PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Análise de Desempenho do Tráfego de Dados Assíncronos sobre *Bluetooth*

Carlos Giovanni Nunes de Carvalho

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://www.cin.ufpe.br
ftp://ftp.cin.ufpe.br/pub/posgrad

Recife (PE), fevereiro/2003

UNIVERSIDADE FEDERAL DE PERNAMBUCO – UFPE
CENTRO DE INFORMÁTICA – CIN
MESTRADO EM CIÊNCIAS DA COMPUTAÇÃO

Análise de Desempenho do Tráfego de Dados Assíncronos sobre *Bluetooth*

Carlos Giovanni Nunes de Carvalho

Dissertação apresentada como parte dos requisitos para obtenção do grau de Mestre em Ciências da Computação.
Área de Concentração: Redes de Computadores
Orientador: Prof. Dr. Djamel F. H. Sadok

Recife (PE), fevereiro/2003

Agradecimentos

Agradeço em primeiro lugar a Deus, por ter permitido que eu cumprisse mais uma etapa da minha vida, entre outras que virão.

Aos meus pais, Antônio Carlos e Hercília, que me deram força para que se tornasse possível à participação no mestrado.

Aos meus irmãos Joselisse, Clarisse e Gilvan, que sempre estiveram prontos para me ajudar e contribuíram nas atividades do curso.

A minha namorada Fabíola, que incentivou, torceu e apoiou, motivando-me cada vez mais para concluir os meus trabalhos.

A meu orientador, Prof. Dr. Djamel Sadok, que me guiou e mostrou a importância desse trabalho na minha vida.

A Prof. Dra. Judith Kelner, que me ajudou no desenvolvimento da dissertação.

Aos meus amigos, que me apoiaram e acompanharam todas as etapas desse trabalho.

Aos amigos da FAPEPI, AESPI e UESPI, por terem entendido a importância desse trabalho e a necessidade de me ausentar algumas vezes.

Índice

CAPÍTULO 1.0 – INTRODUÇÃO 1

1.1. MOTIVAÇÃO 2
1.2. CARACTERIZAÇÃO DO PROBLEMA 3
1.3. TRABALHOS RELACIONADOS 3
1.4. ESTRUTURA DA DISSERTAÇÃO 3

CAPÍTULO 2.0 – TECNOLOGIA *BLUETOOTH* 5

2.1. VISÃO GERAL 6
2.1.1. APLICAÇÕES *BLUETOOTH* 7
2.1.2. DOCUMENTOS DA PADRONIZAÇÃO *BLUETOOTH* 8
2.1.3. ARQUITETURA DE PROTOCOLOS 9
2.1.4. MODELOS DE USO 11
2.1.5. PICONETS E *SCATTERNETS* 12
2.2. ESPECIFICAÇÃO DO RÁDIO 14
2.3. ESPECIFICAÇÃO DA BANDA BÁSICA 15
2.3.1. SALTOS DE FREQÜÊNCIA 16
2.3.2. LINKS FÍSICOS 18
2.3.3. PACOTES 20
2.3.3.1. Código de Acesso 20
2.3.3.2. Cabeçalho do Pacote 22
2.3.3.3. Formato da Carga Útil 25
2.3.4. CORREÇÃO DE ERRO 26
2.3.5. CANAIS LÓGICOS 28
2.3.6. CONTROLE DE CANAL 29
2.3.6.1. Procedimentos de Investigação 30
2.3.6.2. Procedimento de Paginação 31
2.3.6.3. Estado de Conexão 32
2.3.7. ÁUDIO 33
2.3.8. SEGURANÇA 33
2.4. ESPECIFICAÇÃO DO *LINK MANAGER* 34
2.5. ESPECIFICAÇÃO DO CONTROLE LÓGICO DO *LINK* E PROTOCOLO DE ADAPTAÇÃO 35
2.5.1. CANAIS *L2CAP* 36
2.5.2. PACOTES *L2CAP* 37
2.5.3. MENSAGENS DE SINALIZAÇÃO 38
2.5.4. QUALIDADE DE SERVIÇO (QOS) 40
***2.6. IP* SOBRE BLUETOOTH 43**
2.7. CONSIDERAÇÕES FINAIS 44

CAPÍTULO 3.0 – OTIMIZAÇÃO DO *LINK* ASSÍNCRONO 45

3.1. VISÃO GERAL 46
3.2. ESQUEMAS DE SEGMENTAÇÃO E REMONTAGEM (*SAR*) 46

3.2.1. PROCEDIMENTOS *SAR* 46
3.2.1.1. Procedimentos de Segmentação 47
3.2.1.2. Procedimentos de Remontagem 48
3.2.2. ALGORITMOS *SAR* 49
3.2.2.1. SAR – BF (Best Fit) 49
3.2.2.2. SAR – OSU (Optimum Slot Utilization) 50
3.2.2.3. SAR – Randômico 50
3.3 VARIANTES DO *TCP* 51
3.3.1. MECANISMOS DE CONTROLE DE CONGESTIONAMENTO DO *TCP* 52
3.3.1.1. Temporizador 52
3.3.1.2. Tempo de Ida e Volta *(RTT)* 52
3.3.1.3. Começo Lento (*Slow Start*) 53
3.3.1.4. Evitar Congestionamento (*Congestion Avoidance*) 54
3.3.1.5. Retransmissão Rápida (*Fast Retransmit*) 54
3.3.1.6. Recuperação Rápida (*Fast Recovery*) 55
3.3.2. OPÇÃO DE RECONHECIMENTOS SELETIVOS 55
3.3.2.1. Comportamento do Receptor 55
3.3.2.2. Comportamento do Transmissor 56
3.3.3. IMPLEMENTAÇÕES DO *TCP* 56
3.3.3.1. Tahoe 57
3.3.3.2. Reno 57
3.3.3.3. New Reno 58
3.3.3.4. SACK 58
3.3.3.5. Vegas 59
3.4. OUTROS FATORES QUE INFLUENCIAM O DESEMPENHO DO *LINK* 60
3.4.1. OTIMIZAÇÃO DO TAMANHO DE *BUFFER* 60
3.4.2. ALGORITMOS DE ESCALONAMENTO 60
3.4.3. MANIPULAÇÃO DE ERROS 61
3.4.4. NÚMERO DE CONEXÕES *SCO* 61
3.4.5. MECANISMOS DE *QoS* 62
3.5. CONSIDERAÇÕES FINAIS 62

CAPÍTULO 4.0 – ANÁLISE DE DESEMPENHO DO TRÁFEGO DE DADOS ASSÍNCRONOS 64

4.1. TÉCNICA DE AVALIAÇÃO DE DESEMPENHO 65
4.2. AMBIENTE DE SIMULAÇÃO 65
4.3. MÉTRICAS 65
4.4. TOPOLOGIA E PARÂMETROS DA SIMULAÇÃO 66
4.5. RESULTADOS 69

CAPÍTULO 5.0 – CONCLUSÃO 75

5.1. CONSIDERAÇÕES FINAIS 76
5.2. CONTRIBUIÇÕES 77
5.3. TRABALHOS FUTUROS 77
5.4. REFERÊNCIAS BIBLIOGRÁFICAS 79

ANEXOS 82

***SCRIPTS* DA SIMULAÇÃO 83**
CÓDIGO ALTERADO 84

Índice de Figuras

Figura 1: Pilha de Protocolos *Bluetooth* 9
Figura 2: *Piconet* e Scatternet 13
Figura 3: *TDD* e *Timing* 17
Figura 4: Pacotes *Multislots* 17
Figura 5: Formato do Pacote Padrão 20
Figura 6: Formato do Cabeçalho 22
Figura 7: Formato do Pacote *DV* 23
Figura 8: Diagrama de Estados do *LC* 30
Figura 9: Arquitetura de Protocolos *Bluetooth* 35
Figura 10: *L2CAP* dentro das Camadas de Protocolos 36
Figura 11: Canais entre Dispositivos 37
Figura 12: Segmentação L2CAP 47
Figura 13: Primeiro Cenário da Simulação (Variantes do *TCP*) 67
Figura 14: Segundo Cenário da Simulação (Tráfego *CBR*) 68
Figura 15: Atraso com *SAR-BF* e *TCP New Reno* 70
Figura 16: Vazão com *SAR-BF* e *TCP New Reno* 72

Índice de Tabelas

Tabela 1: Parâmetros de Rádio e Banda Básica do *Bluetooth* 14
Tabela 2: Alocações de Freqüências Internacionais *Bluetooth* 15
Tabela 3: Taxa de Dados Possíveis no *Link ACL* 19
Tabela 4: Tipos de Código de Acesso 21
Tabela 5: Tipos de Pacotes *Bluetooth* 24
Tabela 6: Parâmetros do Cenário 1 – Variantes do *TCP* 66
Tabela 7: Parâmetros do Cenário 2 – Tráfego *CBR* 67
Tabela 8: Parâmetros *QoS* da Simulação 69
Tabela 9: Distribuição do Atraso com *SAR-BF* e Variantes do *TCP* 70
Tabela 10: Distribuição do Atraso com *SAR-OSU* e Variantes do *TCP* 71
Tabela 11: Distribuição de Atraso com *SAR*-Randômico e Variantes do *TCP* 71
Tabela 12: Vazão com *SAR-BF* e Variantes do *TCP* 73
Tabela 13: Vazão com *SAR-OSU* e Variantes do *TCP* 73
Tabela 14: Vazão com *SAR*-Randômico e Variantes do *TCP* 73
Tabela 15: Perda de pacotes (%) em todos os algoritmos *SAR* e Variantes do *TCP* 74

Lista de Acrônimos

ACK	*Acknowledgment*
ACL	*Link Asynchronous Connection-Less*
AM_ADDR	*Active Member Address*
ARQ	*Automatic Repeat reQuest*
BCH	Tipo de código *Bose, Chaudhuri & Hocquenghem.* As pessoas a qual descobriram este código em 1959 *(H)* e 1960 *(B&C)*
BD_ADDR	*Bluetooth Device Address*
BE	*Best Effort*
BER	*Bit Error Rate*
BT	*Bluetooth*
CAC	*Channel Access Code*
CBR	*Constant Bit Rate*
CL	*ConnectionLess*
CRC	*Cyclic Redundancy Check*
CVSD	*Continuous Variable Slope Delta Modulation*
CWND	*Congestion Window*
DAC	*Device Access Code*
DH	*Data-High Rate Data* tipo de pacote para alta taxa de dados
DIAC	*Dedicated Inquiry Access Code*
DM	*Data-Medium Rate* tipo de pacote para taxa média de dados
DV	*Data Voice* tipo de pacote para dados e voz
ETSI	*European Telecommunications Standards Institute*
FEC	Código *Forward Error Correction*
FH	*Frequency Hopping*
FHS	*Frequency Hop Synchronization*
FSK	*Frequency Shift Keying* tipo de modulação
FTP	*File Transfer Protocol*
GFSK	*Gaussian Frequency Shift Keying*
GIAC	*General Inquiry Access Code*
HCI	*Host Controller Interface*
HEC	*Header Error Check*
HTTP	*Hyper Text Transfer Protocol*
HV	*High quality Voice* (pacote HV1)
IAC	*Inquiry Access Code*
IEEE	*Institute of Electronic and Electrical Engineering*
IP	*Internet Protocol*
IrDA	*Infra-red Data Association*
IrMC	*Ir Mobile Communications*
ISDN	*Integrated Services Digital Networks*
ISM	*Industrial, Scientific, Medical*
L_CH	*Logical Channel*
L2CAP	*Logical Link Control and Adaption Protocol*
LAN	*Local Área Network*
LAP	*Lower Address Part*
LC	*Link Controller* (ou *Baseband*) parte da pilha de protocolos *Bluetooth* manipulador de protocolo da camada inferior *Baseband*
LCP	*Link Control Protocol*

