FreeBSD Handbook.* The FreeBSD Documentation Project, 2006. [Online]. Available: http://www.freebsd.org/doc/handbook/

[41] J. Elischer and A. Cobbs. *Netgraph(4) man page*. [Online]. Available: http://www.freebsd.org/cgi/man.cgi?query=netgraph&sektion=4

[42] Various contributors. *freebsd-bluetooth mailing list archives*. [Online].
Available: http://lists.freebsd.org/pipermail/freebsd-bluetooth/

[43] Various contributors. *Bluez-devel mailing list archives*. [Online].
Available: http://sourceforge.net/mailarchive/forum.php?forum_id=1881

[44] FreeBSD. *FreeBSD Status Report: Sept - Oct 2002. Oct 2002*. [Online].
Available: http://www.freebsd.org/news/status/report-sept-2002-oct-2002.html

[45] A. Huang. *An introduction to bluetooth programming in GNU/Linux*. June 2006.
[Online]. Available: http://people.csail.mit.edu/albert/bluez-intro/

[46] A. Huang, *The Use of Bluetooth in Linux and Location Aware Computing*. M.S. thesis. Massachusetts Institute of Technology, 2005.

[47] J. Lawson. *Datapipe. 1.0 source code.* 2005. [Online].
Available: ftp://ftp.distributed.net/pub/dcti/unsupported/datapipe-1.0.tar.gz

[48] T. Vierling, *Datapipe (orginal) source code*. [Online].
Available: http://packetstormsecurity.nl/unix-exploits/tcp-exploits/datapipe.c

[49] Linux Programmer's Manual. *Select(2) man page*. Feb 2001. [Online].
Available: http://man.cx/select(2)

[50] A. Lin, J. Heinanen, G. Armitage and A. Malis. *A Framework for IP Based Virtual Private Networks.* RFC 2764. Feb 2000.

[51] S. Kent and K. Seo. *Security Architecture for the Internet Protocol.* RFC 4301. Dec 2005

[52] D. J. Barrett and R. E. Silverman, *SSH, the Secure Shell: The Definitive Guide.* First ed. O'Reilly & Associates, Inc, Sebastopol, CA, 2001.

[53] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol.* RFC 4346. April 2006.

[54] cryptlib. *Cryptlib encryption toolkit*. [Online]
Available: http://www.cs.auckland.ac.nz/~pgut001/cryptlib/

[55] The OpenSSL Project. *OpenSSL: The open source toolkit for SSL/TLS*. [Online].
Available: http://www.openssl.org/

[56] J. Viega and G. McGraw. *Building secure software: How to avoid security problems the right way.* Addison-Wesley, Boston, 2002.

[57] Bluetooth SIG. *Bluetooth Specification Documents website*. [Online].
Available: https://www.bluetooth.org/spec/

[58] Bluetooth SIG. *Part K:7 dial-up networking.* Feb 2001. [See attached CD].
Available: \bluetooth_specs\7_dun0.pdf

[59] ITU-T. *Recommendation G.114 mean one-way propagation time*. ITU-T. 1988.

[60] Bluetooth SIG. *Human Interface Device (HID) Profile Version 1.0.* May 2003. [See attached CD]. Available: \bluetooth_specs\HID Spec v1_0.pdf

[61] USB Implementers Forum. *Device Class Definition for Human Interface Devices (HID) Version 1.11.* June 2001. [Online].
Available: http://www.usb.org/developers/devclass_docs/HID1_11.pdf

[62] USB Implementers Forum. *USB.org - HID tools*. [Online].
Available: http://www.usb.org/developers/hidpage/

[63] Bluetooth SIG. *Audio/Video Control Transport Protocol Specfication Version 1.0.* May 2003. [See attached CD].
Available: \bluetooth_specs\AVCTP Spec v1_0.pdf

[64] Bluetooth SIG. *Audio/Video Distribution Transport Protocol Specfication Version 1.0.* May 2003. [See attached CD].
Available: \bluetooth_specs\AVDTP Spec v10.pdf

[65] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications.* RFC 3550. July 2003.