UNIVERSIDADE ESTADUAL DE GOIÁS
UNIDADE UNIVERSITÁRIA DE ITABERAI
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

FRANK FERREIRA DA SILVA
CLÁUDIO MENDANHA

BLUETOOTH

Itaberaí-GO
2010

UNIVERSIDADE ESTADUAL DE GOIÁS
UNIDADE UNIVERSITÁRIA DE ITABERAI
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

FRANK FERREIRA DA SILVA
CLÁUDIO MENDANHA

BLUETOOTH

Monografia apresentada à Unidade de Itaberaí da Universidade Estadual de Goiás, como requisito parcial à conclusão do Curso Superior de Sistemas de Informação, sob o título: “Bluetooth”.

Orientador: Professora Eliane

Itaberaí-GO
2010

UNIVERSIDADE ESTADUAL DE GOIÁS
UNIDADE UNIVERSITÁRIA DE ITABERAI
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Bluetooth

Monografia apresentada à Unidade de Itaberaí da Universidade Estadual de Goiás, como requisito parcial à conclusão do Curso Superior de Sistemas de Informação, sob o título: "Bluetooth".

Aprovado por:

Professora Eliane Cristina de Lima, UEG.
(ORIENTADOR)

Professora Lauriana Alves Moreira, UEG.
(EXAMINADOR)

Professora Walkíria Nascente Valle, UEG.
(EXAMINADOR)

Itaberaí, de 2010.

FICHA CATALOGRÁFICA

FERREIRA, Frank Silva, MENDANHA, Cláudio.
Bluetooth. [Itaberaí] 2010.
(UEG / UnU Itaberaí, Bacharelado em Sistemas de Informação, 2010).

Monografia, Universidade Estadual de Goiás, Unidade de Itaberaí.

REFERÊNCIA BIBLIOGRÁFICA

Ferreira, Frank Silva e Mendanha, Cláudio. **BLUETOOTH**. Itaberaí, 2010, Monografia – Curso de Sistemas de Informação, UnU Itaberaí, Universidade Estadual de Goiás.

CESSÃO DE DIREITOS

NOME DOS AUTORES: Ferreira, Frank Silva e Mendanha, Cláudio.
TÍTULO DO TRABALHO: BLUETOOTH.
GRAU/ANO: Graduação /4º Ano.

Frank Ferreira da Silva
Endereço: Rua Rio Verde, Vila Marize

CEP: 75400-000– Inhumas/GO – Brasil

Cláudio Mendanha
Endereço: Rua Ricardo A. Balestra Vila Heitor de Paula.

CEP: 75400-000– Inhumas/GO – Brasil

Dedicamos a presente monografia aos nossos pais, que sempre nos educaram, incentivaram e tantas outras coisas que serviram como exemplo para que tornássemos pessoas melhores e a nossa orientadora sem a qual não teríamos conseguido a conclusão desse trabalho.

“Aprende que não importa em quantos pedaços seu coração foi partido, o mundo não pára para que você o conserte. Aprende que o tempo não é algo que possa voltar. Portanto, plante seu jardim e decorre sua alma, em vez de esperar que alguém lhe traga flores.”

(William Shakespeare)

LISTA DE ABREVIATURAS

Siglas	Descrição
8PSK	Phase Shift Keying com 8 níveis
ACL	Asynchronous Connection-Less
DUN	Dial-up Networking Profile
EDR	Enhanced Data Rate
FHS	Frequency Hopping-Synchronization
FH-CDMA	Frequency Hopping - Code-Division Multiple Access
FH/TDD	Frequency Hopping/Time-Division Duplex
FTP	File Transfer Profile
GSFK	Gaussian Frequency Shift Keying
HID	Human Interface Device Profile
HSP	Headset Profile
ISM	Industrial, Scientific, Medical
L2CAP	Logical Link Control and Adaptation Protocol
LMP	Link Manager Protocol
OBEX	Object Exchange Protocol
OPP	Object Push Profile
PAN	Private Area Network
PPP	Point-to-Point Protocol
RFCOMM	Radio frequency communications
SCO	Synchronous Connection-Oriented
SDP	Service Discovery Protocol
SIG	Special Interest Group
TCS BIN	Telephony control protocol-binary
WAE	Wireless Application Environment
WAP	Wireless Application Protocol

RESUMO

Essa monografia tem o intuito de mostrar a tecnologia BLUETOOTH de um modo acessível e de fácil entendimento mostrando sua historia e ressaltando seus pontos fortes e fracos, vantagens e desvantagens, sua implementação e funcionamento utilizando de embasamento teórico e conhecimento técnico. Foi utilizado de pesquisa e estudo de sites e monografias sobre o assunto para a aquisição de conhecimento para a formulação deste trabalho, fazendo um levantamento na historia da tecnologia e suas versões, fazendo uma explanação sobre a tecnologia de um modo geral, aprofundando no seu funcionamento e recursos utilizados, esclarecendo suas finalidades e utilizações atuais e futuras, fazendo levantamento de seus pontos fortes e fracos, levantando necessidades para sua implementação.

Palavras-chave: BLUETOOTH, tecnologia, funcionamento, pontos fortes e fracos, desenvolvimento, implementação.

ABSTRACT

This monograph aims to show the BLUETOOTH technology in a manner accessible and understandable showing its history and highlighting their strengths and weaknesses, advantages and disadvantages, its implementation and operation using the theoretical and technical knowledge. Was used for research and study sites and monographs on the subject to acquire knowledge for the formulation of this work, making a survey in the history of technology and their versions, making an explanation of the technology in general, approved funding the their operation and resources used, clarifying their purposes and current uses and future, like lifting their strengths and weaknesses, raising needs for its implementation.

Keywords: BLUETOOTH, technology, operation, strengths and weaknesses, development, implementation.

LISTA DE FIGURA

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

As comunicações em redes são outras depois das novas tecnologias *Wireless*, indo desde o nível individual até o corporativo. A utilização de tecnologias *Wireless* e o padrão de banda larga móvel são cada vez maiores, aumentando as possibilidades de formas de conexão.

Recentemente uma tecnologia *Wireless* tem ganhado espaço pela capacidade de permitir que equipamentos eletrônicos dos mais diversos tipos como computadores pessoais, modens, PDA's, impressoras, telefones celulares, etc., se comuniquem em curtas distâncias sem a utilização de algum tipo de cabeamento.

Essa é a tecnologia Bluetooth, uma especificação aberta (*royalty free*) para comunicação de curto alcance, sem fio e baixo custo que usa de radiofrequência para efetuar as conexões, e que tem por principal objetivo facilitar a vida de seus usuários possibilitando as transmissões de voz e dados em tempo real, e que assegura proteção contra interferências e fornece segurança para os dados transmitidos.

2. VISÃO GERAL DE BLUETOOTH

Bluetooth é um padrão de comunicação *Wireless* de curto alcance, baixo custo e baixo consumo de energia que utiliza tecnologia de radiofrequência. Foi criado pela Ericsson (a maior fabricante de celulares, hoje Sony-Ericsson *Corporation*) em 1994 pensando-se que seria uma forma de substituição para, o Bluetooth tem sido usado de forma ampla em diversos dispositivos o que o torna uma parcela significativa do mercado *wireless*. Os dispositivos que utilizam Bluetooth são dispositivos inteligentes como PDAs, telefones celulares, PCs,

2.1 HISTÓRIA

A história do Bluetooth começa em meados de 1994. Na época, a empresa Ericsson começou a estudar a viabilidade de desenvolver uma tecnologia que permitisse a comunicação entre telefones celulares e acessórios utilizando sinais de rádio de baixo custo, ao invés dos tradicionais cabos. O estudo era feito com base em um projeto que investigava o uso de mecanismos de comunicação em redes de telefones celulares, que resultou em um sistema de rádio de curto alcance que recebeu o nome *MCLink*. Com a evolução do projeto, a Ericsson percebeu que o *MCLink* poderia dar certo, já que o seu principal atrativo era uma implementação relativamente fácil e barata.

Em 1997, o projeto começou a despertar o interesse de outras empresas que, logo, passaram a fornecer apoio. Por conta disso, em 1998 foi criado o consórcio Bluetooth SIG (*Special Interest Group*), formado pelas empresas Ericsson, Intel, IBM, Toshiba e Nokia. Note que esse grupo é composto por dois "gigantes" das telecomunicações (Ericsson e Nokia), dois nomes de peso na fabricação de PCs (IBM e Toshiba) e a líder no desenvolvimento de chips e processadores (Intel). Essa diversidade foi utilizada para permitir o desenvolvimento de padrões que garantissem o uso e a interoperabilidade da tecnologia nos mais variados dispositivos.

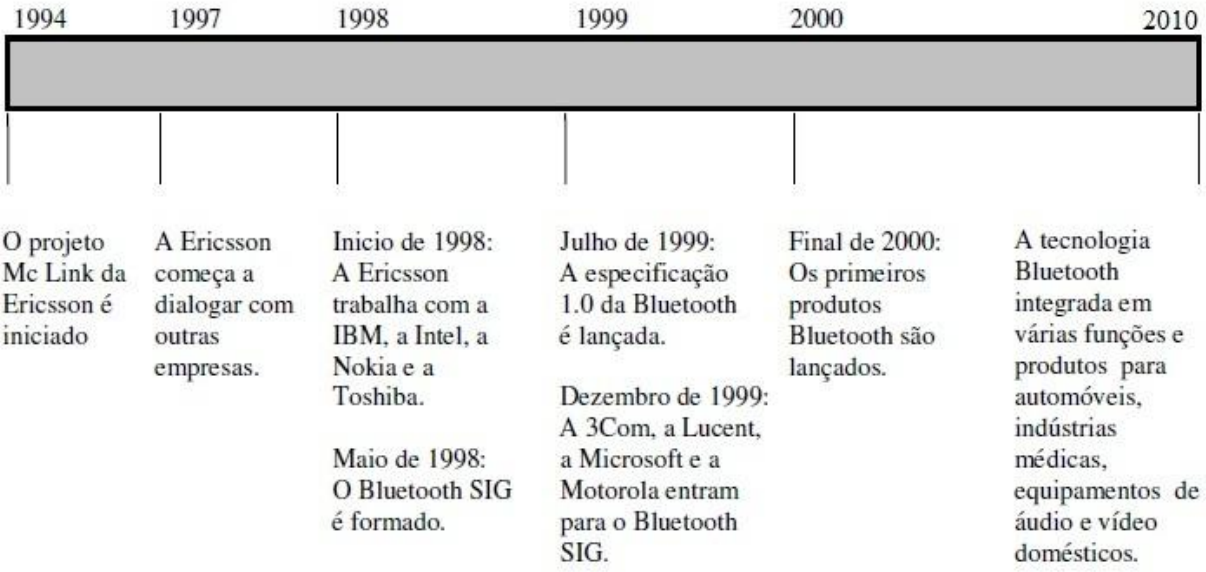

Figura 1 – Cronograma Bluetooth

[**PEREIRA**, Magno C. Artigo Bluetooth. Acessado em: 10/ 11/ 2010]

A partir daí, o Bluetooth começou a virar realidade, inclusive pela adoção desse nome. A denominação Bluetooth é uma homenagem a um rei dinamarquês chamado *Harald Blåtand*, mais conhecido como *Harald Bluetooth* (*Haroldo Dente-Azul*). Um de seus grandes feitos foi a unificação da Dinamarca, devido ao fato da tecnologia Bluetooth ter sido criada com o intuito de comunicar dispositivos heterogêneos, em função e fabricantes, optou-se por fazer referência ao findado rei. O logotipo do Bluetooth é a união das runas nórdicas ᚼ(Hagall) e ᛒ(Berkanan) correspondentes às letras H e B no alfabeto latino [1] (Ver na figura 2).