LM	*Link Manager*
LMP	*Link Manager Protocol* para comunicações *LM peer to peer*
LSB	*Least Significant Bit*
MAC	*Medium Access Control*
MSB	*Most Significant Bit*
MTU	*Maximum Transmission Unit*
NAK	*Negative Acknowledgment*
NS	*Network Simulator*
OBEX	Protocolo *OBject EXchange*
OSI	*Organization Standard International*
OTCL	*Object Tool Control Language*
PAN	*Personal Área Network*
PC	*Personal Computer*
PCM	*Pulse Coded Modulation*
PDA	*Personal Digital Assistent*
PDU	*Protocol Data Unit*
PIN	*Personal Identification Number*
PM_ADDR	*Parked Member Address*
PPP	*Point-to-Point Protocol*
PSTN	*Public Switched Telephone Network*
QoS	*Quality of Service*
RAND	*Random number*
RF	*Radio Frequency*
RFC	*Request For Comments*
RFCOMM	Protocolo emulador de cabo *serial* baseado no *ETSI TS 07.10*
RSSI	*Received Signal Strength Indication*
RTT	*Round Trip Time*
RX	*Receiver*
SAR	*Segmentation and Reassembly*
SCO	*Link Synchronous Connection-Oriented*
SDP	*Service Discovery Protocol*
SEQN	Esquema *Sequential Numbering*
SIG	*Special Interest Group*
SRES	*Signed Response*
SSTHRESH	*Slow Start Threshold*
TCL	*Tool Control Language*
TCP	*Transmission Control Protocol*
TCP/IP	*Transport Control Protocol/Internet Protocol*
TCS	Especificação *Telephony Control Protocol*
TDD	*Time Division Duplex*
TX	*Transmit*
UA	Dados *User Asynchronous*
UDP	*User Datagram Protocol*
UDP/IP	*User Datagram Protocol/Internet Protocol*
UI	Dados de usuário *User Isochronous*
US	Dados *User Synchronous*
WAN	*Wide Area Network*
WAP	*Wireless Application Protocol*
xDSL	*Digital Subscribe Line*

Resumo

Aplicações de dados, que funcionam sobre o *Bluetooth*, tais como *HTTP*, *FTP* e *real audio,* irão necessitar de protocolos da camada de transporte da arquitetura *TCP/IP* (*TCP* e *UDP*) para enviar pacotes sobre os *links wireless*. Para aumentar o desempenho destas aplicações é preciso otimizar os recursos existentes, que na maioria dos dispositivos para este tipo de tecnologia são escassos.

Alguns fatores influenciam o desempenho do tráfego de dados sobre estes dispositivos, normalmente, de tamanho reduzido e pouca autonomia de uso. Com isso, há uma crescente tentativa de torná-los mais eficientes, através do melhor aproveitamento de seus recursos.

Este trabalho contempla a análise e o aumento de performance do tráfego de dados usando o serviço assíncrono *Bluetooth*. Mostramos, através dos resultados, que o melhor algoritmo de Segmentação e Remontagem *(SAR)* é o de Melhor Ajuste *(BF)*, que na média, apresenta maior vazão, menor atraso e variação de atraso nas transmissões e menor porcentagem de perda de pacotes. Quanto as variantes do *TCP*, o *Vegas* apresentou-se a melhor escolha, com uma vazão média maior, uma menor porcentagem de perda de pacotes, menor atraso e variação de atraso, mas devido não haver implementações dessa variante, a solução mais conveniente é a utilização do *New Reno*, uma vez que utilizado com o algoritmo *SAR-BF* obteve melhor desempenho que os demais.

Palavras chave: *Wireless*, *Bluetooth*, *Ad-hoc*, Variantes do *TCP*, *Segmentação e Remontagem (SAR).*

Abstract

Applications of data, that work over Bluetooth, such as HTTP, FTP and real audio, they will need protocols of the layer of transport of the architecture TCP/IP (TCP and UDP) to send packages on the links wireless. To increase the performance of these applications it is necessary to optimize the existent resources, that in most of the devices for this technology tipo are scarce.

Some factors influence the performance of the traffic of data on these devices, usually, of reduced size and little use autonomy. With that, there is a tentative crescent of turning them more efficient, through of the best use of their resources.

This work contemplates the analysis and the increase of performance of the traffic of data using the asynchronous service Bluetooth. We showed, through the results, that the best algorithm of Segmentation and Reassembly (SAR) is the Best Fit (BF), that in the average, it presents larger throughput, smaller delay and variation of delay in the transmissions and smaller percentage of loss of packages. As the variants of TCP, Vegas came the best choice, with a larger medium throughput, a smaller percentage of loss of packages, smaller delay and variation of delay, but should not have implementations of that variant, the most convenient solution is New Reno's use, once used with the algorithm SAR-BF obtained better performance than the others.

Key Words: Wireless, Bluetooth, Ad-hoc, Variants of TCP, Segmentation and Reassembly (SAR).

Capítulo 1.0 – Introdução

Este capítulo apresenta a motivação para este trabalho, o problema proposto, os trabalhos relacionados com o assunto e a estrutura da dissertação.

1.1. Motivação

O Bluetooth[1] é capaz de conectar uma grande variedade de dispositivos como o *Personal Digital Assistant (PDA)*, telefones móveis, *headsets*, *desktops*, *notebooks*, câmeras digitais e outros. Isto o torna um ambiente adequado para a formação de *Personal Area Netwoking (PAN)*. As aplicabilidades do *Bluetooth* incluem, entre outras: eliminar cabos entre dispositivos como *PCs*, impressoras, modem, projetores; auto sincronização entre *PDAs* e *PCs*; conexão sem fio em Redes Locais (*LANs)* através de pontos de acesso e a *Internet* pelos telefones móveis.

O Bluetooth [1] [2] opera na faixa de 2,4 GHz da banda *ISM*, o canal físico é definido por uma Seqüência de Saltos de Freqüência, para o qual os dispositivos de uma única rede, chamada de *piconet*, são "sintonizados". Os dispositivos na mesma *piconet* compartilham este canal em uma base de divisão do tempo.

O Bluetooth tem um número de características [2] distintas, comparadas a outras *LANs wireless* existentes, tais como:

- Suporte para tráfego de dados e voz;
- Saltos de Freqüência para evitar interferência;
- Um dispositivo Mestre com sistema *Time Division Duplex (TDD)* na camada Controle de Acesso ao Meio *(MAC)* para suportar transmissões *full duplex*;
- Segmentação e Remontagem *(SAR)* para manipular pacotes de dados grandes;
- Suporte ao nível de *link,* dos esquemas Requisição de Repetição Automática *(ARQ)* e *Forward Error Correction (FEC)*.

[1] O nome veio do Rei Harald Blaatand *(Bluetooth)* da Dinamarca, que viveu no século 10 D.C. Diferente de seus antecedentes Viking, o Rei Harald possuía cabelos escuros (dai o nome *Bluetooth*, significando um estado das coisas escuras) e acredita-se que trouxe o Cristianismo para Scandinava unificando Dinamarca e Noruega. A marca *blue,* que identifica os dispositivos com *Bluetooth,* é derivada do alfabeto escandinavo antigo.

1.2. Caracterização do Problema

Essas características terão um impacto significante no desempenho do tráfego de dados sobre *Bluetooth*. A fragmentação de pacotes de dados grandes executando *SAR*, a qual permite a transmissão em pequenos pacotes Banda Básica, pode aumentar seus atrasos fim a fim. O dispositivo Mestre, ao realizar escalonamento no nível *MAC* afetará na vazão e no atraso do enfileiramento. A taxa de sucesso da transmissão de dados será afetada pela presença de mecanismos *FEC* e *ARQ* ao nível do *link*. A largura de banda disponível para o tráfego de dados será reduzida na presença de conexões de voz. As classes de Qualidade de Serviço (*QoS)* do tráfego, podem aumentar a utilização da largura de banda do canal.

O efeito destes assuntos precisa ser bem entendido para otimizar o desempenho do tráfego de dados assíncronos sobre *Bluetooth*.

1.3. Trabalhos Relacionados

A análise de desempenho do tráfego de dados assíncronos sobre redes *Bleutooth* é de muita importância para um melhor aproveitamento dos recursos disponíveis e das características constantes na especificação.

Os trabalhos relacionados com este assunto e importantes para a conclusão do mesmo podem ser encontrados em [3], [4] e [5].

1.4. Estrutura da Dissertação

A dissertação está estruturada da seguinte maneira:

O Capítulo 1 – Introdução, apresenta a motivação que determina a elaboração deste trabalho, caracteriza os problemas relacionados e cita os trabalhos relacionados ao assunto.

O Capítulo 2 – Tecnologia *Bluetooth*, traz as características e especificações técnicas da tecnologia, necessárias para se estudar o comportamento dos *links* de dados assíncronos em dispositivos *Bluetooth*.

O Capítulo 3 – Otimização do *Link* Assíncrono, descreve os fatores que influenciam o desempenho do tráfego de dados assíncrono sobre os *links Bluetooth*, dando ênfase aos algoritmos de Segmentação e Remontagem *(SAR)* e as variantes do *TCP*.

O Capítulo 4 – Análise de Desempenho do Tráfego de Dados Assíncronos, avalia o desempenho do tráfego de dados assíncronos, sob a influência de dois fatores (*SAR* e variantes de *TCP*) que interferem na otimização dos *links*.

O Capítulo 5 – Conclusão, apresenta o resultado final da análise e testes realizados neste trabalho. São apontados trabalhos futuros e referências bibliográficas utilizadas.

Capítulo 2.0 – Tecnologia *Bluetooth*

Este capítulo descreve a especificação do *Bluetooth*, dando ênfase nos fatores que influenciam o desempenho do tráfego de dados sobre a tecnologia.

2.1. Visão Geral

O *Bluetooth* é um rádio de transmissão simultânea de curto alcance que geralmente reside em um *microchip*. Ele foi inicialmente desenvolvido pela fabricante sueca de telefone móvel Ericsson em 1994 [6] como uma maneira de permitir computadores *palmtop* realizar chamadas sobre um telefone móvel. Desde então, mais de mil companhias se registraram para fazer do *Bluetooth* um padrão *wireless* de baixa potência e curto alcance para uma grande quantidade de dispositivos. Observadores da indústria estimam que o *Bluetooth* será instalado em bilhões de dispositivos até 2005 (Business Week, 18 de setembro de 2000).

O padrão *Bluetooth* foi publicado por um consórcio de indústrias conhecido como *Bluetooth SIG (*Grupo de Interesse Especial*)*.

O *Bluetooth* foi concebido para oferecer uma capacidade universal de se ter comunicação de dispositivos *wireless* de curto alcance. Usando a banda 2,4GHz *ISM*, disponível globalmente para uso sem licença, de baixa potência, dois dispositivos *Bluetooth* com 10m um do outro podem compartilhar até 720Kbps de velocidade na transmissão de dados [1]. O *Bluetooth* tem o objetivo de suportar uma lista ampla de aplicações, incluindo dados (compromissos e números de telefone), áudio, gráficos e vídeo. Por exemplo, dispositivos de áudio podem se comunicar com *handsets*, telefones sem fio e convencionais, *home stereos* e *digital MP3 players*. O *Bluetooth* possibilita pessoas a fazer, entre outras coisas [6]:

- Estabelecer chamadas de *headset wireless* conectado remotamente para um telefone celular.
- Eliminar cabos ligando computadores à impressora, teclado e mouse.
- Unir *MP3 players wireless* a outros dispositivos para fazer *download* de músicas.

