

Bluetooth hotspots: Extending the reach of Bluetooth by seamlessly transporting Bluetooth communications over IP Networks

David Mackie and Peter Clayton
Department of Computer Science
Rhodes University, Grahamstown, 6140
Email: d.mackie@ru.ac.za | p.clayton@ru.ac.za
Tel: 046 603-8291; Fax: 046 636-1915

Abstract— This paper 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, since Bluetooth was designed as cable-replacement technology for wireless communication over short ranges. 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 Internet Protocol (IP). Two methods were investigated; spoofing of distant devices by the local hotspot to allow seamless communication, and creating a distributed service discovery database to offer alternative means for devices to locate and use each other's services. The latter method was used to develop applications that can tunnel the RFCOMM and alter the L2CAP layers of Bluetooth's protocol stack between two machines.

Index Terms— Bluetooth, Tunnelling, Hotspots, Service Discovery, Piconets

I. BLUETOOTH

Bluetooth is a specification [1] for a low-cost, low-power, short-range wireless communication technology. It was born out of a study undertaken by Ericsson in 1995 looking for a technology to replace cables and provides wireless connectivity between mobile devices such as cellphones, personal digital assistants (PDA) and portable computers. [2]

Though originally developed as a cable-replacement for point-to-point connections, Bluetooth was expanded into a wireless ad-hoc network technology allowing users to create networks with any devices within range on demand; what has been called Personal Area Networks (PAN), allowing them to interact with information technology on a totally new level.

The Bluetooth Special Interest Group (SIG) [3] is the trade association that is responsible for the Bluetooth specifications and consists of over 2000 different companies

comprising the leaders in telecommunication, networking and computing industries and who are driving the development of Bluetooth. Founded by Ericsson, IBM, Intel, Nokia and Toshiba in beginning of 1998 and later joined by Microsoft, Lucent, Motorola and 3Com in December 1999, they are now called SIG Promoters and are responsible for the SIG administration such as legal, marketing and setting criteria for devices to qualify to be official Bluetooth devices.

Bluetooth was envisioned as a global communication system and this has some bearing on the extensive detail the specification documents go into. The complete Bluetooth system is meticulously defined in the specification from the lower physical radio layer up to and including software applications, thus preventing potential inconsistencies when marketed by various independent companies. All the Bluetooth specifications are available on the Bluetooth SIG's website [3]. They are open for anyone to read and are patent and license free for SIG members, thus making it easier for a multitude of vendors to implement them without interoperability problems.

II. RESEARCH AIM

Currently Bluetooth devices cannot communicate with each other unless they are within radio range by design. This paper is an investigation into the creation of Bluetooth "hotspots" that would allow remote Bluetooth devices to communicate with each other. These hotspots would be connected to each other via an IP network and thus allow Bluetooth devices in range of different hotspots to communicate with each other by transporting the Bluetooth communications across the IP network. Both the feasibility of such a system and the extent to which it could be done seamlessly and transparently to the user were investigated.

III. BLUETOOTH RADIO

Bluetooth operates in the globally available 2.4GHz ISM (Industrial, Scientific & Medical) band, which is reserved for general applications with a set of basic power, emission and interference specifications. To be able to operate effectively in such a noisy radio-frequency environment, which it shares with microwaves ovens and other Wi-Fi technologies, Bluetooth employs Frequency Hopping Spread Spectrum (FHSS). FHSS is a modulation scheme that uses a narrowband carrier that changes frequencies in a pattern known to both the transmitter and receiver.. The band has been divided up into 79 channels each 1 MHz in size, and the hopping is done to a pseudo random pattern at

This work was undertaken in the Distributed Multimedia Centre of Excellence at Rhodes University, with financial support from Telkom SA, Comverse, Verso Technologies, THRIP, and the National Research Foundation. Scholarship funding was kindly provided by the National Research Foundation and DAAD.

1600 times per second. Data is broken up into very small packets which are transmitted one packet per hop

As a low-power, short-range wireless technology, Bluetooth was designed to be deployed primarily in mobile devices such as PDA's and mobile phones that run off batteries and needed to use as little power as possible. Because of this the range of the radio is limited. The Bluetooth specification defines three power classes; Class 1 – 100 meters, Class 2 – 10 meters and Class 3 – 10 centimetres. Most Bluetooth devices are Class 2 devices and therefore have an approximate range of only 10 meters.

