

*Departement Elektrotechnik
Professur für Technische Informatik
Professor Dr. Albert Kündig*

Alain Pellmont
Andreas Petralia

Bluetooth Networking for Smartcards

Diploma Thesis WS-2001.05
Winter 2000/2001

Supervisors:
Prof. Dr. Albert Kündig
Dr. George Fankhauser
Bernard Stauffer

Public Release

Supervisors:

Prof. Dr. Albert Kündig, kuendig@tik.ee.ethz.ch

Dr. George Fankhauser, gfa@acter.ch

Bernard Stauffer, stauffer@tik.ee.ethz.ch

Students:

Alain Pellmont, alain@pellmont.ch

Andreas Petralia, andreas@petralia.ch

Acknowledgments

Our special appreciation goes to our supervisors Prof. Dr. Albert Kündig, Bernard Stauffer and Dr. George Fankhauser for entrusting us with this project. Prof. Dr. Albert Kündig and Bernard Stauffer work for the *Computer Engineering and Networks Laboratory* [49] at the *Swiss Federal Institute of Technology Zurich* [48]. Dr. George Fankhauser works for *acter ag* [3] where the authors were enabled to work in a stimulating atmosphere and wonderful environment.

Furthermore, we thank *acter ag* [3] and their staff for their support, *AXIS* [11] for the Bluetooth stack and Lesley Brack, Emmanuelle Graf, Kathy Grolimund and Ian Maloney for their proof-reading and comments. Remaining mistakes are ours.

Zurich, 17th March 2001

Alain Pellmont

Andreas Petralia

Aufgabenstellung

Bluetooth Networking for Smartcards

Alain Pellmont und Andreas Petralia
Diplomarbeit

TIK-DA-2001.05
Winter 2000/2001

Betreuer: Bernard Stauffer
Betreuer (extern): George Fankhauser
Verantwortlich: Prof. Dr. Albert Kündig

Einführung

Die Firma acter ag entwickelt eine neuartige Smartcard, die im Gegensatz zur traditionellen Kontaktschnittstelle via Bluetooth drahtlos kommuniziert. Bei dieser neuen Form der Kommunikation ergeben sich völlig neue Probleme: die Übertragung ist weder zuverlässig noch sicher. Zudem ist eine Gruppierung von drahtlosen Geräten notwendig, um die Kommunikation zu initiieren. Obwohl die Applikationen meist im Punkt-zu-Punkt Modus arbeiten, ist die darunterliegende Radioverbindung inhärent ein Broadcast-Medium. Bluetooth ist ein von einem grossen Industriekonsortium gebildeter Standard. Er liegt in der ersten Version vor, die eine einfache Punkt-zu-Punkt Verbindung mit PPP über eine emulierte serielle Verbindung (RFCOMM) vorsieht. Probleme dieses Ansatzes zeigen sich, wenn anspruchsvollere Internet-Anwendungen in einer Bluetooth Umgebung eingesetzt werden sollen:

- Adresskonfiguration:
Alle Endpunkte von Verbindungen müssen mit IP-Adressen versehen werden. In Netzwerken, die eine Verbindung ins Internet (oder Intranet) haben, müssen diese Adressen bei einem Konfigurationsserver angefragt (DHCP) und via PPP gesetzt werden. In ad-hoc Netzwerken

muss die Vergabe von einem temporären Server durchgeführt werden. Bei mehreren Servern muss einer ausgewählt werden.

- **Routing:**
In lokalen Netzwerken ist kein Routing auf dem lokalen Subnetz notwendig. In einem aus einzelnen Verbindungen bestehenden Netzwerk muss die lokale Kommunikation bereits durch das Vermitteln von Paketen durchgeführt werden.
- **Mobilität:**
Bei häufigem Verbindungsauf- und abbau erweist sich die PPP/RFCOMM Lösung als langsam und kompliziert.
- **Multicast:**
Verlangen Anwendungen die Kommunikation mit mehreren Partnern gleichzeitig, muss im PPP/RFCOMM Ansatz ein Paket multipliziert und über verschiedene Verbindungen versendet werden.

Der Wunsch nach einfacheren und effizienteren Netzwerkanbindungen via Bluetooth wird mit der Ausbreitung von Internet-Anwendungen auf eingebetteten und mobilen Systemen ebenfalls immer breiter. Neben dem Bluetooth Konsortium versucht nun auch das IEEE 802.15 Standardisierungsgremium eine solche, dem drahtlosen Ethernet (802.11) ähnliche Lösung zu entwickeln. Da die Technik rund um Bluetooth aber noch im Prototyp-Stadium steckt, ist auch dieser Effort für 'Personal Area Networks' (PAN) noch nicht sehr weit fortgeschritten. In dieser Diplomarbeit sollen nun solche Konzepte unabhängig von der aktuellen Standardisierung als angewandter Forschungsbeitrag erarbeitet werden.

Aufgaben

Die Diplomarbeit setzt sich aus zwei Teilen zusammen, die jeweils in mehreren Phasen von den Kandidaten gemeinsam bearbeitet werden sollen:

- Es soll ein allgemeines Konzept für die Vernetzung von mobilen Geräten erarbeitet werden, wobei grundsätzlich auf der Netzwerkschicht aufgebaut werden soll. Es muss beachtet werden, wie neue Geräte erkannt und konfiguriert werden können und diese Informationen den beteiligten Applikationen weitergeleitet werden können. Um das Konzept zu verifizieren, sollen kleine, typische Demonstrationsanwendungen auf der Basis von Webserver und Webclient entwickelt werden. Dieser Entwicklungsschritt ist zu Beginn der Arbeit in einem drahtgebundenen Netz zu testen und am Ende auf der drahtlosen Plattform (angepasst) aufzubauen.
- Die technische Implementation eines Bluetooth Protokoll Stacks auf dem Smartcard Prototypen soll zuerst gemäss den Vorgaben des

Bluetooth Standards 1.0B erfolgen. Die meisten Teile dieser Arbeit sind verfügbar, müssen aber noch integriert werden:

- Der Linux Bluetooth Stack muss an die ARM7 Umgebung der Smartcard angepasst werden.
- Die für das 'LAN Access Profile' nötigen Voraussetzungen müssen gegeben sein (RF/Baseband Hardware auf Basis des Cambridge Silicon Radio Single-Chip Moduls BC01; HCI H4 Schnittstelle zu Linux; L2CAP und RFCOMM im Bluetooth Stack).
- Konfiguration von PPP um die notwendigen Links zwischen Karten und PCs aufzubauen.

Der Standard-Ansatz dient hauptsächlich zum Testen der Hardware und des Betriebssystems auf der Smartcard. Alternativ dazu soll auch der Ansatz des 'Native Networking' verfolgt werden. Dabei ist zu beachten:

- Gegenüber dem Betriebssystem soll ein neues Netzwerk-Interface implementiert werden, das mit den gängigen Werkzeugen bearbeitet werden kann.
- Im Bluetooth Stack muss die geeignete Stelle gefunden werden, wo Pakete im verbindungslosen Modus versendet und empfangen werden können.
- Schliesslich sollen die Ansätze IP/BT und IP/RFCOMM/BT verglichen und quantitativ analysiert werden.

Vorgehen

Die Arbeiten sollen in folgende Phasen aufgeteilt werden:

1. Konzept für Interaktion der Geräte (Pairing, Identifikation).
2. Visualisierung des alpha-Prototyps (aka fake demo).
3. Linux Bluetooth Stack auf der Smartcard integrieren (PPP/RFCOMM auf dem ARM7).
4. Konzept für Bluetooth Native Networking.
5. Implementation von 'BT Native Networking' (zuerst auf i386 Plattform, dann ARM7).
6. Anpassung des alpha-Prototypen an die Kartenumgebung (ARM7) im Rahmen der beschränkten Hardware Ressourcen.

Organisation der Arbeit

- Mit dem Betreuer sind wöchentliche Sitzungen zu vereinbaren. In diesen Sitzungen sollen die Studenten mündlich über den Fortgang der Arbeit und die Einhaltung des Zeitplanes berichten und anstehende Probleme diskutieren.
- Am Ende der zweiten Woche ist ein Zeitplan für den Ablauf der Arbeit vorzulegen und mit dem Betreuer abzustimmen.
- Nach der Hälfte der Arbeitsdauer soll ein kurzer mündlicher Zwischenbericht abgegeben werden, der über den Stand der Arbeit Auskunft gibt. Dieser Zwischenbericht besteht aus einer viertelstündigen, mündlichen Darlegung der bisherigen Schritte und des weiteren Vorgehens.
- Am Schluss der Arbeit ist eine Präsentation von 15 Minuten im Fachgruppen- oder Institutsrahmen wie auch bei acter ag fällig. Anschliessend (evtl. eingebaut) an die Schlusspräsentation soll die Arbeit Interessierten praktisch vorgeführt werden.
- Bereits vorhandene Software kann übernommen und gegebenenfalls angepasst werden.
- Die Dokumentation ist entweder mittels des Satzsystemes \LaTeX oder mit dem Textverarbeitungsprogramm Star/Open-Office zu erstellen. Es ist ein Schlussbericht über die geleisteten Arbeiten abzuliefern (vier Exemplare). Dieser Bericht ist in Deutsch oder Englisch zu halten und beinhaltet sowohl eine deutsche wie auch eine englische Zusammenfassung, die Aufgabenstellung und den Zeitplan. Der Bericht besteht aus einer Einleitung, einer Analyse über verwandte und verwendete Arbeiten, sowie einer vollständigen Dokumentation der Programme und Tools, die für weitere, darauf aufbauende Arbeiten praktisch brauchbar sind.
- Die Arbeit wird als CD-R archiviert. Die Studenten können die vorhandene Infrastruktur ausnützen, um Ihre Arbeit auf CD zu brennen.

Abstract

Bluetooth has recently obtained an unprecedented success in gaining a wide industry support. Today, many people carry numerous portable devices such as laptops, mobile phones and PDAs, for use in their everyday life. In the future, these devices will probably use Bluetooth to interact directly, thus building a spontaneous, ad-hoc wireless communication network.

One of those devices is the acterCARD developed by the Swiss high-tech company acter ag. The acterCARD is a hardware device in the size of a credit card integrating combined functionality such as autonomous localization (GPS), biometric user identification and high-speed wireless communication.

This thesis presents a general concept for spontaneous wireless communication and an implementation of the Bluetooth protocol stack for the acterCARDs hardware target.

In particular, the communication concept for wireless devices in general and for the acterCARD in detail are pointed out. It will mention the necessity of a service discovery protocol (SDP and SLP), IP capable network profiles (LAP, DUNP, PAN), DHCP Server and security needs. The concept solutions are related with the given acterCARD user interface to provide a simple device handling for the user. To verify the concepts, a communication demonstration will be carried out by means of small webserver and webclient applications. This will confirm the chosen approach of a remote display as a user interface. The approach treats functionality from the user's point of view. The demonstration should be used as an inspiration for further applications. Moreover, some network issues that limit Bluetooth applicability in network environments will be pointed out. We give an overview about how networking is actually solved by the current standard and discuss how it differs from current approaches. Last but not least we outline a set of possible designs and present the design we name *RedNose - A New Approach to Bluetooth Networking*. RedNose will make use of the scatternet functionalities allowing interconnection of multiple piconets. An analysis of the interference of colocated piconets is also provided.

The implementation of the Bluetooth protocol stack is based on a freely available stack distributed by AXIS, a company headquartered in Sweden and developing network appliance solutions. We will present the modifica-

tions that were necessary to run the stack on the acterCARD's hardware defined by a low-power consumption processor (ARM7TDMI), a Bluetooth reference design based on the CSR bluecore chip and an embedded version of Linux. The main problems for the ARM port were alignment and endianness and the resource restrictions on the target (RAM, ROM).

Kurzfassung

Bluetooth hat vor kurzem einen beispiellosen Erfolg durch die breite Unterstützung der Industrie erlangt. In der heutigen Zeit tragen viele Leute zahlreiche den Arbeitstag begleitende, portable Geräte mit sich, wie zum Beispiel Laptops, Handys und PDAs. Zukünftig werden diese Geräte mit hoher Sicherheit über Bluetooth kommunizieren um direkt miteinander zu interagieren, und so spontan ein ad hoc Kommunikationsnetzwerk aufzubauen.

Eines dieser Geräte ist die acterCARD, welche von der schweizerischen high-tech Firma acter ag entwickelt wird. Die acterCARD ist ein kreditkartengrosses Gerät, welches verschiedene Funktionseinheiten wie autonome Positionsbestimmung (GPS), biometrische Benutzererkennung und high-speed Funkverbindung integriert.

In der folgenden Diplomarbeit wird ein allgemeines Konzept für die Vernetzung von mobilen Geräten und die Implementierung eines Bluetooth Protocol Stack für die acterCARD vorgestellt.

Insbesondere wird ein Kommunikationskonzept für kabellose Geräte im allgemeinen und für die acterCARD im Speziellen erwähnt. Es werden die Notwendigkeit eines Service Discovery Protocols (SDP und SLP), IP fähigen Netzwerkprofilen (LAP, DUNP, PAN), eines DHCP Servers und Sicherheitsanforderungen unterstrichen. Das Konzept steht in Beziehung zum User Interface der Karte, um eine einfache Handhabung zu gewähren. Um das Konzept zu verifizieren werden kleine, typische Demonstrationsanwendungen an hand von Webserver- und Webclient-Applikationen entwickelt. Dies wird den gemachten Entscheid für das Remote Display bestätigen, welches die Funktionalität aus Benutzersicht behandelt. Die Demonstration soll als Inspiration für weitere Anwendungen dienen. Ausserdem wird durch sie einige Netzwerkaspekte, welche die Bluetooth Verwendbarkeit in Netzwerkkumgebungen limitieren, unterstrichen. Wir geben einen Überblick, wie der Netzwerk-Support durch den jetzigen Standard unterstützt wird und diskutieren die zur Zeit vorhandenen Bestrebungen für Alternativen. Anschliessend skizzieren wir diese Alternativen und stellen eine mögliche Lösung vor, welche wir *RedNose - A New Approach to Bluetooth Networking* nennen. RedNose macht von den im Bluetooth Standard erwähnten Scatternet-Möglichkeiten Gebrauch, welche Verbindungen über mehrere Pi-

conetze unterstützen. Eine Analyse der Störungen in solchen Piconet-Netzen wird ebenfalls durchgeführt.

Die Implementierung des Bluetooth Protocol Stack basiert auf einem frei erhältlichen Stack von AXIS, einer in Schweden ansässigen Firma, welche sich mit der Entwicklung von Netzwerkanwendungslösungen beschäftigt. Wir zeigen die Änderungen auf, welche notwendig waren, um den Stack auf die acterCARD-Hardware zu portieren. Die Hardware beinhaltet einen stromsparenden Prozessor (ARM7TDMI), ein Bluetooth Referenzdesign basierend auf dem CSR bluecore Chip und ein Linux Betriebssystem. Hauptproblematik bei der ARM-Portierung waren alignment- und endianness-Probleme, sowie die dürftigen Hardware-Ressourcen auf der Zielplattform.

Contents

Acknowledgments	iii
Aufgabenstellung	v
Abstract	ix
Kurzfassung	xi
I Concept	1
1 Communication Concept	5
1.1 Bluetooth Overview	5
1.2 Bluetooth Networking	6
1.2.1 Bluetooth Physical Layer	8
1.2.2 Bluetooth Device Addressing	12
1.2.3 Device Authentication	12
1.2.4 Pairing	12
1.3 Bluetooth Profiles	14
1.3.1 New Profiles	16
1.4 IP Networking Model	17
1.4.1 Ad-hoc Network	17
1.4.2 Routing	18
1.4.3 Mobile IP	20
1.5 Address Configuration	20
1.5.1 DHCP	20
1.6 Service Discovery	21
1.6.1 UPnP	22
1.6.2 Jini	22
1.6.3 Salutation	23
1.6.4 SLP	23
1.6.5 SDP	24
1.6.6 Comparison	25
1.7 RedNose	25

II	ARM Port	27
2	AXIS Bluetooth Stack	31
2.1	Stack Overview	31
2.1.1	Profiles	32
2.2	Development Environment	33
2.3	Running Stack on PC Platform	33
2.3.1	Ericsson ROK Module	33
2.3.2	CSR Kit	38
3	ARM Port	41
3.1	Product and System Environment	41
3.2	Development Environment	41
3.2.1	Source Tree	42
3.3	Modifications	44
3.3.1	Loadable Module	45
3.3.2	Stack Linked in Kernel	46
3.3.3	User Program	46
3.4	Object File Size	49
3.5	Running Stack on ARMulator	50
3.5.1	Starting Bluetooth Stack	50
3.5.2	Starting Bluetooth Application	51
3.5.3	acter Kit	51
III	Native Networking	53
4	RedNose	57
4.1	Bluetooth Overview	57
4.2	Bluetooth IP Architectures	57
4.2.1	Piconet	58
4.2.2	Scatternet	58
4.2.3	Broadcast	60
4.3	Bluetooth IP Architecture Goals	60
4.4	Bluetooth IP Architecture Solutions	62
4.4.1	Current Solution: LAN Access Profile	64
4.4.2	Solution under Development: PAN Profile	65
4.4.3	Possible Solution: IP over SCO	65
4.4.4	Possible Solution: IP over ACL	66
4.4.5	Possible Solution: IP over L2CAP	67
4.4.6	Possible Solution: IP over PPP over L2CAP	68
4.4.7	Handoffs Between Piconets	68
4.5	RedNose IP Architecture	71

IV	Conclusion	75
5	Conclusion	77
5.1	Summary and Contributions	77
5.2	Future Work	77
V	Appendix	79
A	Software	81
A.1	Software Sources	81
A.2	VMware	81
A.3	Kernel Configuration File	82
A.4	Makefile Module	83
A.5	Makefile User Program	84
A.6	Data Between Stack and Ericsson Module	85
A.7	Data Between Stack and CSR Module	87
B	Hardware	89
B.1	CSR Radio Module Test Report	89
C	Glossary	93
	Bibliography	99

List of Tables

1.1	Definition of Input Variables	10
1.2	Comparison of common Service Discovery Protocols	25
3.1	Software Repository Source Tree	44
4.1	Overview of Bluetooth IP Architecture Solutions	63
4.2	Minimum/Maximum Inquiry/Paging Time	72

List of Figures

1.1	Bluetooth Protocol Stack	5
1.2	Piconet	7
1.3	Scatternet	8
1.4	Bluetooth Device States	8
1.5	Bluetooth Packet Structure	9
1.6	Piconet Throughput	12
1.7	Maximum Throughput	13
1.8	Bluetooth 48-bit Device Address	13
1.9	Bluetooth Profile Structure	14
1.10	IP Solution using LAN Access Profile	15
1.11	PAN Profile	17
1.12	L2CAP-based Solution	18
1.13	IP-based Solution using LAN Access Profile	18
1.14	IP-based Solution using PAN Profile	19
1.15	SLP Overview	23
1.16	SDP Service Registry Structure	24
2.1	PC Development Environment	34
2.2	USB, RS232, and PC Card Transport Layers	35
3.1	ARM Development Environment	42
3.2	Source Tree	43
4.1	Timing in Multi-Slave configuration	58
4.2	Single Bluetooth Piconet with multiple Slaves	59
4.3	Scatternet Topologies	59
4.4	Relation between OSI and Bluetooth Layers	61
4.5	IP Protocol Stack using LAN Access Profile	64
4.6	Pan Profile for IP Networking	65
4.7	SCO packets	66
4.8	ACL packets	67
4.9	LAP's Role in a Piconet: Master or Slave	68
4.10	Access Point maintains a Lookup Table	70
4.11	Connection to another Piconet	72

4.12 Bluetooth Device maintains a Lookup Table 73

Part I
Concept

This part of the diploma thesis provides the interaction and communication concept of the acterCARD. The concept includes different layers of the communication hierarchy.

physical layer and link layer: The physical and link layer use Bluetooth wireless modules to transport information over the air.

network layer: This layer performs communication between services. In ad-hoc networking automatic parameter deployment is needed. The parameters on TCP/IP networks are for example the IP address.

service management layer: The service management layer is concerned with exchange of information about services. The description of services, service requests, protocols to register and request services are located in this layer.

Relevant topics for the concept are:

- Bluetooth Communication
- IP Networking
- Service Discovery
- Security

With these fundamentals and the acterCARD's specification [4], the resulting interaction and communication concept is built and presented in Chapter ?? on page ??.

Chapter 1

Communication Concept

1.1 Bluetooth Overview

The Bluetooth Special Interest Group (SIG) has released the Bluetooth Specification [45] allowing the development of interactive services and applications over interoperable radio modules and data communication protocols. The objective of the specification is to allow applications written in a manner that is conformal to the specification to interoperate with each other. The following figure shows the complete Bluetooth stack (for details refer to [45] and [44]):

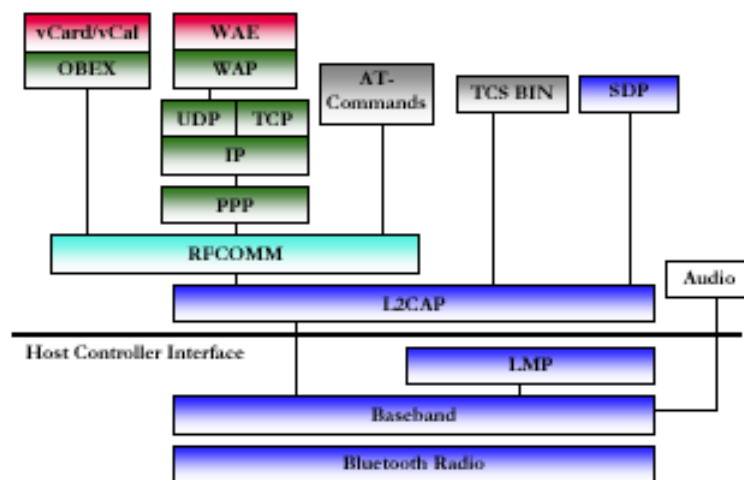


Figure 1.1: Bluetooth Protocol Stack

The complete protocol stack comprises both Bluetooth specific protocols like Link Manager Protocol (LMP), Logical Link Controller and Adaptation Protocol (L2CAP), and non-Bluetooth-specific protocols like Point-to-Point Protocol (PPP) and Telephony Control protocol Specification (TCS). In

designing the protocols and the whole protocol stack, the main principle has been to maximize the re-use of existing protocols for different purposes at higher layers instead of re-inventing the wheel, so to speak. The Bluetooth protocol defines some important protocols.