- Configurar redes domésticas tais como; um usuário pelo controle remoto da televisão, pode manipular aparelhos de ar-condicionado, forno e até controlar a navegação da *Internet* dos seus filhos.
- Chamadas locais remotamente para ligar e desligar aparelhos, ativar alarmes e monitorar atividades.

2.1.1. Aplicações *Bluetooth*

O *Bluetooth* foi desenvolvido para operar em um ambiente de vários usuários. Até oito dispositivos podem comunicar-se em uma pequena rede chamada *piconet*. Dez dessas *piconets* podem coexistir no mesmo alcance de cobertura do radio *Bluetooth*. Para prover segurança, cada *link* pode ser codificado e protegido contra escuta e interferência.

O *Bluetooth* tem suporte para três áreas [6] de aplicações em geral, usando conectividade de curto alcance *wireless*:

- Pontos de Acesso de Dados e Voz: facilita transmissões de voz em tempo real e dados para estabelecer, sem esforço, conexões *wireless* de dispositivos de comunicação portáteis e fixos.
- Substituição de cabos: eliminam a necessidade de numerosos, freqüentemente proprietários, cabos para conexões de dispositivo de comunicação. As conexões são instantâneas e mantidas em um alcance de rádio de aproximadamente 10m, mas podem ser estendidas para 100m com um amplificador opcional.
- Redes *Ad-hoc*: um dispositivo equipado com um rádio *Bluetooth* pode estabelecer conexão instantaneamente para outro rádio *Bluetooth* à medida que ele vem aproximando-se da área de alcance.

2.1.2. Documentos da Padronização *Bluetooth*

A padronização *Bluetooth* apresenta um volume considerável de documentos: bem mais de 1500 páginas, divididas em dois grupos: núcleo e perfil. A especificação do núcleo descreve os detalhes das várias camadas da arquitetura de protocolos *Bluetooth*, da *interface* de rádio até o controle do *link*. Tópicos relacionados são abrangidos, tal como interoperabilidade com tecnologias agregadas, requisitos de teste e uma definição de vários temporizadores *Bluetooth* e seus valores associados.

As especificações de perfil se preocupam com o uso da tecnologia *Bluetooth* para suportar várias aplicações. Cada especificação de perfil discute o uso da tecnologia definida na especificação do núcleo para implementar um particular modelo de uso [6].

A especificação do perfil inclui uma descrição de quais aspectos da especificação do núcleo são obrigatórios, opcionais e não aplicáveis. O propósito da especificação do perfil é para definir um padrão de interoperabilidade, tal como os produtos de diferentes fabricantes, que irão trabalhar junto, devem suportar um dado modelo de uso. Em termos gerais, a especificação do perfil cai em uma das duas categorias: substituição de cabo ou áudio *wireless*.

O perfil de substituição de cabo provê um conveniente meio para conectar logicamente dispositivos em proximidade a outro e para trocar dados. Por exemplo, quando dois dispositivos vêm ao encontro um do outro, eles podem automaticamente consultar um ao outro pelo mesmo perfil. Isto pode ser feito através de um alerta ao usuário final do dispositivo ou uma troca de dados automática.

O perfil de áudio *wireless* preocupa-se com o estabelecimento de conexões de voz de alcance curto, como no caso dos *headset*, por exemplo.

2.1.3. Arquitetura de Protocolos

O Bluetooth foi definido por um número de protocolos residentes nas camadas física e *datalink,* no modelo *Organization Standard International (O-SI)*, como mostra a Figura 1 [1].

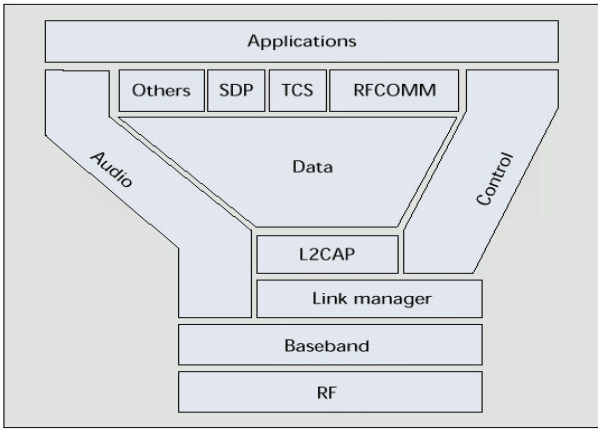


Figura 1: Pilha de Protocolos *Bluetooth*

A arquitetura de protocolos *Bluetooth* consiste de protocolos núcleo, substituição de cabos, protocolos de controle de telefone e protocolos adotados.

Os protocolos núcleo formam uma pilha de cinco camadas, consistindo dos seguintes elementos:

- Rádio: especifica detalhes da *interface* aérea, incluindo freqüência, o uso do Salto de Freqüência, esquema de modulação e potência de transmissão.
- Banda Básica: preocupa-se com estabelecimento de conexão em uma *piconet*, endereçamento, formato do pacote, temporização e controle de potência.
- Protocolo de Gerenciamento do *Link (LMP)*: responsável pelo estabelecimento do *link* entre dispositivos *Bluetooth* e existência do Gerenciador do *Link*. Isto inclui aspectos de segurança (autenticação e criptografia), controle e negociação do tamanho de pacotes Banda Básica.

- Controle do Link Lógico e Protocolo de Adaptação *(L2CAP)*: adapta protocolos das camadas superiores para a camada Banda Básica. O *L2CAP* oferece serviços de dados sem conexões e orientado à conexão.
- Protocolo de Descoberta de Serviço *(SDP)*: informação de dispositivos, serviços e as características dos serviços podem ser consultadas para habilitar o estabelecimento de uma conexão entre dois ou mais dispositivos *Bluetooth*.

O *RFCOMM* é o protocolo de substituição de cabos incluído na especificação *Bluetooth*. O *RFCOMM* apresenta uma porta serial virtual que foi desenvolvida para fazer substituição de tecnologias de cabo. Portas seriais são um dos tipos de *interfaces* de comunicação mais comuns, usadas por dispositivos de computação e comunicações. O *RFCOMM* habilita a substituição de cabos de portas seriais com o mínimo de modificações nos dispositivos existentes. O *RFCOMM* habilita o transporte de dados binários e emula sinais de controle *EIA-232* sobre a camada Banda Básica. O *EIA-232* (formalmente conhecido como *RS-232*) é um padrão de *interface* de porta serial largamente utilizado.

O *Bluetooth* especifica um protocolo de controle de telefone. O *TCS BIN* (Especificação do Controle de Telefone Digital) é um protocolo orientado a *bit* que define a sinalização do controle de chamadas para o estabelecimento de chamadas de conversa e de dados entre dispositivos *Bluetooth*. Além disso, ele define procedimentos de gerenciamento de mobilidade para manipular grupos de dispositivos *TCS Bluetooth*.

Outros protocolos adotados são definidos na especificação emitida por outras organizações fabricantes de padrões e incorporado na arquitetura completa do *Bluetooth*. A estratégia do *Bluetooth* é para desenvolver somente protocolos necessários e sempre que possível, usar padrões existentes. Outros protocolos adotados [1] são:

- *PPP*: o protocolo ponto-a-ponto é um protocolo padrão *Internet* para transportar datagramas *IP* sobre *link* ponto-a-ponto.
- *TCP/UDP/IP*: estes são os protocolos base da arquitetura de protocolos *TCP/IP*.
- *OBEX*: o Protocolo de Troca de Objetos é um protocolo de nível de sessão desenvolvido para a Associação de Dados Infravermelho *(IrDA)* para a troca de objetos. O *OBEX* provê funcionalidade similar ao *HTTP*, mas em um uso simples. Ele também provê um modelo para representar objetos e operações. Exemplos de formatos de conteúdos transferidos pelo *OBEX* são *vCard* e *vCalendar*, o qual provê o formato de um cartão de negócios eletrônico, calendário pessoal e informações de compromissos, respectivamente.
- *WAE/WAP*: o *Bluetooth* incorpora na arquitetura o ambiente de aplicações *wireless* e os protocolos de suas aplicações.

2.1.4. Modelos de Uso

Vários modelos de uso foram definidos nos documentos do perfil *Bluetooth*. Em essência, um modelo de uso é um conjunto de protocolos que implementam uma aplicação particular baseada em *Bluetooth*. Cada perfil define os protocolos e as características de protocolos suportados em um modelo de uso particular. A seguir, alguns modelos de uso [1]:

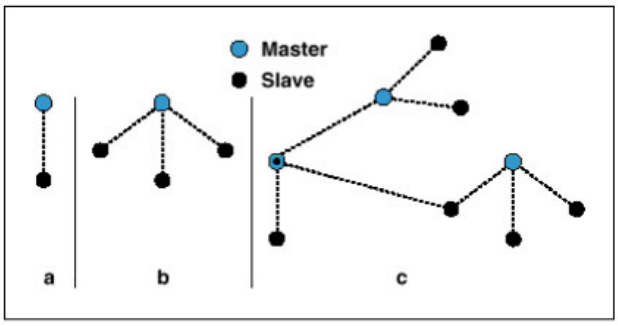
- Transferência de arquivos: suporta a transferência de diretórios, arquivos, documentos, imagens e formatos de mídia *streaming*. Este modelo de uso também inclui a capacidade de visualizar *folders* em dispositivo remoto.
- Ponte *Internet*: um *PC* pode ser conectado sem fio, para um telefone móvel ou um *modem* sem fio para realizar conexões *dial-up* e *fax*. Para rede *dial-up*, comandos *AT* são usados para controlar o telefone móvel ou modem e uma outra pilha de protocolo (ex: *PPP* sobre *RFCOMM*) é usada para transferir dados. Para transferir *fax*, o software de *fax* existente opera diretamente sobre *RFCOMM* de maneira transparente.

- Acesso a *LAN*: habilita dispositivos em uma *piconet* para acessar uma *LAN*. Uma vez conectado, um dispositivo funciona como se estivesse conectado diretamente (com fio) para uma *LAN*.
- Sincronização: provê uma sincronização de dispositivo para dispositivo, de informações *PIM* (Gerenciamento de Informações Pessoais), tal como uma agenda de telefone, calendário, mensagem e lembretes. O *IrMC* (Comunicações Móveis Infravermelho) é um protocolo *IrDA* que possibilita transações cliente/servidor para transferir atualizações de informações *PIM* de um dispositivo para outro.
- Telefone três em um: *handset* de telefone que implementa este modelo de uso pode atuar como um telefone sem fio, conectando para uma estação base de voz, como um dispositivo *intercom* para conectar para outros telefones e como um telefone celular.
- *Headset*: o *headset* pode atuar como um dispositivo de áudio de *interface* de entrada e saída remota, usado para ouvir música, por exemplo.

2.1.5. *Piconets* e *Scatternets*

Como foi mencionada, a unidade básica de rede em *Bluetooth* é uma *piconet*, consistindo de um mestre e de até sete dispositivos escravos ativos. O rádio, designado como mestre, faz a determinação do canal (Seqüência de Saltos de Freqüência) e a fase (compensação de tempo, ou seja, quando transmitir) que será usada por todos os dispositivos nesta *piconet*. O mestre faz esta determinação usando seu próprio endereço de dispositivo como um parâmetro, enquanto que o dispositivo escravo deve direcionar para o mesmo canal e fase. Um escravo, somente pode se comunicar com o mestre e quando for concedida a permissão. Um dispositivo em uma *piconet* pode também fazer parte de outra *piconet* e funcionar como um mestre ou um escravo em cada *piconet* (Figura 2) [1]. Esta forma de sobreposição é chamada de *scatternet*.