IV. BLUETOOTH NETWORK

To allow Bluetooth devices to hop to a new frequency after each data transmission, they need to all agree on what that next frequency is and when to hop. To this end, a Bluetooth device can act in one of two roles; master or slave. A master device sets the frequency hopping sequence that the slave devices then match in both timing and frequency. Each Bluetooth device has a unique address, the Bluetooth Device Address (BD_ADDR) similar to an Ethernet MAC address, and a timing clock. The specification defines an algorithm that uses these to generate the pseudo random frequency hopping sequence. When a slave device connects to a master device it has to know the masters address for the connection and is told the masters clock. Using this information the slave then calculates the master's hopping sequence and matches it allowing it to connect to the master. Any other devices wanting to connect to the master as well would need to do so as a slave device and then match the master's hopping sequence as shown in Fig. 1(a).

A collection of multiple slave devices connected to one master device is called a piconet. All devices in the same piconet would follow the same frequency hopping sequence, that of the master's. A maximum of 7 slaves are able to connect to a single master as stipulated in the specification but a slave can be in more than one piconet and therefore link them into a larger scatternet shown in Fig. 1(c), though the specification does not make it clear how this can be done.

The master does not only set the hopping sequence but also manages all communications between devices by controlling when different devices can communicate. In a piconet all communication is via its master; slaves cannot connect directly to each other.

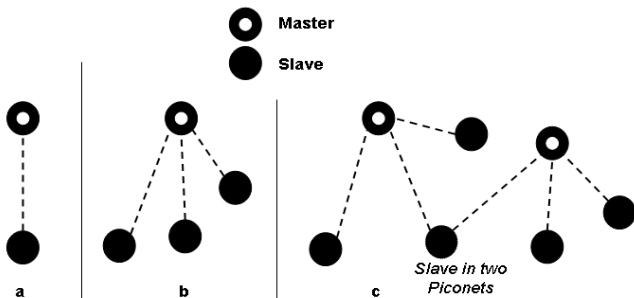


Figure 1: Bluetooth piconet with a) one slave and a master, b) multiple slaves and master and c) a scatternet of two piconets joined by common slave

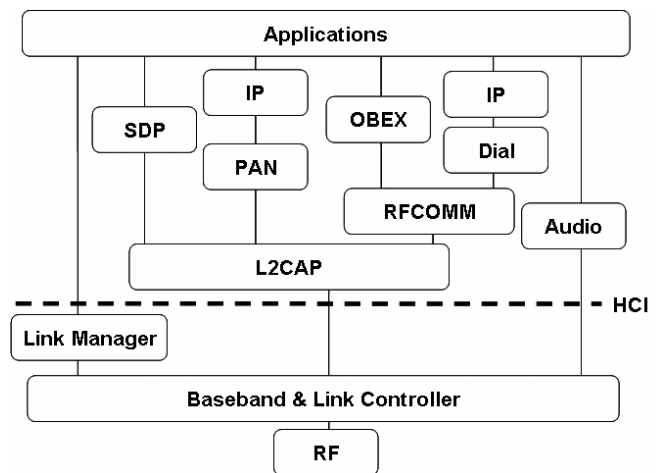


Figure 2: Bluetooth Protocol Stack

V. BLUETOOTH PROTOCOL STACK

The Bluetooth protocol stack is defined in the Bluetooth Specification and provides a set of layers as shown in Fig. 2. Each layer has a well-defined functionality, thus ensuring the interoperability of Bluetooth devices and therefore encouraging adaptation of Bluetooth technology. The specification defines the Bluetooth Core System which details what every Bluetooth device must contain for it to carry the Bluetooth name. The criteria are as follows:

- Four lowest layers and associated protocols.
- Service Discovery Protocol (SDP)
- The overall profile requirement specified in the Generic Access Profile.

The Bluetooth protocol stack can be broken down into two broad areas, the lower layers (usually implemented in hardware/firmware) and the upper layers (usually implemented in software). The lowest three layers are normally grouped into a subsystem called the Bluetooth controller, but more commonly known as a Bluetooth module or dongle.

A. Radio Layer

At the bottom of the Bluetooth protocol stack is the radio module. This layer deals with the conversion of the data into RF signals for the transmission through the air.