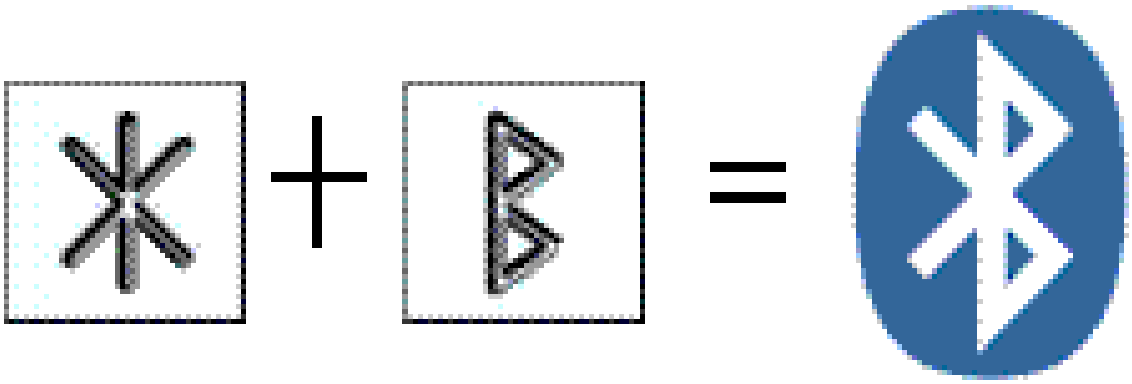

Figura 2 – Origem do logotipo Bluetooth

[http://2.bp.blogspot.com/_j8A6KKlMxQA/S4qL5G-ZGeI/AAAAAAAAAQU/1H6dzOnBpqE/s320/logo-Bluetooth.GIF
Acessado em: 10/ 11/ 2010]

2.2 INTRODUÇÃO AO BLUETOOTH

O Bluetooth é um modelo para comunicação *Wireless* de baixo custo e de curto alcance. Usando ondas de rádio na freqüência de 2.4 GHz, que não precisa de licença e está disponível em quase todo o mundo. Usando-se o mesmo e possibilitada a comunicação sem fio entre dispositivos eletrônicos que podem ser telefones celulares, Palmtops, computadores, scanners, impressoras, equipamentos de escritório, ou seja, qualquer dispositivo que tenha um chip Bluetooth. A arquitetura do Bluetooth consiste essencialmente de dois elementos: um *transceiver* (*hardware*) e uma pilha de protocolos (*software*), que proporcionam as funcionalidades básicas para se formar uma conexão Bluetooth.

Os dispositivos Bluetooth se comunicam constituindo uma rede que se chama piconet ou picorede, onde podem existir até oito dispositivos conectados. Basicamente um deles é o *master*, ou seja, o principal, sendo os demais os dispositivos escravos (*slave*). Embora oito dispositivos parecerem pouco, é possível interconectar vários piconets, aumentando os pontos de comunicação. Esse artifício é conhecido como scatternet e com esse método pode-se reunir até 10 piconets simultaneamente numa mesma área de cobertura de rádio. Tendo segurança graças ao fato de que cada piconet decodifica-se e protege contra interferências de intrusos.

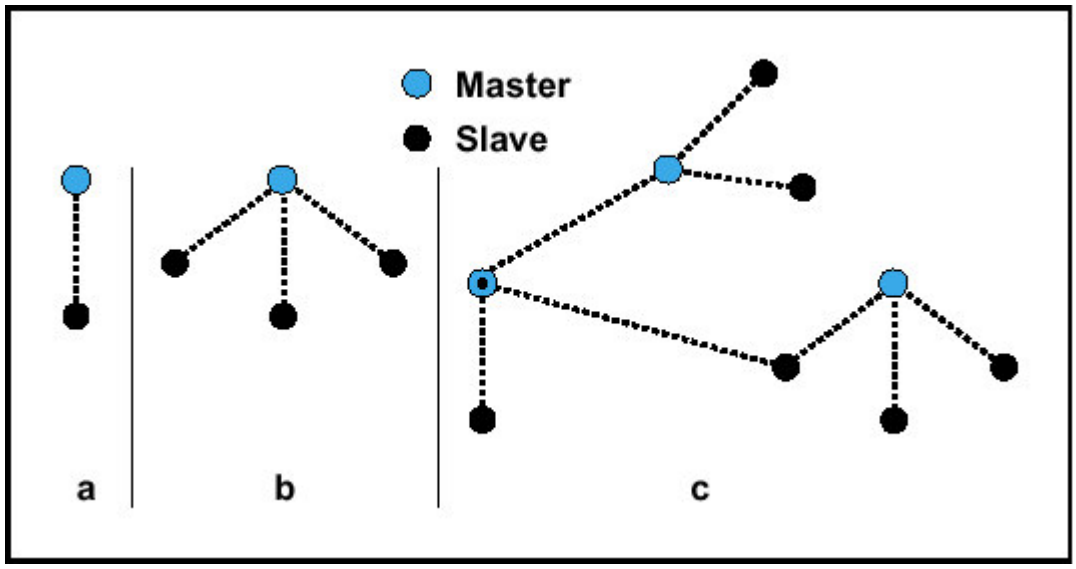

Figura 3 – Rede Bluetooth

[http://2.bp.blogspot.com/_ 1H6dzOnBpqE/s320/logo-Bluetooth.GIF
Acessado em: 02/ 09/ 2010]

A *mostra* uma piconet com uma única conexão. ***B*** *mostra* uma piconet com múltiplas conexões. ***C*** *mostra* uma possível configuração de uma scatternet.

Apesar de o Bluetooth ter sido desenvolvido para atender o segmento das PAN's (*Private Area Network*), que tem um alcance máximo de 100 metros, ele é geralmente utilizado em situações com alcances bem mais restritos. Na Figura 4 temos um infográfico, que mostra o uso da tecnologia Bluetooth, em comparação com as outras tecnologias de acesso sem fios disponíveis.

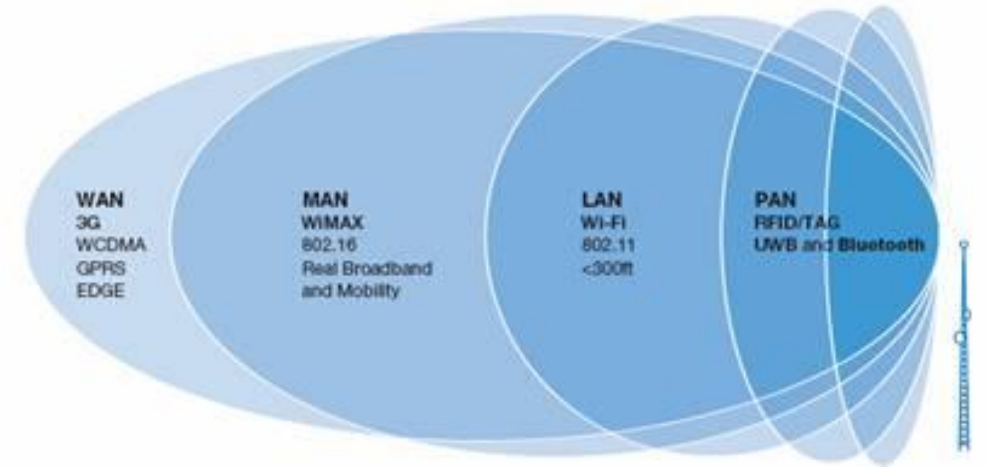

Figura 4 – Localização Espacial dos Dispositivos

[http://www2.eletronica.org/artigos/eletronica-digital/bluetooth/image002.jpg
Acessado em: 10/ 11/ 2010]

São divididos em classes de acordo com seu alcance com um total de três classes. Cada classes possui uma faixa de potencia onde opera, de acordo com a representação na tabela abaixo:

Tabela 1 – Potência de operação

	Potência Máxima	**Alcance**
Classe1	100 mW	100 metros
Classe2	2,5 mW	10 metros
Classe3	1 mW	1 metros

Para o Bluetooth operar na faixa ISM de 2,45 GHz, foram definidas 79 portadoras com espaçamento de 1 MHz. Ou seja, tem-se 79 frequências nas quais imediatamente um dispositivo pode estar transmitindo. A sequência optada deve ser constituída pelo dispositivo *master* da pico-net e os dispositivos *slave* devem adotar essa sequência para poder se comunicar. Isso é feito a-través de sincronismo. Para diminuir interferências, o dispositivo *master* pode mudar sua fre-quência 1600 vezes por segundo!

Para estabelecer conexões no Bluetooth, são necessários três elementos: *scan, page* e *in-quiry*.

SCAN - É usado para economia de energia. Quando dispositivos estiverem ociosos, eles entram em modo *stand-by*.e passam a verificar a cada 10 ms se existe algum dispositivo tentando estabelecer uma conexão.

PAGE - É utilizado pelo dispositivo que deseja estabelecer conexão. A cada 1,25 ms são transmitidos dois pedidos de conexão seguidos em diferentes portadoras. O dispositivo verifica também duas vezes se há respostas.

INQUIRY - São mensagens enviadas por um dispositivo para determinar quais outros dispositivos estão em sua área e quais suas características. Ao receber esta mensagem, um dispositivo deve retornar um pacote chamado FHS (*Frequency Hopping-Synchronization*) contendo além de sua identidade, informações para o sincronismo entre os dispositivos.