Um dispositivo escravo não pode ser mestre em mais de uma *piconet*.



a – *piconet* com um único escravo, b – *piconet* com multi escravos, c – *scatternet*

Figura 2: *Piconet* e Scatternet

A vantagem do esquema *piconet/scatternet* é que ele permite vários dispositivos compartilhar a mesma área física e fazer uso da largura de banda eficientemente. Um sistema *Bluetooth* [1] usa um esquema de Salto de Freqüência com um espaço de portadora de 1MHz. Tipicamente, até 79 diferentes freqüências são usadas para uma largura de banda total de 80MHz. Se o Salto de Freqüência não for usado, um único canal corresponderá para uma única banda de 1MHz. Com o Salto de Freqüência, um canal lógico é definido pela Seqüência de Saltos de Freqüência. Em qualquer momento, a largura de banda disponível é de 1MHz, com um máximo de oito dispositivos compartilhando-a. Canais lógicos diferentes (usando uma Seqüência de Saltos diferente) podem simultaneamente compartilhar os mesmos 80MHz de largura de banda. As colisões irão ocorrer quando dispositivos em diferentes *piconet*s, em diferentes canais lógicos, utilizam o mesmo Salto de Freqüência ao mesmo tempo. Quando o número de *piconets* em uma área aumenta, o número de colisões aumenta e degrada a performance. Em resumo, a área física e a largura de banda total são compartilhadas pela *scatternet*. O canal lógico e a transferência de dados são compartilhados por uma *piconet*.

2.2. Especificação do Rádio

A especificação do rádio *Bluetooth* é um documento curto que dá os detalhes básicos da transmissão de rádio para dispositivos *Bluetooth*. Alguns dos parâmetros chaves são resumidos na Tabela 1[1].

Topologia	**Até 7 *links* simultâneos em uma estrela lógica**
Modulação	*GFSK*
Pico da taxa de dados	1Mbps
Freqüência de Rádio*(RF)* da largura de banda	220KHz (-3dB), 1MHz (-20dB)
RF da banda	2.4GHz, *ISM band*
Portadoras da *RF*	23/79
Espaço da portadora	1MHz
Potência transmitida	0.1W
Acesso a *piconet*	*FH-TDD-TDMA*
Taxa de Salto de Freqüência	1600 saltos por segundo
Acesso a *scatternet*	*FH-CDMA*

Tabela 1: Parâmetros de Rádio e Banda Básica do *Bluetooth*

O *Bluetooth* faz uso da banda 2,4GHz *ISM* (Industrial, Científica e Médica). Na maioria das regiões, a largura de banda é suficiente para definir 79 canais físicos de 1MHz, mas sofre variações de freqüência em outras, veja a Tabela 2 [1]. O controle de potência é usado para manter o dispositivo emitindo somente potência *RF* necessária. O algoritmo de controle de potência é implementado usando o protocolo de Gerenciamento do *Link* entre um mestre e os escravos em uma *piconet*.

A modulação para o *Bluetooth* é a *Gaussian Frequency Shift Keying (GFSK)*, com o binário 1 representado por um desvio de freqüência positiva e o binário 0 representado por um desvio de freqüência negativa. O desvio mínimo é de 115kHz.

Área	Faixa Regulamentada	Canais *RF*
EUA, maioria da Europa, e maioria de outros países.	2,4 até 2,4835GHz	*F*=2,402 + *n*MHz, *n*=0,...,78
Japão	2,471 até 2,497GHz	*F*=2,437 + *n*MHz, *n*=0,...,22
Espanha	2,445 até 2,475GHz	*F*=2,449 + *n*MHz, n=0,...,22
França	2,4465 até 2,4835GHz	F=2,454 + nMHz, n=0,...,22

Tabela 2: Alocações de Freqüências Internacionais *Bluetooth*

2.3. Especificação da Banda Básica

Um dos mais complexos documentos do *Bluetooth* é a especificação Banda Básica [6].

A camada Banda Básica oferece temporização, *framing*, transmissão de pacotes, controle de fluxo, detecção e correção de erro. A entidade na Banda Básica a qual controla as rotinas dos *links* de baixo nível é o Controlador de *Link* (LC).

Dois tipos de *links* são suportados: o Orientado a Conexão Síncrona *(SCO)* – usado para transmitir voz e o Não Orientado a Conexão Assíncrona *(ACL)* – usado para transmitir pacotes de dados. Ambos os tipos de *link* usam o esquema *TDD* para resolver contenção sobre o *link wireless*, onde cada *slot* é de 0,625ms. O *link SCO* é do tipo ponto a ponto entre o mestre e um único escravo, estabelecido por reserva de *slots duplex* em intervalos regulares. O *link ACL* é um ponto a multiponto entre o mestre e todos os escravos na *piconet*. Um pacote Banda Básica pode ocupar 1, 3 ou 5 *slots*. O tamanho do pacote Banda Básica pode ser decidido baseado em critérios, tais como, a quantidade de dados para serem transmitidos ou o número de *slots* contínuos disponíveis na presença do tráfego de voz.

As conexões de dados podem ser síncronas ou assíncronas, com taxa de dados de 432 Kbps e 721 Kbps, respectivamente. Até três conexões simultâneas de voz podem ser mantidas com taxa de dados de 64 Kbps, sendo sempre síncronas [1] [2].

Um esquema *ARQ* é usado para informar o sucesso ou a falha da transferência da carga útil. Os pacotes da camada Banda Básica são retransmitidos até um reconhecimento *(ACK)* positivo ser retornado ou até um *timeout* ser excedido. Um código *FEC* opcional de taxa 2/3 pode ser usado na carga útil dos dados para reduzir o número de retransmissões. O cabeçalho do pacote é sempre protegido por um *FEC* de taxa 1/3, visto que ele contém valiosas informações de *link* e devendo ser capaz de suportar mais erros.

2.3.1. Saltos de Freqüência

Os Saltos de Freqüência *(FH)* no *Bluetooth* serve a dois propósitos:

1. Resistência à interferência e efeitos *multipath*;
2. Uma forma de acesso múltiplo entre dispositivos localizados em diferentes *piconets*.

O esquema *FH* trabalha da seguinte forma [1]: a largura de banda total é dividida em 79 (em quase todas os países, veja a Tabela 2 para mais detalhes) canais físicos, cada largura de banda é de 1MHz. O *FH* ocorre por salto de um canal físico para outro em uma seqüência pseudo-randômica. A mesma Seqüência de Saltos é compartilhada por todos os dispositivos em uma única *piconet*. A taxa de saltos é de 1600 saltos por segundo, tal que cada canal físico é ocupado por uma duração de 0,625ms. Cada período de tempo de 0,625ms é referido como *slot* e estes são numerados seqüencialmente.

Os rádios *Bluetooth* comunicam-se usando uma técnica *TDD,* que é uma técnica de transmissão de *link* na qual dados são transmitidos em uma

direção no tempo, com alternação de transmissão entre as duas direções. Para que mais de dois dispositivos compartilhem o meio da *piconet*, a técnica de acesso é a de Acesso Múltiplo por Divisão de Tempo (*TDMA*). Assim o acesso a *piconet* pode ser caracterizado como *FH-TDD-TDMA*. A Figura 3 [1] ilustra a técnica. Na figura, *k* denota o número de *slot* e *f(k)* é o canal físico selecionado durante o período de *k slot*.

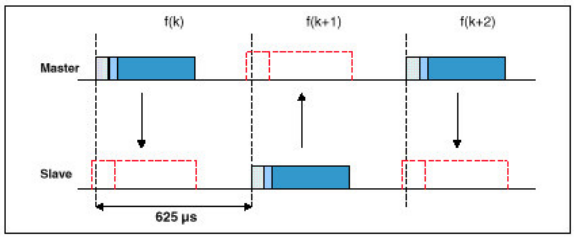


Figura 3: *TDD* e *Timing*

A transmissão de um pacote começa no início de um *slot*. O comprimento do pacote requerido permitido é de 1,3 ou 5 *slots*. Para pacotes *multislot*, o rádio permanece na mesma freqüência até o pacote inteiro ser enviado, veja a Figura 4 [1]. No próximo *slot*, depois do pacote *multislot*, o rádio retorna para a freqüência requerida por sua Seqüência de Saltos, tal que durante a transmissão, dois ou quatro Saltos de Freqüência foram puladas.

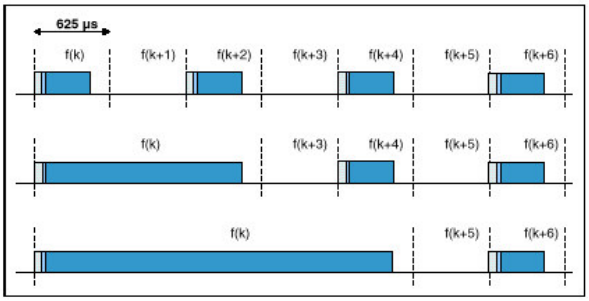


Figura 4: Pacotes *Multislots*

Usando o *TDD* previne-se ligações cruzadas entre operações de transmissão e recepção no transmissor do rádio, o qual é essencial se uma implementação num circuito único é desejada. Devido à transmissão e recepção acontecerem em diferentes time *slots*, diferentes freqüências são usadas.

A Seqüência do *FH* é determinada pelo mestre na *piconet* e é em função do seu endereço. Uma complexa operação matemática envolvendo operações de troca e *OR* exclusivo *(XOR)* é usada para gerar uma pseudo-randômica Seqüência de Saltos.

Devido a diferentes *piconets* na mesma área terem diferentes mestres, eles usam diferentes Seqüência de Saltos. Assim, muitas vezes, transmissões em dois dispositivos em diferentes *piconets* na mesma área serão em diferentes canais físicos. Ocasionalmente, duas *piconets* usarão o mesmo canal físico durante o mesmo *time slot*, causando uma colisão e perdendo dados. Porém, para que isto não aconteça com freqüência, ele é acomodado com o *FEC* e técnicas de detecção de erro *ARQ*. Assim, uma forma de Acesso Múltiplo por Divisão de Código *(CDMA)* é executada entre dispositivos em diferentes *piconets* na mesma *scatternet*; isto é referido como *FH-CDMA*.

2.3.2. *Links* Físicos

Dois tipos [1] de *links* podem ser estabelecidos entre um mestre e um escravo:

- *SCO*: aloca uma largura de banda fixa entre uma conexão ponto-a-ponto envolvendo o mestre e um único escravo. O mestre mantém o *link SCO* usando *slots* reservados em intervalos regulares. A unidade básica de reserva é dois *slots* consecutivos (um em cada direção de transmissão). O mestre pode suportar até três *links SCO* simultâneos para o mesmo escravo ou para escravos diferentes, enquanto um escravo pode suportar até três *links SCO* do mesmo mestre ou dois *links SCO* de diferentes mestres. Os pacotes do *SCO* nunca são retransmitidos.
- *ACL*: um *link* ponto-a-multiponto entre o mestre e todos os escravos na *piconet*. Em *slots* não reservados, para *links SCO*, o mestre pode trocar pacotes com qualquer escravo em uma base por *slot*, incluindo um Escravo já engajado em um *link SCO*. Somente um único *link ACL* po-

de existir. Para a maioria dos pacotes *ACL*, as retransmissões de pacotes são aplicadas.