Host Controller Interface (HCI):

The HCI is the command interface to the baseband controller and link manager and the access to hardware status and control registers. This interface provides a uniform method of accessing the Bluetooth baseband capabilities. It is important to enable the sending of link setup messages in HCI format, for setting-up an ACL link or for transferring data. It should be noted that HCI commands are not manufactured independently. For example the serial speed of the Ericsson module can only be changed by using a special Ericsson HCI command (see [22]).

Logical Link Control and Adaptation Protocol (L2CAP):

The L2CAP adapts upper layer protocols over baseband. It can be regarded as working in parallel with LMP with the difference that L2CAP provides services to the upper layer when the payload data is never sent to the LMP messages (see Figure 1.1 on page 5). It provides connection-oriented and connectionless data services to the upper layer protocols with protocol multiplexing capability, and segmentation and reassembly operation. The maximal packet length from the upper layer is 64 kB.

Cable Replacement Protocol (RFCOMM):

RFCOMM is a serial emulation protocol based on ETSI 07.10 specification [25]. This *cable replacement* protocol emulates RS-232 control and data signals over Bluetooth baseband, providing both transport capabilities for upper level services that use serial line as transport mechanism.

1.2 Bluetooth Networking

Bluetooth is a communication standard for short-distance wireless connections. It replaces the many cables connecting one device to another with one universal short-range radio link. BT¹ defines two types of networks: Ad-hoc and client/server networks. An ad-hoc network is a simple network where communications are established between multiple stations in a defined coverage area without the use of an access point or server. The client/server network uses an access point used to handle traffic from the mobile radio to the wired or wireless backbone of the client/server network. The access

¹Bluetooth (unofficial short form)

point routes data between the stations and other wireless stations or to and from the network server.

Up to eight interconnected devices using the same radio frequency-hopping sequence form a piconet (see Figure 1.2 on page 7). A Bluetooth device can be master or slave. Communication can be initiated by any device. The initiating device will become master of the piconet. If the address of the other device is known, the communication can be initiated by a 'page'. If the address is unknown, communication is initiated by an 'inquiry' followed by a 'page'. The master defines the radio frequency-hopping sequence and synchronizes the piconet. Several independent and non-synchronized piconets can join to form a scatternet (see Figure 1.3 on page 8). The communication between the piconets is done by devices switching between the piconets and so changing their roles. If the device is a slave, he has to inform the master that he is temporarily unavailable and synchronize with the new piconet. If acting as master who wants to change the network, no communication can occur during the absence. He will become slave in the other piconet. A device can have two roles, master and slave, but not at the same time.

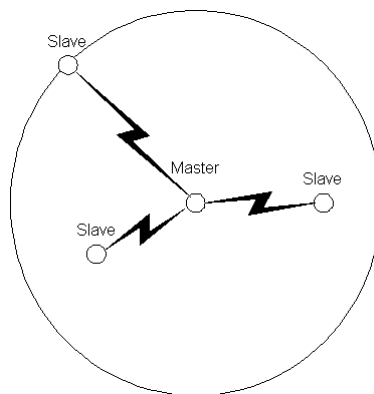


Figure 1.2: Piconet

Each device can be in one of eight states (Figure 1.4 on page 8). Most Bluetooth devices do not have cables and get power from batteries. It is useful to change into power-saving mode when no data has to be transmitted. There are three modes defined in Bluetooth standard: sniff mode, hold mode, and park mode. The park mode is the most power conserving mode. The device has no active member address (docs_ADDR) but rests synchronized with the piconet. The traffic is intercepted from while to while in purpose of re-synchronization. Next less conserving mode is the hold mode: It is analogous to park mode except that the device keeps the docs_ADDR address. The sniff mode is the least conserving mode: The device keeps listening to the traffic regularly.

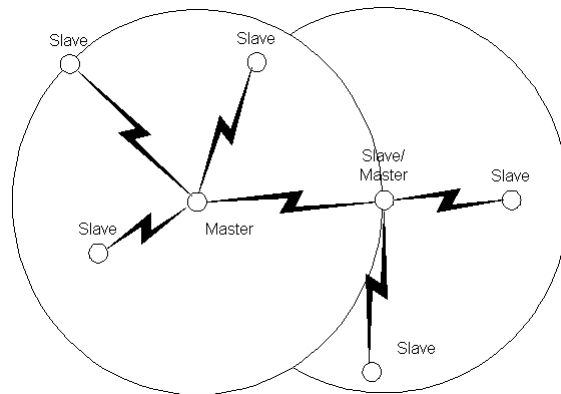


Figure 1.3: Scatternet

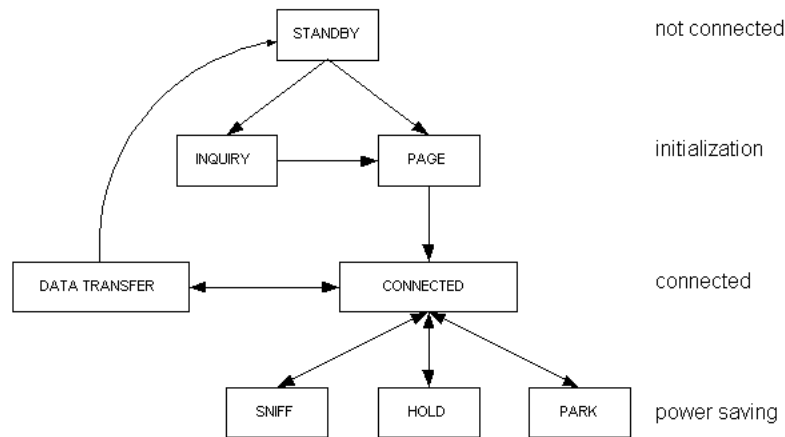


Figure 1.4: Bluetooth Device States

Routing protocols for scatternets have not been defined yet. Moreover, Bluetooth protocol layers present a unique set of features and limitations for which known ad-hoc routing protocols are not optimized. However, a few routing methods have been suggested up to now. For more information about routing refer to Chapter 1.4.2 on page 18.

For general Bluetooth information refer to [13] and [12].

1.2.1 Bluetooth Physical Layer

The short-range radio communication typically extends up to 10 meters in all directions and uses a slotted protocol with a *Frequency Hopping Spread Spectrum* (FHSS) technique in the unlicensed 2.4 GHz *Industrial, Scientific and Medical* (ISM) band. The frequency spectrum is divided into 79 channels of 1 MHz bandwidth and each channel signals data at 1M Symbols per

second (Modulation uses Gaussian Frequency Shift Keying). The nominal hop rate is 1600 hops/sec (corresponding to a time slot of $625 \mu\text{s}$ in width). The hopping structure for this method follows a pseudo-noise (pn) sequence of length $2^{27}-1$. The error correction codes are a $r=\frac{2}{3}$ Hamming block code and a $r=\frac{1}{3}$ repetition code. The Hamming code uses a (15,10) code, 10-bits data and 5-bits redundant data. The repetition code is used only in the header of the packets. Further, header and payload are protected by powerful *Cyclic Redundancy Check* (CRC) codes. The general packet format is shown in Figure 1.5 on page 9.

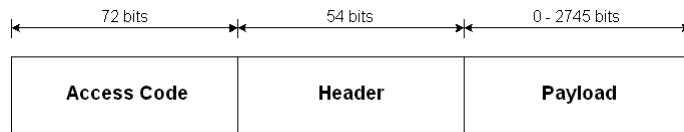


Figure 1.5: Bluetooth Packet Structure

1.2.1.1 Bluetooth Performance in Scatternets

The abilities of scatternet concept allow to build a multiple-hop wireless network. The used frequency hopping spread spectrum makes Bluetooth very robust against RF interference within close proximity of other piconets.

To investigate the interference, some assumptions have to be defined. A piconet is defined by one master and slave that constantly support traffic in both direction. The basic assumption is the existence of a Bluetooth piconet (A) and a number of additional piconets that is greater than or equal to one. This analysis assumes that piconet (A) and the number of piconets are constantly supporting traffic in both directions (master \rightleftharpoons slave). All transmissions of the number of piconets arrive at the piconet (A) with enough power to disrupt the packet. Furthermore, each disruption is assumed to produce more errors than FEC or CRC are capable of correcting. The asynchronous packet format DH1 (Data High rate 1, see Figure 4.8 on page 67), is the high-speed version of the 1-slot packet which supports 172.8 kbits/s in both directions. DH1 format carries up to 28 bytes data per packet. To these 28 bytes, 16 bits are added for error detection CRC, bringing the total payload size to 240 bits. Definitions of input variables are shown in Table 1.1 on page 10. Since there is no error correction block used in DH1, the number of words of the payload is broken into is 240 bits; the whole packet payload. Since there are 240 bits in the payload word and since no errors can be corrected, the probability of no uncorrect errors in the payload is:

$$pCorrectError = binomial(0, 240, pError) \quad (1.1)$$

where $pError$ is the threshold error probability of 10^{-3} and the function $binomial(k, n, p)$ returns the probability density for the binomial distribution

Variable	Value	Description
InBy	28	Information [bytes]
PayCRC	16	CRC bits for payload [bits]
Header	54	Header size including FEC and HEC [bits]
Sync	72	Sync Word [bits]
Tresh	60	Threshold for preamble correlation [bits](not set by standard)
Slots	79	Number of channels in ISM band
Ber	10^{-3}	Maximum Bit Error Rate (BER)

Table 1.1: Definition of Input Variables

and takes the form:

$$binomial(k, n, p) = \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k} \quad (1.2)$$

Based on 1.1, the probability of payload error is:

$$pPayloadError = 1 - pCorrectError = 0.2135 \quad (1.3)$$

The process necessary to calculate the probability of header error is seen in Equations 1.4 through 1.6.

$$HeaderWords = \frac{Header}{3} = 18 \quad (1.4)$$

$$pHeaderWordError = 1 - \sum_{i=0}^1 binomial(i, 3, pError) \quad (1.5)$$

$$\begin{aligned} pHeaderError &= 1 - binomial(0, HeaderWords, pHeaderWordError) \\ &= 3.0 \cdot 10^{-6} \end{aligned} \quad (1.6)$$

The probability of preamble error is seen in Equations 1.7 through 1.9.

$$AcceptableSyncErrors = \frac{SyncWord - Tresh}{2} \quad (1.7)$$

$$\begin{aligned} pSyncError &= 1 - \sum_{i=0}^{AcceptableSyncErrors} binomial(i, SyncWord, pError) \\ &= 1.4 \cdot 10^{-12} \end{aligned} \quad (1.8)$$

$$\begin{aligned}
& pPacketErrorFromBitErrors \\
&= 1 - ((1 - pPayloadError)(1 - pHeaderError)(1 - pSyncError)) \\
&= 6.1 \cdot 10^{-5} \quad (1.9)
\end{aligned}$$

The probability of collision seen in Equation 1.10 assumes that there are `NumberOfPiconets` Bluetooth piconets (each with one master and one slave), and that the interference level received is high enough to disrupt the Bluetooth link between master and slave.

$$\begin{aligned}
& pCollision \\
&= 1 - \frac{Slots}{Slots} \cdot \frac{Slots - 1}{Slots} \cdot \dots \cdot \frac{Slots + 1 - NumberOfPiconets}{Slots} \\
&= 1 - \frac{Slots!}{Slots^{NumberOfPiconets} \cdot (Slots - NumberOfPiconets)!} \quad (1.10)
\end{aligned}$$

The probability of Bluetooth packet error is

$$\begin{aligned}
& pPacketError \\
&= 1 - ((1 - pPacketErrorFromBitErrors)(1 - pCollision)) \quad (1.11)
\end{aligned}$$

As can be expected from the difference in magnitude between `pCollision` and `pPacketErrorFromBitErrors`, `pCollision` dominates the value of `pPacketError`. The probability associated with a certain number of retransmissions per packet can be seen in Equation 1.12.

$$pTransmit(n) = pPacketError^n \cdot (1 - pPacketError) \quad (1.12)$$

The assumption is that the same amount of data is sent over the Bluetooth piconet from slave to master as from master to slave (symmetrical link). Since each packet occupies $625 \mu s$ and since each packet carries at most 28 bytes, the maximum bits per second the channel can support in one direction is

$$BluetoothMaxLoad = \frac{8 \cdot InBy}{625 \cdot 10^{-6}} = 358'400 \text{ bps} \quad (1.13)$$

When taking into account the retransmissions, the expected value of maximum data load in one direction supported by the piconet is

$$E[Throughput] = \sum_{n=0}^{\infty} pTransmit(n) \cdot \frac{BluetoothMaxLoad}{n + 1} \quad (1.14)$$

Since each packet is acknowledged by a packet also carrying a payload, the maximum throughput is twice that seen in Equation 1.13. Figure 1.6 on page 12 shows the throughput within one piconet when other piconets interfere. Figure 1.7 on page 13 shows the maximum throughput within all piconets (the straight line shows the theoretical maximum throughput, the slop turns to zero after 79 piconets). Source: [7]

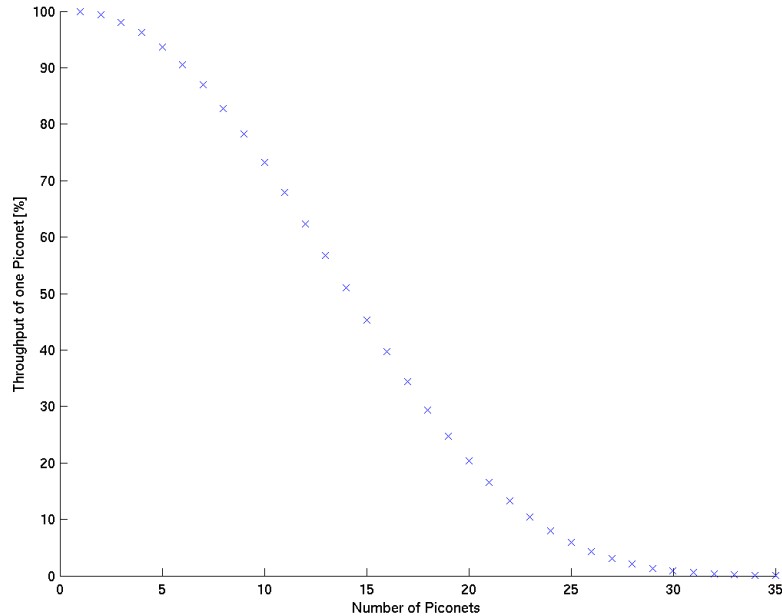


Figure 1.6: Piconet Throughput

1.2.2 Bluetooth Device Addressing

Each Bluetooth device has a 48-bit IEEE MAC address known as the Bluetooth Device Address (BD_ADDR). This is used as a seed in serial bit processing operations for the derivation of the access code. The MAC address is split into *non-significant address part* (NAP), *upper address part* (UAP), and *lower address part* (LAP) as shown in Figure 1.8 on page 13.

1.2.3 Device Authentication

Authentication of Bluetooth devices is based on a challenge-response transaction. In this transaction, a *verifier* challenges a *claimant* by sending a 16-byte random number to the latter. The claimant operates on the random number and returns the result of the operation to the verifier. If the result is the one expected by the verifier, the verifier considers the claimant an authenticated device. Detailed information is found in Chapter ?? on page ??.

1.2.4 Pairing

The Generic Access Profile defines the policies for device communication establishment and categorizes them into discoverability modes, connectivity modes and pairing modes.

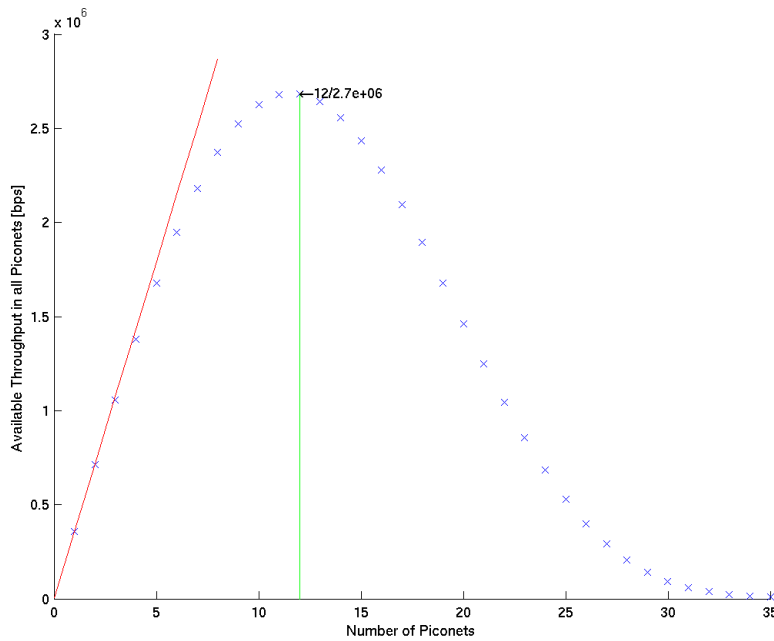


Figure 1.7: Maximum Throughput

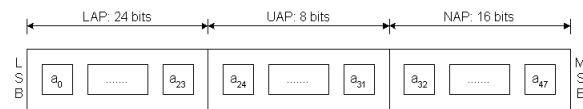


Figure 1.8: Bluetooth 48-bit Device Address

Discoverability Modes: A Bluetooth device is said to be *discoverable* if it can be discovered by other Bluetooth devices. In particular, a discoverable device regularly executes inquiry scans and responds to inquiries sent by inquiring devices.

Connectability Modes: A Bluetooth device is said to be *connectable* if it allows itself to create Bluetooth links with other devices. A connectable device regularly executes page scans and responds to pages sent to it by paging devices.

Pairing Modes: A Bluetooth device is said to be *pairable* if it allows itself to be authenticated by an other Bluetooth device, meaning that it can play the role of a claimant during an authentication transaction. Furthermore, a pairable device must accept an initial authentication request received from a verifier.

1.3 Bluetooth Profiles

Figure 1.9 on page 14 illustrates the Bluetooth profile structure and the dependencies of the profiles. A profile is dependent on another profile if it re-uses parts of that profile. All profiles depend on the Generic Access Profile.

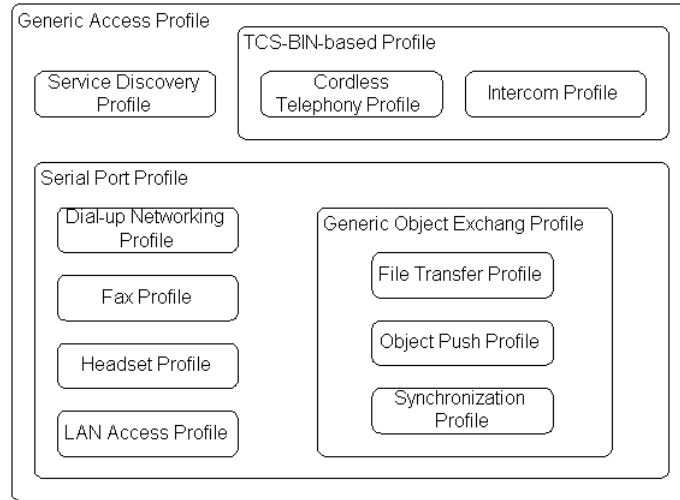


Figure 1.9: Bluetooth Profile Structure

Generic Access Profile: The Generic Access Profile defines the generic procedures related to discovery of Bluetooth devices and link management aspects of connecting to Bluetooth devices. It is the core on which all other Profiles are based.

SDP Profile: The Service Discovery Profile defines the features and procedures for an application to discover services registered in other Bluetooth devices and retrieve any desired available information pertinent to these services.

CTP Profile: The Cordless Telephony Profile defines the features and procedures that are required for interoperability between different units active in the three-in-one phone use case. The three-in-one phone usage model allows a mobile telephone to be used as a cellular phone, as a cordless phone, and as an intercom or phone-to-phone communication. This profile also shows how the use case can be generally applied for wireless telephony in a residential or small office environment.

IP Profile: The Intercom Profile defines the requirements for Bluetooth devices necessary for the support of the intercom functionality within the three-in-one phone use case. It is also referred to as the phone-to-phone usage of Bluetooth.

SPP Profile: The Serial Port Profile defines the requirements for Bluetooth devices necessary for setting up emulated serial cable connections using RFCOMM between two peer devices.

HS Profile: The Headset Profile defines the requirements to be used by devices implementing the usage model called 'Ultimate Headset'.

DUNP Profile: The Dial-up Networking Profile defines the requirements that shall be used by devices (modems, cellular phones) implementing the usage model called 'Internet Bridge'.

FP Profile: The Fax Profile defines the requirements for Bluetooth devices necessary to support the Fax use case. This allows a Bluetooth cellular phone (or modem) to be used by a computer as a wireless fax modem to send/receive a fax message.

LAP Profile: The LAN Access Profile defines how Bluetooth enabled devices can access the services of a LAN using PPP. Also, this profile shows how the same PPP mechanisms are used to form a network consisting of two Bluetooth-enabled devices. Figure 1.10 on page 15 shows the Bluetooth protocols and stacks using the LAN Access Profile. The

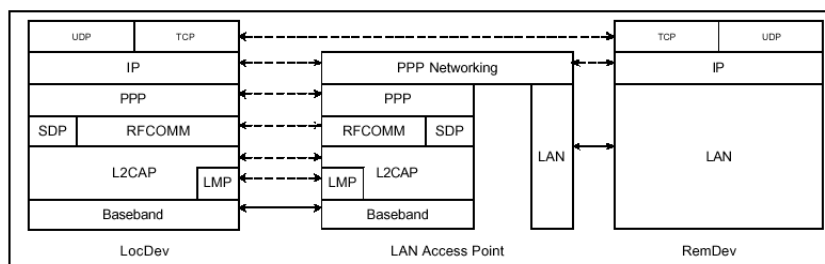


Figure 1.10: IP Solution using LAN Access Profile

profile is divided into LAN Access Points and Data Terminals. A LAN Access Point (LAP) is a Bluetooth device that provides access to a LAN. The LAP provides the services for a PPP server. RFCOMM is used to transport the PPP packets and can also be used for flow control of the PPP data stream. The Data Terminal (DT) uses the services of the LAP. Typical devices are laptops and PDAs. The DT is a PPP client using PPP connection to access a LAN. Communication between DTs is done via the LAP.