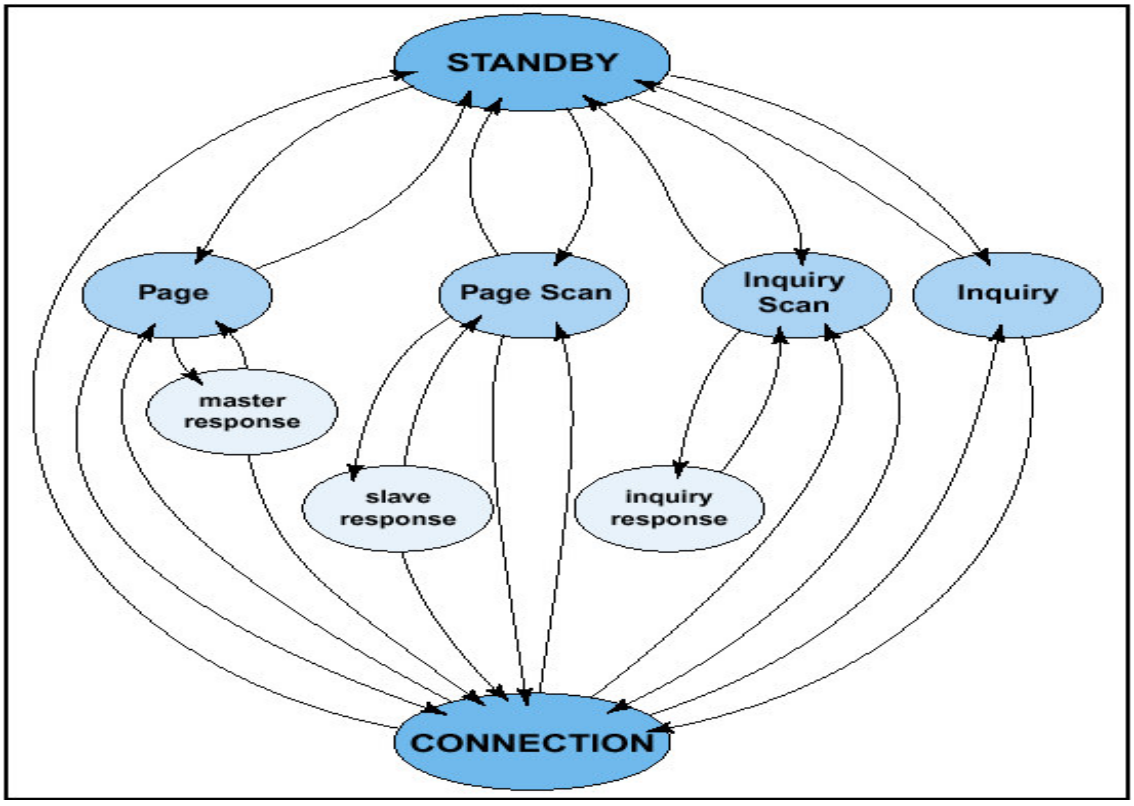

Figura 5 – Conexão Bluetooth

[://2.bp.blogspot.com/_ 1H6dzOnBpqE/s320/logo-Bluetooth.GIF
Acessado em: 02/ 10/ 2010]

3. COMO FUNCIONA A TECNOLOGIA BLUETOOTH

Numa rede Bluetooth, a transmissão de dados é feita através de pacotes, como na Internet. Para evitar interferências, e aumentar a segurança, existem 79 canais possíveis (23 em alguns países onde o governo reservou parte das freqüências usadas). O dispositivo Bluetooth tem capacidade de localizar dispositivos próximos, formando as redes de transmissão, chamadas de *piconet*. Uma vez estabelecida à rede, os dispositivos determinam um padrão de transmissão, usando os canais possíveis. Isto significa que os pacotes de dados serão transmitidos cada um em um canal diferente, numa ordem que apenas os dispositivos da rede conhecem.

Isto anula as possibilidades de interferência com outro dispositivo Bluetooth próximo (assim como qualquer outro aparelho que trabalhe na mesma frequência), e torna a transmissão de dados mais segura, já que um dispositivo "intruso", que estivesse próximo, mas não fizesse parte da rede simplesmente não compreenderia a transmissão. Naturalmente existe também um sistema de verificação e correção de erros, um pacote que se perca ou chegue corrompido ao destino será retransmitido, assim como acontece em outras arquiteturas de rede.

Para tornar as transmissões ainda mais seguras, o padrão inclui também um sistema de criptografia. Existe também a possibilidade de acrescentar camadas de segurança via software, como novas camadas de criptografia, autenticação, etc. [7]

3.1 PROTOCOLOS UTILIZADOS

O Bluetooth tem seu funcionamento baseado em pilhas de protocolo as quais carregam a funcionalidade do sistema, sendo desde a transmissão via ondas de radio o estabelecimento de links síncronos e assíncronos e suporte a criptografia e outras funções.

3.1.1 PROTOCOLOS NÚCLEO

Os protocolos nucleo são divididos em uma pilha de 5 camadas:

- *Bluetooth Radio* — Faz a especificação dos detalhes da interface com o ar também incluindo a frequência, salteamento, esquema de modulação e a força da transmissão.
- *Baseband* — Fala do estabelecimento de conexão com a piconet, endereçamento, formato do pacote, temporização e controle de energia
- *Link Manager Protocol (LMP)* — Estabelece as configurações do link entre dispositivos Bluetooth e faz o gerenciamento de links em andamento, incluindo aspectos de segurança como autenticações e Encriptações, e controle e negociação do tamanho do pacote da banda base
- *Logical Link Control and Adaptation Protocol* (L2CAP) — Adapta os protocolos das camadas superiores à camada de banda base, fornecendo tanto serviços sem orientação à conexão quanto serviços orientados à conexão.
- *Service Discovery Protocol* (SDP) — manipula informações do dispositivo, serviços e consultas para características de serviço entre dois ou mais dispositivos Bluetooth. [11]

3.1.2 PROTOCOLO DE SUBSTITUIÇÃO DE CABO

Radio frequency communications (RFCOMM) é o protocolo de substituição de cabo usado para criar uma porta serial virtual para fazer com que a substituição de tecnologias de cabo seja transparente através de mínimas modificações a dispositivos existentes. FRCOMM provê transmissão de dados binários e emula os sinais de controle do EIA-232 (também conhecido como RS-232) sobre uma camada de banda-base Bluetooth.

3.1.3 PROTOCOLO DE CONTROLE DE TELEFONIA

Telephony control protocol-binary (TCS BIN) é o protocolo orientado a bit que define o controle de chamada de sinalização para estabelecimento de chamadas de voz e dados entre dispositivos Bluetooth. Ainda, TCS BIN define procedimentos de gerenciamento de mobilidade para manipular grupos de dispositivos Bluetooth TCS.

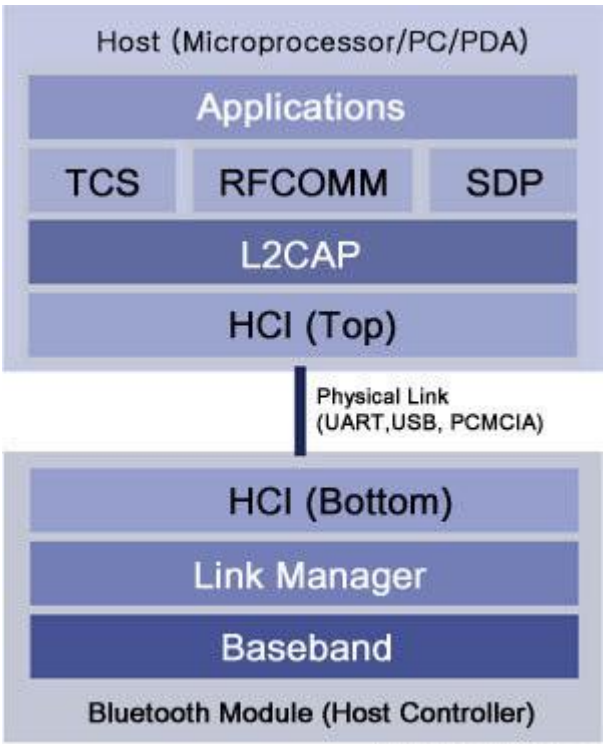

Figura 6 – Pilha de Protocolos

[**BILLO,** Eduardo Afonso, Artigo Uma pilha de protocolos Bluetooth adaptável à aplicação, Acessado em: 25/ 09/ 2010]

3.1.4 PROTOCOLOS ADOTADOS

Protocolos adaptados são definidos por outras organizações criadoras de padrões e são incorporados na pilha de protocolos do Bluetooth, permitindo ao Bluetooth criar protocolos apenas quando necessário. Os protocolos adotados incluem:

- *Point-to-Point Protocol* (PPP) — Protocolo padrão da Internet para transportar data gramas de IP sobre um link ponto-a-ponto
- TCP/IP/UDP — Protocolos fundamentais para a suite de protocolos TCP/IP
- *Object Exchange Protocol* (OBEX) — Protocolo de camada de sessão para intercâmbio de objetos, fornecendo um modelo para representação de objeto e operação

- *Wireless Application Environment / Wireless Application Protocol* (WAE/WAP) — WAE especifica um framework de aplicação para dispositivos sem fio e WAP em um padrão aberto para fornecer acesso a telefonia e serviçõs de informação aos usuários de "*mobiles*".

3.2 FREQUÊNCIA E COMUNICAÇÃO

O Bluetooth é uma tecnologia criada para funcionar no mundo todo, razão pela qual se fez necessária a adoção de uma frequência de rádio aberta, que seja padrão em qualquer lugar do planeta. A faixa ISM (*Industrial, Scientific, Medical*), que opera à frequência de 2,45 GHz, é a que me mais se aproxima dessa necessidade e é utilizada em vários países, com variações que vão de 2,4 GHz a 2,5 GHz.

Como a faixa ISM é aberta, isto é, pode ser utilizada por qualquer sistema de comunicação, é necessário garantir que o sinal do Bluetooth não sofra e não gere interferências. O esquema de comunicação *FH-CDMA* (*Frequency Hopping - Code-Division Multiple Access*), utilizado pelo Bluetooth, permite tal proteção, já que faz com que a frequência seja dividida em vários canais. O dispositivo que estabelece a conexão vai mudando de um canal para outro de maneira muito rápida. Esse esquema é chamado "salto de frequência" (*frequency hopping*). Isso faz com que a largura de banda da frequência seja muito pequena, diminuindo sensivelmente as chances de uma interferência. No Bluetooth, pode-se utilizar até 79 frequências (ou 23, dependendo do país) dentro da faixa ISM, cada uma espaçada da outra por 1 MHz.