Os *links SCO* são usados primeiramente para trocar dados sensitivos ao tempo requerendo uma garantia da taxa de dados, mas sem garantia de entrega. Um exemplo, usado em muitos dispositivos *Bluetooth*, é a codificação de áudio digital com tolerância a perda de dados. A garantia da taxa de dados é executada por reserva de um número particular de *slots*.

Tipo	Simétrico (Kbps)	Assimétrico (Kbps)	
DM1	108.8	108.8	108.8
DH1	172.8	172.8	172.8
DM3	256.0	384.0	54.4
DH3	384.0	576.0	86.4
DM5	286.7	477.8	36.3
DH5	432.6	721.0	57.6

DMx = x-slot FEC-encoded; DHx = x-slot desprotegido

Tabela 3: Taxa de Dados Possíveis no *Link ACL*

Os *links ACL*, utilizam um estilo de comutação de pacote. Nenhuma reserva de largura de banda é possível e a entrega pode ser garantida por detecção de erro e retransmissão. A um escravo, é permitido retornar em pacote *ACL* no *slot* do escravo para o mestre, se e somente se, ele for endereçado no *slot* anterior do mestre para o escravo. Para os *links ACL*, foram definidos os pacotes de 1, 3 e 5 *slots*. Os dados podem também ser enviados desprotegidos (embora o *ARQ* possa ser usado em uma camada superior) ou protegido com um código de taxa de 2/3 *FEC*. A taxa máxima de dados que pode ser executada é com um pacote de 5 *slots* desprotegido, com capacidade assimétrica de alocação, resultando em 721Kbps na direção *forward* e 57,6Kbps na direção inversa. A Tabela 3 resume todos as possibilidades de taxas reportadas pela norma [1].

2.3.3. Pacotes

O formato de todos os pacotes *Bluetooth* é mostrado na Figura 5 [1]. Ele consiste de três campos:

- Código de Acesso: usado para a sincronização de tempo e nos procedimento de Paginação e Investigação.
- Cabeçalho: usado para identificar o tipo de pacote e executar o protocolo de informações de controle.
- Carga útil: se presente, contém voz ou dados de usuário e na maioria dos casos, um cabeçalho da carga útil.

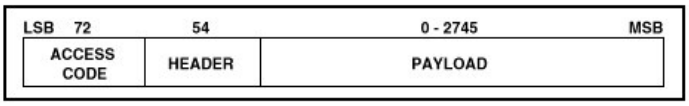


Figura 5: Formato do Pacote Padrão

2.3.3.1. Código de Acesso

Existem três tipos [1] de Código de Acesso (Tabela 4):

- Código de Acesso do Canal *(CAC)*: identifica uma *piconet* (único para uma *piconet*).
- Código de Acesso do Dispositivo *(DAC)*: usado para o procedimento de *Paging* e suas subseqüentes respostas.
- Código de Acesso da Investigação *(IAC)*: usado para propósito do procedimento de Investigação.

Um Código de Acesso consiste de um preâmbulo, uma informação de sincronização e uma cauda. O preâmbulo consiste de um padrão 0101, se o *Bit* Menos Significativo (*LSB*), ou seja, mais à esquerda na informação de sincronização é 0 e o padrão 1010, se o *LSB* na informação de sincronização é 1. Similarmente, a cauda é 0101, se o *Bit* Mais Significativo *(MSB), ou seja,* mais à direita da informação de sincronização é 1 e 1010, se o *MSB* é 0.

A informação de sincronização de 64 *bits*, consiste de três componentes. A cada dispositivo *Bluetooth* é designado um único endereço mundial de 48 *bits*. Os 24 *LSB* são referidos como uma Parte Baixa do Endereço *(LAP)* e é usado na formação da informação de sincronização. Para um *CAC*, o *LAP* do mestre é usado; para um *DAC*, o *LAP* da unidade paginada. Existem dois *IACs* diferentes (Tabela 4). O Geral *IAC (GIAC)*, é uma mensagem *Inquiry* geral usada para descobrir qual o dispositivo *Bluetooth* está no alcance e para isto um valor especial reservado do *LAP* está disponível. Um Dedicado *IAC (DIAC)*, é comum para um grupo dedicado de unidades *Bluetooth* que compartilham uma característica comum e um *LAP* previamente definido correspondendo a qual característica é usada.

Tipo de Código	*LAP*	Comprimento do Código
CAC	Mestre	72
DAC	Unidade paginada	68/72*
GIAC	Reservado	68/72*
DIAC	Dedicado	68/72*

* Comprimento 72 é usado somente em combinação com pacotes de Sincronização do Salto de Freqüência – *FHS*

Tabela 4: Tipos de Código de Acesso

Usando o *LAP* apropriado, a informação de sincronização é formada como segue [1]:

1. Para o *LAP*, adiciona-se os 6 *bits* 001101, se o *MSB* do *LAP* é 0 e adiciona-se 110010, se o *MSB* é 1. Isto forma uma seqüência de *Barker* de 7-*bit*. O propósito de incluir uma seqüência de *Barker* é para mais adiante melhorar as propriedades de auto-relacionamento da informação de sincronização.
2. Gera uma seqüência de ruídos de 64 *bits (PN)*, p_0, p_1, ..., p_{63}. A seqüência é definida pela equação $P(X) = 1 + X + X^3$ e pode ser implementada com um retorno linear da troca de registro de *6 bits*. O valor da origem para a seqüência *PN* é 100000.
3. Pega o *bitwise XOR* do p_{34}, p_{35}, ..., p_{63} e a seqüência de 30 *bits* produzida no passo 1. Esta "mistura" de informação, remove irregularidades.

4. Gera um código de correção de erro de 34 *bits* para misturar informações do bloco e colocá-lo para formar a informação do código de 64 *bits*. Assim, nós temos um código (64, 30). Para gerar este código, inicia com um código *BCH* (63, 30). Então define o gerador polinomial $g(X) = (1 + X)g'(X)$, onde $g'(X)$ é o gerador polinomial para o código *BCH* (63, 30). Isto produz o desejado código de 34 *bits*.
5. Pega o *bit* inteligente *XOR* de $p_0, p_1, ..., p_{63}$ e a seqüência de 64 *bits* produzida no passo 4. Este passo separa a parte da informação do código, tal que o original *LAP* e a seqüência de *Barker* são transmitidas. O passo também mistura o código do bloco.

A mistura de parte da informação do código no passo 3 foi desenvolvida para reforçar as propriedades de correção de erro do código do bloco. A subseqüente separação habilita o receptor para recuperar o *LAP* facilmente. O embaralhamento do código de erro de 34 *bits* remove as propriedades cíclicas do código a baixo. Isto talvez dê melhor qualidade de transmissão *spectral* e também melhora as propriedades de auto-relacionamento.

Figura 6: Formato do Cabeçalho

2.3.3.2. Cabeçalho do Pacote

O formato do cabeçalho para todos os pacotes *Bluetooth* é mostrado na Figura 6 [1]. Ele consiste de seis campos:

- *AM_ADDR*: a *piconet* acrescenta na maioria dos sete escravos ativos. O *AM_ADDR* de 3 *bits* contém o endereço do "modo ativo" (endereço temporário determinado para este escravo na *piconet*) de um dos escravos. Uma transmissão do mestre para um escravo contém qual o endereço do escravo; uma transmissão de um escravo contém seu endereço. O valor 0 é reservado para um *broadcast* do mestre para todos escravos na *piconet*.

- Tipo: identifica o tipo de pacote, veja a Tabela 5 [2]. Quatro tipos de códigos são reservados para controlar pacotes comuns aos *links SCO* e *ACL*. O restante dos tipos de pacotes é usado para conduzir informações de usuários. Para *links SCO*, são usados os pacotes *HV1*, *HV2* e *HV3,* cada um executando 64Kbps de voz. A diferença é a quantidade de proteção de erro oferecida. O pacote *DV* (veja o formato na Figura 7 [1]) transporta voz e dados. Para *links ACL*, seis pacotes diferentes são definidos. Estes, junto com o pacote *DM1*, carregam dados de usuário com diferentes quantias de proteção de erro e diferentes taxas de dados (Tabela 5). Existe outro tipo de pacote comum para ambos *links* físicos; ele consiste de somente o Código de Acesso, com um comprimento fixo de 68 *bits* (não incluindo a cauda). É o pacote *ID*, usado nos procedimentos de Investigação e Acesso.

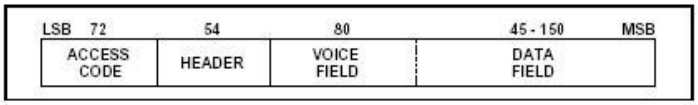


Figura 7: Formato do Pacote *DV*

Código do Tipo	*Link* Físico	Nome	Número de *Slots*	Descrição
0000	Ambos	*NULL*	1	Sem carga útil. Usado para retornar informações do *link* para a origem relativa ao sucesso da transmissão anterior *(ARQN)* ou o *status* do *buffer RX (*Fluxo*)*. Sem reconhecimento.
0001	Ambos	*POLL*	1	Sem carga útil. Usado pelo mestre para o *Poll* em um escravo. Com reconhecimento.
0010	Ambos	*FHS*	1	Pacotes de controle especial para revelar o endereço do dispositivo e o relógio do remetente. Usado na Resposta da Paginação do Mestre, na Resposta da Investigação e na Sincronização dos Saltos de Freqüência. Com codificação de 2/3 *FEC*.
0011	Ambos	*DM1*	1	Suporte a controle de mensagens e pode também carregar dados de usuários. Com 16 *bits CRC*. Com codificação de 2/3 *FEC*.
0101	*SCO*	*HV1*	1	Transporta 10 *bytes* de informações; tipicamen-

				te usado para 64Kbps de voz. Com codificação de 1/3 *FEC*.
0110	*SCO*	*HV2*	1	Transporta 20 *bytes* de informações; tipicamente usado para 64Kbps de voz. Com codificação de 2/3 *FEC*.
0111	*SCO*	*HV3*	1	Transporta 30 *bytes* de informação; tipicamente usado para 64Kbps de voz. Sem codificação *FEC*.
1000	*SCO*	*DV*	1	Combina pacotes de dados (150 *bits*) e voz (50 *bits*). Campo de dados com codificação de 2/3 *FEC*.
0100	*ACL*	*DH1*	1	Transportam 28 *bytes* de informações mais 16 *bits CRC*. Sem codificação *FEC*. Tipicamente usado para dados de alta velocidade.
1001	*ACL*	*AUX1*	1	Transportam 30 *bytes* de informações sem *CRC* ou *FEC*. Tipicamente usado para dados de alta velocidade.
1010	*ACL*	*DM3*	3	Transportam 123 *bytes* de informações mais 16 *bits* CRC. Com codificação de 2/3 *FEC*.
1011	*ACL*	*DH3*	3	Transportam 185 *bytes* de informações mais 16 *bits CRC*. Sem codificação *FEC*.
1110	*ACL*	*DM5*	5	Transportam 226 *bytes* de informações mais 16 *bits CRC*. Com codificação de 2/3 *FEC*.
1111	*ACL*	*DH5*	5	Transportam 341 *bytes* de informações mais 16 *bits CRC*. Sem codificação *FEC*.