GOEP Profile: The Generic Object Exchange Profile lays the basis (defines the protocols and procedures) for Bluetooth devices necessary for the support of the object exchange usage models. The usage model can be the Synchronization, File Transfer or Object Push model.

OPP Profile: The Object Push Profile defines the requirements for applications providing the object push usage model. Typical scenarios covered by this profile involve the pushing/pulling of data objects between Bluetooth devices.

FTP Profile: The File Transfer Profile defines the requirements for applications providing the file transfer usage model. Typical scenarios involve a Bluetooth device browsing, transferring, and manipulating objects on/with another Bluetooth device.

SP Profile: The Synchronization Profile defines the requirements for applications providing the synchronization usage model. Typical scenarios covered by this profile involve manual or automatic synchronization of PDA data when two Bluetooth devices come within range.

1.3.1 New Profiles

Different Bluetooth SIG-Working Groups² are currently working on:

- Personal Area Network Profile, described in Section 1.3.1.1 on page 16
- Extended Service Discovery Profile, described in Section 1.3.1.2 on page 16
- Car Profile
- Printing Profile
- Local Positioning Profile
- ...

1.3.1.1 Personal Area Network Profile

The PAN Profile working group is defining specifications for dynamic ad-hoc IP-based personal area networks (PAN). The usage model is to guarantee access to a PAN in a public environment and co-exist with other Bluetooth devices, enabling multi-user collaboration such as overhead-projectors, displays, chatting, and file/application sharing. PAN stack is shown in Figure 1.11 on page 17.

1.3.1.2 Extended Service Discovery Profile

The Extended Service Discovery Profile [34] defines how devices with Bluetooth wireless communication can use the Bluetooth Service Discovery Profile initially to discover other devices that support Universal Plug and

²<http://www.bluetooth.org/bluetooth/member/workinggroups.htm>

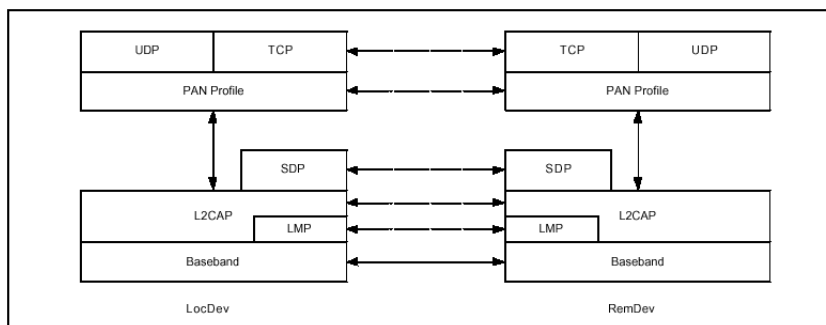


Figure 1.11: PAN Profile

Play (UPnP) services and retrieve information about these services (UPnP is discussed in Section 1.6.1 on page 22).

In a networked computing environment, devices should be able to discover services beyond their current Bluetooth piconet, Bluetooth SDP is limited to a single piconet. The use of the Extended Service Discovery Profile removes this limitation. In addition, extended service discovery could enable devices with Bluetooth wireless technology to control remote resources on other devices (that may or may not employ Bluetooth wireless communications) within and outside their piconets.

The Extended Service Discovery Profile defines two approaches of implementing UPnP within Bluetooth: a L2CAP-based and two IP-based solutions. The L2CAP-based solution is dependent on the Generic Access Profile and is shown in Figure 1.12 on page 18. The IP-based solutions are dependent on at least one of the IP profiles. The solution using LAN Access is shown in Figure 1.13 on page 18 and the solution using PAN is shown in Figure 1.14 on page 19.

1.4 IP Networking Model

1.4.1 Ad-hoc Network

One of the most obvious features of ad-hoc networks is that no structure is defined. These networks can be built spontaneously and are independent of existing infrastructure. All members can join and leave the network which permanently changes structure. Each node is able to communicate with each other. The communication has not to be direct if the nodes are able to route communication. The links may be symmetric or asymmetric and uni- or bi-directional. Ad-hoc networks may be connected to a local area network.

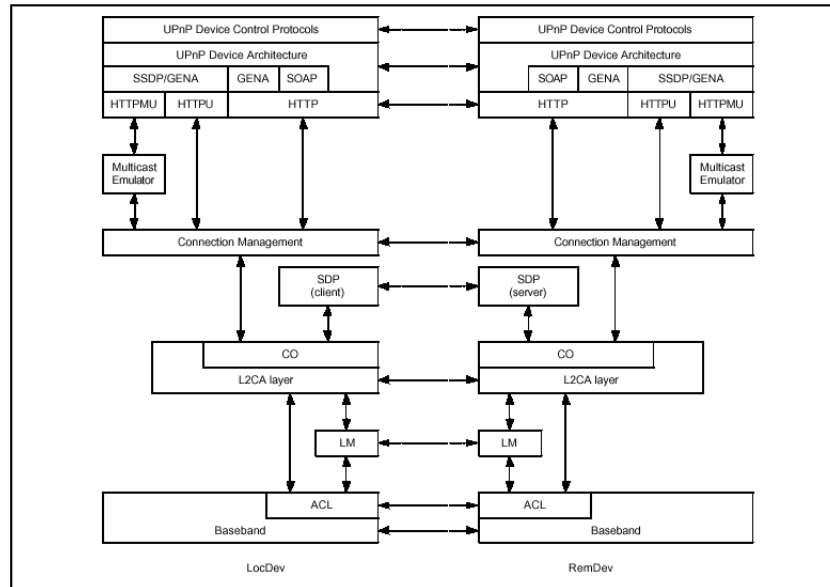


Figure 1.12: L2CAP-based Solution

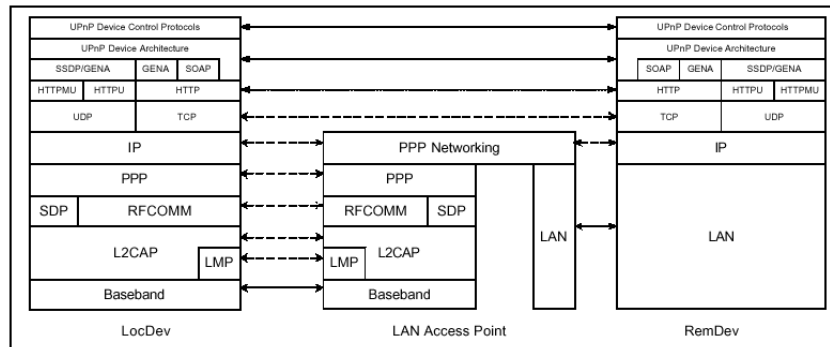


Figure 1.13: IP-based Solution using LAN Access Profile

1.4.2 Routing

The biggest problem in realizing an ad-hoc network is the routing. Each node has to forward the traffic because no default router is available. Or each node has to be connected to a node with the possibility to forward traffic. The routing problem is extended by the strong dynamic change of the network topology and the related changes of links, signal quality, and also the quantity change of members and their location. The common routing algorithms are designed for fix installed networks. Therefore, new algorithms have to be developed first. In addition, general problems of mobile communication like weak security, hard energy and bandwidth restrictions have to be considered. MANET has proposed some routing algorithms which are

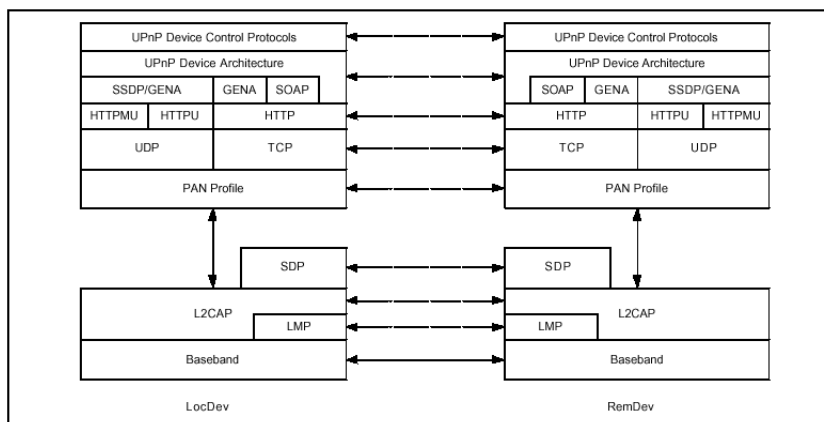


Figure 1.14: IP-based Solution using PAN Profile

discussed next.

1.4.2.1 MANET

MANET IETF working group is defining specifications for mobile ad-hoc networking [35]. The main target is to develop an efficient routing algorithm for mobile, wireless networks. Here are some of the algorithms, the group is working on:

- Ad-hoc On Demand Distance Vector (AODV) Routing
- Temporally-Ordered Routing Algorithm (TORA)
- Dynamic Source Routing Protocol for Mobile ad-hoc Networks
- Optimized Link State Routing Protocol
- Differential Destination Multicast (DDM)
- Multicast ad-hoc On-Demand Distance Vector (MAODV) Routing
- Topology Broadcast based on Reverse-Path Forwarding(TBRPF)
- Landmark Routing Protocol (LANMAR) for Large Scale ad-hoc Networks
- A Simple Protocol for Multicast and Broadcast in Mobile ad-hoc Networks
- Fisheye State Routing Protocol (FSR) for ad-hoc Networks

Only a few of them have yet been implemented.

1.4.3 Mobile IP

Mobile IP is a mechanism for maintaining transparent network connectivity to mobile hosts. The Mobile IP protocol enables a mobile host to be addressed by the IP address it uses in its home network (home IP address), regardless of the network to which it is currently physically attached. Therefore, ongoing network connections to a mobile host can be maintained even when the mobile host is moving from one subnet to another. Mobile IP protocol is specified in RFC 2002 [36] and further improvements are introduced in RFC 2344 [36].

1.5 Address Configuration

1.5.1 Dynamic Host Configuration Protocol

All terminals normally have a unique IP address for identification. Data packets sent to these terminals are passed on the appropriated subnetwork which passes the packets to the appropriated host. In case of a mobile device it is theoretically possible to access the network at any Access Point (AP) regardless of its IP address. If the mobile device has a fix IP address it has to belong to a network informed of its location to forward the data packets. Dynamic Host Configuration Protocol (DHCP) [21] is used to fullfil this task.

A computer joining a network receives the following information from a DHCP server:

- DNS Server Address
- Routing Table
- Subnetwork Mask
- Domain Name
- IP Address
- Options (e.g. SLP entry)

Procedure runs as follows: A client (a mobile device) broadcasts a DHCPDISCOVER to the available DHCP servers on the subnetwork. The reachable servers reply a DHCPOFFER. The client compares all received configurations and selects one. From the client the selected server receives a DHCPREQUEST with the parameter 'options'. The other servers receive the request with the parameter 'reject', and can re-use the offered resources for other requests. The selected server replies with DHCPACK or DHCPNACK. The client is initialized in case of an acknowledgment with

the assigned IP address. On leaving the network, a client must free the configuration for other hosts by sending a DHCPRELEASE to the server. The IP address is temporarily assigned during the 'lease time'. A client must repeat the request after expiration of the 'lease time' to renew the configuration. This is done to free unused configurations for other clients. Due to the DHCP protocol, a DHCP server knows on which access point a mobile device is connected to and where to send the packets for that mobile device.

1.5.1.1 AutoIP

AutoIP [8] is an enhancement to DHCP allowing devices to claim IP addresses in absence of a DHCP server or similar IP configuration authority. IP addresses are claimed from a reserved range that is not allowed to be transmitted on the open Internet: they are only good for the local network. A device claims an address by randomly generating an address in the reserved range and then making an ARP request to see if anyone else has already claimed that address.

AutoIP systems will continually check for the presence of a DHCP server. In case one should come online, all the AutoIP devices will attempt to switch their IP addresses to one provided by the DHCP server.

1.6 Service Discovery

There are some well-known service discovery protocols currently under development, namely Universal Plug and Play (UPnP), Jini, Salutation, Service Location Protocol (SLP) and Bluetooth Service Discovery Protocol (SDP). Service discovery protocols provide mechanisms for dynamically discovering available services in a network and contain the necessary information to

- search and browse for services,
- choose the right service (with desired characteristics), and
- utilize the service.

Service discovery plays an essential role in ad-hoc communications, where no fixed infrastructure is present but nodes themselves form the network. Due to the very dynamic nature of such networks (devices just link together spontaneously and then move away again), static service configuration makes no sense at all, but dynamic and automatic service discovery functionality is required.

This section summarizes and compares the most important service discovery protocols. The overview is summed up from [2].

1.6.1 Universal Plug and Play

Universal Plug and Play uses the Simple Service Discovery Protocol (SSDP) for discovery of devices on IP networks. SSDP is based on profiles. A single identifier specifies a profile that defines a contract between client and service. By identifying itself with the profile, the service advertises compliance with the associated contract. Using a single identifier makes it possible to implement an extremely simple discovery system. Clients send out a User Datagram Protocol (UDP) multicast packet containing the identifier of the desired service on some standard channel. Services listening on the standard channel read the request, see whether they can provide the service, and respond if so. Directories provide a mechanism to allow discovery to scale. The directory is also responsible for communicating with other directories in order to determine whether the service is available within the local network, the WAN, and potentially the Internet. SSDP uses UDP- and TCP-based HTTP to provide service discovery. It uses a URI to represent the service and the OPTIONS method to provide for discovery. The discovery process only returns the basic information needed to connect to a device. Once a service has discovered its peers, the service often needs to find out more information to best work with them. The description process returns a schema providing descriptive data about the service. A schema is a structured data definition defining a set of structured values that provide descriptive information about a service. Universal Plug and Play uses Extensible Markup Language (XML) for schema, because XML's self-describing structured data format provides the level of expressiveness and extensibility needed by a universal schema and data format.

1.6.2 Jini

Jini (Java Intelligent Network Infrastructure) technology is an architecture for the construction of systems from objects and networks. The Jini architecture lets programs use services in a network without knowing anything about the wire protocol that the service uses. One implementation of a service might be XML-based, and another RMI-based, and a third CORBA-based. The client is, in effect, taught by each service how to talk to it. A service is defined by its programming API, declared as a Java programming language interface.

When a service is plugged into a network of Jini technology-enabled services and/or devices, it advertises itself by publishing a Java programming language object that implements the service API. The object's implementation can work in any way the service chooses. The client finds services by looking for an object that supports the API. When it gets the service's published object, it will download any code it needs in order to talk to the service, thereby learning how to talk to the particular service implementa-

tion via the API.

In other words, the Jini architecture uses objects that move around the network to make each service, as well as the entire network of services, adaptable to new strategies over time.

1.6.3 Salutation

The Salutation architecture is developed by an open industry consortium. It consists of Salutation Managers (SLMs) that have the functionality of service brokers. Services register their capabilities with an SLM, and clients query the SLM when they need a service. After discovering a desired service, clients are able to request the utilization of the service through the SLM.

1.6.4 Service Location Protocol

The Service Location Protocol (SLP) is designed to simplify discovery and use of network resources and services. It is suited for client-server applications and connection establishing between network peers that offer or consume services (e.g. intranet). It defines three types of agents:

- User Agents
- Service Agents
- Directory Agents

The user agent requests services. The service agent advertises services, and the directory agent organizes the services into directories (see Figure 1.15 on page 23). SLP is designed for interactive and non-interactive

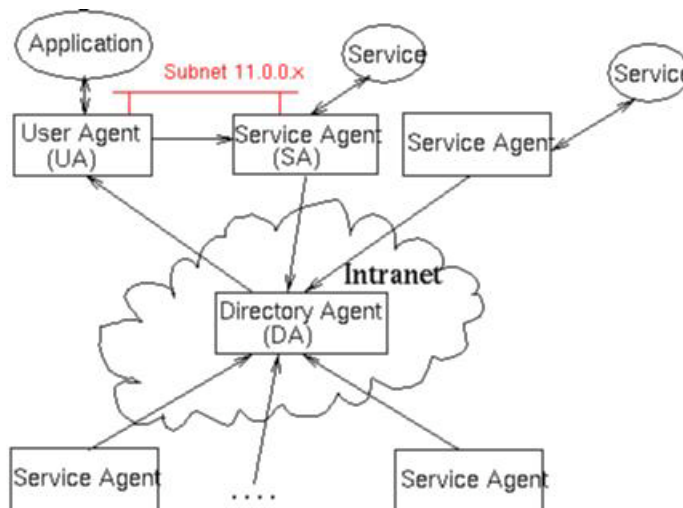


Figure 1.15: SLP Overview

use. SLP also offers support for allowing network resources to be collected into administrative domains called 'scopes'. Resources under a common administrative control are good candidates for inclusion in the same scope. User agents are typically configured (dynamically by DHCP, or as derived from initial setup) with the name of their administrative scope. Afterwards, each user agent includes its configured scope in service requests, which enables access to services configured to operate within that scope. Scopes may also indicate geographic proximity or network topology.

A concept to use LDAP servers as back-end for SLP is under development. Furthermore, SLP will be adapted to use with IPv6 and DHCPv6. Source: [43].

1.6.5 Bluetooth Service Discovery Protocol

The Bluetooth Service Discovery Protocol (SDP) was developed for discovering services in Bluetooth environments. The main objectives were simplicity, compactness, versatility, service location by class and by attribute, and service browsing. SDP is based on a client/server relationship, where servers provide information of the services. Servers maintain a list called the service registry containing service records. Figure 1.16 on page 24 illustrates the structure of the service registry. A service record consists of a service

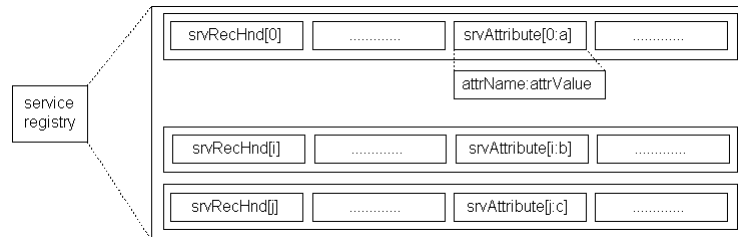


Figure 1.16: SDP Service Registry Structure

record handler (srvRecHnd) and a collection of service attributes (srvAttribute). These attributes contain information about the class of service, the profiles to use, user information, and how to access the service. For further information please refer to [45](p. 323) .

A single Bluetooth device may act as a SDP server and as a SDP client. If multiple applications on a device provide services, a SDP server may handle requests for information about the services provided. Similarly multiple client applications may utilize a SDP client to query the servers.

The discovery service is limited to active devices within a given Bluetooth piconet. No services outside a piconet can be seen. An improvement would be the use of ESDP described in Chapter 1.3.1.2 on page 16.

1.6.6 Comparison

	UPnP	Jini	Salutation	SLP	SDP
Flexibility of Service Description and Request	-	++	(?)	++	+
Centralized/ Decentralized	√/ √	√/ -	√/ √	√/√	-/ √
Group concept	?	√	?	√	-
Code mobility	-	√	-	-	-
Scalability	+	+	(?)	+	-
Security	+	-	+	+	?
Open Source	-	√	√	√	(√)
Ready to Use	-	√	√	√	(√)

Table 1.2: Comparison of common Service Discovery Protocols

The mentioned protocols are shown in Table 1.2 on page 25. Compared are the flexibility of discovering services and of discovering descriptions, use of centralized or decentralized or mixed infrastructure, possibilities of group administration and scalability, safety-conscious development of protocols, license agreements used and immediate ready for use. Table 1.2 on page 25 shows advantages and drawbacks of the protocols. Each of these protocols approaches the vision of service discovery from different perspective. We expect that there will be various kinds of service discovery protocols also in the future. This makes it important to have bridges between the different protocols to enable service discovery with various devices.

SLP will play an important role in the future, that is why SLP should be one of the supported protocols by the acterCARD.

1.7 RedNose

Bluetooth was originally developed as a cable replacement for short-range links. Expanding Bluetooth to a short-range wireless LAN leads to RedNose, which is an approach to Bluetooth IP networking with better IP support than with the current LAP or Dial-up profile. It is discussed in Chapter 4 on page 57.

Part II
ARM Port

As introduced in Chapter 1.1 on page 5, a Bluetooth stack implementation according the Bluetooth Standard [45] has to be developed for the Linux based target environment.

AXIS [11], a company headquartered in Sweden and developing network appliance solutions distributes such a Bluetooth implementation under the terms of GNU General Public License [26]. Their stack was the basis for the ARM port, which will be introduced in Chapter 3 on page 41. An overview of the stack and how to start up in a PC based Linux environment using the Ericsson Bluetooth module or the CSR Bluetooth module will be given in Chapter 2 on page 31.

Chapter 2

AXIS Bluetooth Stack

2.1 Stack Overview

The AXIS Bluetooth stack is distributed under the terms of GNU General Public License and can be downloaded here [10]. The source code is work in progress and the newest version of the software will be made available there. This work is based on the version 20001115 that was released on the 15th of November 2000.

To run the stack without any hardware, HCLEMULATION must be switched on in the file *btconfig.h* and the stack must be recompiled. The HCI emulation as of today simply converts some HCI commands to the corresponding events and simply forwards all ACL/SCO data. If you get messages from the stack that something fails when initiating the stack, it is because those HCI commands are not supported in the HCI emulator.

The stack does not support using PCMCIA drivers for any PCMCIA based Bluetooth card (due to the lack of documentation) and only supports the HCI interface towards that hardware in terms of setting baudrate and other hardware specific functions. *Digianswer Bluetooth Demo PC Card (MK I)* [18] is supported using the Digianswer Bluetooth HCI Router application which routes HCI Packets between the RS232 connection and the Bluetooth PC Card.

There are thoughts about how to generalize the hardware interface so you can use any type of hardware (see Figure 2.2 on page 35), but nothing has been done in the version 1115 so far.

There is no client functionality in SDP due to changes in the design. However, Serial/LAN/DialUp profile is supported as server. Please refer to the next section for any information on profiles.

The stack supports the use of several tty's (currently seven, *ttyBT0* - *ttyBT6*). The control of the stack is dedicated to a tty called *ttyBTC*. This device will be used for all *ioctl*'s dedicated to the Bluetooth driver.

In general the AXIS stack implements all the necessary Bluetooth layer

for their LAN Access Profile. The specification limits the number of Slaves in a piconet to seven, with each Slave only communicating with the shared Master, which is the reason for the seven tty *ttyBT0 - ttyBT6*. However, the specification does not say how the connection should be managed, although the security white paper [14] gives some hints. Putting all together gives a lack of the current AXIS stack, namely a missing device and security manager that can handle establishment and configuration of the links.