Como um dispositivo se comunicando por Bluetooth pode tanto receber quanto transmitir dados (modo *full-duplex*), a transmissão é alternada entre slots para transmitir e slots para receber, um esquema denominado *FH/TDD* (*Frequency Hopping/Time-Division Duplex*). Esses slots são canais divididos em períodos de 625 µs (microssegundos). Cada salto de frequência deve ser ocupado por um slot, logo, em 1 segundo, tem-se 1600 saltos.

No que se refere ao enlace, isto é, à ligação entre o emissor e receptor, o Bluetooth faz uso, basicamente, de dois padrões: *SCO* (*Synchronous Connection-*

Oriented) e *ACL* (*Asynchronous Connection-Less*). O primeiro estabelece um link sincronizado entre o dispositivo master e o dispositivo escravo, onde é feito uma reserva de slots para cada um. Assim, o SCO acaba sendo utilizado principalmente em aplicações de envio contínuo de dados, como voz. Por funcionar dessa forma, o SCO não permite a retransmissão de pacotes de dados perdidos. Quando ocorre perda em uma transmissão de áudio, por exemplo, o dispositivo receptor acaba reproduzindo som com ruído. A taxa de transmissão de dados no modo SCO é de 432 Kbps, sendo de 64 Kbps para voz.

O padrão ACL, por sua vez, estabelece um link entre um dispositivo *master* e os dispositivos *slave* existentes em sua rede. Esse link é assíncrono, já que utiliza os slots previamente livres. Ao contrário do SCO, o ACL permite o reenvio de pacotes de dados perdidos, garantindo a integridade das informações trocadas entre os dispositivos. Assim, acaba sendo útil para aplicações que envolvam transferência de arquivos, por exemplo. A velocidade de transmissão de dados no modo ACL é de até 721 Kbps.

É importante colocar que diferentes tipos de link podem ser aplicados entre diferentes pares de *master-slave* numa mesma piconet e o tipo de link pode mudar arbitrariamente durante uma seção. O tipo do link define quais os tipos de pacotes podem ser usados. [1]

3.3 SISTEMA RÁDIO

O *Bluetooth* utiliza modulação GSFK (*Gaussian Frequency Shift Keying*), onde os dados são codificados na forma de variações de freqüência em uma portadora, de maneira similar à modulação FSK. Todavia, antes dos pulsos entrarem no modulador, eles passam por um filtro gaussiano, de modo a reduzir a largura espectral dos mesmos, servindo como uma espécie de formatador de pulso suavizando a transição entre os valores dos pulsos. No *Bluetooth*, o um "1" binário é representado por um desvio positivo de freqüência, e um zero binário é representado por um desvio negativo de freqüência, com variação nunca menor que 115 KHz.

A partir da inclusão do *Modo Enhanced Data Rate* (EDR) – oficialmente a partir da versão 2.0, mas já presente na implementação v1.2 de muitos fabricantes – passou-se a utilizar também modulação *Phase Shift Keying* com 8 níveis (8PSK).

O espalhamento espectral é feito através de *Frequency Hopping*. Neste método, o transmissor envia um sinal sobre uma série randômica de freqüências de rádio. Um receptor captura o sinal, através de uma sincronia com o transmissor. A mensagem somente é recebida se o receptor conhecer a série de freqüências na qual o transmissor trabalha para enviar o sinal.

Em um canal físico básico de *piconet* do *Bluetooth* a sua frequência muda de forma pseudo-aleatória 1.600 vezes por segundo (cada 0,625 useg). A sequência de salto de frequência é definida pelo relógio e endereço *Bluetooth* do dispositivo mestre. Os dispositivos em uma *piconet* compartilham este canal físico de comunicação. Quando ocorre um salto de frequência os seus transmissores e receptores são sintonizados ao mesmo tempo na nova frequência.

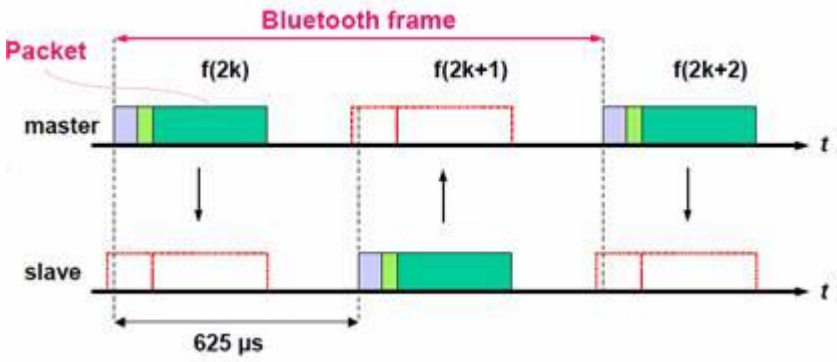

Figura 7 – Salto de frequência

[http://wiki.sj.cefetsc.edu.br/wiki/index.php/Imagem:Blue_frame.jpg]

O intervalo de tempo de 0,625 useg que dura a transmissão em um frequência é chamado de *slot*. Um pacote de dados é transmitido em cada *slot* de tempo. É possível também estender o pacote para ocupar 3 ou 5 *slots* de modo a aumentar a taxa de dados transmitida como apresentado na figura a seguir.

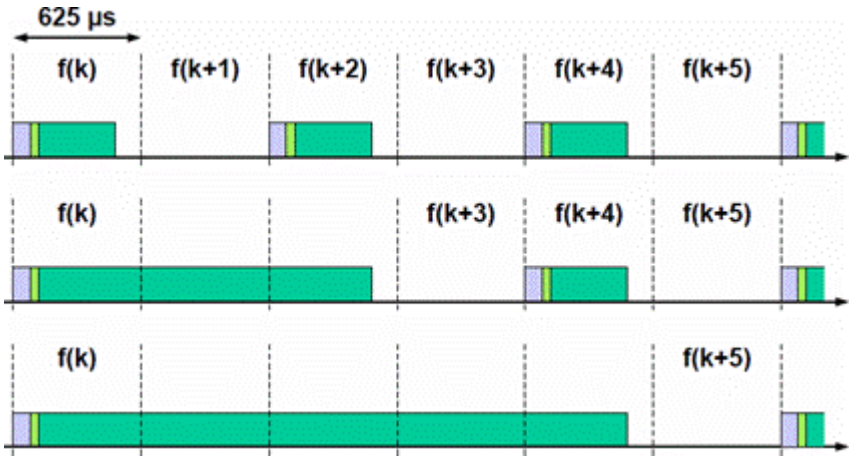

Figura 8 – Gráfico pacotes e slots

[http://wiki.sj.cefetsc.edu.br/wiki/index.php/Imagem:Multi-slot_packets.jpg]

O *release* 1.2 da especificação do *Bluetooth* definiu também um canal de *piconet* adaptado que apresenta as seguintes diferenças em relação ao canal básico descrito acima:

As frequências nas quais um escravo transmite são as mesmas que o mestre acabou de transmitir. Ou seja, não há um salto de frequência entre um pacote do mestre e o pacote do escravo que vem logo a seguir.

Para evitar a colisão entre as múltiplas transmissões de dispositivos escravos, o dispositivo mestre utiliza uma técnica chamada "*polling*", que permite somente ao dispositivo indicado no slot mestre-para-escravo transmitir no slot escravo-para-mestre seguinte.

Como existem muitas tecnologias trabalhando nessa faixa ISM, a maioria não utilizando sistemas *Frequency Hopping*, é possível excluir algumas frequências entre as 79 disponíveis para a sequência de salto de frequências, que são marcadas como fora de uso. Evita-se desta forma a utilização de frequências com alto grau de interferência. [14]

3.4 ESTABELECENDO UMA CONEXÃO BLUETOOTH

Um dispositivo Bluetooth pode operar em dois estados. O primeiro é o estado *Connection*, o dispositivo está conectado a outro ou executando alguma atividade. Já no estado *standby*, o dispositivo está conectado, mas não está envolvido em nenhuma atividade com outro dispositivo.

Os dispositivos Bluetooth sempre iniciam no modo *standby* sendo que quando um dispositivo percebe o outro, um se torna o mestre e o outro se torna o escravo. Numa conexão é emitido um comando de solicitação quando ainda não se conhece o endereço do dispositivo. Uma vez conhecido o endereço, é emitido o comando página despertando o outro dispositivo estabelecendo a conexão.

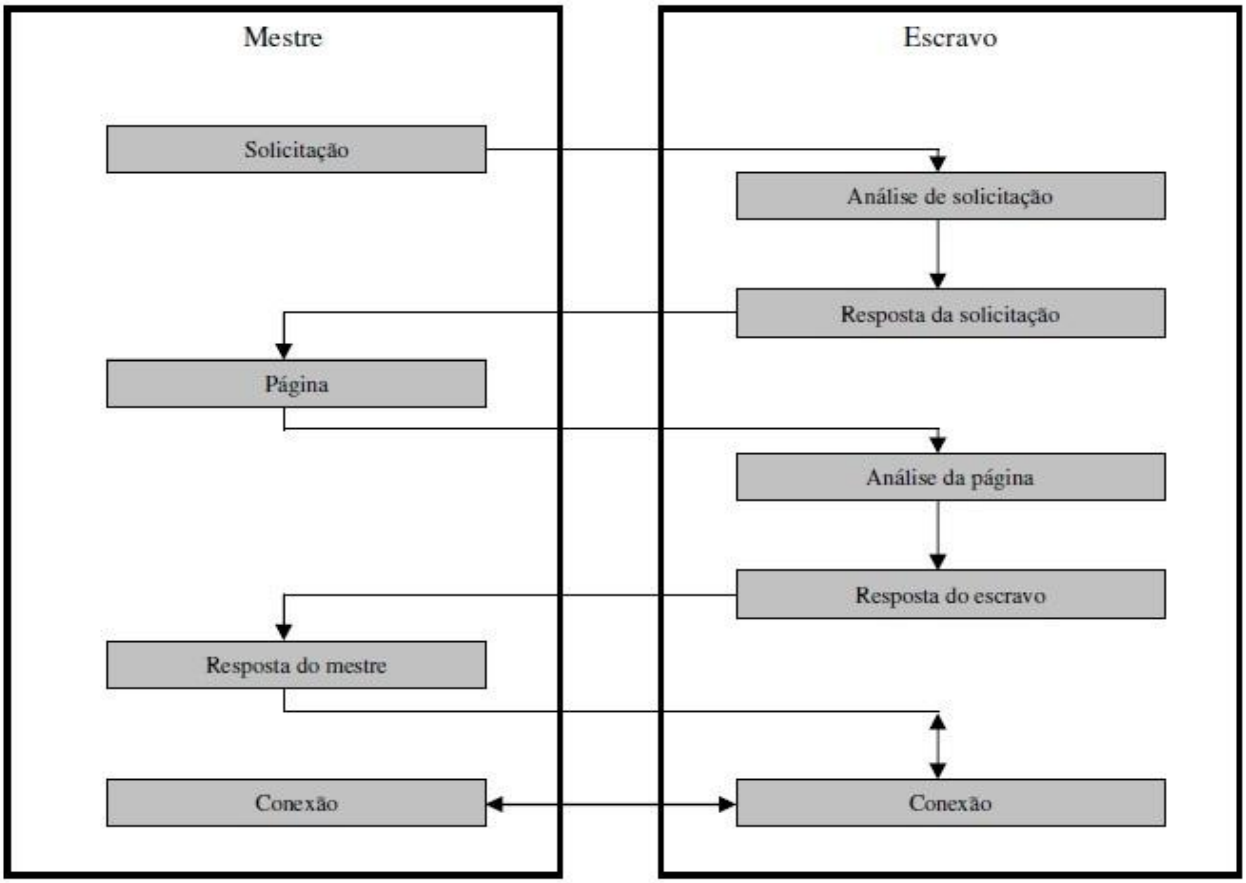

Figura 9 – Estabelecimento de Conexão Bluetooth

[http://2.bp.blogspot.com/_j8A6KKlMxQA/S4qL5G-ZGeI/AAAAAAAAAQU/1H6dzOnBpqE/s320/logo-Bluetooth.GIF Acessado em: 02/ 11/ 2007]