Tabela 5: Tipos de Pacotes *Bluetooth*

- Fluxo: mecanismo de controle de fluxo de 1 *bit,* somente para tráfego *ACL*. Quando um pacote com Fluxo=0 é recebido, a estação que recebeu o pacote deve temporariamente parar a transmissão de pacotes *ACL* neste *link*. Quando um pacote com Fluxo=1 é recebido, as transmissões podem voltar.
- *ARQN*: mecanismo de reconhecimento de 1 *bit* para tráfego *ACL* protegido por *CRC*. Se a recepção for com sucesso, um *ACK* (*ARQN*=1) é retornado; se não, um *NAK* (*ARQN*=0) é retornado. Quando não for retornada uma mensagem relativa ao reconhecimento, um *NAK* é assu-

mido implicitamente. Se um *NAK* é recebido, o pacote relevante é retransmitido.

- *SEQN*: esquema de numeração seqüencial de 1 *bit*. Pacotes transmitidos são alternadamente rotulados com 1 ou 0. Isto é requerido para filtrar a retransmissão no destinatário; se uma retransmissão ocorrer devido a uma falha *ACK*, o destinatário recebe o mesmo pacote duas vezes.
- Controle de Erro do Cabeçalho *(HEC)*: um código de detecção de erro de 8 *bits* é usado para proteger o cabeçalho do pacote.

2.3.3.3. Formato da Carga Útil

Para alguns tipos de pacotes, a especificação da Banda Básica define um formato para o campo da carga útil. Para a carga útil de voz, nenhum cabeçalho é definido. Para todos os pacotes *ACL* e para a porção de dados do pacote *SCO DV*, um cabeçalho é definido. Para a carga útil de dados, o formato da carga útil [1] consiste de três campos:

- Cabeçalho da Carga Útil: um cabeçalho de 8 *bits* é definido para pacotes *slot* único e um cabeçalho de 16 *bits* é definido para pacotes *multislot*.
- Corpo da Carga Útil: contém informações de usuários.
- *CRC*: um código de 16 *bits* é usado em todos as cargas úteis de dados, exceto o pacote *AUX1*.

O cabeçalho da carga útil [1], quando presente, consiste de três campos:

- *L_CH*: identifica o canal lógico. As opções são, mensagem *LMP* (11); uma mensagem *L2CAP* não fragmentada ou o início de uma mensagem *L2CAP* fragmentada (10); a continuação de uma mensagem *L2CAP* fragmentada (01); ou outra (00).

- Fluxo: usado para controle de fluxo a nível *L2CAP*. Este é um mesmo mecanismo de liga/desliga provido pelo campo Fluxo no cabeçalho do pacote para tráfego *ACL*.
- Comprimento: o número de *bytes* de dados na carga útil, excluindo o cabeçalho da carga útil e o *CRC*.

2.3.4. Correção de Erro

No nível Banda Básica, o *Bluetooth* faz uso de três esquemas de correção de erro [1]:

- Taxa de 1/3 *FEC*
- Taxa de 2/3 *FEC*
- *ARQ*

Estes esquemas de correção de erro são designados para satisfazer requisitos de competição. O esquema de correção de erro deve ser adequado para lidar com o ambiente *wireless*, mas deve também ser fluído e eficiente.

A taxa de redundância de 1/3 *FEC* é usada no cabeçalho do pacote de 18 *bits* e também para o campo de voz em um pacote *HV1*. O esquema, simplesmente envia três cópias de cada *bit*. Uma maioria lógica é usada: de cada tríade de *bits* aceita é mapeado dentro de um *bit* com maioria.

A taxa de redundância de 2/3 *FEC* é usada em todos os pacotes *DM*, no campo de dados do pacote *DV (*Figura 7*)*, no pacote *FHS* e no pacote *HV2*. A codificação é uma forma de código de *Hamming,* com parâmetros (15, 10). Este código pode corrigir todos os erros simples e detectar todos os erros duplos em cada informação de código.

O esquema *ARQ* é usado com pacotes *DM, DH* e o campo de dados de pacotes *DV*. O esquema é similar ao esquema *ARQ* usado no Protocolo

de Controle do *Link (LCP)* de dados. *O* esquema *ARQ* tem os seguintes elementos:

- Detecção de erro: o destinatário detecta erros e descarta os pacotes que estão errados. A correção de erro é executada com um código de detecção de erro *CRC,* suplementado com o código *FEC*.
- Reconhecimento positivo: o destinatário retorna um reconhecimento positivo para a recepção com sucesso (pacotes livres de erro).
- Retransmissão depois de *timeout*: a origem retransmite um pacote que não recebeu um reconhecimento depois de um predeterminado tempo.
- Reconhecimento negativo e retransmissão: o destinatário retorna um reconhecimento negativo para pacotes na qual um erro foi detectado. A origem retransmite tal pacote.

O *Bluetooth* usa o esquema chamado de *ARQ* Rápido, o qual pega vantagem do fato que um mestre e um escravo comunicam-se em *time slots* alternados. Quando uma estação recebe um pacote, ela determina se um erro ocorreu usando um *CRC* de 16 *bits*. Se ocorrer erro, o *bit ARQN* no cabeçalho é modificado para 0 *(NAK)*; se nenhum erro for detectado, então *ARQN* é modificado para 1 *(ACK)*. Quando uma estação recebe um *NAK*, ela retransmite o mesmo pacote enviado no *slot* anterior, usando o mesmo Número Seqüencial (*SEQN)* de 1 *bit* no cabeçalho do pacote. Com esta técnica, um emissor é notificado no próximo *time slot* e se uma transmissão falhar ele retransmite. O uso do *SEQN* de 1*bit* e a imediata retransmissão do pacote, minimizam a sobre carga e maximiza a eficiência.

Na recepção de um pacote, o dispositivo primeiro checa se o cabeçalho é válido usando o *HEC*; se nenhum pacote é rejeitado e o *ARQN* for *NAK* no próximo *time slot*, então o dispositivo tem um endereço igual e checa se este é o tipo de pacote que usa o mecanismo *ARQ*. Passando neste teste, o próximo dispositivo checa se este é um novo *SEQN* ou o mesmo. Se os *SEQNs* são os mesmos, então isto é uma retransmissão, o qual é ignorada. Se o *SEQN* é novo, então o dispositivo checa o *CRC*. Se nenhum erro é de-

tectado, a próxima saída de pacote tem o *ARQN = ACK* e se um erro é detectado, a próxima saída de pacote tem o *ARQN = NAK*.

No lado do transmissor, no próximo pacote *DM*, *DH* e *DV* a ser transmitido são determinados o valor da entrada imediatamente anterior ao *ARQN*. Quando os *ACKs* são recebidos, o dispositivo envia uma carga útil nova, alternando os valores do *SEQN,* entre 0 e 1. Se um *NAK* ou nenhum reconhecimento é recebido, o dispositivo retransmitirá a carga útil antiga repetidamente até um *ACK* ser recebido ou algum limite ser alcançado, tempo que a carga útil antiga é esvaziada do *buffer* do transmissor e uma nova carga útil é transmitida.

2.3.5. Canais Lógicos

O *Bluetooth* define [1] cinco tipos de canais de dados lógicos, designados para transportar diferentes tipos de cargas de tráfegos:

- Controle do *Link (LC)*: usado para gerenciar o fluxo de pacotes sobre a *interface* do *link*. O canal *LC* é mapeado para o cabeçalho do pacote. Este canal carrega informações de baixo nível do *LC* com *ARQ*, o controle de fluxo e a caracterização da carga útil. O canal *LC* é executado em cada pacote, exceto no pacote *ID*, o qual não tem cabeçalho do pacote.
- Gerenciador do *Link (LM)*: transporta informações de gerenciamento do *link* entre estações participantes. Este canal lógico suporta tráfego *LMP* e pode ser mapeado sobre um *link SCO* ou *ACL*.
- Usuários Assíncronos *(UA)*: carrega dados de usuários assíncronos. Este canal é normalmente mapeado sobre o *link ACL*, mas pode ser transportado em um pacote *DV* no *link SCO*.

- Usuários Isócronos *(UI)*: carrega dados de usuário isócronos[2]. Este canal é normalmente executado sobre o *link ACL*, mas pode ser executado em um pacote *DV* no *link SCO*. No nível Banda Básica, o canal *UI* é tratado na mesma maneira que um canal *UA*. O tempo para provê propriedades isócronas é fornecido pela camada superior.
- Usuários Síncronos *(US)*: carrega dados de usuários síncronos. Este canal é mapeado sobre o *link SCO*.

2.3.6. Controle de Canal

A operação de uma *piconet* pode ser entendida em termos de estados da operação durante o estabelecimento do *link* e a manutenção. A Figura 8 [1], mostra o diagrama de estados dos *links*. Existem dois estados principais [1]:

- *Standby*: é o estado padrão. Este é um estado de baixa potência, no qual somente o relógio nativo é executado.
- Conexão: o dispositivo é conectado a *piconet* como um mestre ou escravo.

Além disso, existem sete subestados intermediários que são usados para adicionar novos escravos a *piconet*. Para mover, de um estado para outro, comandos do *LM* ou sinais internos no *LC* são usados. Os subestados são os seguintes:

- Paginação: dispositivo emite uma Paginação. Usado pelo mestre para ativar e conectar a um escravo. O mestre envia a mensagem de Paginação para transmitir o *DAC* do escravo em diferentes Saltos de Canais*;*
- Busca de Paginação: dispositivo está procurando por uma Paginação com seu próprio *DAC*;

[2] O termo isócrono refere aos blocos de dados que retorna com conhecimento do *timing* periódico.

- Resposta do Mestre: um dispositivo atua como mestre, recebendo uma Resposta de Paginação de um escravo. O dispositivo pode agora, entrar no estado de Conexão ou retornar para o estado de Paginação para realizar uma Paginação a outros escravos.

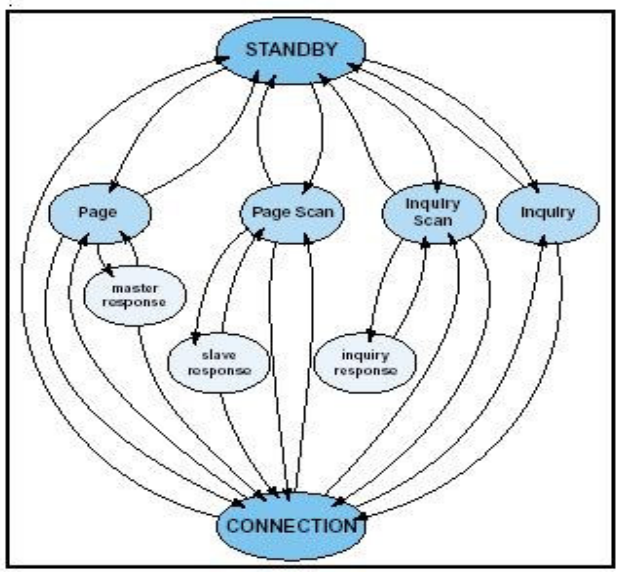


Figura 8: Diagrama de Estados do *LC*

- Resposta do Escravo: o dispositivo atua como escravo, respondendo por uma mensagem de Paginação de um mestre. Se a conexão for com sucesso, o dispositivo entra no estado de Conexão; se não, ele retorna para o estado de Busca de Paginação.
- Investigação: o dispositivo realiza uma Investigação para encontrar a identidade dos dispositivos dentro do alcance.
- Busca da Investigação: o dispositivo está procurando por uma Investigação.
- Resposta da Investigação: o dispositivo a qual emitiu uma Investigação recebe uma Resposta da Investigação.

2.3.6.1. Procedimentos de Investigação

O primeiro passo no estabelecimento de uma *piconet* é um potencial mestre identificar os dispositivos em seu alcance, a qual desejem participar da *piconet*. O dispositivo inicia um procedimento de Investigação para este

propósito através do usuário ou de uma aplicação. O procedimento de Investigação inicia quando o potencial mestre transmite um pacote *ID* com um *IAC*, o qual é um código comum para todos os dispositivos *Bluetooth*. Lembrando que o pacote *ID* não tem cabeçalho e nem carga útil.