The AXIS stack can run in user mode or in kernel mode. *btduser* and *sdp_user* are built with the compiler flag `-DBTD_USERSTACK` which is missing when building *btd* and *sdp_server*. The usermode stack should only be used for development and testing of basic stack functionalities and does not provide the full stack functionalities. The userstack just opens the device to the Bluetooth module (e.g. `/dev/ttyS0`). It has disabled the Bluetooth driver and all its ioctl's which are bypassed in *btd.c* to not involve kernel mode. Due to a bug in *btd*, no connection can be initiated (the command `con <xx:xx:xx:xx:xx:xx> <line> <server channel> <profile>` is disabled). A workaround is to initiate connection from *btd_user* using `rf_conn <xx:xx:xx:xx:xx:xx> <server channel>`.

The stack shows some problems, especially if trying to use the PPP-connection for other things than pining each other. Fast/big transmissions will crash the stack.

For a generic overview based on the AXIS stack of the SDP, RFCOMM, L2CAP and HCI layer and the driver concept please refer to [19].

2.1.1 Profiles

The AXIS stack implements primarily RFCOMM (Serial Port Profile), so any of the profiles which go through RFCOMM can use it. Currently their application (*btd*) has a way to use PPP over RFCOMM (LAN Access Profile). Refer to Figure 1.9 on page 14.

Basically, the stack gives you an RFCOMM tty (or a set of them, actually) which can be used as a lower driver if you want to use another profile, like for example

- Dial-Up Networking
- Fax
- Headset
- Generic OBEX.

Profiles which don't use RFCOMM don't currently have an interface into the stack. So you don't have a lot of support (yet) if you want to use for example one of the TCS-BIN-based profiles over an SCO link (requires an interface to L2CAP/HCI and application support outside the stack).

One exception is made to support the Generic Access Profile. The control tty *ttyBTC* is used to start ioctl's which will be handed over to the L2CAP/HCI layer.

The Service Discovery Profile is still under construction. According to [45](p. 68) SDP Server database application is implemented. The database uses XML. What is still missing is the SDP Client, the service discovery user application (SrvDscApp) and the interface to applications. The Bluetooth Standard left this up to implementers.

The executable *sdp_user* and *sdp_server* do the same thing (which is to act as SDP server) but for different contexts. *sdp_user* is used when running the stack in usermode and *sdp_server* when running in standard kernel mode. The reason is that the usermode stack communicates differently (uses local AF unix sockets) with the sdp server compared to running it in the kernel (uses /proc file). The format of the data sent and received is described in *sdp_proc_protocol.h* in */include/linux/bluetooth/*. See */apps/sdp_server/README* for details.

2.2 Development Environment

First tests with the AXIS Bluetooth stack were made on two virtual machines (VMware [52]) running inside a PC compatible host. The host (*atlas*) was running under *SUSE Linux 7.0 Professional* [47] with an updated Linux Kernel [32] (Kernel 2.4.0-test12). The two virtual machines (*dev1*, *dev2*) were running under *SUSE Linux 7.0 Professional* with the pre-installed Kernel 2.2.16. The Ericsson Bluetooth devices (Bluetooth Application Tool Kit [23]) or the CSR Bluetooth devices (CASIRA[tm] Development Kit [16]) were connected to the virtual machines using UART (Figure 2.1 on page 34). *dev1:/dev/ttyS0* is mapped to *atlas:/dev/ttyS0* and *dev1:/dev/ttyS0* is mapped to *atlas:/dev/ttyS1*. Please refer to Appendix A.2 on page 81 for the VMware configuration file. For the Ericsson module, the power supply was USB [50], and for the CSR module the shipped power supply was used. The Ericsson Bluetooth chip was a ROK 101 007 with firmware P9A. The CSR Bluetooth chip was a BCO1B-URT flashed with the firmware beta9.

2.3 Running Stack on PC Platform

2.3.1 Ericsson ROK Module

The AXIS stack supports different Bluetooth hardware. Whereas the Ericsson module works without problems, the developer community encountered some problems with the CSR module.

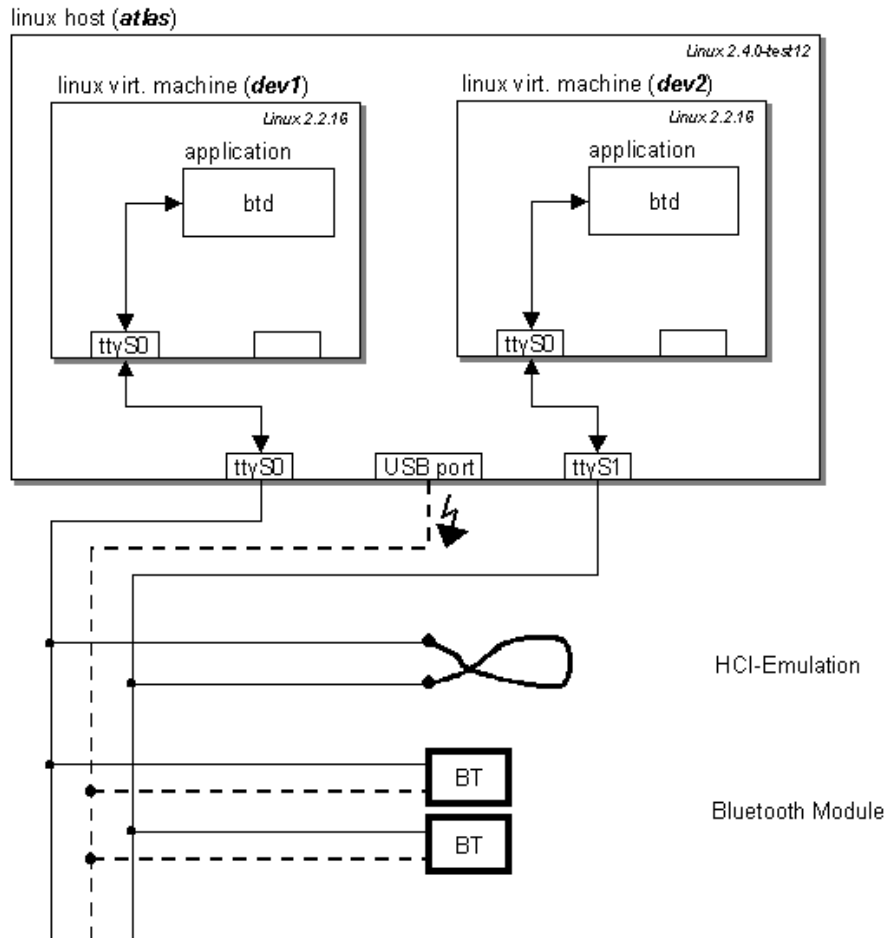


Figure 2.1: PC Development Environment

To enable the serial interface on the Ericsson module, the jumper between pin 8 and 10 has to be removed. If you connect the module to a USB hub, you can use it as a power supply (make sure that the host has loaded the USB driver *usb-uhci.o*).

The AXIS stack can talk through the serial driver or the USB driver to the Ericsson ROK Module. See Figure 2.2 on page 35. The most recent version of the Bluetooth USB driver (*bluetooth.o*) is included in Kernel 2.2.18pre15 or newer. It does not have anything to do with the AXIS stack, but you can get the AXIS stack to work with it (you can also get other vendors' stack to work with it, too).

To make the Bluetooth stack (version 1115) work with the Kernel 2.2.18pre15 follow these steps:

1. Download and install Kernel 2.2.18pre15. Be sure to enable ppp, usb and the Bluetooth usb driver. Be sure to insert the module

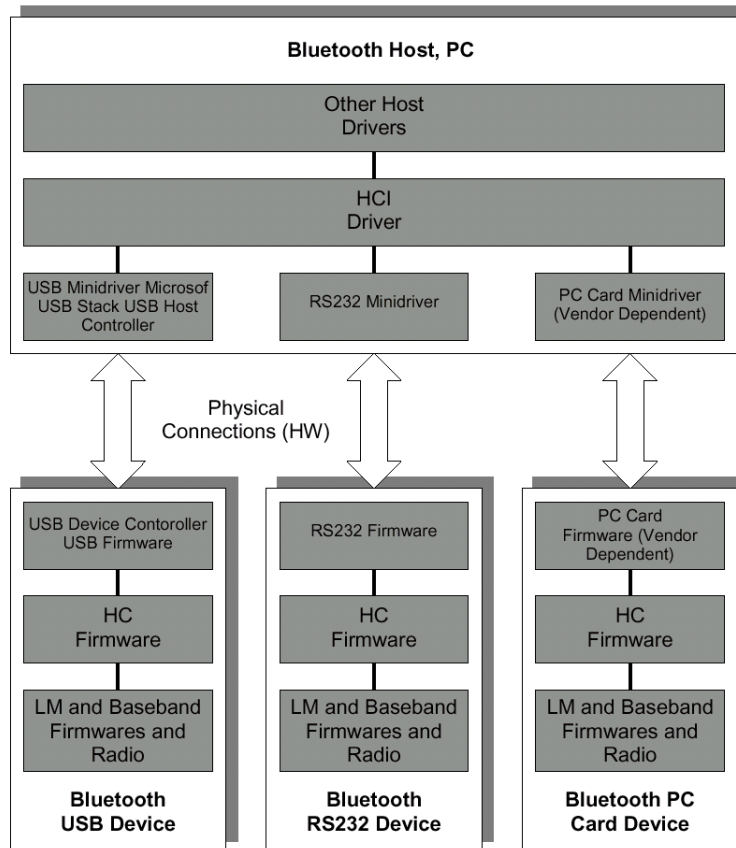


Figure 2.2: USB, RS232, and PC Card Transport Layers

bluetooth.o and *usb-uhci.o* if you compiled it as a module and make the character device */dev/ttyUB0*.

- #> insmod bluetooth.o
- #> insmod usb-uhci.o
- #> mknod -m 0666 /dev/ttUB0 c 240¹ 0
- #> chgrp tty /dev/ttyUB0

2. Download the AXIS stack (version 1115) and install it. Be sure to insert the module *bt.o* and to make the character devices */dev/ttyBT** and */dev/ttyBTC*.

- #> insmod bt.o

¹use major number 216 with Kernel 2.4.x

- #> mknod² -m 0666 /dev/ttyBTy c 124 y
 - #> mknod -m 0666 /dev/ttyBTC c 124 7
 - #> chgrp tty /dev/ttyBT*
3. Install the jumper between pin 8 and 10 on EBMK JP2
 4. Now start the **btd** application and simply point the stack at the **/dev/ttyUB0** node instead of the **/dev/ttyS0** node.
 - #> btd -u /dev/ttyUB0 -e 0 -m -i usb
 5. Type **readbd** at the menu and you should get the hardware bt address. It looks something like this: **00:d0:b7:03:28:c4**

If you prefer the serial driver instead of the usb driver, follow these steps:

1. Download the AXIS stack (version 1115) and install it. Be sure to insert the module **bt.o** and to make the character devices **/dev/ttyBT*** and **/dev/ttyBTC**.
 - #> insmod bt.o
 - #> see README.txt for the character devices
2. Remove the jumper between pin 8 and 10 on EBMK JP2
3. Now start the **btd** application and simply point the stack at the **/dev/ttyS0** node (or wherever you have put your serial connection at).
 - #> btd -u /dev/ttyS0 -s 57600 -e 0 -m
4. Type **readbd** at the menu and you should get the hardware bt address. It looks something like this: **00:d0:b7:03:28:c4**

The AXIS stack will only compile under Kernel Version 2.0.x or 2.2.x. Changes in the wait_queue data structure and in setting up /proc file have to be made to port the AXIS stack to 2.4.x. A patch for the version 20010108 is available [10].

To test your virtual environment for correct connection to the physical UART devices, try this simple test:

- connect the UARTs with a null-modem cable
- atlas: #> sane³

²y=0..6

³see Chapter 3.3.3.4 on page 49

- dev1: #> cat > /dev/ttyS0
- dev2: #> cat < /dev/ttyS0

If you write something on the virtual machine dev1, it should appear on the virtual machine dev2. If not, you have a problem :)

Due to an error in the SUSE Configuration you have to manually make the following link

- #> cd /usr/lib
- #> ln -s termcap/libtermcap.a libtermcap.a

before you can compile the stack.

Once the stack is compiled without error you can start on both virtual machines the stack doing the following (only the baudrate 57'600 will work with the Ericsson modules):

- dev2 #> insmod src/bt.o
- dev2 #> btd -u /dev/ttyS0 -s 57600 -e 0 -r server -i ericsson
- dev1 #> btduser -u /dev/ttyS0 -s 57600 -e 0 -m -r client -i ericsson
- dev1 #> inq
- dev1 #> rf_conn 00:d0:b7:03:27:cd 0
- dev1 #> rf_conn 00:d0:b7:03:27:cd 1
- dev1 #> ppp

Doing so sets up the rfcomm control message server channel 0 and the rfcomm data server channel 1 (see [45](p. 386) for details on RFCOMM). For the difference between *btd* and *btduser* refer to Chapter 2 on page 31.

If ppp connection has completed you will see the following routing information if you start *route*:

```
dev1:~ # route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
dev2.acter.ch * 255.255.255.255 UH 0 0 0 ppp0
129.132.164.64 * 255.255.255.192 U 0 0 0 eth0
loopback * 255.0.0.0 U 0 0 0 lo
default firewall.acter. 0.0.0.0 UG 0 0 0 eth0
```

```
dev2:~ # route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
dev1.acter.ch * 255.255.255.255 UH 0 0 0 ppp0
129.132.164.64 * 255.255.255.192 U 0 0 0 eth0
loopback * 255.0.0.0 U 0 0 0 1
default firewall.acter. 0.0.0.0 UG 0 0 0 eth0
```

To test the link you can try to make a ftp connect to your virtual machine (supposing that you have installed a ftp server on dev2):

- dev1 #> ftp dev2

Try to download something and see how fast your link is.

It's also possible to connect a Ericsson module with a CSR module. Just use the following command to start the client on the CSR module (further details will be given in the next section):

- dev2 #> btd -u /dev/ttyS0 -s 115'200 -e 0 -r server -i csr

2.3.2 CSR Kit

To get the Casira to work with the AXIS stack, upgrade to the beta9 firmware. Before you load the beta9, make sure to save important parameters in the Persistent Storage (e.g. crystal trim, ...) ⁴. Make sure you erase the flash before downloading the beta9, or the PSTool will not work properly. Before you are going to flash a new firmware, be sure you have the BC01b chip, since BC01a will be destroyed with firmware newer than beta7. Set the following parameters in the Persistent Storage using PSTool:

- “Host Interface” to “UART link running H4”
- “UART: Non-BCSP Enabled” to “Enable”
- “UART: Parity Bits” to “No parity”
- “UART: Stop Bits” to “1”
- “UART: RTS Auto Enabled” to “Disable”
- “UART: RTS Enabled” to “Enable”
- “UART Baud Rate” to “115.2 kbaud”
- “UART configuration bitfields” to “168⁵”

⁴see Appendix B.1 on page 89 for the Radio Module Test Reports

⁵Warning: the value 168 is a summarized value for the above mentioned settings.

- “UART: Flow control Enabled” to “Enable”

Make sure to reset the Casira module for the changes to take effect (you can use the “Reset BC” button on the PSTool application). The BC01 Module will now be configured to use H4 over the UART.

Note: You won’t be able to see the “Host Interface” parameter unless you set the access level to developer (Access Key 1812).

Use the following command lines to start the stack (use **btd** not **bt-duser**):

- dev1 #> insmod src/bt.o
- dev2 #> insmod src/bt.o
- dev1 #> btd -u /dev/ttyS0 -s 115'200 -e 0 -m -r client -i csr
- dev2 #> btd -u /dev/ttyS0 -s 115'200 -e 0 -m -r server -i csr

You can then execute **readbd** and **inq** from the client. It is also possible to connect a Ericsson module to the CSR module.

Chapter 3

ARM Port

There are some processor-specific aspects you have to deal with when running the stack on ARM. The concern is alignment and endianness which will be introduced in the following chapter. Another problem are the resource restrictions on the target (RAM, ROM, uC-lib).

3.1 Product and System Environment

The acterCARD is a high-end smart card that has capabilities and resources that reach far beyond traditional embedded systems. As of today the card is defined by a low-power consumption processor (ARM7TDMI - an ARM core with UARTs and 2 MB Flash) with a simple protection unit and 512KB SRAM (external). The operating system running on the card is uC-Linux (Kernel 2.0.x) [1] - an embedded version of Linux.

3.2 Development Environment

On a host running under *SUSE Linux 7.0 Professional* [47] with an updated Linux Kernel [32] (Kernel 2.4.0-test12), we installed one virtual machine as explained in Chapter 2.2 on page 33 and one software compatible target emulator. The Ericsson Bluetooth devices (Bluetooth Application Tool Kit [23]) or the CSR Bluetooth devices (CASIRA[tm] Development Kit [16]) were connected to host UARTs. The virtual machine's tty and the emulator's tty where software mapped to the host serial devices (Figure 3.1 on page 42). For the virtual machine, this mapping is configured in its configuration file (Appendix A.2 on page 81), and for the emulator, the environment variable **ARM_UART=/dev/ttyS0** is needed. The emulator is basically a simulator for ARM6/7 [1] and the GNU debugger gdb 5.0 with AT91 hardware extension [1]. The target operating system is uC-Linux 2.0.38-pre7 (patched for ARM7) [1]

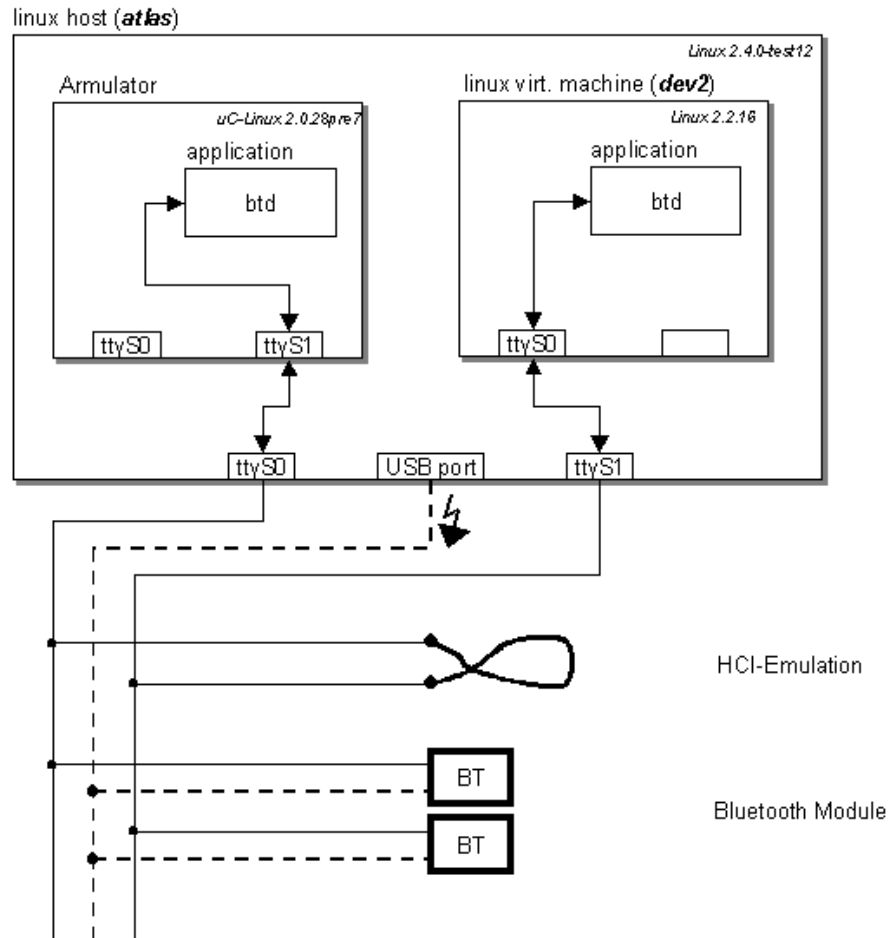


Figure 3.1: ARM Development Environment

For the emulator a diskimage is needed. It contains the linked kernel with all the user programs and the user file system and is generated with the tool *genromfs*. For uC-Linux user programs, a special version of the compiler is needed [1]. It is able to produce fully relocatable code. For linking the user programs uC-lib* are needed. Modules for the kernel are elf-binaries. User programs must be converted with *elf2flt* to a flat-binary before the tool *fltzip* is applied.

3.2.1 Source Tree

The installed software version repository is organized as shown in Figure 3.2 on page 43.

The repository is created using

- user@server:/ > cvs -d /<path>/cvsroot init

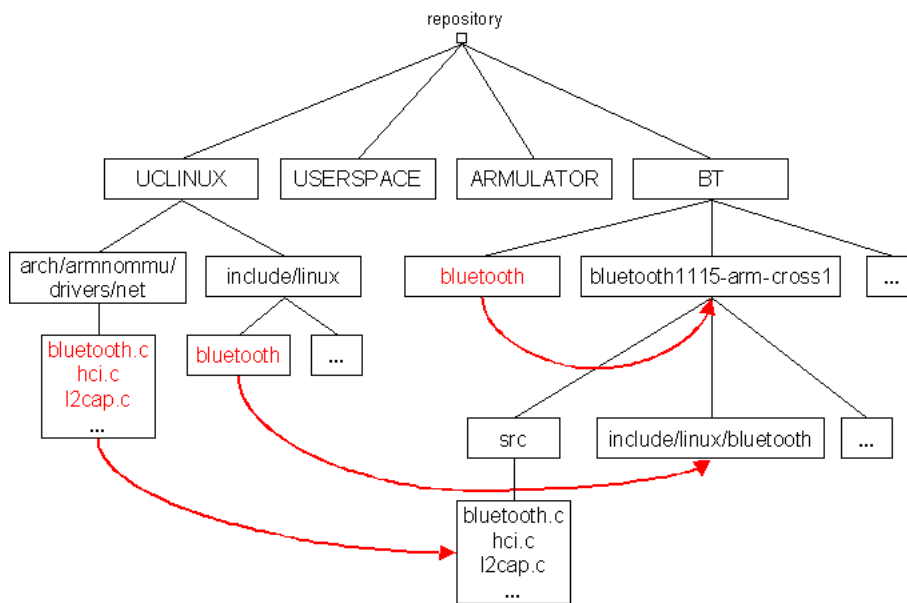


Figure 3.2: Source Tree

The following shell variables have to be added

- CVS_RSH=ssh
- CVSROOT=
:ext:petralia@octopussy:/export/home1/acter/Software/Bluetooth/cvsroot

To add a new project to the repository, change to the directory where the project sources are and use the command

- cvs import -m “Imported sources” SOURCE axis start

To checkout the project use

- cvs checkout SOURCE

If you have changed something in your local project you may commit your changes to the repository

- cvs commit

If someone else has changed something in the repository you may update your local copy

Directory	Description
USERSPACE	plenty of stuff, system utilities, includes run-time.
ARMULATOR	simulator and GNU debugger (gdb 5.0) with AT91 extension.
UCLINUX	uC-Linux 2.0.38-pre7, with ARM7 and EB01/ActerCard patches.
BT	ARM compatible AXIS stack based on version 20001115.