B. Baseband and Link Controller Layer

The role of the baseband is to properly format data for transmission to and from the radio module. It also performs timing and frequency hop selection. The link controller's role is to establish and maintain the links that are set by link manger. We investigated spoofing the BD_ADDR in this layer but this proved problematic as discussed later.

C. Link Manger

The link manger acts as a contact between applications that sit on higher layers and the link controller. Its responsibility is establishing and configuring the different links that are needed by applications.

D. Host Controller Interface Layer

The Host Controller Interface (HCI) is not a physical layer but forms the hardware abstraction layer between firmware

on modules and software on the host computer and exposes the services of the module to the higher layers. Communication between the module and the host computer is done through the HCI, usually across a wired-bus such as USB or PCMCIA (PC card). In small devices such as headsets the module and host components are combined for simplicity and size, and therefore the HCI layer is not needed.

E. Logical Link Control and Adoption Protocol Layer

The Logical Link Control and Adoption Protocol (L2CAP) Layer acts as the middle manager between the application and the Bluetooth link controller. Once a connection has been established by the link manager, it handles the actual data communication between Bluetooth devices from the higher layers. By multiplexing the different higher layers it allows many different applications to use a single data link between Bluetooth devices. Each higher protocol is assigned a unique protocol service multiplexer (PSM) number, which is then used when data is segmented and reassembled between the lower layers. This layer offers the most promise in seamlessly transporting Bluetooth communication since it is the multiplexer of all higher data layers, but we used the next layer to test first.

F. RFCOMM Layer

RCOMM is based on the “GSM 07.10: Multiplexing Protocol” standard, with a few minor differences. It provides a serial port emulation which can be used by legacy applications. It also is used by several other Bluetooth profiles for their data transfer such as OBEX Object Push for file transfers. Once the setup messages have been passed, a unique channel is opened to be used for data transfer [2]. Though RFCOMM is not defined as part of the Core specification, it is included in practically every non-embedded Bluetooth system. This was the first layer we attempted and where successful to transport.

G. Service Discovery Protocol Layer

The SDP is a very important part of the Bluetooth protocol stack as it allows Bluetooth devices to locate services offered by other Bluetooth devices. An SDP server maintains a database of service records which record information on available services and how to access them. To allow for easy access to the SDP database, a reserved PSM is used for all SDP communication. An application was written to locate and distribute services between the hotspots that were offered by devices within their range.

H. Audio

Audio information between Bluetooth devices such as headset to mobile phone are carried via Synchronous Connection-Oriented (SCO) channels. These channels bypass most of the Bluetooth protocol stack and via a direct PCM connection to the baseband layer. This direct connection avoids problems with flow-control across the HCI layer. Due to the problems caused by delays transporting data across an IP network this layer was not investigated for transporting.

VI. TOOLS USED

One of the first decisions of this project was to decide which operating system (OS) and Bluetooth protocol stack the

development work should be carried out on. Microsoft Windows with Microsoft’s own Bluetooth stack or Broadcom’s stack (formerly Widcomm), FreeBSD or Linux distributions (each with their respective official stacks) were considered as options.

It was found that Open Source Software (OSS) offers many advantages in the research environment. Open standards are implied as the implementation of protocols or systems are viewable making additional development work easier. Modification of previously written code such as libraries or even kernel code is thus allowed, making the coding process simpler by allowing debugging messages to be placed in any code that is running.

A. Bluetooth Protocol Stack

At the beginning of the project the three most prominent Bluetooth protocol stacks that were OSS were FreeBSD’s stack [4], OpenBT which was later discontinued [5] and Linux’s official stack, BlueZ [6]. FreeBSD’s stack is implemented using FreeBSD’s Netgraph system. Netgraph provides a uniform and modular framework for various networking functions. It has “nodes” which are different protocol implementations which can be arbitrary connected to each other. BlueZ on the other hand was implemented using standardised UNIX socket interface to all protocol layers thus allowing for easy communication to any layer directly. This made new applications easy to implement, with no steep learning curve as the standard socket calls to *bind*, *listen* and *connect* are used as with TCP and UDP network sockets. BlueZ was therefore chosen for development.

BlueZ has been developed under the principle of modularity, with each part of the protocol stack as defined by the Bluetooth specification written as either a kernel module or separate application. These modules closely match the Bluetooth protocol stack as defined by the Bluetooth specification meaning that boundaries in developed applications also match boundaries in the official stack, making development and understanding previous code far easier.