Das 79 portadoras de rádio, 32 são consideradas portadoras do tipo *wake-up (acordar)*. O mestre envia um *broadcast* com o *IAC* sobre cada uma das 32 portadoras *wake-up*. Isto ocorre no estado de Investigação. Por enquanto, os dispositivos no estado *Standby,* periodicamente entram no estado de Busca de Investigação para procurar por mensagens *IAC* nas portadoras *wake-up*. Quando um dispositivo recebe a Investigação, ele entra no estado de Resposta da Investigação e retorna um pacote *FHS* (Tabela 5) contendo seu endereço de dispositivo e as informações de tempo requerida pelo mestre para iniciar a conexão. O mestre não responde ao pacote *FHS* e pode ficar no estado de Investigação até que todos os rádios sejam encontrados.

Uma vez um dispositivo responde a uma Investigação, ele passa para o estado de Busca de Paginação para esperar uma Paginação do mestre e estabelecer uma conexão. Porém, se uma colisão ocorrer na fase de Resposta da Investigação (dois ou mais dispositivos simultaneamente responder a uma Investigação), nenhuma Paginação será recebida e o dispositivo pode necessitar retornar ao estado de Busca de Investigação para tentar outra Resposta de Investigação[3].

2.3.6.2. Procedimento de Paginação

Quando o mestre encontra os dispositivos em seu alcance, torna-se capaz de estabelecer conexões para cada dispositivo, mantendo uma *piconet*. Para cada dispositivo ser paginado, o mestre usa o Endereço de Dispositivo (*BD_ADDR*) para calcular uma Seqüência de Saltos da Freqüência de Paginação, alvo a qual serve para contatar o dispositivo durante a Paginação. O mestre pagina um escravo usando um pacote *ID*, neste momento, com um

[3] O diagrama de estados da Figura 8 é baseado na especificação da Banda Básica e não mostra uma transição de Resposta de Investigação para a Busca de Paginação, mas mostra uma transição da Resposta de Investigação para a Busca de Investigação.

DAC do escravo específico. Lembrando que o *DAC* é a parte baixa do endereço do dispositivo escravo. O escravo responde retornando o mesmo pacote *ID* do *DAC* para o mestre na mesma Seqüência de Saltos (conhecido como a Seqüência de Saltos do Modo de Paginação) que foi usada pelo mestre. O mestre responde a ele no próximo *slot* mestre para escravo, com seu próprio pacote *FHS* contendo seu endereço de dispositivo e o valor do tempo real de seu relógio. Novamente, o escravo envia uma resposta no pacote *ID* do *DAC* para o mestre, confirmando a recepção do pacote *FHS* do mestre. O escravo, neste ponto, transita do estado de Resposta do Escravo para o estado de Conexão e começa a usar a Seqüência de Saltos da Conexão, definido no pacote *FHS* do mestre. Por enquanto, o mestre pode continuar realizando Paginação até todos os escravos (que desejarem) serem conectados; o mestre então, entra no estado de Conexão.

2.3.6.3. Estado de Conexão

Para cada escravo, o estado de Conexão inicia com um pacote *Poll,* enviado pelo mestre para verificar se o escravo mudou para o *timing* do mestre e a Freqüência de Saltos do canal. O escravo pode responder com qualquer tipo de pacote.

Quando o escravo está no estado de Conexão, ele pode estar em um dos quatro modos de operação [1]:

- Ativo: o escravo ativo participa na *piconet,* escuta, transmite e recebe pacotes. O mestre periodicamente transmite para o escravo mantendo a sua sincronização.
- *Sniff*: o escravo não escuta em cada *slot* recebido (cada outro *slot*), mas somente nos *slot*s especificados por esta mensagem. O escravo pode operar em um *status* de potência reduzida o resto do tempo. Para operar no modo *Sniff*, o mestre designa um número reduzido de *time slot*s na transmissão para um específico escravo.
- *Hold*: o dispositivo neste modo, não suporta pacote *ACL* e vai para o status de potência reduzida. O escravo pode ainda, participar em tro-

cas *SCO*. Durante períodos sem atividade, o escravo fica livre e ocioso, em um status de potência reduzida ou possivelmente participando em outra *piconet*.

- *Park*: quando um escravo não precisa participar da *piconet,* mas ainda preciso ficar retido como parte dela, pode entrar no modo *Park*, o qual é um modo de baixa potência, com pouquíssima atividade. Ao dispositivo é dado um Endereço de Membro *Parking (PM_ADDR),* perdendo seu Endereço de Membro Ativo *(AM_ADDR)*. Com o uso do modo *Park*, uma *piconet* pode ter mais de sete escravos.

2.3.7. Áudio

A especificação de Banda Básica traz que um dos dois esquemas de codificação de voz pode ser usado: a Modulação de Código de Pulso *(PCM)* ou a Modulação de Continuamente Variável Declínio Delta *(CVSD)*. A escolha é feita pelo *LM* dos dois dispositivos de comunicação, os quais negocia o mais apropriado esquema para a aplicação.

2.3.8. Segurança

A especificação de Banda Básica define uma facilidade para a segurança do *link* entre quaisquer dois dispositivos *Bluetooth*, consistindo dos seguintes elementos [1]:

- Autenticação
- Criptografia (privacidade)
- Gerenciamento de chaves

Os algoritmos de segurança fazem uso de quatro parâmetros:

- Endereço da Unidade: o endereço de dispositivo de 48 *bits*, o qual é conhecido publicamente;

- Chave Secreta de Autenticação: uma chave secreta de 128 *bits*;
- Chave Secreta Privada: uma chave secreta de comprimento entre 4 e 128 *bits*;
- Número Randômico: um número randômico, de 128 *bits,* derivado de um algoritmo de geração pseudo-randômica, executado na unidade *Bluetooth*.

As duas chaves secretas são geradas e configuradas pela unidade e não são reveladas.

O propósito da autenticação é verificar a identidade exigida de um dos dois dispositivos *Bluetooth* envolvidos na troca.

2.4. Especificação do *Link Manager*

LMP e *L2CAP* são camadas acima do protocolo de Banda Básica e residem na camada *datalink*. O *LMP* assume a responsabilidade de gerenciamento do estado das conexões, obrigando justiça entre os escravos, gerenciamento de potência e outras tarefas de gerenciamento.

O *LMP* gerencia vários aspectos do *link* de rádio entre um mestre e um escravo. O protocolo envolve a troca de mensagens na forma de Unidades de Protocolo de Dados (*PDUs) LMP* entre a entidade *LMP* no mestre e no escravo. As mensagens são sempre enviadas como pacotes de *slot* único, com cabeçalho da carga útil de 1 *byte,* o qual identifica o tipo da mensagem e um corpo da carga útil*,* o qual contém informações adicionais pertinentes a esta mensagem.

Os procedimentos definidos pelo *LMP* são agrupados em áreas funcionais, cada qual envolvendo a troca de uma ou mais mensagens.

O *LMP* suporta vários serviços de segurança com mecanismo para gerenciar autenticação, criptografia e distribuição de chave. Também, oferece

mecanismos [1] para sincronização de relógios em vários participantes da *piconet.*

2.5. Especificação do Controle Lógico do *Link* e Protocolo de Adaptação

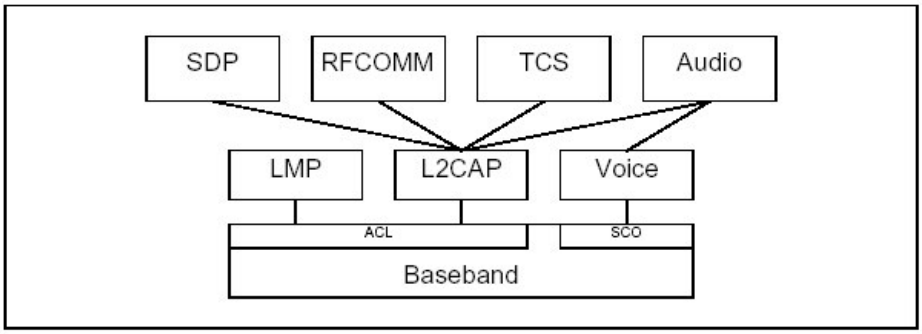


Figura 9: Arquitetura de Protocolos *Bluetooth*

O *L2CAP* suporta multiplexação de protocolos de alto nível, visto que a Banda Básica não suporta qualquer tipo de campo identificando os protocolos da camada alta. A Figura 9 [1], mostra a arquitetura de protocolos. O *L2CAP* também suporta Segmentação e Remontagem *(SAR)* de pacotes e carrega as informações de *Qualidade de Serviço (QoS)*. Ele permite que protocolos de alto nível e aplicações, transmitam e recebam pacotes de dados *L2CAP* de até 64 *kilobytes* de comprimento [2].

Como o *LLC* na especificação *IEEE802*, o *L2CAP* provê um protocolo da camada de *link* entre as entidades por uma rede meio compartilhada. A Figura 10 [1], mostra o L2CAP nas camadas de protocolos. O *L2CAP* provê também um número de serviços e deixa para a camada baixa (neste caso, a camada de Banda Básica) o fluxo e o controle de erro.

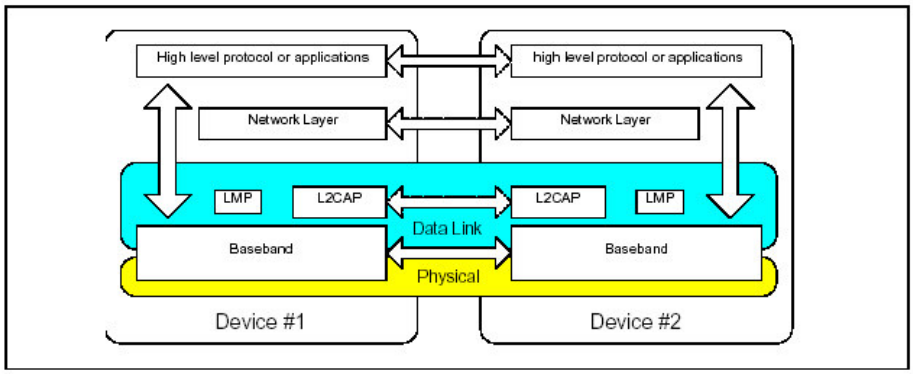


Figura 10: *L2CAP* dentro das Camadas de Protocolos

O *L2CAP* faz uso de *link*s *ACL*; ele não suporta *link*s *SCO*. Usando *link*s *ACL*, o *L2CAP* oferece dois serviços [6] para os protocolos das camadas superiores: o serviço não orientado à conexão e o serviço com conexão.

2.5.1. Canais *L2CAP*

O *L2CAP* oferece três tipos [1] de canais lógicos:

- Não orientado a conexão *(CL)*: suporta o serviço não orientado a conexão. Cada canal é unidirecional. Este tipo de canal é tipicamente usado para *broadcast* do mestre para múltiplos escravos.
- Orientado a conexão: suporta o serviço orientado a conexão. Cada canal é bidirecional (*full duplex*). Uma especificação do fluxo no *QoS* é nomeada em cada direção.
- Sinalização: utilizado para a troca de mensagens de sinalização entre entidades *L2CAP*.