4. APLICAÇÕES DA TECNOLGIA BLUETOOH

Atualmente com a popularização do Bluetooth se encontram inúmeras aplicações para a tecnologia, principalmente para as situações onde se exige uma baixa de transferência de dados e curto alcance, entre as aplicações práticas para essa tecnologia, encontramos os telefones celulares, aplicações em computadores e periféricos, indústria automobilística e de eletroeletrônicos, incluindo ainda aplicações nas áreas de entretenimento (jogos), automação predial e industrial, medicina, etc.

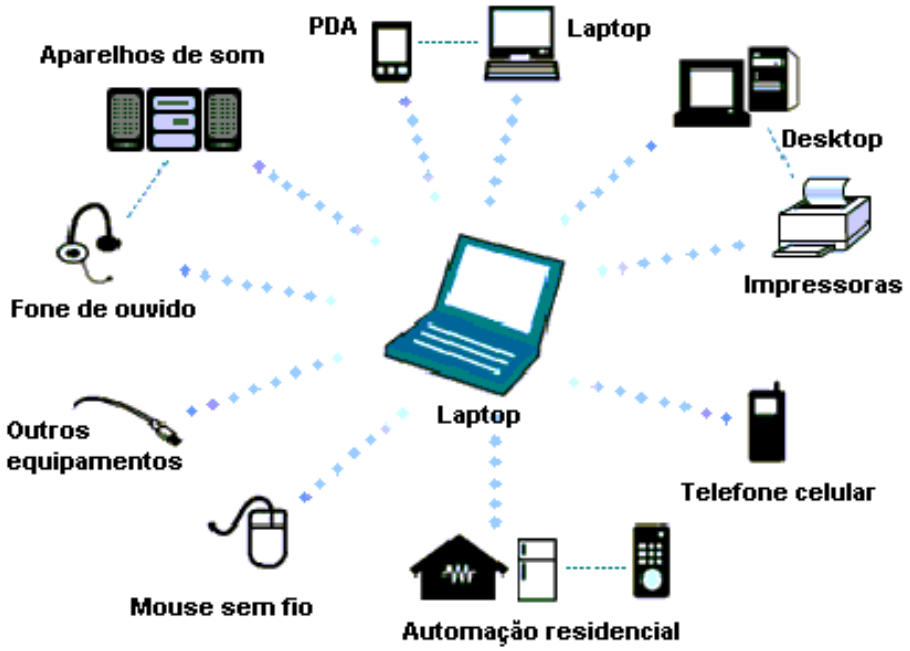

Figura 10 – Dispositivos Bluetooth

[http://www.projetoderedes.com.br/artigos/imagens/image58.gif
Acessado em: 15/ 11/ 2010]

O padrão Bluetooth prevê o uso de diversos "*profiles*", que são diferentes protocolos de comunicação, desenvolvidos de forma a atender diversos cenários de uso. Os cinco *profiles* mais usados são o HSP *(Headset Profile*), que é utilizado por *headsets* Bluetooth, o HID (*Human Interface Device Profile*), usado por teclados, mouses, joysticks e outros dispositivos de entrada, o FTP (*File Transfer Profile*), que permite transferir arquivos, o OPP (*Object Push Profile*) um protocolo de transferência de dados de uso geral, que pode ser usado para transferir contatos, fotos e outras informações e o DUN (*Dial-up Networking Profile*), que é usado por celulares para permitir o acesso à web através do PC.

Cada *profile* faz com que o transmissor Bluetooth e o dispositivo do outro lado sejam vistos de forma diferente pelo sistema. No HSP, o *headset* é visto como uma placa de som remota, que permite o envio de *streams* de áudio. No HID o teclado ou mouse Bluetooth é visto pelo sistema como se fosse um dispositivo de entrada conectado a uma das portas USB do micro, enquanto no DUN o celular é visto pelo sistema como um modem ligado a uma porta serial, que é usado para "discar" para o provedor e, assim, estabelecer a conexão.

Existe ainda o PAN que é um profile pouco usado na prática, já que é muito mais fácil ligar dois micros usando um cabo *cross-over*, ou uma rede *wireless* ad-hoc, que são mais fáceis de configurar e oferecem uma velocidade maior. [7]

5. VANTAGENS E DESVANTAGENS

5.1 VANTAGENS

Com Bluetooth não é necessário usar conexões por cabo, tornando a conexão mais prática e barata, hoje o SIG já conta com mais de 10 mil empresas integrantes de áreas diferentes, o que proporciona uma grande quantidade de dispositivos com chips Bluetooth no mercado; o Bluetooth suporta comunicação tanto por voz quanto por dados, sendo útil nas mais diversas aplicações; a tecnologia pode ser facilmente integrada aos protocolos de comunicação, como o TCP/IP, por exemplo.

Outra grande cartada do Bluetooth e o seu sistema de uso inteligente da potência do sinal. Se dois dispositivos estão próximos, é usado um sinal mais fraco, com o objetivo de diminuir o consumo elétrico, se por outro lado eles estão distantes, o sinal vai ficando mais forte, até atingir a potência máxima.

Dentro do limite dos 10 metros ideais, o consumo de cada transmissor fica em torno de 50 microampères, algo em torno de 3% do que um celular atual, bem menos do que outras tecnologias sem fio atuais. O baixo consumo permite incluir os transmissores em notebooks, celulares e handhelds sem comprometer muito a autonomia das baterias.

5.2 DESVANTAGENS

O Bluetooth também traz inúmeras desvantagens podemos citar como principais o número máximo de dispositivos que podem se conectar ao mesmo tempo é limitado, são somente 8 dispositivos por rede, principalmente se compararmos com a rede cabeada. O alcance é bastante curto, por isso uma rede pode ser apenas local, o alcance ideal e 10 metros. A taxa de transferência de dados é muito baixa isso inviabiliza muitas das aplicações multimídia atuais.

6. CONCLUSÃO

A tecnologia Bluetooth é um novo recurso tecnológico muito promissor como alternativa para as redes *wireless*, apesar de ter limitações como pouco alcance ou falta de padronização da frequência utilizada pela tecnologia por outro lado tem seus pontos fortes como baixo consumo de energia e a ausência de cabeamento, utilizando-se a arquitetura de sistemas de distribuição, pode ter sua abrangência aumentada.

Com sua simples arquitetura e capacidade de adaptação tende a trazer muita praticidade para a vida moderna. A especificação Bluetooth suporta transferências tanto de voz quanto de dados, tornando-se uma tecnologia ideal na comunicação de dispositivos heterogêneos.

E por ser um tema um tanto recente ainda não se tem muito material especifico sobre o mesmo o que dificulta sua pesquisa, muitas das fontes encontradas repetem as suas referencias e tem conteúdo muito semelhante. Mas seu conteúdo e de fácil entendimento com um mínimo de conhecimento teórico e pesquisa.

REFERÊNCIAS

[1] **ALECRIM**, Emerson. Tecnologia Bluetooth. Disponível em: <www.infowester.com/Bluetooth.php> Acessado em: 09 de outubro de 2010.

[2] **BILLO,** Eduardo Afonso. Artigo Uma pilha de protocolos Bluetooth adaptável à aplicação, Disponível em: <http://www.lisha.ufsc.br/teaching/theses/billo.pdf>. Acessado em: 22 de outubro de 2010.

[3] **BLUETOOTH**. Soluções sem fio, Disponível em: <http://www2.eletronica.org/artigos/eletronica-digital/bluetooth-solucoes-sem-fio> Acessado em 15 de outubro de 2010.

[4] **BLUETOOTH SIG**. Bluetooth Sig Selects Wimedia Alliance Ultra-Wideband Technology for High Speed Bluetooth® Applications. Disponível em: <www.Bluetooth.com> Acessado em: 09 de outubro 2010.

[5] **CORPORATION,** Microsoft. Disponível em: <http://msdn.microsoft.com/pt-br/library/aa931801.aspx> Acessado em: 15 de Outubro de 2010.

[6] **KOBAYASHI,** Carlos Yassunori. A Tecnologia Bluetooth e aplicações, Disponível em: <http://grenoble.ime.usp.br/movel/monografia_bluetooth.pdf> Acessado em: 21 de Outubro de 2010.

[7] **MORIMOTO**, Carlos E. Conheça o Bluetooth – Disponível em: <http://www.guiadohardware.net/analises/saiba-bluetooth/> Acessado em: 28 de setembro de 2010.

[8] **PEREIRA,** Magno C. Artigo Bluetooth. Disponível em: <http://www.aedb.br/seget/artigos05/16_artigo_Bluetooth.pdf> Acessado em: 06 de outubro de 2010

[9] **PINHEIRO**, José Mauricio Santos. Por dentro do Bluetooth. Disponível em: <http://www.projetoderedes.com.br/artigos/artigo_dentro_bluetooth.php> Acessado em: 15 de outubro de 2010.

[10] **SIQUEIRA**, Thiago Senador. Bluetooth Características, Protocolos e Funcionamento, Disponível em: <http://www.ic.unicamp.br/~ducatte/mo401/1s2006/T2/057642-T.pdf> - Bluetooth – Características, protocolos e funcionamento. Acessado em: 14 de outubro de 2010.

[11] **WIKIPEDIA.** Bluetooth – Disponível em: <http://pt.wikipedia.org/wiki/Bluetooth> Acessado em: 17 de outubro de 2010.