B. Operating System

As BlueZ is written for a Linux kernel a Linux distribution had to be chosen for the development platform for this project. Gentoo [7] which is a source-based distribution was chosen, meaning that instead of using pre-compiled binaries of applications, each application, library and kernel is compiled specifically for your own machine. This allowed for the ease of re-compiling the modified system and kernel files and their incorporation into the system.

VII. BLUETOOTH HOTSPOTS

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. The three layers that have been investigated so far are the Baseband layer, RFCOMM layer and L2CAP Layer. These could be broadly broken down into two different processes: hotspots masquerading as other Bluetooth devices, or hotspots proxying services offered by other devices. These two processes are discussed in detail here and how they been implemented is clarified in the next section.

A. Spoofing

The first route of examination was to investigate whether it was possible to enable the hotspots to advertise themselves as other Bluetooth devices, to spoof devices that were located at other hotspots as shown in Fig. 3. This would allow for easy access to and use of the hotspots, as once a pairing of devices is accomplished, they will communicate seamlessly whether within range of the device directly, or via the hotspots, without user intervention.

B. Using Bluetooth Services Discovery

The investigation then examined Bluetooth’s Service Discovery Protocol (SDP) which is defined in the specification as a core component and must be apart of every Bluetooth device. This is used by individual devices to discover what services are available from other devices, and what functionality these services offer. The protocol is used in two different ways; either a found device is queried and requests are made to find what specific services it offers, or all local devices are searched for a particular service or a service with a particular property. The feasibility of using SDP to provide facilities to discover remote devices and facilitate the setting up of communication channels between them was also investigated.

The scenario that was envisioned is shown in Fig. 4, where a user (*Device A*) within the range of a hotspot (*Repeater 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 & C*) are within range of other distant hotspots (*Repeater Y*) and what services (“Sync”) they can offer to allow for communication with a distant device. If the user then for example chooses to “Sync with *Device C*,” a communication channel would then be set up between *Device A* and *Device C* by the hotspots that will

transparently tunnel 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.

VIII. IMPLEMENTATION AT DIFFERENT LAYERS

A. Baseband Layer

An illustration of transporting Bluetooth data via spoofing would be to change the BD_ADDR address of a Bluetooth device. For this to be able to work it would require that the address on the hotspot be able to change as needed. An investigation was done into the possibility of changing the address but it was found that this was problematic. It is indeed possible for the BD_ADDR of practically every Bluetooth module to be changed (though this would need to be done by proprietary vender methods as a process is not defined in the specification) but the Bluetooth devices would need to be reset for the new address to take effect. This reset would mean that any previous connections with devices would be lost as the timing clock is reset on power-up. Also, considering Bluetooth devices perform 1600 hops-per-second with each hop being a single data packet, many data packets would be lost and have to be resent. To further complicate this approach, users would not be able to locate a particular distant device as the hotspot would not know in advance what device it should masquerade as before communication starts.

B. RFCOMM Layer

To investigate the viability of transporting Bluetooth communication across an IP network, an application was written to tunnel RFCOMM communication. The reason that RFCOMM was chosen is that it a simple protocol and once the RFCOMM channel has been set up only application data is passed along the channel.

The application is called rcpipe as it’s conceptually based on the idea of datapipe [8], which is a well-known, simple TCP/IP socket redirection application that can be found in various versions on the Internet. Rcpipe was based on rctest, a testing utility of the BlueZ package that is used for testing RFCOMM links. The core part of datapipe is a *select* statement, which performs synchronous I/O multiplexing by watching specified file descriptors to detect when their status changes and then acting on the data being received on those file descriptor.