Table 3.1: Software Repository Source Tree

- cvs update

Refer to the manual page (man cvs) or to [17] for detailed information.

3.3 Modifications

The AXIS mailing list [9] has a ARM patch¹ available for the version 1115. It fixes the alignment and endianness problems so it can run on MIPS, ARM big/little endian. These modifications have never been included in the stack. At every new release, you have to redo the modifications by hand.

The following list shows you which files were changed by the patch and which files were changed only because of the above mentioned ARM problems (marked with ***)

```

root@atlas:/tmp/arm/bluetooth# patch -p1 -N < bluetooth_20001115_gmcnuttp1
patching file Makefile
patching file README
patching file apps/btd/Makefile
patching file apps/btd/btattach.c
patching file apps/btd/btd.c
patching file apps/btd/btd.h
patching file apps/btd/startBt
patching file apps/sdp_server/Makefile
patching file apps/sdp_server/sdp_server.c
patching file apps/sdp_server/sdp_server.h
patching file apps/userstack/Makefile
patching file apps/userstack/unplug_test.c
patching file apps/userstack/unplug_test.h
patching file include/linux/bluetooth/bluetooth.h
patching file include/linux/bluetooth/bt_proc.h
patching file include/linux/bluetooth/btcommon.h          ***
patching file include/linux/bluetooth/btconfig.h
patching file include/linux/bluetooth/btdebug.h
patching file include/linux/bluetooth/btmem.h

```

¹contributed by Gordon McNutt, gmcnuttp1@ridgerun.com

```

patching file include/linux/bluetooth/hci.h
patching file include/linux/bluetooth/l2cap.h          ***
patching file include/linux/bluetooth/local.h         ***
patching file include/linux/bluetooth/rfcomm.h       ***
patching file include/linux/bluetooth/sdp.h
patching file include/linux/bluetooth/sdp_proc_protocol.h ***
patching file include/linux/bluetooth/sysdep-2.1.h
patching file include/linux/bluetooth/tcs.h          ***
patching file include/linux/bluetooth/test.h
patching file init_env
patching file libs/expat/xmlparse/Makefile
patching file libs/expat/xmlparse/xmlparse.c
patching file libs/expat/xmltok/Makefile
patching file libs/expat/xmltok/xmltok.c
patching file libs/expat/xmltok/xmltok_impl.c
patching file src/Makefile
patching file src/bluetooth.c
patching file src/bt_proc.c
patching file src/btdebug.c
patching file src/btmem.c
patching file src/hci.c
patching file src/l2cap.c                             ***
patching file src/rfcomm.c                           ***
patching file src/sdp.c
patching file src/tcs.c                              ***
patching file src/test.c

```

As mentioned at the beginning of this chapter, the axis stack lacks of modularity and platform independency. It is well written for Linux operating system. In view that in future Bluetooth will be widely used on a lot of platforms (Linux, Windows, VxWorks, QNX, PSOS, PalmOS [20], ...) it could be good to try to change these disadvantages. This would include a configuration file for configuring the stack for different operating systems. Memory management functions like *malloc*, *free* and functions like *memcpy* and *strcpy* could be made part of the Bluetooth stack since a lot of embedded controllers have limited memories and can't include big library files. In this way the stack could easily be ported to any embedded environments.

3.3.1 Loadable Module

The first approach was to build a loadable module (*bt.o*) with enabled version information on all symbols. Therefore, changes in *BT/bluetooth/src/Makefile* to adapt it to the ARM target were necessary (see also Chapter 3.1 on page 41). A typically *Makefile* for a kernel module can be found in Appendix A.4 on page 83.

To finally be able to include the module, RAM and SRAM size had to be increased (*UCLINUX/include/asm/arch/hardware.h*) and the block size had to be adjusted (*UCLINUX/mmnommu/kmalloc.c* and

UCLINUX/mmnommu/page_alloc.c).

3.3.2 Stack Linked in Kernel

The second approach was to include the Bluetooth source into the UCLINUX source tree. With this approach you can select to link the Bluetooth stack directly into the kernel. Changes to the file *UCLINUX/arch/armnommu/drivers/net/Config.in* and *UCLINUX/arch/armnommu/drivers/net/Makefile* were necessary to make the Bluetooth source appear in the *make menuconfig* script, and a suitable place during kernel start up had to be found (*UCLINUX/init/main.c*) to start the stack. The same function as for the module is called to start the stack (*bt_init()*). The kernel configuration file *.config* is shown in Appendix A.3 on page 82.

The Bluetooth stack sources were linked from *UCLINUX/arch/armnommu/drivers/net/* to the *BT/bluetooth/src* directory whereas the link *BT/bluetooth* points to the Bluetooth arm source directory *BT/bluetooth/src*. For the include files a link points from *include/linux/bluetooth/* to *BT/bluetooth/include/linux/bluetooth/*. All those links are done in *UCLINUX/arch/armnommu/drivers/net/Makefile*. Figure 3.2 on page 43 gives an overview.

Do the following to include the Bluetooth stack to your kernel:

- #> make menuconfig
- #> choose AXIS Bluetooth Stack
- #> make clean dep
- #> make

3.3.3 User Program

A typical *Makefile* for a user program can be found in Appendix A.5 on page 84.

There were no building problems with the user program *sdp_server*. But the target did not include the readline development package which is linked to the application *btd*. Therefore a new define called *USE_READLINE* was added:

```

//#define USE_READLINE

#ifdef USE_READLINE
#define MAXLINE 100
static char line[MAXLINE];

int getline(char s[], int lim) {

```

```

int c,i;
for (i=0;i<lim-1 && (c=getchar())!=EOF && c!='\n';++i)
    s[i]=c;
if (c == '\n') {
    s[i] = c;
    ++i;
}
s[i] = '\0';
return i;
}

void read_history(char *hist_file_name){}

void write_history(char *hist_file_name){}

void add_history(char *command){}

char *readline(char *prompt)
{
    printf("%s",prompt);
    fflush(stdout);
    getline(line,MAXLINE);
    return line;
}

#endif

```

If *USE_READLINE* is defined, the *btd* application works as before. If it is not defined some dummy functions emulate the readline functions by serving a simple readline function.

To parse the command options without the use of *getopt_long()* which was not available on our target, a new define called *USE_GETOPT_LONG* was added:

```

//#define USE_GETOPT_LONG

#ifndef USE_GETOPT_LONG
static char *optarg;

int getopt_long(int argc, char *argv[], const char *optstring,
                const struct option *longopts, int *longindex) {

    int opt= -1;
    static int anz = 0;

    while ( ++anz < argc) {
        if ( (argv[anz])[0] == '-' ) {
            opt = (argv[anz])[1];
            optarg = NULL;
        }
        else {
            optarg = argv[anz];
            return opt;
        }
    }
}

```

```

    return -1;
}
#endif

```

If *USE_GETOPT_LONG* is defined, the *btd* application works as before. If it is not defined some dummy functions emulate the *getopt_long* functions.

To decrease the available HCI input buffers a new define called *CONFIG_REDUCED_MEMORY* was added to the file *hci.c*:

```

#ifndef CONFIG_REDUCED_MEMORY
#define NBR_OF_HCI_INBUFFERS 5
#define HCI_IN_SIZE 1024
#else
#define NBR_OF_HCI_INBUFFERS 7
#define HCI_IN_SIZE 17000
#endif
#define NBR_CMD_BUFS 10

```

If *CONFIG_REDUCED_MEMORY* is defined², the input buffers for HCI commands are changed.

Please note that *btd_user* is not available on the target due to the lack of functions for pseudo TTY handling.

btd implements all necessary function for *inquiry* and *connection establishment*. It has been tested with the Ericsson and CSR Module.

Please refer to Chapter 3.5 on page 50 how the stack and *btd* is started and can be tested on the target.

All other user programs that do not belong to the AXIS stack are described next:

3.3.3.1 sertest(-arm)

A small program to test a serialport. It simply sends (client) a string of data over a serial cable. On the other side (server) the data is echoed back if it was correct, and checked once again when received.

3.3.3.2 hello-arm

A small “hello world” program that was used to test the ARM user program makefile.

²only possible when using *make menuconfig*

3.3.3.3 eric

This program emulates a Ericsson ROK 101 007 module in software. The emulation is restricted to the replies a hardware module would give, when the host sends the commands shown in Appendix A.6 on page 85.

3.3.3.4 sane

A shell script that changes and prints terminal line settings. It uses the program *stty*.

3.4 Object File Size

The following list provides an overview of the section sizes and the total size for each of the object files (with *CONFIG_REDUCED_MEMORY* enabled).

text	data	bss	dec	hex	filename

USER PROGRAM:					
31561	6140	1904	39605	9ab5	apps/btd/btd.o
KERNEL WITHOUT LINKED STACK:					
435123	37964	36288	509375	7c5bf	linux
a) BT DEBUG ENABLED					
KERNEL MODULE:					
136	0	0	136	88	src/btdebug.o
968	144	0	1112	458	src/bt_proc.o
2612	176	112	2900	b54	src/sdp.o
3536	0	32	3568	df0	src/btmem.o
3668	0	0	3668	e54	src/tcs.o
10980	36	468	11484	2cdc	src/bluetooth.o
15784	28	1280	17092	42c4	src/rfcomm.o
27256	28	18728	46012	b3bc	src/hci.o
25308	76	112	25496	6398	src/l2cap.o
90248	488	20732	111468	1b36c	src/bt.o
KERNEL WITH LINKED STACK:					
524771	38296	61836	624903	98907	linux
b) BT DEBUG DISABLED					
136	0	0	136	88	src/btdebug.o
968	144	0	1112	458	src/bt_proc.o
2104	176	112	2392	958	src/sdp.o
3272	0	0	3272	cc8	src/tcs.o

```

3372      0      32      3404      d4c src/btmem.o
6452     36     468     6956     1b2c src/bluetooth.o
14728    28    1280    16036    3ea4 src/rfcomm.o
18144    28   18728    36900    9024 src/hci.o
23232    76     112    23420    5b7c src/l2cap.o

72408    488   20732   93628   16dbc src/bt.o

KERNEL WITH LINKED STACK:

507139   38296   61836   607271   94427 linux

```

The stack enlarges the kernel for 19% in size. To make the stack smaller, TCS (and hence SCO support) could be made removable and unused features³ could be wrapped with preprocessor conditionals. Also, each HCI input buffer could be allocated dynamically for each new active (non parked/hold) HCI handle.

3.5 Running Stack on ARMulator

3.5.1 Starting Bluetooth Stack

Once the stack is compiled as a loadable module (using the Makefile in *BT/bluetooth/*) or included to the kernel (using *make menuconfig* in UCLINUX) do the following to start the diskimage:

- #> cd USERSPACE
- #> make
- #> export ARM_UART=/dev/ttyS0
- #> ../ARMULATOR/gdb/gdb linux
- #> tar sim
- #> load
- #> run

If you build a loadable module, insert it with the command *insmod ./bt.o*. If the stack is included to the kernel it will start automatically on start up⁴.

The data sent between the stack and the Bluetooth module depends on whether the Ericsson or the CSR module is attached. Appendix A.6 on page 85 and Appendix A.7 on page 87 give you an overview

³e.g. most of the HCI commands are not needed in order to run the basic functionalities of the stack

⁴*bt_init* in *UCLINUX/init/main.c*

about the data the stack should send and receive after starting the application *btd*.

3.5.2 Starting Bluetooth Application

Just start the user program *btd* and enter the command

- #> ppp

The application will open the device */dev/ttyS1* on the target which is forwarded to the device */dev/ttyS0* on the host. Make sure the Bluetooth hardware is connected to this device. To make a PPP connection start *btd_user* on your host and do the following

- host #> btduser -u /dev/ttyS0 -s 57600 -e 0 -m -r client -i ericsson
- host #> inq
- host #> rf_conn 00:d0:b7:03:27:cd 0
- host #> rf_conn 00:d0:b7:03:27:cd 1
- host #> ppp

3.5.3 acter Kit

Acter ag has its own Bluetooth hardware design using the BlueCore chip and acting as the described CSR Kit in Chapter 2.3.2 on page 38.

Part III

Native Networking

Bluetooth is a wireless standard for low cost, low power, shortrange radio communications. This cheap but powerful technology will be available in a wide range of mobile devices and the result will be an increasing demand to local and/or global IP based services. This part describes some issues and goals for IPv4/IPv6 and end-to-end IP network design that are candidates for use in an IP over Bluetooth solution. We give an overview about how this problem is solved by the current standard and discuss how it differs from current approaches. Last but not least we outline a set of possible designs and present the design we name *RedNose*.

Chapter 4

RedNose - A New Approach to Bluetooth Networking

The current Bluetooth provisions for IP (Internet Protocol) networking are limited to a LAN (Local Area Network) access configuration, with IP over PPP over RFCOMM over L2CAP. There are limitations on flexibility and efficiency in such a configuration: IP only goes over point-to-point (PPP) links and multi-hop ad-hoc IP networks are not practical with the existing profiles. In this Chapter, we outline a set of possible IP network designs and present the design we name *RedNose*.

4.1 Bluetooth Overview

The radio specification is resumed in Chapter 1.2.1 on page 8. Refer to Chapter 1.1 on page 5 for a Bluetooth Protocol Stack overview.

4.2 Bluetooth IP Architectures

Communication between Bluetooth devices follows a strict master-slave scheme, i.e. there is no way for slave devices to communicate directly with each other. A device acting as master can have up to 7 active slaves connected. The master always starts the transmission in the even-numbered slots whereas the slaves start their transmission in the odd-numbered slots (see Figure 4.1 on page 58). Only the slave addressed by its active member address (AM_ADDR¹) can return a packet in the next slave-to-master slot. If no valid AM_ADDR is received, the slave may only respond if it concerns

¹To identify each slave separately, each slave is assigned a temporary 3-bit address to be used when it is active. Packets exchanged between the master and the slave all carry the AM_ADDR of this slave. Slaves that are disconnected or parked give up their AM_ADDR. A new AM_ADDR has to be assigned when they re-enter the piconet.

its reserved SCO slave-to-master slot. In case of a broadcast message, no slave is allowed to return a packet.

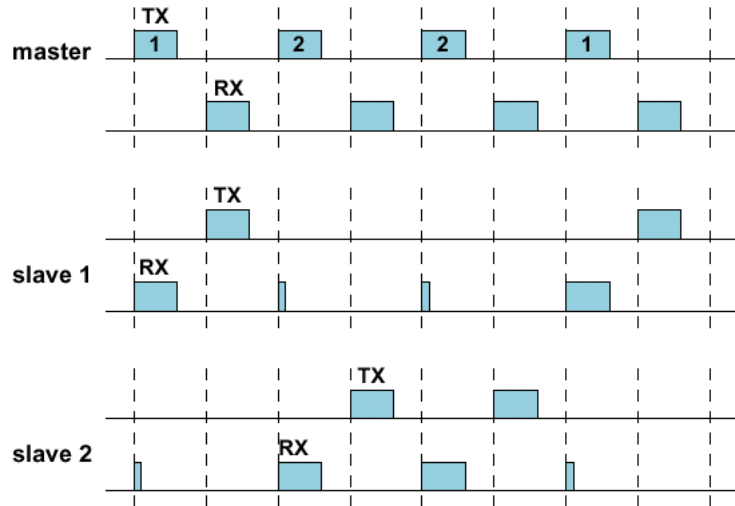


Figure 4.1: Timing in Multi-Slave configuration

4.2.1 Piconet

A Piconet can be formed between any two Bluetooth devices. The discovered device (using *inquiry*) joins the piconet of the paging master (using *paging*) as a slave. Since the paging unit always starts out as master, a master-slave role exchange is required if a slave role is desired.

A master can have up to seven active slaves associated to it at any time. Slaves not participating in normal communication can be in the parked state. A slave can request to be “parked” or “un-parked”. Also, a master can at any time “park” or “un-park” any Bluetooth devices “belonging” to his piconet (see Figure 1.4 on page 8). The slaves have to stay synchronized to the master while participating in the piconet.

4.2.2 Scatternet

If multiple piconets cover the same area, a unit can participate in two or more overlaying piconets by applying time division multiplexing (TDM). A group of piconets in which connections consist between different piconets is called a scatternet and can be formed between at least three devices, where at least one device is able to switch between the hopping sequences of the other two devices: A master or slave can become a slave in another piconet by being paged by the master of this other piconet. On the other hand, a unit participating in one piconet can page the master or slave of another piconet. These abilities allow to build a multiple-hop wireless network. In

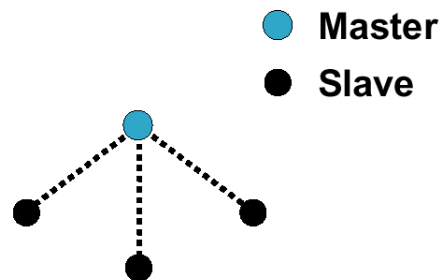


Figure 4.2: Single Bluetooth Piconet with multiple Slaves

case of ACL links only, a unit can request to enter the hold (preferably, as it could retain the AM address) or park mode in the current piconet during which time it may join another piconet by just changing the channel parameters.

Since the clocks of two masters of different piconets are not synchronized, a slave unit participating in two piconets has to take care of two offsets that, added to its own native clock, create one or the other master clock. The two master clocks drift independently which requires regular updates of the offsets in order for the slave unit to keep synchronization to both masters.

Since the paging unit always starts out as master, a master-slave role exchange is required if a slave role is desired (see [45](p. 123)).

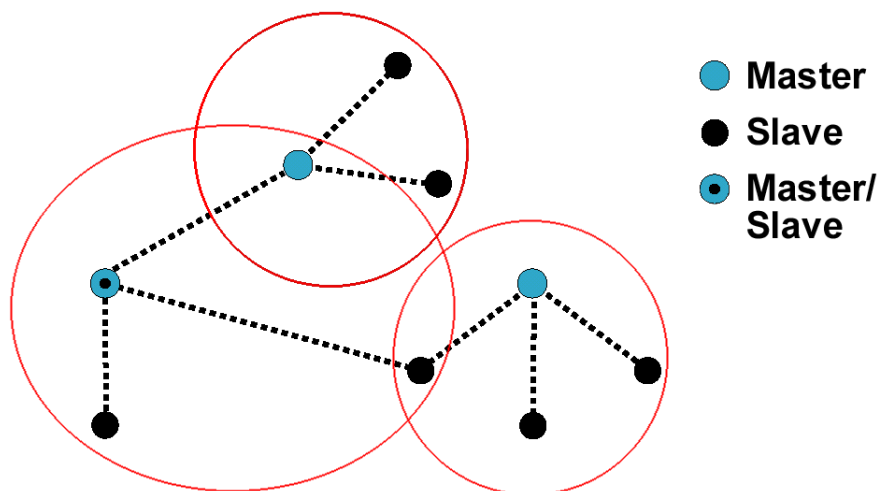


Figure 4.3: Scatternet Topologies

4.2.3 Broadcast

ACL packets not addressed to a specific slave are considered as broadcast packets and are read by every slave. Because slaves only transmit on ACL links when they are addressed by the master, only the master is allowed to broadcast on an ACL link. A broadcast packet is indicated by the all-zero AM_ADDR. Since broadcast messages are not acknowledged, each broadcast packet is repeated for a fixed number of times. This gives a reasonable chance that one of the broadcast transmissions will get through. The packets are checked on errors using the CRC.

The *Broadcast_Flag* is located in bit 6 and 7 in the second byte of the HCI ACL Data packet. It can be set to

- No broadcast: Only point-to-point.
- Active broadcast: Packet is sent to all active slaves.
- Piconet broadcast: Packet is sent to all slaves, including slaves in park mode.

4.3 Bluetooth IP Architectures Goals

Our goals assume that Bluetooth will be available in a wide range of mobile devices that allow users to get access to data and/or communication services within a private or public area such as airports, buildings, super-markets and so on. The services may vary from information services, paying services, access services ... to services using wide area networks (media streaming over IP, voice over IP, Internet, ...). IP support in Bluetooth will enable a large number of applications and systems to use Bluetooth as a link layer. Thus our goal is to support IP which can be done in many ways. Saying we want to support IP means that we want to realize as much flexibility as with IP over Ethernet (broadcast support, Point-to-Point connections) without changing the radio, baseband or link manager specifications viewed as the Physical and Medium Access Layer as shown in Figure 4.4 on page 61. Mostly these layers are implemented in hardware, so they have to be used anyway. Thus, among other things, the solutions have to be compatible with the given master-slave scheme.

The following IP based technology will have an impact when doing IP over Bluetooth:

IPv6 must be considered as an important point for the long term success of Bluetooth IP devices. Due to the low cost radio communication, any consumer device can become a Bluetooth IP device. Upgrading Bluetooth to IP and finally IPv6 could also have an impact on the global IP network.