[12] **WIKIPEDIA.** Redes de computadores - conceitos – II – Disponível em: <http://wiki.pm.sc.gov.br/Redes_de_computadores_-_conceitos_-_II> Acessado em: 09 de outubro de 2010.

[13] **WIKIPEDIA.** Bluetooth from Wikipedia – Disponível em: <http://en.wikipedia.org/wiki/Bluetooth> Acessado em: 14 de outubro de 2010.

[14] **WIKIPEDIA.** Bluetooth – Disponível em: <http://wiki.sj.cefetsc.edu.br/wiki/index.php/Bluetooth> Acessado em: 10 de outubro de 2010.

ANEXOS

ANEXO A – Código HCI

```
#include <linux/config.h>
#include <linux/module.h>
#include <linux/types.h>
#include <linux/errno.h>
#include <linux/kernel.h>
#include <linux/major.h>
#include <linux/sched.h>
#include <linux/slab.h>
#include <linux/poll.h>
#include <linux/fcntl.h>
#include <linux/init.h>
#include <linux/skbuff.h>
#include <linux/interrupt.h>
#include <linux/socket.h>
#include <linux/skbuff.h>
#include <linux/proc_fs.h>
#include <linux/list.h>
#include <net/sock.h>
#include <asm/system.h>
#include <asm/uaccess.h>
#include <asm/unaligned.h>
#include "Bluetooth.h"
#include "hci_core.h"
#define BT_INQUIRY_LENGTH 8
#define BT_INQUIRY_NUM_RESPONSES 10
int (*list_receptors[10]) (char * data);
int num_receptors = 0;
struct sk_buff * wait_acl_data = 0;
/* This is just a first version of the billotooth stack, so it will just support
device which will be handled by the mydev struct. */
struct hci_dev *mydev;
/* inquiry device list */
struct inquiry_cache inq_cache;
static void billotooth_event_packet(struct sk_buff *skb);
/*-----------------------------------------------------------------------------------
GENERAL-USE ROUTINES
-----------------------------------------------------------------------------------*/
```

```
char *batostr(bdaddr_t *ba)
{
static char str[2][18];
static int i = 1;
i ^= 1;
sprintf(str[i], "%2.2X:%2.2X:%2.2X:%2.2X:%2.2X:%2.2X",
ba->b[0], ba->b[1], ba->b[2],
ba->b[3], ba->b[4], ba->b[5]);
return str[i];
}
void data_dump(struct sk_buff *skb) {
int count;
char byte;
BT_DUMP("DATA: ");
for (count=0 ; count < skb->len; count++) {
byte = skb->data[count];
BT_DUMP("%2.2X ",byte);
}
BT_DUMP("\n");
}
/*-----------------------------------------------------------------------------------------
DATA TRANFER ROUTINES
-----------------------------------------------------------------------------------------*/
/* Send Data to the lower stack layer */
int billotooth_send_frame(struct sk_buff *skb)
{
/* Get rid of skb owner, prior to sending to the driver. */
skb_orphan(skb);
BT_DBG("Sending data to the USB driver, len = %d",skb->len);
data_dump(skb);
return mydev->send(skb);
return 0;
}
/* Send ACL data */
int billotooth_send_acl(struct hci_conn *conn, char * data, int len)
{
struct sk_buff * skb;
hci_acl_hdr *ah;
__u16 flags = 0;
skb = bluez_skb_alloc(HCI_ACL_HDR_SIZE+len, GFP_ATOMIC);
if (!(skb)) {
BT_ERR("Lack of memory in kernel to allocate structure");
}
skb->dev = (void *) mydev;
skb->pkt_type = HCI_ACLDATA_PKT;
/* Assembles the acl header */
```

```
ah = (hci_acl_hdr *) skb_put(skb, HCI_ACL_HDR_SIZE);
ah->handle = __cpu_to_le16(acl_handle_pack(conn->handle, flags | ACL_START));
ah->dlen = __cpu_to_le16(len);
skb->h.raw = (void *) ah;
if (len)
memcpy(skb_put(skb, len), data, len);
BT_DBG("Sending ACL %d bytes of data through connetion %p flags 0x%x", skb->len,conn,
flags);
//data_dump(skb);
return billotooth_send_frame(skb);
//TODO: fragmented packets
}
/* Process received packet */
void hci_acldata_packet(struct sk_buff * skb)
{
int i;
BT_DBG("ACL packet received");
data_dump(skb);
//Send packet to upper layers
for (i=0;i<num_receptors;i++)
{
list_receptors[i](skb->data+HCI_ACL_HDR_SIZE);
}
kfree_skb(skb);
}
/* Receive a frame from the lower stack layer */
int billotooth_recv_frame(struct sk_buff *skb)
{
/* Make sure that the hci interface is UP */
if (!mydev || (!test_bit(HCI_UP, &mydev->flags) &&
!test_bit(HCI_INIT, &mydev->flags)) ) {
kfree_skb(skb);
return -1;
}
BT_DBG("Frame received. Type = %d, Size = %d", skb->pkt_type, skb->len);
/* Process frame */
switch (skb->pkt_type) {
case HCI_EVENT_PKT:
billotooth_event_packet(skb);
break;
case HCI_ACLDATA_PKT:
hci_acldata_packet(skb);
break;
case HCI_SCODATA_PKT:
BT_DBG("SCO packet received. HANDLER NOT IMPLEMENTED!!!");
kfree_skb(skb);
```

```
break;
default:
kfree_skb(skb);
break;
}
return 0;
}
/*---------------------------------------------------------------------------------------
COMMAND ROUTINES
---------------------------------------------------------------------------------------*/
/* Send a command to the HCI controller */
int billotooth_send_cmd(__u16 ogf, __u16 ocf, __u32 plen, void *param)
{
int len = HCI_COMMAND_HDR_SIZE + plen;
hci_command_hdr *hc;
struct sk_buff *skb;
/* allocate memory to skb */
if (!(skb = bluez_skb_alloc(len, GFP_ATOMIC))) {
BT_ERR("Error allocating memory");
return -ENOMEM;
}
/* fill command operands */
hc = (hci_command_hdr *) skb_put(skb, HCI_COMMAND_HDR_SIZE);
hc->opcode = __cpu_to_le16(cmd_opcode_pack(ogf, ocf));
hc->plen = plen;
if (plen)
memcpy(skb_put(skb, plen), param, plen);
BT_DBG("Sending command. Len = %d Ogf=0x%X Ocf=0x%X Plen=0x%X", skb-
>len,ogf,ocf,plen);
skb->pkt_type = HCI_COMMAND_PKT;
skb->dev = (void *) mydev;
/* send command to the lower stack layer */
return billotooth_send_frame(skb);
}
/*---------------------------------------------------------------------------------------
INQUIRY ROUTINES
---------------------------------------------------------------------------------------*/
/* Request for an inquiry (location the devices nearby) */
void billotooth_inq_req(void)
{
inquiry_cp ic;
BT_DBG("Initializing inquiry...");
/* fill command operands */
ic.lap[0] = 0x33;
ic.lap[1] = 0x8B;
ic.lap[2] = 0x9E;
```

```
ic.length = BT_INQUIRY_LENGTH;
ic.num_rsp = BT_INQUIRY_NUM_RESPONSES;
/* send command */
billotooth_send_cmd(OGF_LINK_CTL, OCF_INQUIRY, INQUIRY_CP_SIZE, &ic);
}
/* Updates the list with a new device just found */
void billotooth_inquiry_list_update(inquiry_info *info)
{
struct inquiry_entry *e;
/* allocate memory to the inquiry entry */
if (!(e = kmalloc(sizeof(struct inquiry_entry), GFP_ATOMIC)))
return;
/* insert device in the inquiry list */
memset(e, 0, sizeof(struct inquiry_entry));
e->next = inq_cache.list;
inq_cache.list = e;
memcpy(&e->info, info, sizeof(inquiry_info));
BT_DBG("New Device. Address = %s", batostr(&info->bdaddr));
}
/* Destroy the inquiry list */
void billotooth_inquiry_list_destroy(void) {
struct inquiry_entry *e = inq_cache.list;
/* Deallocate all the inquiry entries */
while (e) {
kfree(e);
e = e->next;
}
BT_DBG("Inquiry list destroyed");
}
/* Destroy the inquiry list */
struct inquiry_entry *billotooth_inquiry_lookup(bdaddr_t *dst) {
struct inquiry_entry *e = inq_cache.list;
while (e) {
if (!bacmp(&e->info.bdaddr, dst))
break;
e = e->next;
BT_DBG("%s\n","Found device in inquiry lookup list");
}
return e;
}
/*-----------------------------------------------------------------------------------
CONNECTION ROUTINES
-----------------------------------------------------------------------------------*/
struct hci_conn *billotooth_create_connection(bdaddr_t *dst)
{
struct hci_conn *conn;
```

```
BT_DBG("Creating connection with remote host %s", batostr(dst));
if (!(conn = kmalloc(sizeof(struct hci_conn), GFP_ATOMIC)))
return NULL;
memset(conn, 0, sizeof(struct hci_conn));
bacpy(&conn->dst, dst);
conn->type = ACL_LINK; //so far, only ACL supported
conn->hdev = mydev;
conn->state = BT_OPEN;
conn_hash_add(mydev, conn);
return conn;
}
void billotooth_acl_connect(struct hci_conn *conn)
{
struct inquiry_entry *ie;
create_conn_cp cp;
BT_DBG("Connecting to remote host %s", batostr(&conn->dst));
conn->state = BT_CONNECT;
conn->out = 1;
conn->link_mode = HCI_LM_MASTER;
memset(&cp, 0, sizeof(cp));
bacpy(&cp.bdaddr, &conn->dst);
if ((ie = billotooth_inquiry_lookup(&conn->dst))) {
cp.pscan_rep_mode = ie->info.pscan_rep_mode;
cp.pscan_mode = ie->info.pscan_mode;
cp.clock_offset = ie->info.clock_offset | __cpu_to_le16(0x8000);
}
cp.pkt_type = __cpu_to_le16(mydev->pkt_type & ACL_PTYPE_MASK);
cp.role_switch = 0x01;
billotooth_send_cmd(OGF_LINK_CTL, OCF_CREATE_CONN,
CREATE_CONN_CP_SIZE, &cp);
}
struct hci_conn * billotooth_connect(bdaddr_t *dst)
{
struct hci_conn *acl;
if (!(acl = conn_hash_lookup_ba(mydev, ACL_LINK, dst))) {
if (!(acl = billotooth_create_connection(dst)))
return NULL;
}
if (acl->state == BT_OPEN || acl->state == BT_CLOSED)
billotooth_acl_connect(acl);
BT_DBG("%s","Leaving billotooth_connect");
return acl;
}
void billotooth_acl_disconnect(struct hci_conn *conn, __u8 reason)
{
disconnect_cp cp;
```

```
BT_DBG("%s","Disconnecting...");
conn->state = BT_DISCONN;
cp.handle = __cpu_to_le16(conn->handle);
cp.reason = reason;
billotooth_send_cmd(OGF_LINK_CTL, OCF_DISCONNECT,DISCONNECT_CP_SIZE, &cp);
}
void billotooth_conn_complete_evt(struct sk_buff *skb)
{
evt_conn_complete *cc = (evt_conn_complete *) skb->data;
struct hci_conn *conn = NULL;
conn = conn_hash_lookup_ba(mydev, cc->link_type, &cc->bdaddr);
if (!conn) {