Rcpipe is designed to transport RFCOMM traffic between

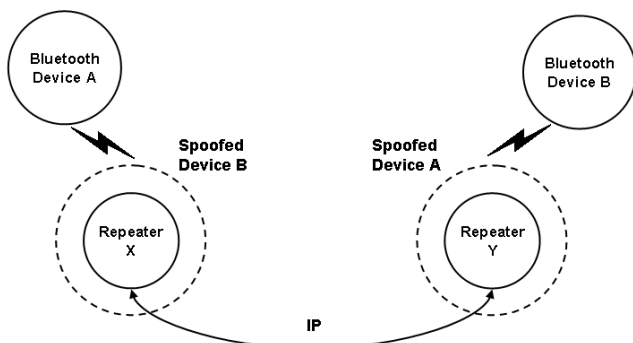


Figure 3: Using Spoofing to create communications

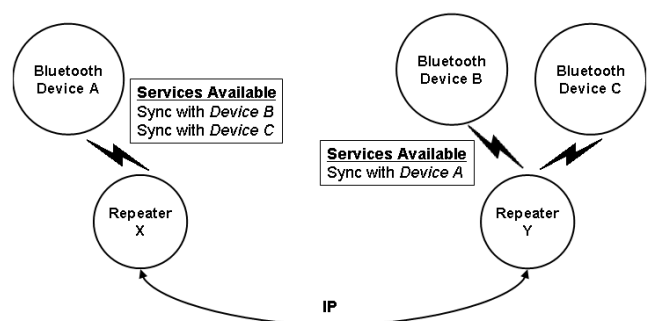


Figure 4: Using Service Discovery to create communications

two hotspots. It does this by opening a RFCOMM channel on each hotspot. When a Bluetooth device connects to one of these channels all data sent to the channel is transported or piped across an existing TCP/IP connection to the other hotspot where an RFCOMM connection is made to a predefined Bluetooth address as given on the command-line when the program is started.

Rcpipe can be started as either a server or client. As a server the program will open a TCP socket and bind it to a given port. An rcpipe client when run will open a TCP socket and connect to the port and IP address given on the command-line. A server will wait for an rcpipe client to connect to its port. Once a TCP connection is established the roles of server/client fall away and both hotspots become peers and will then open RFCOMM sockets and bind them, either to the channels given on the command-line or to the default channels if none was set. Both programs are now in the same state and are listening on both the open TCP connection and non-connected RFCOMM channel. When a Bluetooth device connects to either RFCOMM channel the data is then piped directly to the other hotspot via the open TCP connection. The remote hotspot detects the data on the TCP socket and then opens another RFCOMM channel and connects to a pre-defined Bluetooth device whose address was given on the command-line. The data from the TCP socket is then piped to this RFCOMM channel therefore connecting the two remote Bluetooth devices together via their RFCOMM connections and the hotspot's TCP connection.

The transporting of the RFCOMM communication is made very simple by the fact that RFCOMM is a stream based protocol much the same as TCP. The piping is performed by simply reading in the data from one socket with the read statement to a temp buffer and then writing it out to the other socket using the write statement. The function of transporting is solely performed by the TCP/IP connection and we therefore rely solely on TCP for reliability and data transport. Detection of errors in transmission is also not checked by rcpipe and applications on either side of the RFCOMM channels are relied upon to perform their own error detection.

C. L2CAP Layer

The next step in the project after the success with the rcpipe application was to develop an application that was similar in operation but transported L2CAP communication instead. As L2CAP is a multiplexer of higher protocols, including RFCOMM, being able to tunnel its communication would be valuable. To this end a transporting application that does just that is being developed (called l2pipe).

BlueZ's consistent coding design and APIs for the various Bluetooth protocol layer applications lead to l2pipe being very similar to rcpipe both in user interface and coding structure. This also meant that there was a lot of code re-use from rcpipe and the L2CAP test utility, l2test, speeding up application development.

L2pipe operates very similarly to rcpipe in that two hotspots are first connected together by a TCP connection and then L2CAP sockets are opened on either hotspot. There is a slight difference in terminology in that rather than using

channel numbers L2CAP uses PSM numbers. The major difference between the RFCOMM sockets and L2CAP sockets is that various signals besides user data can be sent across the latter making implementing the l2pipe more complex. L2pipe is still in development; presently the application opens up the L2CAP sockets, accepts connections to them and then tunnels the data to the other hotspot via the IP network but so far is unable to make the second L2CAP connection out from the hotspot.

IX. DISTRIBUTED SERVICE DATABASE

The transporting applications that have been developed are static in so far that the channels/pipes between hotspots are set up when the application is started and they are point-to-point. The next step in the development of a complete solution was to design a distributed SDP database application that would locate and advertise services from all devices within reach of the hotspots, and use this information to manage the creation of the different transporting applications as needed.

The Distributed Service Database (DSDB) was written as a daemon (called dsdbd) which can be communicated with via a TCP socket. The distributed part in the name comes from that multiple DSDBs connect to each other in a peer-to-peer network via TCP and pass between them the different services they locate on devices local to them.