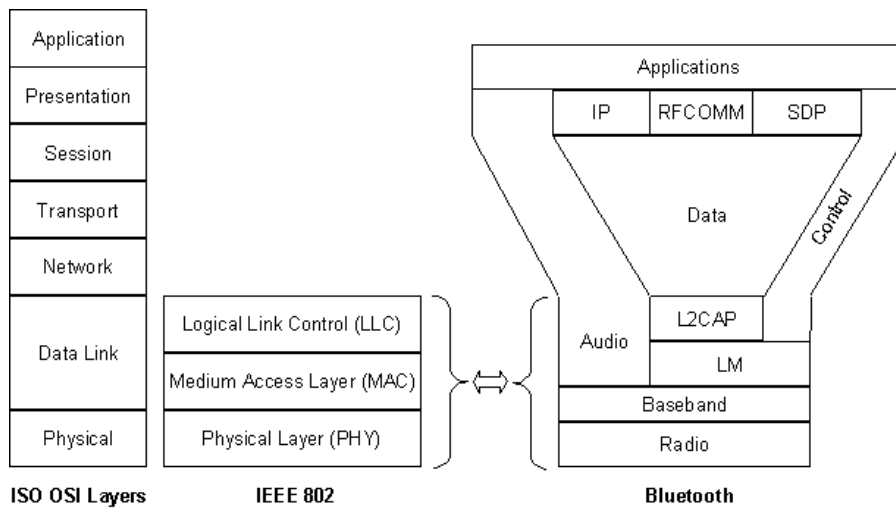


Figure 4.4: Relation between OSI and Bluetooth Layers

Header Compression will become an issue since Bluetooth is currently bandwidth limited. Many suggestions for IP over Bluetooth tend to leave away the Point-to-Point (PPP) layer which provides data compression and encryption.

IP Multicast is required by many real-time streaming applications. It will become important for mobile wireless real-time streaming.

IP Security provides various security services for traffic at the IP layer in both the IPv4 and IPv6 environments. It could be implemented in addition to the already existing Bluetooth Host Controller SAFER+ encryption.

Quality of Service based on RFC 1363 is employed by the Bluetooth Host Controller. A goal will be to provide QoS for IP.

End-to-end IP Network in a mobile wireless environment has to be designed with care, since growth and use of Bluetooth IP devices are expected to place a significant load on the global IP network. Since Bluetooth devices accessing the IP network can be in motion, routing plays an important role. For this purpose Mobile IP [36] may be applied. Since IP based devices need an IP address, IP address assignment plays an important role. The Dynamic Host Configuration Protocol (DHCP [21]) is well suited to accomplish this task.

4.4 Bluetooth IP Architectures Solutions

The master-slave scheme is viewed as a given fact. That is why all listed solutions in this Chapter will remind the reader of the LAN Access Profile where Bluetooth-enabled devices can access services of a LAN connecting to a LAN access point. They differ in the efficiency of how IP is supported and whether Mobile IP can be supported or not. With Mobile IP we mean that we take care of enabling Bluetooth devices to migrate between different Access Points. The Layer 3 protocols for supporting this are routing protocols and DHCP. On Layer 2, handoffs between piconets are necessary. The requirements for supporting handoffs are resumed in Chapter 4.4.7 on page 68.

Solutions based on the L2CAP layer should get along with the other protocols (SDP, RFCOMM, TCS) being multiplexed in L2CAP. Since protocol multiplexing is what L2CAP does, no problems are expected. Running IP over L2CAP could allow IP multicasting/broadcasting to take advantage of L2CAP point-to-multipoint channels for efficient multicasting/broadcasting.

When looking at Figure 1.1 on page 5 you could ask yourself whether HCI support is necessary or not. The HCI provides a uniform interface method of accessing the Bluetooth hardware capabilities for the HCI driver in the host system. The Host Controller's firmware implements the HCI Commands for the Bluetooth hardware by accessing baseband commands, link manager commands, hardware status registers, control registers, and event registers. If you wish to develop a Bluetooth embedded solution in form of a chipset hosting the entire Bluetooth stack (right from baseband to application), then you most probably would not require HCI, since HCI is required for applications having a clear division between the Host and the Host Controller parts of the stack.

An overview of Bluetooth IP Architecture Solutions is given in Table 4.1 on page 63.

	LAP	PAN	IPoSCO	IPoACL	IPoL2CAP	IPoPPPoL2CAP	RedNose
Broadcast/Point-to-multipoint	n	y	n	n	y	n	y
Baseband changes	n	n	y	y	n	n	n
Affect other applications	n	n	y	y	n	n	n
Mobile support	n	y	y	y	y	n	y
CRC ^a /ARQ ^b	y	y	n	y	y	y	y
FEC ^c	y	y	y	y	y	y	y
New protocols to handle MTU negotiation/fragmentation	n	n	y	y	n	n	n
Authentication and authorization support	y	?	n	n	n/y	y	n/y
Compression supported	y	y	n	n	n	y	n
Scalability	n	?	y	y	y	n	y
Piconet handoff support	n	?	n/y	n/y	n/y	n	y
IPv6 preferred	n	y	n	n	y	n	y

Table 4.1: Overview of Bluetooth IP Architecture Solutions

^aMost ACL packets have CRC. The AUX1 packet has no CRC.

^bThe ARQ scheme only works on the payload in the ACL packet (only that payload which has a CRC).

^cMost ACL/SCO packets have FEC.

4.4.1 Current Solution: LAN Access Profile

LAN Access Profile [46](p. 265) is the proposed solution by SIG for IP over Bluetooth. It is using the IETF [29] Point-to-Point Protocol (PPP) [41], RFCOMM [45](p. 387), L2CAP [45](p. 247) over the Host Controller as shown in Figure 4.5 on page 64.

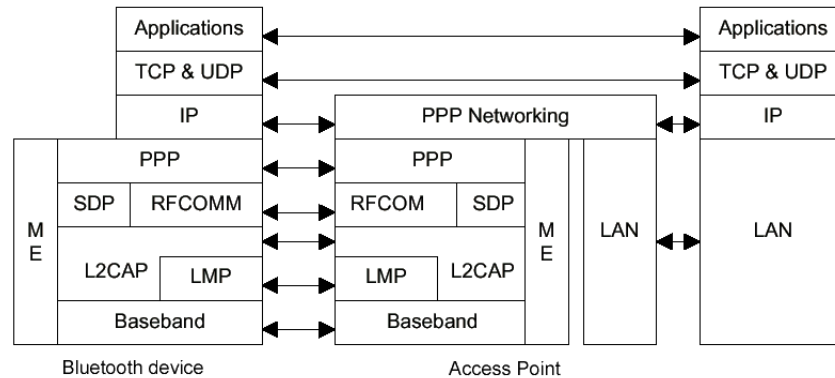


Figure 4.5: IP Protocol Stack using LAN Access Profile

PPP provides authentication, encryption, data compression, and multi-protocol facilities (e.g IP, IPX, etc.) and has been chosen because it is widely used. RFCOMM is the Bluetooth adaption of GSM TS 07.10. L2CAP supports packet segmentation and reassembly, and higher level protocol multiplexing.

LAP does not deal with LAN emulation or ad-hoc networking. This fails to exploit the point-to-multipoint capabilities of the Bluetooth core and in particular the point-to-multipoint channels that L2CAP can provide. It converts a broadcast medium into a non-broadcast medium and leads to inefficiencies in IP multicasting and broadcasting. This introduces complications into IP networking functions that expect LANs to have broadcast media. A typical scenario where LAP can be used is in an environment where a Bluetooth device is connected directly to a wired IP-backbone. Such a device is called an Access Point (AP) and provides access to up to seven active client nodes to a LAN (e.g. Ethernet, Token Ring, Fiber Channel, Cable Modem, USB, ...). Once connected, the clients will operate as if they were connected to the LAN via dial-up networking. The PPP protocol is used exclusively to transport IP traffic to and from the clients to the AP and out to the wired network. The clients can only communicate with each other via the LAP using IP forwarding. Assuming that client A knows the IP address of client B, he could become master in a new piconet and establish a connection to client B. To have this working, client A must provide the services of a PPP Server and client A and B must support scatternet functionality (baseband and PPP). This shows all the limiting problems of

running a network over serial PPP link. Moreover, PPP is not sufficient in a mobile environment that contains multiple wireless AP's and does not scale to operate efficiently.

A major component of the long time it takes for a PPP connection to be set up are the authentication and authorization functions. As long as a node wanders around within zones (trusted areas such as the inside of a building where authentication and authorization need only take place once), a faster way to set up the PPP and IP connections could be implemented.

4.4.2 Solution under Development: PAN Profile

At present the Bluetooth SIG together with the IEEE 802.15 working group are working out a new profile for the IP over Bluetooth problem. Their solution is known as the Personal Area Networking (PAN) profile [15]. The PAN profile implies an IP over Bluetooth protocol (Bluetooth Network Encapsulation Protocol (BNEP)). The specification for BNEP can be downloaded by associate members of the SIG from their website. PAN networking enables multi-user collaboration by creating ad-hoc IP-based personal area networks for data, voice, video and other forms of communications. Using PAN for ad-hoc network does not require the existence of any special purpose products or infrastructure, but it will inter-operate with such devices if they exist. The network solution uses L2CAP to provide enhanced features and performance.

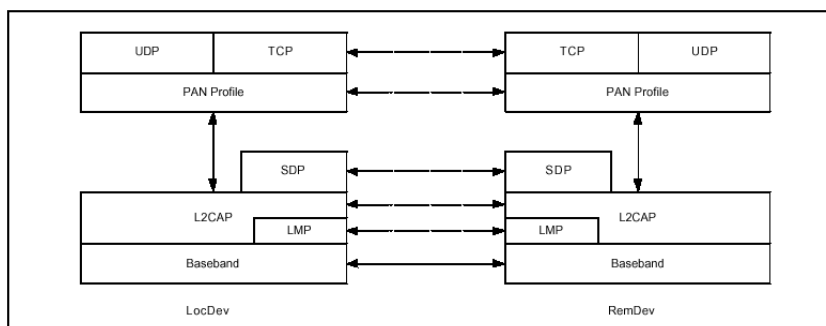


Figure 4.6: Pan Profile for IP Networking

4.4.3 Possible Solutions: IP over SCO

SCO packets are used on synchronous SCO links (point-to-point). The packets do not include a CRC and are never retransmitted. The master maintains the SCO links by using reserved slots at regular intervals. He can support up to three SCO links to the same slave or to different slaves. A slave can support up to three SCO links from the same master, or two SCO links if the links originate from different masters. Up to now, three pure SCO packets

have been defined (see Figure 4.7 on page 66) which are used for 64 kb/s speech transmission.

Type	Payload Header (bytes)	User Payload (bytes)	FEC	CRC	Symmetric Max. Rate (kb/s)
HV1	na	10	1/3	no	64.0
HV2	na	20	2/3	no	64.0
HV3	na	30	no	no	64.0
DV*	1 D	10+(0-9) D	2/3 D	yes D	64.0+57.6 D

Figure 4.7: SCO packets

- Advantages:
 - Low overhead packets
- Disadvantages:
 - No CRC check.
 - Only point-to-point link are supported.
 - Bandwidth inefficiency.
 - Connection-Oriented.
 - New protocols are needed to handle MTU negotiation/fragmentation² which creates overhead.
 - Changes in the Baseband are probably necessary (ACL header, etc.).

4.4.4 Possible Solutions: IP over ACL

The ACL link provides a packet-switched connection between the master and all active slaves participating in the piconet (point-to-multipoint). Between a master and a slave only a single ACL link can exist. For most ACL packets (see Figure 4.8 on page 67), packet retransmission is applied to assure data integrity. The AUX1 packet has no CRC and is not retransmitted. It could be used to support UDP. A slave is permitted to return an ACL packet in the slave-to-master slot if and only if it has been addressed in the preceding master-to-slave slot. If the slave fails to decode the slave address in the packet header, it is not allowed to transmit.

- Advantages:

²not necessary with IPv6

Type	Payload Header (bytes)	User Payload (bytes)	FEC	CRC	Symmetric Max. Rate (kb/s)	Asymmetric Max. Rate (kb/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

Figure 4.8: ACL packets

- Connection-Less.
- Low overhead, since we will have some overhead because of the additional framing.
- Disadvantages:
 - New protocols are needed to handle MTU negotiation/fragmentation³ which creates overhead.
 - Changes in the Baseband are probably necessary (ACL header⁴).
 - Bandwidth sharing problems with other applications. L2CAP is supposed to have a channel scheduler to make sure that all applications will get a fair amount of bandwidth.

4.4.5 Possible Solutions: IP over L2CAP

In order to reduce the layering overhead, it may be desirable to run IP directly over L2CAP. This would allow IP multicasting/broadcasting to take advantage of L2CAP point-to-multipoint channels for efficient multicasting/broadcasting.

- Advantages:
 - No need to change Baseband (ACL header⁵).
 - Handles fragmentation and MTU negotiation.
 - Will not affect other applications (bandwidth, etc).
 - Low data overhead (4 bytes per frame).

³not necessary with IPv6

⁴TYPE field (see [45](p. 51))

⁵TYPE field (see [45](p. 51))

4.4.6 Possible Solutions: IP over PPP over L2CAP

Since PPP supports authentication and authorization it may be suitable to use PPP and to leave out the RFCOMM layer.

- Advantages:
 - PPP can encapsulate multi-protocol datagrams.
 - PPP can establish and configure different network-layer protocols.
 - PPP provides multiple compression choices.
- Disadvantages:
 - Overhead.
 - PPP does not scale to operate efficiently.
 - PPP is not sufficient in a mobile environment

4.4.7 Handoffs Between Piconets

As described in Chapter 4.2.2 on page 58, the Piconet/Scatternet abilities allow to build a multiple-hop wireless network. The LAN Access Point provides access to a LAN whereas the mobile devices use the services of the LAP. For the question who should be the master and who the slave in those piconets, two approaches can be found. They are shown in Figure 4.9 on page 68.

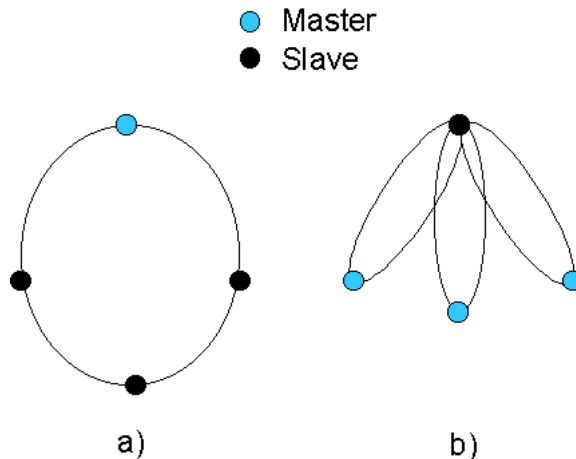


Figure 4.9: LAP's Role in a Piconet: Master or Slave

In the first piconet design approach (a) the Access Point acts as the master whereas the connected Bluetooth devices are slaves. Such a piconet is restricted to a maximum of seven active slaves. A new approaching Bluetooth

device establishes a connection to the Access Point. The Access Point being the slave and the Bluetooth device being the master. After a master/slave switch, the new Bluetooth device is added to the existing piconet as a slave.

A second approach (b) to connect multiple Bluetooth devices to an Access Point results when each Bluetooth device establishes a new piconet to the Access Point. In this case the Access Point acts as a slave and the Bluetooth devices act as masters in each piconet. As soon as the Access Point periodically enters the page scan and inquiry scan modes to respond to approaching mobile devices, the mobile device can start connecting. Using a Time-Division-Multiplexing scheme⁶ the slave will be addressable in all the piconets. The disadvantage of this approach is that the necessary switches between the piconets are too time-consuming. This results in a deterioration of performance.

Solutions have to be found, when mobile devices leave the range of their piconet to enter the range of a new one. The following problems have to be solved:

- If a link to the Access Point is broken (this may happen because the mobile Bluetooth device has left the range of the piconet or because any abnormal error has broken the connection), the Bluetooth device has to find an Access Point within range. We assume that, in case we connect to a new Access Point, the device's IP address will not change and routing paths to the new Access Point for our device are adjusted. Furthermore, as aforementioned, the mobile Bluetooth device will start connection as a master and perform a master/slave switch later on.
- The IP layer and any other upper layer should not be affected by piconet handoffs.

The reader will now be introduced to a possible solution for the most promising way of supporting IP over Bluetooth (IP over L2CAP). The solution is not restricted to IP over L2CAP but can also be applied to any other possibilities using L2CAP. A small sublayer behaving like a network interface is needed to place the IP layer above the L2CAP.

4.4.7.1 Discovery State

After starting up or when a broken link has been detected, the mobile Bluetooth device starts finding an available Access Point within range. For this an interface to SDP is needed or a dedicated inquiry access code should

⁶By applying time division multiplexing a unit can participate in two or more piconets. Therefore the unit has to use the same channel access code, channel hopping sequence and clock synchronisation as the master of the piconet. Since a master schedules the slave communication in his piconet this could also be seen as a time division multiplexing. We will use the term TDM only for the first mentioned case.

be reserved. Unless the maximum number of devices for the respective piconet has been reached, the Access Point periodically enters the page scan and the inquiry scan modes to respond to approaching mobile devices.

4.4.7.2 Configuration State

After discovery, the Bluetooth device performs a master/slave-switch and initiates the negotiation of the L2CAP and baseband MTU. By L2CAP MTU we refer to the largest datablock the IP layer is allowed to send to the L2CAP layer, and by baseband MTU we mean the size of the baseband segments. The master/slave-switch is initiated by the Access Point.

4.4.7.3 Connected State

When the connected state has been reached the first time, the Bluetooth device initiates IP address assignment using DHCP. Every IP datagram coming from the Bluetooth device is sent to the Access Point's Bluetooth hardware address (which has been provided during inquiry). Every IP datagram coming from the network has to be submitted to the corresponding Bluetooth device. For this, the Access Point maintains a lookup table with data records containing IP address, Bluetooth hardware address, and L2CAP connection handle of every Bluetooth device (see Figure 4.10 on page 70).

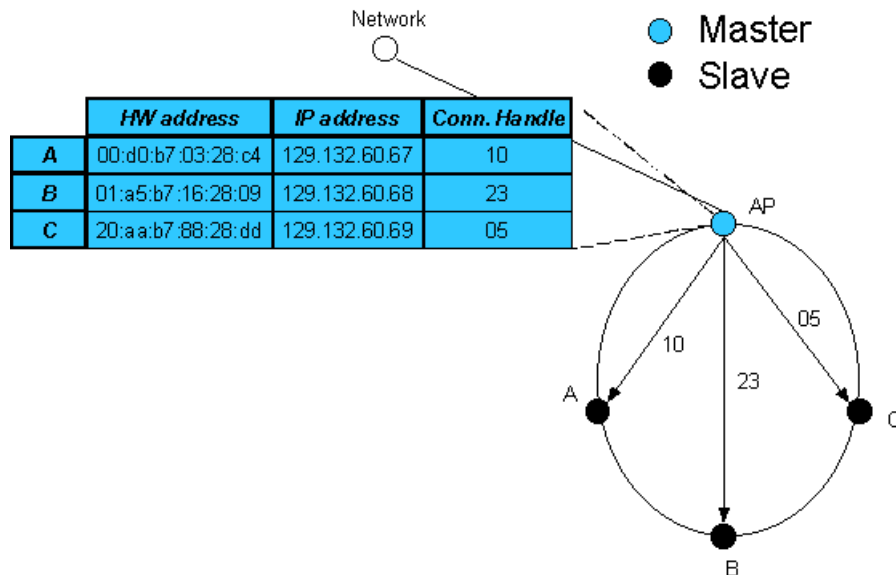


Figure 4.10: Access Point maintains a Lookup Table

4.4.7.4 Link Down Detection

A connection may break down due to various reasons such as a device moving out of range or a power failure condition. In order to supervise link loss, each Bluetooth link has a timer used for link supervision. The IP sublayer needs an interface to this information. If a broken link is detected by the Bluetooth device, the device attempts to reconnect to the same Access Point or to any other available Access Point. If the Access Point detects a broken link, the corresponding data records for this connection are deleted in the lookup table. The Access Point will not try to reconnect to the Bluetooth device.

4.5 RedNose IP Architecture

In the previous section we have shown several designs to support IP over Bluetooth. We pointed out that the most promising way is IP over L2CAP, and we showed how an additional layer above L2CAP can handle handoffs between piconets.

Looking at Figure 4.11 on page 72, someone could ask himself whether it is necessary for slave-slave connections to forward the datagrams to the Access Point. In general, any two Bluetooth devices within radio range could directly form their own piconet. Due to the stringent TDM scheme, forming a new piconet is the only possibility of preventing a slave from communicating with other slaves. The devices can still be part of their original piconets if traffic flows to or from them, or if they need to receive control information. Since the frequency-hopping spread-spectrum (FHSS) system makes Bluetooth very robust against interference, new piconets gain substantially more capacity than they lose as a result of increased interference between them. An analysis of the interference of collocated piconets is given in Chapter 1.2.1.1 on page 9 with the expected value of the throughput as a function of the number of piconets (Figure 1.7 on page 13).

Let us have a closer look at this sublayer. It still behaves like a network interface for the IP layer. In addition to the previously mentioned function, new features are needed (similar to those of the Access Point sublayer):

- Periodically initiate an inquiry and look for Bluetooth devices within range.
- Determine the IP address of all inquired Bluetooth devices.
- Maintain a lookup table with records containing IP address, Bluetooth hardware address, and L2CAP connection handle of every Bluetooth device within range.

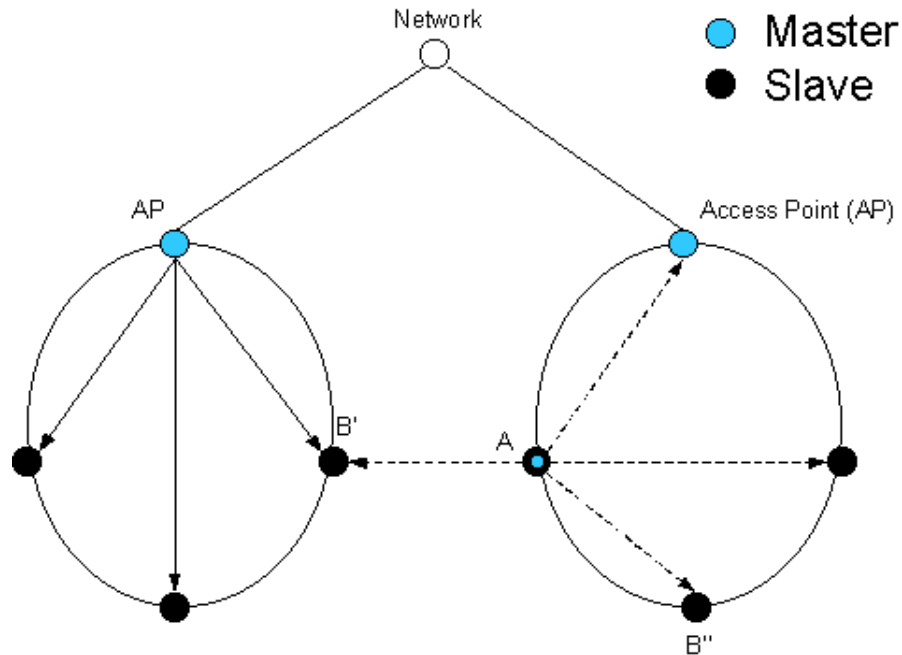


Figure 4.11: Connection to another Piconet

	Minimum Time	Average Time	Maximum Time
Inquiry	0.00125 s	3–5 s	10.24–30.72 s
Paging	0.0025 s	1.28 s	2.56 s
Total	0.00375 s	4.28–5.28 s	12.8–33.28 s

Table 4.2: Minimum/Maximum Inquiry/Paging Time

As the inquiry takes a substantial amount of time, an appropriate value has to be selected for the inquiry interval (see Table 4.2 on page 72).