return;
}
conn->handle = __le16_to_cpu(cc->handle);
conn->state = BT_CONNECTED;
/* Set packet type for incomming connection */
if (!conn->out) {
change_conn_ptype_cp cp;
cp.handle = cc->handle;
cp.pkt_type = (conn->type == ACL_LINK) ?
__cpu_to_le16(mydev->pkt_type & ACL_PTYPE_MASK):
__cpu_to_le16(mydev->pkt_type & SCO_PTYPE_MASK);
billotooth_send_cmd(OGF_LINK_CTL, OCF_CHANGE_CONN_PTYPE,
CHANGE_CONN_PTYPE_CP_SIZE, &cp);
}
BT_DBG("Connected with %s thought conn 0x%x", batostr(&cc->bdaddr),conn->handle);
}
void hci_disconn_complete_evt(struct sk_buff *skb)
{
evt_disconn_complete *dc = (evt_disconn_complete *) skb->data;
struct hci_conn *conn = NULL;
__u16 handle = __le16_to_cpu(dc->handle);
BT_DBG("%s","Disconnected...");
if (dc->status)
return;
conn = conn_hash_lookup_handle(mydev, handle);
if (conn) {
conn->state = BT_CLOSED;
conn_hash_del(mydev,conn);
}
}
/*---------------------------------------------------------------------------------------
EVENT ROUTINES
---------------------------------------------------------------------------------------*/
/* Command Complete OGF HOST_CTL */
```

```
static void billotooth_cc_host_ctl(__u16 ocf, struct sk_buff *skb)
{
__u8 status;
status = *((__u8 *) skb->data);
switch (ocf) {
case OCF_SET_EVENT_FLT:
if (status) {
BT_DBG("SET_EVENT_FLT failed %d", status);
} else {
BT_DBG("SET_EVENT_FLT succeseful");
}
break;
case OCF_WRITE_CA_TIMEOUT:
if (status) {
BT_DBG("OCF_WRITE_CA_TIMEOUT failed %d", status);
} else {
BT_DBG("OCF_WRITE_CA_TIMEOUT succeseful");
}
break;
case OCF_WRITE_PG_TIMEOUT:
if (status) {
BT_DBG("OCF_WRITE_PG_TIMEOUT failed %d", status);
} else {
BT_DBG("OCF_WRITE_PG_TIMEOUT succeseful");
}
break;
case OCF_WRITE_SCAN_ENABLE:
if (!status) {
BT_DBG("Scan and inquiry enabled");
}
}
}
/* Command Complete OGF INFO_PARAM */
static void billotooth_cc_info_param(__u16 ocf, struct sk_buff *skb)
{
read_local_features_rp *lf;
read_buffer_size_rp *bs;
read_bd_addr_rp *ba;
switch (ocf) {
case OCF_READ_LOCAL_FEATURES:
lf = (read_local_features_rp *) skb->data;
if (lf->status) {
BT_DBG("READ_LOCAL_FEATURES failed %d", lf->status);
break;
}
memcpy(mydev->features, lf->features, sizeof(mydev->features));
```

```
/* Adjust default settings according to features
* supported by device. */
if (mydev->features[0] & LMP_3SLOT)
mydev->pkt_type |= (HCI_DM3 | HCI_DH3);
if (mydev->features[0] & LMP_5SLOT)
mydev->pkt_type |= (HCI_DM5 | HCI_DH5);
if (mydev->features[1] & LMP_HV2)
mydev->pkt_type |= (HCI_HV2);
if (mydev->features[1] & LMP_HV3)
mydev->pkt_type |= (HCI_HV3);
BT_DBG("Device features 0x%x 0x%x 0x%x", lf->features[0], lf->features[1], lf-
>features[2]);
break;
case OCF_READ_BUFFER_SIZE:
bs = (read_buffer_size_rp *) skb->data;
if (bs->status) {
BT_DBG("READ_BUFFER_SIZE failed %d", bs->status);
break;
}
mydev->acl_mtu = __le16_to_cpu(bs->acl_mtu);
mydev->sco_mtu = bs->sco_mtu ? bs->sco_mtu : 64;
mydev->acl_pkts = mydev->acl_cnt = __le16_to_cpu(bs->acl_max_pkt);
mydev->sco_pkts = mydev->sco_cnt = __le16_to_cpu(bs->sco_max_pkt);
BT_DBG("Buffer sizes. mtu: acl %d, sco %d max_pkt: acl %d, sco %d",
mydev->acl_mtu, mydev->sco_mtu, mydev->acl_pkts, mydev->sco_pkts);
break;
case OCF_READ_BD_ADDR:
ba = (read_bd_addr_rp *) skb->data;
if (!ba->status) {
bacpy(&mydev->bdaddr, &ba->bdaddr);
} else {
BT_DBG("READ_BD_ADDR failed %d",ba->status);
}
break;
};
}
/* Inquiry Result */
static inline void billotooth_inquiry_result_evt(struct sk_buff *skb)
{
inquiry_info *info = (inquiry_info *) (skb->data + 1);
int num_rsp = *((__u8 *) skb->data);
BT_DBG("%d detected device(s)", num_rsp);
for (; num_rsp; num_rsp--)
billotooth_inquiry_list_update(info++);
}
/* Connect Request */
```

```
static inline void billotooth_conn_request_evt(struct sk_buff *skb)
{
evt_conn_request *cr = (evt_conn_request *) skb->data;
struct hci_conn *conn;
accept_conn_req_cp ac;
BT_DBG("Connection request: %s type 0x%x",
batostr(&cr->bdaddr), cr->link_type);
/* Connection accepted */
conn = conn_hash_lookup_ba(mydev, cr->link_type, &cr->bdaddr);
if (!conn) {
if (!(conn = billotooth_create_connection(&cr->bdaddr))) {
BT_ERR("No memmory for new connection");
return;
}
}
conn->state = BT_CONNECT;
bacpy(&ac.bdaddr, &cr->bdaddr);
ac.role = 0x01; /* Remain slave */
billotooth_send_cmd(OGF_LINK_CTL, OCF_ACCEPT_CONN_REQ,
ACCEPT_CONN_REQ_CP_SIZE, &ac);
}
/* Event Handling */
static void billotooth_event_packet(struct sk_buff *skb)
{
hci_event_hdr *he = (hci_event_hdr *) skb->data;
//evt_cmd_status *cs;
evt_cmd_complete *ec;
__u16 opcode, ocf, ogf;
skb_pull(skb, HCI_EVENT_HDR_SIZE);
BT_DBG("%s evt 0x%x", mydev->name, he->evt);
switch (he->evt) {
case EVT_INQUIRY_RESULT:
billotooth_inquiry_result_evt(skb);
break;
case EVT_CONN_REQUEST:
billotooth_conn_request_evt(skb);
break;
case EVT_CONN_COMPLETE:
billotooth_conn_complete_evt(skb);
break;
case EVT_DISCONN_COMPLETE:
hci_disconn_complete_evt(skb);
break;
case EVT_CMD_COMPLETE:
ec = (evt_cmd_complete *) skb->data;
skb_pull(skb, EVT_CMD_COMPLETE_SIZE);
```

```
opcode = __le16_to_cpu(ec->opcode);
ogf = cmd_opcode_ogf(opcode);
ocf = cmd_opcode_ocf(opcode);
switch (ogf) {
case OGF_INFO_PARAM:
billotooth_cc_info_param(ocf, skb);
break;
case OGF_HOST_CTL:
billotooth_cc_host_ctl(ocf, skb);
break;
default:
BT_ERR("EVT_CMD_COMPLETE not handled");
break;
};
break;
default:
BT_ERR("Event not handled");
break;
};
}
/*----------------------------------------------------------------------------------------
INTERFACE TO UPPER LAYERS
----------------------------------------------------------------------------------------*/
int billotooth_register_mod(char * desc, int * id, int (*preceptor)(char * data))
{
BT_DBG("Registering module: %s",desc);
list_receptors[num_receptors] = preceptor;
num_receptors++;
billotooth_inq_req();
return 0;
}
int billotooth_unregister_mod(void)
{
BT_DBG("Unregistering module");
//TODO: reorganize list order
num_receptors--;
return 0;
}
/*----------------------------------------------------------------------------------------
INITIALIZATION ROUTINES
----------------------------------------------------------------------------------------*/
/* Open device. When interface this is called when the hci interface goes up */
int billotooth_dev_open(void)
{
int ret = 0;
set_event_flt_cp ef;
```

```
__u16 param;
/* Verify if interface is already open */
if (test_bit(HCI_UP, &mydev->flags)) {
ret = -EALREADY;
goto done;
}
/* Lower layer opening */
if (mydev->open(mydev)) {
ret = -EIO;
goto done;
}
/* Now, the interface is up */
set_bit(HCI_UP, &mydev->flags);
/* Mandatory initialization */
/* Read Local Supported Features */
billotooth_send_cmd(OGF_INFO_PARAM, OCF_READ_LOCAL_FEATURES, 0, NULL);
/* Read Buffer Size (ACL mtu, max pkt, etc.) */
billotooth_send_cmd(OGF_INFO_PARAM, OCF_READ_BUFFER_SIZE, 0, NULL);
#if 0
/* Host buffer size */
{
host_buffer_size_cp bs;
bs.acl_mtu = __cpu_to_le16(HCI_MAX_ACL_SIZE);
bs.sco_mtu = HCI_MAX_SCO_SIZE;
bs.acl_max_pkt = __cpu_to_le16(0xffff);
bs.sco_max_pkt = __cpu_to_le16(0xffff);
billotooth_send_cmd(OGF_HOST_CTL, OCF_HOST_BUFFER_SIZE,
HOST_BUFFER_SIZE_CP_SIZE, &bs);
}
#endif
/* Read BD Address */
billotooth_send_cmd(OGF_INFO_PARAM, OCF_READ_BD_ADDR, 0, NULL);
/* Optional initialization */
/* Clear Event Filters */
ef.flt_type = FLT_CLEAR_ALL;
billotooth_send_cmd(OGF_HOST_CTL, OCF_SET_EVENT_FLT, 1, &ef);
/* Page timeout ~20 secs */
param = __cpu_to_le16(0x8000);
billotooth_send_cmd(OGF_HOST_CTL, OCF_WRITE_PG_TIMEOUT, 2, &param);
/* Connection accept timeout ~20 secs */
param = __cpu_to_le16(0x7d00);
billotooth_send_cmd(OGF_HOST_CTL, OCF_WRITE_CA_TIMEOUT, 2, &param);
/* Inquiry and Page scans */
param = 3;
billotooth_send_cmd(OGF_HOST_CTL, OCF_WRITE_SCAN_ENABLE, 1, &param);
/* Initializes inquiry device list */
```

```
inq_cache.list = NULL;
/* Init list of receptors */
num_receptors = 0;
/* Inquiry devices nearby */
billotooth_inq_req();
done:
return ret;
}
/* Register HCI device - Interface to the lower stack layer */
int billotooth_register_dev(struct hci_dev *hdev)
{
int id = 0;
BT_DBG("Registering device");
/* Check for lower layer routines presence */
if (!hdev->open || !hdev->close || !hdev->destruct)
return -EINVAL;
/* Set device parameters */
sprintf(hdev->name, "hci%d",id);
hdev->id = id;
atomic_set(&hdev->refcnt, 1);
hdev->flags = 0;
hdev->pkt_type = (HCI_DM1 | HCI_DH1 | HCI_HV1);
hdev->link_mode = (HCI_LM_ACCEPT);
memset(&hdev->stat, 0, sizeof(struct hci_dev_stats));
atomic_set(&hdev->promisc, 0);
MOD_INC_USE_COUNT;
mydev = hdev;
billotooth_dev_open();
conn_hash_init(mydev);
return id;
}
/* Unregister HCI device - Interface to the lower stack layer */
int billotooth_unregister_dev(struct hci_dev *hdev)
{
BT_DBG("Unregistering device");
MOD_DEC_USE_COUNT;
return 0;
}
int billotooth_init(void)
{
BT_DBG("*** BILLOTOOH PROTOCOL STACK IS RUNNING ***");
return 0;
}
void billotooth_cleanup(void)
{
BT_DBG("Performing cleanup");
```

```
billotooth_inquiry_list_destroy();
}
module_init(billotooth_init);
module_exit(billotooth_cleanup);
```