DSDB can be started as either a server or client. The only difference between the two modes is that as a server the application will open a TCP port and listen for connections whereas the client will actively make a connection to another DSDB given on the command-line. Once two DSDBs are connected the rest of the system is peer-to-peer with a command and response paradigm. To simplify design and keep operational boundaries the DSDBs do not search for Bluetooth devices but are notified by a higher program when a device comes within range, and again when it is no longer contactable. When it is notified of a new device it will get a list of all the services that device is advertising via its SDP server. These services are then checked to see which ones have transporting applications available and the ones that do and can be tunnelled are then added to its internal database. The application executes the relevant transporting application (rcpipe or l2pipe) in server mode giving it a port number on the local machine, the BD_ADDR of the "found" device and the channel the service available on that device. The applications will then open a TCP socket and listen on the given port and wait for a connection.

The DSDB will inform all the other remote DSDBs that are connected to it about the new services it has detected giving the service type, what machine to connect to and on what port. The remote DSDBs will then execute the relevant transporting application in client-mode giving it a channel number and the machine and port it should connect to. The application will then connect to its local partner via TCP and once connected it will start listening on the given channel for a connection

Once the transporting application is running the remote DSDB will add the service to its local SDP server giving the channel the transporting application is running on. When another Bluetooth device makes a connection to the remote

transporting application it will then tunnel the data across the TCP connection where the local application will make the relevant connection to the original Bluetooth device.

X. PRELIMINARY RESULTS

Transporting RFCOMM communication was proved possible by the use of the developed application rcpipe. This application does not operate transparently to the user. Furthermore it is static in various respects, specifically that the links between the hotspots, what RFCOMM channels should be opened, and what Bluetooth device needs to be connected to, are selected on the command-prompt at start. This application would be used as part of a distributed SDP database to allow the dynamic setting up of RFCOMM channels when needed.

RFCOMM is not a core component of the Bluetooth specification and is thus limited in the number of applications that make use of it, severely limiting its application in real-world situations. Moving development down a layer in the stack to the L2CAP Layer thus extends the applicability of the project.

L2CAP is an aggregator of all data communication from and to higher layer protocols, including RFCOMM, and therefore by facilitating the transporting of L2CAP communication all Bluetooth user data would be able to be transported across an IP network. Transporting L2CAP data though has the difficulty that unlike RFCOMM's simple data stream, the L2CAP sockets can also carry various signals to manage security and reliability options. These signals and options still need to be implemented in the transporting application for it to be complete.

The Distributed Service Database application allows for Bluetooth devices to locate services advertised by other remote devices via the hotspots. It runs on the hotspot and tracks all services offered by devices within its range, while informing remote hotspots about services it has located. Once a new service is located the DSDBs will then execute the relevant transporting applications to set up the tunnel between the hotspots and the remote DSDB will advertise the found service.

XI. FURTHER WORK

Given that the piconets on either end of the hotspots (consisting of a hotspot and Bluetooth devices that are within range of the hotspot) can be considered to be a scatternet, with the hotspots and IP network being the common slave device (Fig 1(c)), investigations into service discovery in scatternets such as [9] are being further investigated for inclusion into the development of a distributed SDP database application.

REFERENCE

- [1] Bluetooth SIG (2003) "Core specification of the Bluetooth Systems". Version 1.2
- [2] Jennifer Bray and Charles F Sturman, Bluetooth: Connect Without Cables, Prentice-Hall, 2001
- [3] Bluetooth SIG. <http://www.bluetooth.org/>
- [4] FreeBSD Bluetooth. <http://www.freebsd.org/handbook/network-bluetooth.html>
- [5] OpenBT. <http://sourceforge.net/projects/openbt/>
- [6] BlueZ. <http://www.bluez.org/>
- [7] Gentoo/Linux. <http://www.gentoo.org/>
- [8] <http://packetstormsecurity.nl/unix-exploits/tcp-exploits/datapipe.c>
- [9] Nils Agne Nordbotten, Tor Skeie and Niels D. Aakvaag, Methods for service discovery in Bluetooth scatternets, Computer Communications, Volume 27, Issue 11, 1 July 2004, Pages 1087-1096.

David Mackie is currently studying towards a Masters degree in the Department of Computer Science at Rhodes University. His interests are in mobile communication focusing on making Bluetooth more accessible to users.