We will now look closer at the problem of how to obtain the IP address of all inquired Bluetooth devices: The sublayer of every Bluetooth device is aware of his IP and hardware address. If device A wants to know the IP address of device B

- a new protocol in the sublayer could exchange this information. This would entail opening a connection to every inquired device using paging. Instead of a new protocol, we could use *ping* (i.e. ICMP ECHO/REPLY), one of the most widely available tools bundled with TCP/IP software packages. The sublayer could take the address from the source field of the first IP datagram received from the Bluetooth device.
- something similar to the Address Resolution Protocol (ARP) [24] could

be used.

- device A could perform a service discovery which implies that this information be included in the SDP database.
- and both are using IPv6 [30] the problem can easily be solved due to the concept of IPv6: Interface IDs which are part of IP addresses and are required to be constructed in IEEE EUI-64 format [27] may have global scope when a global token is available (e.g. Bluetooth hardware address). Thus, device A is aware of all IP traffics belonging to device B, just by looking at the 64-bit interface identifier of an IP address.
- device A could distinguish between finding IP addresses of devices within his piconet and finding IP addresses of devices that are outside his piconet. For the first group of devices a protocol in the sublayer could exchange the missing information with the Access Point maintaining such a database (see Figure 4.12 on page 73). For the second group, one of the above mentioned ideas can be taken.

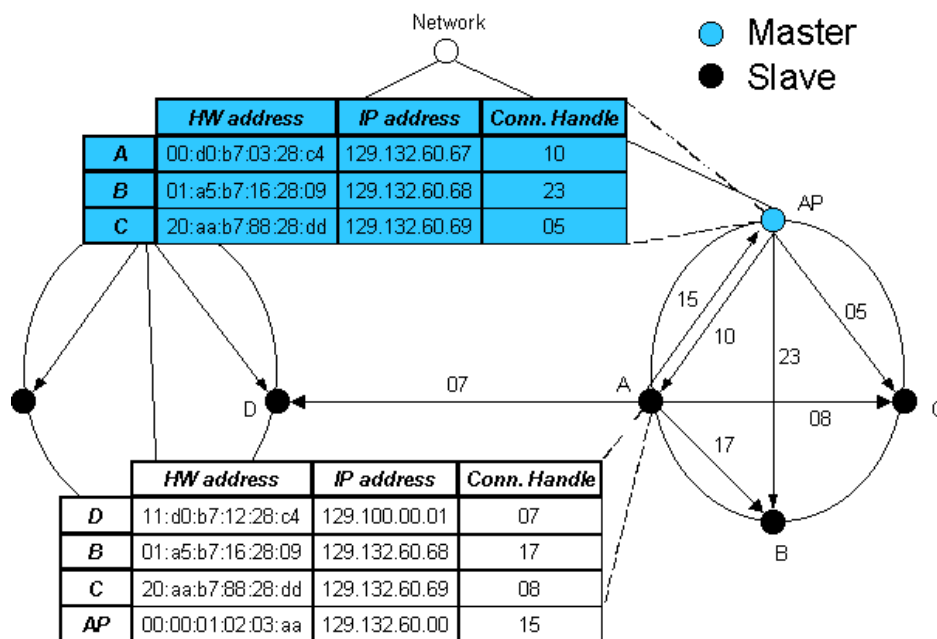


Figure 4.12: Bluetooth Device maintains a Lookup Table

As soon as a slave inside the AP piconet can become a slave or master in a new piconet, the schedule of the presence of the interpiconet nodes in their different piconets has to be considered as important. Given that the interpiconet node is a single transceiver unit, only one of its entities (master or slave) can be active at a time. To efficiently manage scatternet traffic, the

intrapiconet scheduler (AP) must consider the interpiconet scheduler when it polls the slaves of a piconet. For instance, the intrapiconet scheduler in the AP might not schedule an interpiconet node when the latter is active in another piconet.

In summary, RedNose with the proposed sublayer between the IP and L2CAP layer provides

- a solution to keep the IP layer independent of the link layer technology.
- a broadcast segment-like interface to IP.
- functions interacting closely with the Bluetooth baseband during the establishment or tear-down of a Bluetooth-specific piconet. It also covers the migration between different Access Points. Slave-to-slave communication is supported by maintaining a lookup table with devices within radio range.

Part IV

Conclusion

Chapter 5

Conclusion

5.1 Summary and Contributions

In this thesis, we first introduce Bluetooth, IP networking and address configuration.

We then show the problems of networking for mobile devices without user interfaces, summarize and explain the acterCARD's problematics. We also propose some possible solutions for implementation.

We programme the visualization demonstration for the acterCARD. The software presents a possible usage of the acterCARD. Additionally we present the concept that may lead to a solution for acterCARD's interaction.

Furthermore, the AXIS Bluetooth stack was successfully ported to the ARM environment on the acterCARD.

Finally, we provide RedNose, a concept for IP over Bluetooth and make some suggestions for implementation.

Overall, we show a possible interaction implementation for mobile devices without a local user interface.

5.2 Future Work

The acterCARD is now ready for building new applications communicating over Bluetooth with the rest of the world using the Bluetooth Profiles. To build these applications may also be a part of future works. This thesis is about the theoretical framework for RedNose networking.

It would be interesting to see an implementation of the RedNose concept and to show emerging problems and possible solutions. The RedNose concept might lead to a solution for the acterCARD and be integrated by acter ag.

It could be worthwhile to investigate how service discovery protocols - such as Jini, and especially the Service Location Protocol - can be integrated into the acterCARD.

As seen in this thesis, service discovery, auto configuration, IPv6, and many other topics are possible tasks for future works. It is up to the successor to choose the appropriate task. Furthermore, it will be interesting to see what RedNose becomes in a few years. As actor quotes:

Shoot for the moon. Even if you miss, you'll land among the stars.

Les Brown

Part V
Appendix

Appendix A

Software

A.1 Software Sources

Please refer to the CDROM.

A.2 VMware

```
#!/usr/local/bin/vmware
config.version = 2

# CD-ROM
ide1:0.present = TRUE
ide1:0.fileName = /dev/cdrom
ide1:0.deviceType = atapi-cdrom
ide1:0.startConnected = TRUE

# Virtual hard disk on primary master
ide0:0.present = TRUE
ide0:0.fileName = /root/vmware/linux1/linux.dsk
ide0:0.deviceType = ata-hardDisk
ide0:0.mode = persistent

# Floppy
floppy0.present = TRUE
floppy0.fileName = /dev/fd0
floppy0.startConnected = TRUE

# Networking bridged to real ethernet
ethernet0.present = TRUE
ethernet0.connectionType = bridged

# Networked to host only subnet
ethernet1.present = TRUE
ethernet1.connectionType = hostOnly

# Memory size
memsize = 16

# Nvram
nvram = /root/vmware/linux1/linux.nvram
```

```
# Log file
log.fileName = /root/vmware/linux1/linux.log

# Hints
guestOS = linux

# added by me
# -oOo-
ethernet0.address = 00:50:56:33:22:11

serial0.present = TRUE
serial0.fileName = "/dev/ttyS0"

tools.remindInstall = TRUE
```

A.3 Kernel Configuration File

```
#
# Automatically generated by make menuconfig: don't edit
#
CONFIG_ARM=y
CONFIG_UCLINUX=y

#
# Code maturity level options
#
CONFIG_EXPERIMENTAL=y

#
# Loadable module support
#
CONFIG_MODULES=y

#
# General setup
#
CONFIG_ARCH_TRIO=y
CONFIG_AT91=y
CONFIG_ACTER=y
CONFIG_CPU_ARM7=y
CONFIG_BINUTILS_NEW=y
CONFIG_NET=y
CONFIG_REDUCED_MEMORY=y
CONFIG_BINFMT_FLAT=y
CONFIG_KERNEL_ELF=y

#
# Floppy, IDE, and other block devices
#
CONFIG_BLK_DEV_BLKMEM=y

#
# Networking options
#
```

```
CONFIG_INET=y
CONFIG_IP_FORWARD=y

#
# SCSI support
#

#
# Network device support
#
CONFIG_NETDEVICES=y
CONFIG_BT_AXIS=y|n
CONFIG_PPP=y

#
# Filesystems
#
CONFIG_PROC_FS=y
CONFIG_ROMFS_FS=y
CONFIG_JFFS=y

#
# Character devices
#
CONFIG_SERIAL_ECHO=y
CONFIG_SERIAL_TRIO=y
CONFIG_UMISC=y
CONFIG_TRIO_PORTS=y

#
# Sound
#

#
# Kernel hacking
#
CONFIG_READA_SMALL=y
```

A.4 Makefile Module

```
#
# Makefile for the Bluetooth device driver.
#

CROSS_COMPILE = arm-uclinux-

LD      =$(CROSS_COMPILE)ld
CC      =$(CROSS_COMPILE)gcc
CPP     =$(CROSS_COMPILE)cpp
AR      =$(CROSS_COMPILE)ar
RANLIB  =$(CROSS_COMPILE)ranlib

PROGS := bt.o
OBSJS = bluetooth.o btdebug.o btmem.o hci.o l2cap.o rfcomm.o sdp.o tcs.o bt_proc.o
M_OBSJS = $(PROGS)
```

```

MODFLAGS --DMODVERSIONS -include $(DEVTREE_ROOT)/../linux/include/linux/modversions.h
CFLAGS   =-D__KERNEL__ -DMODULE -I../include -I$(DEVTREE_ROOT)/../linux/include \
          -Wstrict-prototypes -fno-strength-reduce $(MODFLAGS)
CFLAGS   +=-O2 -Wall -g \
          -mcpu=arm7tdmi -msoft-float -fno-exceptions \
          -fno-rtti -symbolic -shared -nostartfiles

LDLFLAGS +=-g -O2 -fno-exceptions -fno-rtti -symbolic -nostartfiles -nostartfiles

all:    $(PROGS)

$(PROGS):    $(OBJS)
             $(LD) $(LDLFLAGS) -r -o $@ $^
             chmod 644 $(PROGS)

clean:
        rm -f $(OBJS) $(PROGS)

```

A.5 Makefile User Program

```

#
# Makefile for user program btd
#

# $Id: Makefile,v 1.3 2000/11/14 23:21:47 gmcnut Exp $

ifdef APPS
USE_UCLIBC = 1
include $(APPS)/Rules.elinux
endif

PROGS      = btd
OBJS       = btd.o
INSTDIR    = $(prefix)/bin/
INSTMODE   = 0755
INSTOWNER  = root
INSTGROUP  = root

#LDLIBS += -lreadline -ltermcap
#CFLAGS += -I$(prefix)/include

CROSS_COMPILE = arm-uclinux-

LD      =$(CROSS_COMPILE)gcc
CC      =$(CROSS_COMPILE)gcc
CPP     =$(CROSS_COMPILE)cpp
AR      =$(CROSS_COMPILE)ar
RANLIB  =$(CROSS_COMPILE)ranlib

INC=-I$(DEVTREE_ROOT)/include \

```

```

-I$(DEVTREE_ROOT)/../linux/include \
-I$(DEVTREE_ROOT)/lib/uC-libc/include \
-I$(DEVTREE_ROOT)/lib/uC-libnet \
-I$(DEVTREE_ROOT)/lib/uC-libresolv \
-I/usr/local/lib/gcc-lib/arm-uclinux/2.96/include

LDLIBS+=-lc -lgcc -lnet -lresolv

LDLDFLAGS+=-fno-exceptions -fno-rtti \
-mdisable-got \
-symbolic -shared -nostartfiles -nostartfiles \
-Wl,-m -Wl,armelf_uclinux \
-L$(DEVTREE_ROOT)/lib/uC-libc \
-L$(DEVTREE_ROOT)/lib/uC-libnet \
-L$(DEVTREE_ROOT)/lib/uC-libresolv \
-L/usr/local/lib/gcc-lib/arm-uclinux/2.96 \
$(DEVTREE_ROOT)/lib/uC-libc/sysdeps/machine/crt0.o

#CFLAGS+=-mdisable-got -mcpu=arm7tdmi -msoft-float -fno-exceptions

CFLAGS+=-Wall -g \
-nostdinc \
-mdisable-got \
-mcpu=arm7tdmi -msoft-float -fno-exceptions \
-fno-rtti -symbolic -shared -nostartfiles $(INC) $(DEFS)

all: $(PROGS)

$(PROGS): $(OBJS)
$(LD) $(LDLDFLAGS) $^ -o $@.elf $(LDLIBS)
elf2flt -o $@.flat $@.elf
fltzip $@.flat $@
chmod 755 $@

install:
$(INSTALL) -d $(INSTDIR)
$(INSTALL) -m $(INSTMODE) -o $(INSTOWNER) -g $(INSTGROUP) $(PROGS) $(INSTDIR)

clean:
rm -f $(PROGS) $(OBJS) core

```

A.6 Data Between Stack and Ericsson Module

Ericsson ROK 101007, Bluetooth Stack Version 1115

```

*** READ_BUFFER_SIZE

BT DATA <--|X|      4
0x01 0x05 0x10 0x00

BT DATA -->|X|     14

```

```

0x04 0x0e 0x0b 0x01 0x05 0x10 0x00 0x20 0x03 0x00 0x0a 0x00 0x00 0x00

*** MANUFACTURER_SPEC (Ericsson HW revision info)

BT DATA <--|X|      4
0x01 0x0f 0xfc 0x00

BT DATA -->|X|     88
0x04 0x0e 0x69 0x01 0x0f 0xfc 0x00 0x0d 0x0a 0x20 0x47 0x65 0x6e 0x65 0x72 0x61
0x74 0x65 0x64 0x3a 0x20 0x32 0x30 0x30 0x30 0x2d 0x30 0x34 0x2d 0x32 0x38 0x20
0x31 0x35 0x3a 0x35 0x34 0x0d 0x0a 0x20 0x43 0x6f 0x6d 0x6d 0x65 0x6e 0x74 0x3a
0x20 0x43 0x58 0x43 0x20 0x31 0x32 0x35 0x20 0x32 0x34 0x34 0x20 0x50 0x39 0x41
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

*** WRITE_CLASS_OF_DEVICE

BT DATA <--|X|      7
0x01 0x24 0x0c 0x03 0x00 0x43 0x00

BT DATA -->|X|      7
0x04 0x0e 0x04 0x01 0x24 0x0c 0x00

*** READ_BD_ADDR

BT DATA <--|X|      4
0x01 0x09 0x10 0x00

BT DATA -->|X|     13
0x04 0x0e 0x0a 0x01 0x09 0x10 0x00 0x11 0x17 0x03 0xb7 0xd0 0x00

*** CHANGE_LOCAL_NAME

BT DATA <--|X|     252
0x01 0x13 0x0c 0xf8 0x64 0x65 0x76 0x31 0x2e 0x61 0x63 0x74 0x65 0x72 0x2e 0x63
0x68 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00

BT DATA -->|X|      7
0x04 0x0e 0x04 0x01 0x13 0x0c 0x00

*** WRITE_SCAN_ENABLE

BT DATA <--|X|      5
0x01 0x1a 0x0c 0x01 0x03

BT DATA -->|X|      7
0x04 0x0e 0x04 0x01 0x1a 0x0c 0x00

*** MANUFACTURER_SPEC (set_ericsson_baudrate 57600 baud)

BT DATA <--|X|      5
0x01 0x09 0xfc 0x01 0x03

BT DATA -->|X|      7
0x04 0x0e 0x04 0x01 0x09 0xfc 0x00

```

A.7 Data Between Stack and CSR Module

CASIRA BC01b, Bluetooth Stack Version 1115

*** READ_BUFFER_SIZE

BT DATA <--|X| 4
0x01 0x05 0x10 0x00

BT DATA -->|X| 14
0x04 0x0e 0x0b 0x01 0x05 0x10 0x00 0x80 0x00 0x40 0x08 0x00 0x08 0x00

*** WRITE_CLASS_OF_DEVICE

BT DATA <--|X| 7
0x01 0x24 0x0c 0x03 0x00 0x43 0x00

BT DATA -->|X| 7
0x04 0x0e 0x04 0x01 0x24 0x0c 0x00

*** READ_BD_ADDR

BT DATA <--|X| 4
0x01 0x09 0x10 0x00

BT DATA -->|X| 13
0x04 0x0e 0x0a 0x01 0x09 0x10 0x00 0x00 0x00 0x00 0x5b 0x02 0x00

*** CHANGE_LOCAL_NAME

BT DATA <--|X| 252
0x01 0x13 0x0c 0xf8 0x64 0x65 0x76 0x31 0x2e 0x61 0x63 0x74 0x65 0x72 0x2e 0x63
0x68 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

BT DATA -->|X| 7
0x04 0x0e 0x04 0x01 0x13 0x0c 0x00

*** WRITE_SCAN_ENABLE

BT DATA <--|X| 5
0x01 0x1a 0x0c 0x01 0x03

BT DATA -->|X| 7
0x04 0x0e 0x04 0x01 0x1a 0x0c 0x00

*** WRITE_PAGESCAN_ACTIVITY

BT DATA <--|X| 8
0x01 0x1c 0x0c 0x04 0x50 0x00 0x20 0x00

BT DATA -->|X| 7
0x04 0x0e 0x04 0x01 0x1c 0x0c 0x00

Appendix B

Hardware

B.1 CSR Radio Module Test Report

Radio Module Test Report

Radio Module Serial Number: 11530

Stack Loaded to Flash: PASS
UART Communications check: PASS
Serial Number written to persistent store: PASS

Crystal Tuning test: PASS
Crystal Tuning value: 23
Crystal Tuning Frequency: 2.499999e+005

Crystal Low frequency TEST: PASS
Crystal Low frequency: 2.500122e+005
Crystal high frequency TEST: PASS
Crystal high frequency: 2.499834e+005

BIST function test results

VCO Trim LUT 1 monotonic: PASS
VCO Trim minimum value test 1: PASS Value: 147
VCO Trim maximum value test 1: PASS Value: 178
VCO Trim gradient (difference) test 1: PASS Value: 31

VCO Trim LUT 2 monotonic: PASS
VCO Trim minimum value test 2: PASS Value: 148
VCO Trim maximum value test 2: PASS Value: 177
VCO Trim gradient (difference) test 2: PASS Value: 29

RF IQ Match: Not Tested
IF IQ Match: Not Tested

IF Response Bandwidth test: PASS Value: 1.191406e+000
IF Response centre frequency test: PASS Value: 1.489258e+000

```

IF Response centre RSSI test:                PASS    Value: 59
IF Response stopband LL2 test in MHz:       PASS    Value: 5.361328e-001
IF Response stopband UL2 test in MHz:       PASS    Value: 2.323242e+000

```

```

Synth settle response test:    PASS
Synth settle deviation test:   PASS    2
Synth final settle value:     76

```

RF test results

```

Transmit Current test:  PASS    Value: 2.671022e-001
Receive Current test:  PASS    Value: 1.507195e-001

```

```

RSSI value at 2.402GHz and -85dbm test:  PASS    Value: 44
RSSI Noise at 2.402GHz test:              PASS    Value: 4

```

```

RSSI value at 2.432GHz and -85dbm test:  PASS    Value: 44
RSSI Noise at 2.432GHz test:              PASS    Value: 5

```

```

RSSI value at 2.480GHz and -85dbm test:  PASS    Value: 41
RSSI Noise at 2.480GHz test:              PASS    Value: 5

```

```

Reference RSSI value at 2.432GHz and -65dbm test:  PASS    Value: 91
Reference RSSI value at 2.432GHz and -80dbm test:  PASS    Value: 60
Image test (wanted signal / unwanted image):        PASS    Value: 75

```

```

TX power at 2.402GHz:  PASS    Value: 2.112000e+001
TX power at 2.432GHz:  PASS    Value: 2.071000e+001
TX power at 2.480GHz:  PASS    Value: 1.981000e+001

```

BER test results

```

Bit Error Rate at -75 dBm test:  PASS    Value: 0

```

Persistent store values
All values below in Hex

```

Local Name:                CASIRA Bluetooth development kit
Bluetooth address written:  PASS    Value: FF AD2 5B 2
Module ID written:         Value: 2DOA
Module Manufacturer ID:    PASS    Value: 1
Module Design ID:          PASS    Value: 1
Local Name Length Value written:  PASS    Value: 21
Marketing Task Value written:  PASS    Value: 0
BCCMD security Value NOT written:  PASS    Value: 0
Crystal Tuning Value written:  PASS    Value: 17
Higher RSSI Value written:  PASS    Value: 5B
Lower RSSI Value written:    PASS    Value: 3C
IQ Trim Value written:      PASS    Value: CD

```

Radio Module Test Report

```

Radio Module Serial Number: 11603

```

Stack Loaded to Flash: PASS
 UART Communications check: PASS
 Serial Number written to persistent store: PASS

 Crystal Tuning test: PASS
 Crystal Tuning value: 21
 Crystal Tuning Frequency: 2.499998e+005
 Crystal Low frequency TEST: PASS
 Crystal Low frequency: 2.500114e+005
 Crystal high frequency TEST: PASS
 Crystal high frequency: 2.499821e+005

BIST function test results

VCO Trim LUT 1 monotonic: PASS
 VCO Trim minimum value test 1: PASS Value: 143
 VCO Trim maximum value test 1: PASS Value: 174
 VCO Trim gradient (difference) test 1: PASS Value: 31
 VCO Trim LUT 2 monotonic: PASS
 VCO Trim minimum value test 2: PASS Value: 144
 VCO Trim maximum value test 2: PASS Value: 174
 VCO Trim gradient (difference) test 2: PASS Value: 30

RF IQ Match: Not Tested
 IF IQ Match: Not Tested

IF Response Bandwidth test: PASS Value: 1.191406e+000
 IF Response centre frequency test: PASS Value: 1.489258e+000
 IF Response centre RSSI test: PASS Value: 57
 IF Response stopband LL2 test in MHz: PASS Value: 5.957031e-001
 IF Response stopband UL2 test in MHz: PASS Value: 2.323242e+000

Synth settle response test: PASS
 Synth settle deviation test: PASS 2
 Synth final settle value: 76

RF test results

Transmit Current test: PASS Value: 2.684440e-001
 Receive Current test: PASS Value: 1.482668e-001

RSSI value at 2.402GHz and -85dbm test: PASS Value: 37
 RSSI Noise at 2.402GHz test: PASS Value: 5

RSSI value at 2.432GHz and -85dbm test: PASS Value: 37
 RSSI Noise at 2.432GHz test: PASS Value: 5

RSSI value at 2.480GHz and -85dbm test: PASS Value: 36
 RSSI Noise at 2.480GHz test: PASS Value: 5

Reference RSSI value at 2.432GHz and -65dbm test: PASS Value: 85
 Reference RSSI value at 2.432GHz and -80dbm test: PASS Value: 51
 Image test (wanted signal / unwanted image): PASS Value: 73

TX power at 2.402GHz: PASS Value: 2.048000e+001

TX power at 2.432GHz: PASS Value: 1.984000e+001
 TX power at 2.480GHz: PASS Value: 1.923000e+001

 BER test results

Bit Error Rate at -75 dBm test: PASS Value: 6.250000e-003

Persistent store values
 All values below in Hex

Local Name:		CASIRA Bluetooth development kit			
Bluetooth address written:	PASS	Value: FF	B1B	5B	2
Module ID written:		Value: 2D53			
Module Manufacturer ID:	PASS	Value: 1			
Module Design ID:		PASS	Value: 1		
Local Name Length Value written:	PASS	Value: 21			
Marketing Task Value written:	PASS	Value: 0			
BCCMD security Value NOT written:	PASS	Value: 0			
Crystal Tuning Value written:	PASS	Value: 15			
Higher RSSI Value written:	PASS	Value: 55			
Lower RSSI Value written:	PASS	Value: 33			
IQ Trim Value written:	PASS	Value: 1FA			

Appendix C

Glossary

further information see [\[45\]](#)(p. 891) , [\[53\]](#) and [\[38\]](#).