ANEXO B – Código do módulo para controle centralizado

```
#include <linux/config.h>
#include <linux/module.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/sched.h>
#include <linux/unistd.h>
#include <linux/types.h>
#include <linux/interrupt.h>
#include <linux/slab.h>
#include <linux/errno.h>
#include <linux/string.h>
#include <linux/skbuff.h>
#include <linux/kmod.h>
#include <sys/io.h>
#include "Bluetooth.h"
#include "hci_core.h"
int id;
#define BT_ENABLE_APP 1
#ifdef BT_ENABLE_APP
#define BT_APP(fmt, arg...) printk(fmt , ## arg)
#else
#define BT_APP(D...)
#endif
#define REMOTE_ADDRESS {{0x88, 0x92, 0x09, 0x57, 0x60, 0x00}}
#define MOD_NAME "Mod_automatic"
#define MAJOR 155
#define MAX_NUM_STATES 10
#define MAX_NUM_TRANSITIONS 10
typedef struct {
char inevent;
char outevent;
void (*handle) (void);
int dest_state;
} ttransition;
typedef struct {
ttransition transition[MAX_NUM_TRANSITIONS];
```

```
int num_transitions;
char desc[30];
} tstates;
struct hci_conn * conn;
int current_state = 0;
tstates states[MAX_NUM_STATES];
struct hci_conn * conn; //as a prototype, it first works with a single connection
static int mod_send(char * data);
static int define_protocols(void);
static ssize_t mod_write(struct file *pFile, const char *pData, size_t count, loff_t *off);
static struct file_operations mod_fops = {
write: mod_write,
};
char *batostr(bdaddr_t *ba)
{
static char str[2][18];
static int i = 1;
i ^= 1;
sprintf(str[i], "%2.2X:%2.2X:%2.2X:%2.2X:%2.2X:%2.2X",
ba->b[0], ba->b[1], ba->b[2],
ba->b[3], ba->b[4], ba->b[5]);
return str[i];
}
int handle_event(char inevent) {
int i;
char outdata[10];
for (i=0;i<states[current_state].num_transitions;i++) {
if (states[current_state].transition[i].inevent == inevent) {
if (states[current_state].transition[i].handle != 0) {
states[current_state].transition[i].handle(); //call handle function
}
if (states[current_state].transition[i].outevent != 0) {
outdata[0] = states[current_state].transition[i].outevent;
outdata[1] = 0;
mod_send(outdata);
}
BT_MODAUTOMATIC("Estado Atual: %d Evento entrada: %c ",current_state,inevent);
current_state = states[current_state].transition[i].dest_state;
BT_MODAUTOMATIC("Estado Posterior %d Funcao/Evento saida:
%p/%c\n",current_state,states[current_state].transition[i].handle,outdata[0]);
return 0; //exits function
}
}
return 0;
}
int mod_recv(char * data)
```

```
{
BT_DBG("Module %d received frame. Data[0] = %c",id,data[0]);
handle_event(data[0]);
return 0;
}
static int mod_send(char * data)
{
billotooth_send_acl(conn, data,1);
return 0;
}
static ssize_t mod_write(struct file *pFile, const char *pData, size_t count, loff_t *off)
{
BT_DBG("Writing %c, %d bytes\n", pData[0], count );
handle_event(pData[0]);
return count;
}
int mod_automatic_init(void)
{
int err=0;
char desc[100] = "*** MODULO BLUETOOTH: CONTROLE AUTOMATICO DISPOSITI-
VOS ELETRONICOS
***";
bdaddr_t address = (bdaddr_t) REMOTE_ADDRESS;
BT_DBG("%s",desc);
billotooth_register_mod(desc,&id,mod_recv);
conn = billotooth_connect(&address);
register_chrdev(MAJOR,MOD_NAME,&mod_fops);
define_protocols();
return err;
}
void mod_automatic_cleanup(void)
{
billotooth_acl_disconnect(conn,0x13);
unregister_chrdev(MAJOR,MOD_NAME);
billotooth_unregister_mod();
}
module_init(mod_automatic_init);
module_exit(mod_automatic_cleanup);
```

ANEXO C – Código da aplicação (no computador central)

```
void mostraLigado(void) {
```

```
char str[100] = "Ar condicionado esta ligado";
BT_APP("%s\n",str);
}
void mostraDesligado(void) {
char str[100] = "Ar condicionado esta desligado";
BT_APP("%s\n",str);
}
void ligacaoRealizada(void) {
char str[100] = "Ar condicionado foi ligado";
BT_APP("%s\n",str);
}
void desligamentoRealizado(void) {
char str[100] = "Ar condicionado foi desligado";
BT_APP("%s\n",str);
}
void ausenciaDetectada(void) {
char str[100] = "*** RECINTO DESOCUPADO ***";
BT_APP("%s\n",str);
}
void presencaDetectada(void) {
char str[100] = "*** PRESENCA DETECTADA ***";
BT_APP("%s\n",str);
}
//As a testing code, 2 applications will be designed in the code below... To simplify, they
use the same state// machine
static int define_protocols(void)
{
/*
CONTROLE AR CONDICIONADO
Eventos:
A- Usuario solicita status ar condicionado
B- Usuario solicita que ar condicionado seja ligado
C- Usuario solicita que ar condicionado seja desligado
D- Ar condicionado esta ligado
E- Ar condicionado esta desligado
F- Reconhecimento de que ar condicionado foi ligado
G- Reconhecimento de que ar condicionado foi desligado
CONTROLE VIGILANCIA
Eventos:
M- Presenca de pessoa detectada
N- Ausencia de pessoa detectada
*/
//IDLE
states[0].transition[0].inevent = 'A';
states[0].transition[0].outevent = 'A';
states[0].transition[0].handle = 0;
```

```
states[0].transition[0].dest_state = 1;
states[0].transition[1].inevent = 'B';
states[0].transition[1].outevent = 'B';
states[0].transition[1].handle = 0;
states[0].transition[1].dest_state = 2;
states[0].transition[2].inevent = 'C';
states[0].transition[2].outevent = 'C';
states[0].transition[2].handle = 0;
states[0].transition[2].dest_state = 2;
states[0].transition[3].inevent = 'M';
states[0].transition[3].outevent = 0;
states[0].transition[3].handle = presencaDetectada;
states[0].transition[3].dest_state = 0;
states[0].transition[4].inevent = 'N';
states[0].transition[4].outevent = 0;
states[0].transition[4].handle = ausenciaDetectada;
states[0].transition[4].dest_state = 0;
states[0].num_transitions = 5;
//WAIT STATUS
states[1].transition[0].inevent = 'D';
states[1].transition[0].outevent = 0;
states[1].transition[0].handle = mostraLigado;
states[1].transition[0].dest_state = 0;
states[1].transition[1].inevent = 'E';
states[1].transition[1].outevent = 0;
states[1].transition[1].handle = mostraDesligado;
states[1].transition[1].dest_state = 0;
states[1].transition[2].inevent = 'M';
states[1].transition[2].outevent = 0;
states[1].transition[2].handle = presencaDetectada;
states[1].transition[2].dest_state = 1;
states[1].transition[3].inevent = 'N';
states[1].transition[3].outevent = 0;
states[1].transition[3].handle = ausenciaDetectada;
states[1].transition[3].dest_state = 1;
states[1].num_transitions = 4;
//WAIT ACK
states[2].transition[0].inevent = 'F';
states[2].transition[0].outevent = 0;
states[2].transition[0].handle = ligacaoRealizada;
states[2].transition[0].dest_state = 0;
states[2].transition[1].inevent = 'G';
states[2].transition[1].outevent = 0;
states[2].transition[1].handle = desligamentoRealizado;
states[2].transition[1].dest_state = 0;
states[2].transition[2].inevent = 'M';
```

```
states[2].transition[2].outevent = 0;
states[2].transition[2].handle = presencaDetectada;
states[2].transition[2].dest_state = 2;
states[2].transition[3].inevent = 'N';
states[2].transition[3].outevent = 0;
states[2].transition[3].handle = ausenciaDetectada;
states[2].transition[3].dest_state = 2;
states[2].num_transitions = 4;
current_state = 0;
return 0;
}
```

Código fonte retirado do artigo: Uma pilha de protocolos Bluetooth adaptável à aplicação por Eduardo Afonso Billo.