ACL link

Asynchronous Connectionless link. One of the two types of data links defined for the Bluetooth Systems, it is an asynchronous (packet-switched) connection between two devices created on the LMP level. This type of link is used primarily to transmit ACL packet data. The other data link type is SCO

AM_ADDR

Active Member ADDRESS. During the paging process, the master will assign an AM_ADDR to the slave. This will then form the connection handle used to address all communications to slave and for the master to differentiate between responses from different slaves.

AODV

Ad-hoc On Demand Distance Vector routing is capable of both unicast and multicast routing. It is an on demand algorithm, i.e. it builds routes between nodes only as desired by source nodes. It maintains these routes as long as they are needed by the sources. Additionally, AODV forms trees connecting multicast group members. The trees are composed of the group members and the nodes needed to connect the members. AODV uses sequence numbers to ensure the freshness of routes. It is loop-free, self-starting, and scales to large numbers of mobile nodes.

BT

Bluetooth (unofficial short form). An open specification for wireless communication of data and voice. It is based on a low-cost short-range radio link facilitating protected ad-hoc connections for stationary and mobile communication environments.

CGI

Common Gateway Interface

CSR

Cambridge Silicon Radio

FSM

Final State Machine

DHCP

Dynamic Host Configuration Protocol

GAL

Gate Array Logic

GPS

Global Positioning System

HCI

Host Controller Interface [45](p. 519)

HTTP

Hypertext Transfer Protocol (HTTP) is the set of rules for exchanging files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web. Relative to the TCW/IP suite of protocols (which are the basis for information exchange on the Internet), HTTP is an application protocol. Essential concepts that are part of HTTP include (as its name implies) the idea that files can contain references to other files whose selection will elicit additional transfer requests. Any Web server machine contains, in addition to the HTML and other files it can serve, an HTTP daemon, a program designed to wait for HTTP requests and handle them when they arrive. Your Web browser is a HTTP client, sending requests to server machines. When the browser user enters file requests by either 'opening' a Web file (typing in a Uniform Resource Locator) or clicking on a hypertext link, the browser builds a HTTP request and sends it to the Internet Protocol address indicated by the URL. The HTTP daemon in the destination server machine receives the request and, after any necessary processing, the requested file is returned. The latest version of HTTP is HTTP 1.1.

IETF

Internet Engineering Task Force

L2CAP

Logical Link Control And Adaptation Protocol [45](p. 247) . This protocol supports higher level protocol multiplexing, packet segmentation and reassembly, and the conveying of quality of service information.

LMP

Link Manager Protocol [45](p. 187) . The LMP is used for link setup and control. The LMP PDU signals are interpreted and filtered out by the Link Manager on the receiving side and are not propagated to higher layers.

MAODV

The Multicast Ad hoc On-Demand Distance Vector (MAODV) protocol enables dynamic, self-starting, multihop routing between participating mobile nodes wishing to join or participate in a multicast group within an ad hoc network. The membership of these multicast groups is free to change during the network's lifetime. MAODV enables mobile nodes to establish a tree connecting multicast group members. Mobile nodes are able to respond quickly to link breaks in multicast trees by repairing these breaks in a timely manner. In the event of a network partition, multicast trees are established independently in each partition, and trees for the same multicast group are quickly connected if the network components merge.

MANET

Mobile Ad-hoc NETworking

MTU

Maximum Transmission Unit

PCMCIA

The Personal Computer Memory Card International Association is an industry group organized in 1989 to promote standards for a creditcard-size memory or I/O device that would fit into a personal computer, usually a notebook or laptop computer.

Piconet

A collection of devices connected via Bluetooth technology in an ad-hoc fashion. A piconet starts with two connected devices such as a portable PC and cellular phone, and may grow to eight connected devices. All Bluetooth devices are peer units and have identical implementations. However, when establishing a piconet, one unit will act as a master and the other(s) as slave(s) for the duration of the piconet connection. All devices have the same physical channel defined by the master device parameters (clock and BD_ADDR).

PPP

Point-to-Point Protocol [41] is a protocol for communication between two computers using a serial interface, typically a personal computer connected by phone line to a server. For example, your Internet server provider may provide you with a PPP connection so that the provider's server can respond to your requests, pass them on to the Internet, and forward your requested Internet responses back to you. PPP uses the Internet protocol (IP) (and is designed to handle others). It is sometimes considered a member of the TCP/IP suite of protocols. Relative to the Open Systems Interconnection (OSI) reference model, PPP provides layer 2 (data-link layer) service. Essentially, it packages your computer's TCP/IP packets and forwards them to the server where they can actually be put on the Internet. PPP is a full-duplex protocol that can be used on various physical media, including twisted pair or fiber optic lines or satellite transmission. It uses a variation of High Speed Data Link Control (HDLC) for packet encapsulation. PPP is usually preferred over the earlier de facto standard Serial Line Internet Protocol (SLIP) because it can handle synchronous as well as asynchronous communication. PPP can share a line with other users and it has error detection that SLIP lacks. Where a choice is possible, PPP is preferred.

PSM

Protocol/Service Multiplexer [45](p. 1019)

RFCOMM

[45](p. 387) . Serial Cable Emulation Protocol based on ETSI TS 07.10.

RSA

Rivest-Shamir-Adleman [51]. RSA is an Internet encryption and authentication system that uses an algorithm developed in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman. The RSA algorithm is the most commonly used encryption and authentication algorithm. The encryption system is owned by RSA Security. The company licenses the algorithm technologies and also sells development kits. The technologies are part of existing or proposed Web, Internet, and computing standards.

SCO link

Synchronous Connection Oriented link. One of the 2 bluetooth data link types defined. A synchronous (circuit-switched) connection for reserved bandwidth communications (e.g. voice) between two devices created on the LMP level by periodically reserving slots on a physical channel. This type of link is primarily used to transport SCO packets (voice data). SCO packets do not include a CRC and are never retransmitted. It primarily supports time-bounded information like voice (Master to single slave). SCO links can only be established after an ACL link has been established. See also ACL.

SDP

Service Discovery Protocol [45](p. 325) . It is a Bluetooth defined protocol provided for or available through a Bluetooth device. It essentially provides a means for applications to discover available services and to determine their characteristics.

SIG

Special Interest Group

SSL

Secure Socket Layer is a commonly-used protocol to manage security of a message transmission on the Internet. SSL uses a program layer located between Internet's Hypertext Transfer Protocol (HTTP) and Transport Control Protocol (TCP) layers. SSL was developed by Netscape and is now supported by Microsoft as well as by other Internet client/server developers. The 'sockets' part of the term refers to the sockets method of passing data back and forth between a client and a server program in a network or between program layers in the same computer. SSL uses the RSA public-and-private key encryption system, which also includes the use of a digital certificate. SSL is an integral part of most Web browsers (clients) and Web servers. If a Web site is on a server that supports SSL, SSL can be enabled and specific Web pages can be identified as requiring SSL access. Currently a de facto standard, SSL has been submitted to the Internet Engineering Task Force (IETF) as a proposed official standard.

SLP

Service Location Protocol [43]

TCS

Telephone Control protocol Specification

TORA

Temporally-Ordered Routing Algorithm

Scatternet

Multiple independent and non-synchronized piconets form a scatternet.

Bibliography

- [1] μ C-Linux for ARM9TDMI
 μ c-Linux,
<http://aplionet.aplio.fr/uclinux/page3.htm> 41, 42
- [2] A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol
A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol, Christian Bettstetter and Christoph Renner, Technische Universität München
<http://citeseer.nj.nec.com/christoph00comparison.html> 21
- [3] acter ag
Website,
<http://www.acter.net/> iii
- [4] acter ag
acterCARD Datasheet,
[../docs/acterCARD.pdf](http://docs.acter.net/acterCARD.pdf) 3
- [5] acter ag
acter's Vision of Security, Tarkan Bas
Internal Document
- [6] acter ag
OS/RT for Internetworked High-end Smartcards I/II, George Fankhauser
Internal Document
- [7] Analysis of Throughput of Bluetooth Devices with IEEE 802.11 Interference Sources
Analysis of Throughput of Bluetooth Devices with IEEE 802.11 Interference Sources, Pablo M. Robert, April 2000. 11
- [8] AutoIP
IETF Draft,
<http://search.ietf.org/internet-drafts/draft-ietf-dhc-ipv4-autoconfig-07.txt> 21

-
- [9] AXIS Bluetooth Mailing List
Mailing List,
<http://mhonarc.axis.se/bluetooth-dev/maillist.html> 44
- [10] AXIS Bluetooth Stack
Bluetooth Stack,
<http://www.developer.axis.com> 31, 36
- [11] AXIS
AXIS,
<http://www.axis.com> iii, 29
- [12] Bluetooth Connect Without Cables
Bluetooth Connect Without Cables, Jennifer Bray, Charles F. Sturman,
Prentice-Hall. 1st Edition, 2001. 8
- [13] Bluetooth Revealed
Bluetooth Revealed, Brent A. Miller, Chatschik Bisdikian, Prentice-Hall.
1st Edition, 2001. 8
- [14] Bluetooth Security Architecture
protocol,
[../docs/bluetooth/security architecture.pdf](http://www.bluetooth.com/specifications/protocol-specifications) 32
- [15] Bluetooth Special Interest Group
SIG,
<http://www.bluetooth.com/sig/sig/sig.asp> 65
- [16] Cambridge Silicon Radio (CSR)
CASIRA Development Kit,
<http://www.csr.com> 33, 41
- [17] CVS
Documentation,
<http://www.loria.fr/cgi-bin/molli/wilma.cgi/doc>
<http://dsg.harvard.edu/public/misc/cvs.html> 44
- [18] Digianswer
Digianswer Bluetooth Demo Card,
<http://www.digianswer.com> 31
- [19] DMA - Drop a Message Anywhere
Bluetooth Home Networking - Implementation,
<http://www.tik.ee.ethz.ch/~dma/>
[../docs/SA_Implementation.pdf](http://www.tik.ee.ethz.ch/~dma/SA_Implementation.pdf) 32
- [20] DMA - Drop a Message Anywhere
Bluetooth Stack For Palm,
<http://www.tik.ee.ethz.ch/~dma/pallexport/> 45

-
- [21] Dynamic Host Configuration Protocol
RFC2131, RFC2132,
<http://www.landfield.com/rfcs/rfc2131.html>
<http://www.landfield.com/rfcs/rfc2132.html> 20, 61
- [22] Ericsson
Specific HCI-Commands,
.../Ericsson_HCL_microe.pdf 6
- [23] Ericsson
Bluetooth Application Tool Kit,
<http://bluetooth.ericsson.se> 33, 41
- [24] Ethernet Address Resolution Protocol
RFC826,
<http://www.landfield.com/rfcs/rfc826.html> 72
- [25] ETSI Telecom Standards
07.10 specification,
<http://www.etsi.org> 6
- [26] GNU License
GNU,
<http://www.linux.org/info/gnu.html> 29
- [27] Guidelines for 64-bit Global Identifier (EUI-64) Registration Authority
IEEE,
<http://standards.ieee.org/regauth/oui/tutorials/EUI64.html> 73
- [28] IEEE
862.15 WPAN Task Group 1 (TG1),
<http://www.ieee802.org/16/pub/TG1.html>
- [29] Internet Engineering Task Force
IETF,
<http://www.ietf.org> 64
- [30] IP Version 6 Addressing Architecture
RFC2373,
<http://www.landfield.com/rfcs/rfc2373.html> 73
- [31] IPsec
IPsec Web-Page,
<http://www.cs.umass.edu/lmccarth/ipsec.html>
- [32] Linux Kernel Archives
Kernel.org,
<http://www.kernel.org/> 33, 41

-
- [33] Mandrake
Linux-Mandrake 7.2,
<http://www.linux-mandrake.com/en/>
- [34] Microsoft
Bluetooth ESDP for UPnP, Arun Ayyagari
[../docs/ESDP_UPnP_0_95a.pdf](http://www.microsoft.com/press/presskit/2002/022002/022002_095a.pdf) 16
- [35] Mobile Ad-hoc Network
RFC2501,
<http://www.landfield.com/rfc/rfc2501.html> 19
- [36] Mobile IP
RFC2002, RFC2344,
<http://www.landfield.com/rfc/rfc2002.html>
<http://www.landfield.com/rfc/rfc2344.html> 20, 61
- [37] Netscape Communication Corporation
SSL 3.0 Specification,
<http://home.netscape.com/eng/ssl3/>
- [38] palowireless.com
Bluetooth Resource Centre,
<http://www.padowireless.com/infotooth/> 93
- [39] palowireless.com
Bluetooth Security, Cathal Mu Daid
http://www.palowireless.com/bluearticles/cc1_security1.asp
- [40] Perl
Perl 5.6.0,
<http://www.perl.com/pub/>
- [41] Point-to-Point Protocol
RFC1661,
<http://www.landfield.com/rfc/rfc1661.html> 64, 96
- [42] RedHat
RedHat Linux 7,
<http://www.redhat.com/>
- [43] Service Location Protocol
SLP White Paper,
http://playground.sun.com/srvloc/slp_white_paper.html 24, 97
- [44] Specification of the Bluetooth Protocol Architecture
protocol,
<http://www.bluetooth.com/.../download/download.asp?doc=175> 5

-
- [45] Specification of the Bluetooth System
Core,
http://www.bluetooth.com/developer/specification/core_10_b.pdf 5,
24, 29, 33, 37, 59, 64, 67, 93, 94, 96, 97
- [46] Specification of the Bluetooth System
Profiles,
http://www.bluetooth.com/developer/specification/profile_10_b.pdf
64
- [47] SuSE
SuSE Linux 7.1,
<http://www.suse.de/> 33, 41
- [48] Swiss Federal Institute of Technology Zurich
Swiss Federal Institute of Technology Zurich,
<http://www.ethz.ch> iii
- [49] Swiss Federal Institute of Technology Zurich
Computer Engineering and Networks Laboratory,
<http://www.tik.ee.ethz.ch/> iii
- [50] Universal Serial Bus
USB,
<http://www.usb.org> 33
- [51] VeriSign Inc.
RSA,
<http://www.verisign.com/> 96
- [52] VMware
Virtual machines,
<http://www.vmware.com/> 33
- [53] whatis?com
IT-specific encyclopedia,
<http://www.whatis.com/> 93

Index

- ACL, [66](#)
- ACL link, [93](#)
- acter Module, [51](#)
- Ad-hoc network, [17](#)
- Address Configuration, [20](#)
- AM_ADDR, [93](#)
- AODV, [93](#)
- ARM patch, [44](#)
- ARM Port, [41](#)
- ARM_UART, [41](#)
- ARP, [72](#)
- AutoIP, [21](#)
- AXIS, [29](#)

- Bluetooth
 - Device Address, [12](#)
 - ESDS, [16](#)
 - Pairing, [12](#)
 - PAN Profile, [16](#)
 - Physical Layer, [8](#)
 - Piconet Performance, [9](#)
 - Profiles, [14](#)
- Bluetooth Networking, [6](#)
- Bluetooth Overview, [5](#)
- Bluetooth SDP, [24](#)
- BNEP, [65](#)
- Broadcast, [60](#)
- Broadcast_Flag, [60](#)
- BT, [93](#)
- bt.o, [45](#)
- btd, [32](#)
- BTD_USERSTACK, [32](#)
- btduser, [32](#)

- CGI, [93](#)
- Communication Concept, [5](#)

- Comparison, [25](#)
- Concept
 - Address Configuration, [20](#)
 - Bluetooth Networking, [6](#)
 - Communication Concept, [5](#)
 - IP Networking Model, [17](#)
 - RedNose, [25](#)
 - Service Discovery, [21](#)
- Conclusions
 - Future Work, [77](#)
 - Summary and Contributions, [77](#)
- CONFIG_REDUCED_MEMORY, [48](#)
- CSR, [94](#)
- CSR Module, [38](#)
- CVS, [42](#)

- Define
 - BTD_USERSTACK, [32](#)
 - CONFIG_REDUCED_MEMORY, [48](#)
 - HCLEMULATION, [31](#)
 - USE_GETOPT_LONG, [47](#)
 - USE_READLINE, [46](#)
- Development Environment ARM, [41](#)
- Development Environment PC, [33](#)
- Device Address, [12](#)
- DHCP, [20](#), [94](#)
- Digianswer, [31](#)
- Driver
 - UART, [34](#)
 - USB, [34](#)

- elf2flt, [41](#)

- End-to-end IP Network, 60
- eric, 49
- Ericsson Module, 33
- ESDS, 16
- EUI-64, 72

- FHSS, 71
- ftzip, 41
- FSM, 94
- Future Work, 77

- GAL, 94
- genromfs, 41
- Glossary
 - ACL link, 93
 - AM_ADDR, 93
 - AODV, 93
 - BT, 93
 - CGI, 93
 - CSR, 94
 - DHCP, 94
 - FSM, 94
 - GAL, 94
 - GPS, 94
 - HCI, 94
 - HTTP, 94
 - IETF, 94
 - L2CAP, 94
 - LMP, 94
 - MANET, 95
 - MAODV, 95
 - MTU, 95
 - PCMCIA, 95
 - Piconet, 95
 - PPP, 96
 - PSM, 96
 - RFCOMM, 96
 - RSA, 96
 - Scatternet, 97
 - SCO link, 97
 - SDP, 97
 - SIG, 97
 - SLP, 97
 - SSL, 97
 - TCS, 97
 - TORA, 97
- GPS, 94

- HCI, 62, 94
- HCLEMULATION, 31
- hello, 48
- HTTP, 94

- ICMP, 72
- IETF, 94
- Inquiry Time, 71
- Interference Analysis, 71
- IP Networking Model, 17
 - Ad-hoc network, 17
 - DHCP, 20
 - Routing, 18
- IP over ACL, 66
- IP over Bluetooth
 - IP over ACL, 66
 - IP over L2CAP, 67
 - IP over PPP over L2CAP, 68
 - IP over SCO, 65
 - LAP, 64
 - PAN, 65
- IP over L2CAP, 67
- IP over PPP over L2CAP, 68
- IP over SCO, 65
- IPv6, 60, 72

- Jini, 22

- Kernel 2.2.x, 34
- Kernel 2.4.x, 36
- Kernel Mode, 32
- Kit
 - acter Module, 51
 - CSR Module, 38
 - Ericsson Module, 33

- L2CAP, 62, 67, 94
- LAP, 64
- LMP, 94
- Lookup Table, 71

- MANET, 19, 95

- MAODV, 95
- Mobile IP, 20, 62
- Module
 - bt.o, 45
- MTU, 95
- Multicast, 60

- Object File Size, 49

- Paging Time, 71
- Pairing, 12
- PAN, 65
- PAN Profile, 16
- PCMCIA, 31, 95
- Physical Layer, 8
- Piconet, 58, 95
- Piconet Performance, 9
- PPP, 96
- PPP Server, 64
- Profiles, 14, 32
- PSM, 96
- PSTool, 38

- QoS, 60

- RedNose, 25
- RFCOMM, 96
- Routing, 18
- RSA, 96

- Salutation, 23
- sane, 49
- Scatternet, 58, 97
- SCO, 65
- SCO link, 97
- SDP, 97
- sdp_server, 32, 33
- sdp_user, 32, 33
- sertest, 48
- Service Discovery, 21
 - Bluetooth SDP, 24
 - Comparison, 25
 - Jini, 22
 - Salutation, 23
 - SLP, 23
 - UPnP, 22
- SIG, 97
- SLP, 23, 97
- Source Tree, 42
- SSL, 97
- Stack on ARM, 50
- Stack on PC, 33
- Stack Overview, 31
- Summary and Contributions, 77

- TCS, 97
- TDM, 58, 69, 71
- Tools
 - elf2flt, 41
 - eric, 49
 - ftzip, 41
 - genromfs, 41
 - hello, 48
 - sane, 49
 - sertest, 48
- TORA, 97

- UART, 34
- UART Settings, 38
- UPnP, 22
- USB, 34
- USE_GETOPT_LONG, 47
- USE_READLINE, 46
- User Mode, 32

- VMware, 33, 41