A Figura 11 [1], mostra um exemplo de uso de canais lógicos *L2CAP*. Associado a cada canal lógico está um Identificador do Canal *(CID)*. Para canais orientados a conexão, um único *CID* é nomeado em cada fim de canal para identificar esta conexão e associa-la a um usuário *L2CAP* em cada fim. Canais sem conexão são identificados por um valor *CID* a 2 e a sinalização é

identificada por um valor *CID* a 1. Assim, entre o mestre e qualquer escravo, existe somente um canal sem conexão e uma sinalização de canal, mas podem ser múltiplos canais orientados a conexão.

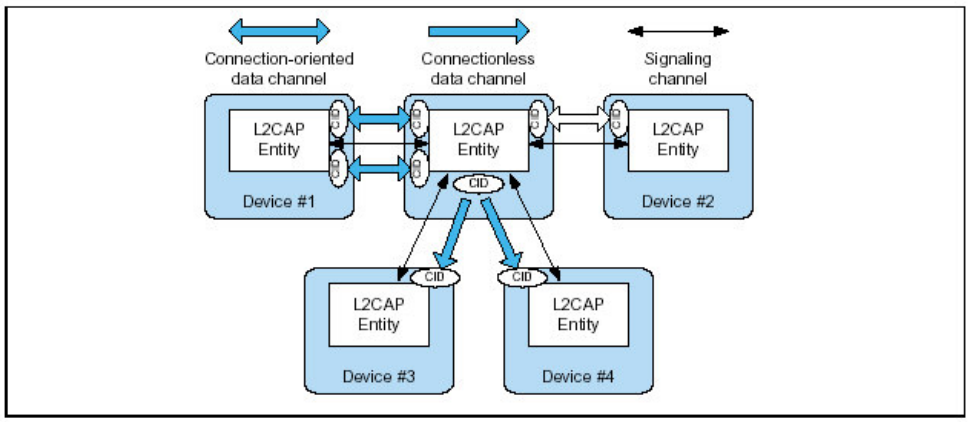


Figura 11: Canais entre Dispositivos

2.5.2. Pacotes *L2CAP*

Para o serviço não orientado a conexão, o formato do pacote [1] *L2CAP* consiste dos seguintes campos:

- Comprimento: comprimento das informações da carga útil, mais o campo *PSM*, em *bytes*.
- *ID* do canal: um valor de 2, indicando o canal sem conexão.
- Multiplexador de Protocolo ou Serviço *(PSM)*: identifica o recipiente da camada superior para a carga útil neste pacote.
- Carga de informações: dados de usuário da camada superior. Este campo pode ter até 65.533 (2^{16} - 3) *bytes* de comprimento.

Pacotes orientados a conexão tem o mesmo formato que os pacotes sem conexão, mas sem o campo *PSM*. O campo *PSM* não é necessário, porque o *CID* identifica o recipiente da camada superior de dados. A informação do campo de carga útil pode ter até 65.535 (2^{16} - 1) *bytes* de comprimento.

Pacotes de comandos de sinalização tem o mesmo formato do cabeçalho que os pacotes orientados a conexão. Neste caso, o valor *CID* é 1, indicando a sinalização do canal. A carga útil de um pacote de sinalização consiste de um ou mais comandos *L2CAP*, cada qual consiste de quatro campos:

- Código: identifica o tipo de comando.
- Identificador: usado para comparar uma requisição com sua resposta. O dispositivo que faz a requisição muda este campo e o dispositivo de resposta usa o mesmo valor na sua resposta. Um diferente identificador deve ser usado para cada comando original.
- Comprimento: comprimento do campo de dados, para este comando, em *bytes*.
- Dados: além de dados, se necessário, relaciona este comando.

2.5.3. Mensagens de Sinalização

Existem onze comandos em cinco categorias. O comando de Rejeitar pode ser enviado na resposta para rejeitar qualquer comando. As razões para rejeição incluem, *CID* inválido ou comprimento excessivo.

Comando de Conexão é usado para estabelecer uma nova conexão lógica. O comando de Requisição inclui um valor *PSM* indicando o usuário *L2CAP* para esta conexão. Três valores foram definidos; *SDP*, *RFCOMM* e Protocolo de Controle de Telefone. Outros valores *PSM* são nomeados dinamicamente e são implementações dependentes. O comando de Requisição também inclui o valor *CID* que será nomeado para esta conexão pela origem. O comando de Resposta inclui o *CID* da origem, o *CID* do destino e a posterior assinatura para este canal, pelo que responder. O parâmetro Resultado indica o resultado (sucesso, pendente, rejeitado) e se o Resulado for pendente, o campo de *status* indica o *status* atual, que fez esta conexão ficar pendente (autenticação pendente, autorização pendente).

O comando de Configuração é enviado para estabelecer uma transmissão inicial de *link* lógico, contratado entre duas entidades *L2CAP* e para renegociar este contrato sempre que apropriado. Cada parâmetro de configuração em uma requisição de configuração é relacionado exclusivamente para tráfego de saída ou entrada de dados. O comando de Requisição inclui um campo *flag*; atualmente, a única *flag* é um indicador de comandos adicionais de configuração. O campo de opções contém uma lista de parâmetros e seus valores para ser negociado.

Os seguintes parâmetros podem ser negociados:

- Unidade Máxima de Transmissão *(MTU)*: a maior carga útil do pacote *L2CAP*, em *bytes*, que o originador da Requisição pode aceitar para aquele canal. O *MTU* é assimétrico e o emissor da Requisição deve especificar o *MTU* que ele pode receber neste canal, caso seja diferente do valor padrão. Implementações *L2CAP* devem suportar um *MTU* mínimo de 48 *bytes* de comprimento. O valor padrão é de 672 *bytes*. Este não é um valor negociado, mas simplesmente informa ao receptor o comprimento do *MTU* que o emissor desta Requisição pode aceitar.
- Opção de *timeout* do esvaziamento: voltando a nossa discussão da especificação da Banda Básica, que como parte do mecanismo *ARQ*, uma carga útil será esvaziada depois de falhar em repetidas tentativas para retransmitir. O *timeout* de esvaziamento é a quantia de tempo que o originador irá tentar para transmitir um pacote *L2CAP* com sucesso, antes de esvaziar o pacote.
- *QoS*: identifica a especificação de fluxo do tráfego para o dispositivo (tráfego de saída de dados) sobre este canal.

Nos últimos dois casos, existem uma negociação, na qual o receptor pode aceitar o *timeout* de esvaziamento e os parâmetros de *QoS* ou requisitar um ajustamento.

O comando de Resposta de Configuração também inclui um campo *flag* com o mesmo significado do comando de Requisição de Configuração. O campo Resultado no comando de Resposta, indica se a requisição anterior foi aceita ou rejeitada. O campo de opções contém a mesma lista de parâmetros do comando de Requisição correspondente. Para um Resultado com sucesso, este parâmetro contém os valores de Retorno para qualquer parâmetro "coringa". Para uma Requisição sem sucesso, parâmetros de Rejeitado devem ser enviados na Resposta com os valores que foram aceitos, se enviados na Requisição original.

O comando de Desconexão é usado para terminar um canal lógico.

O comando de Eco é usado para solicitar uma Resposta de uma entidade *L2CAP* remota. Estes comandos são tipicamente para o teste do *link* ou para passar informações específicas do fabricante, usando o campo de dados opcional.

O comando de Informação é usado para solicitar informações de implementações específicas de uma entidade *L2CAP* remota.

2.5.4. Qualidade de Serviço (QoS)

Os parâmetros *QoS,* no *L2CAP,* definem uma especificação de fluxo de tráfego baseada na *RFC1363*[4]. Na essência, uma especificação de fluxo é um conjunto de parâmetros que indica um nível de desempenho que o transmissor tentará executar.

Quando incluído em uma Requisição de Configuração, esta opção descreve o fluxo do tráfego de saída de dados do dispositivo, enviando a Requisição para o dispositivo receptor. Quando incluído em uma Resposta de Configuração positiva, esta opção descreve o fluxo de tráfego de entrada de dados do acordo, como sendo do dispositivo que enviou a Resposta. Quando

[4] Uma *Proposed Flow Specification*, *RFC1363*, Setembro de 1992.

incluído em uma Resposta de Configuração negativa, esta opção descreve o fluxo de tráfego de entrada de dados preferido, da perspectiva do dispositivo que enviou a Resposta.

A especificação do fluxo consiste dos seguintes parâmetros [6]:

- Tipo do Serviço
- Taxa da Ficha (*bytes*/segundo)
- Tamanho do Balde de Ficha *(bytes)*
- Pico da largura de banda (*bytes*/segundo)
- Latência (microssegundos)
- Variação do atraso (microssegundos)

O parâmetro tipo de serviço indica o nível de serviço para este fluxo. Um valor de 0, indica que nenhum tráfego será transmitido neste canal. Um valor de 1, indica um serviço de Melhor Esforço *(BE)*; o dispositivo irá transmitir dados tão depressa quanto possível, mas sem garantia sobre o desempenho. Um valor de 2, indica um Serviço Garantido; o emissor irá transmitir dados conforme os parâmetros *QoS*.

O parâmetro da taxa de ficha e o tamanho do balde de ficha definem o esquema do balde de ficha (*token bucket),* que é freqüentemente usado nas especificações *QoS*. A vantagem deste esquema é que ele provê uma descrição concisa do pico e a média do tráfego que o receptor pode esperar, ele também provê um mecanismo conveniente, pelo qual o emissor pode implementar a política de fluxo de tráfego.

Uma especificação de tráfego do balde de ficha consiste de dois parâmetros: um reabastecimento da taxa de ficha *R* e um tamanho do balde *B*. O taxa de ficha *R* especifica a taxa de dados, continuamente sustentável; que é sobre um período de tempo relativamente longo, a média da taxa de dados suportada por este fluxo será *R*. O tamanho do balde *B,* especifica a quantidade a qual a taxa de dados pode exceder *R,* por curto período de tempo. A

exata condição é: durante qualquer período de tempo *T*, a quantia de dados enviada não pode exceder *RT + B*.

O balde representa um contador que indica o número permissível de *bytes* de dados que pode ser enviado em qualquer tempo. O balde abastece com fichas de *byte* na taxa de *R* (ou seja, o contador é incrementado *R* vezes por segundos), até a capacidade do balde (até o máximo valor do contador). Os dados, que chegam do *L2CAP* são remontados dentro dos pacotes, os quais são enfileirados para transmissão. Um pacote pode ser transmitido, se existir fichas de *byte* suficiente para o tamanho do pacote. Então, o pacote é transmitido e o balde é esvaziado do número correspondente de fichas. Se não existem fichas suficientes disponíveis, então o pacote excede a especificação para este fluxo. O tratamento para tais pacotes não é especificado na documentação; tipicamente, o pacote simplesmente será enfileirado para transmissão até fichas suficientes estejam disponíveis.

A taxa de dados permitida pelo balde de ficha é *R*. Porém, se existe um período ocioso ou relativamente lento, a capacidade do balde é aumentada de forma que, um adicional de *B bytes* sobre a taxa declarada pode ser aceito. Assim, *B* é uma medida de grau de rajada do fluxo de dados permitida.

Para o *L2CAP*, um valor de 0 para os dois parâmetros, implica que o esquema de ficha não é necessário para esta aplicação e não será usado. Um valor de 1 em todos, é o valor “coringa”. Para o serviço *BE*, o “coringa” indica que o requisitor deseja uma ficha grande ou um tamanho do balde de fichas grande, respectivamente. Para os serviços Garantidos, o “coringa” indica que o máximo da taxa de dados ou o tamanho do balde, respectivamente, estará disponível no tempo da requisição.

O pico da largura de banda, expresso em *bytes* por segundos, limita como os pacotes rápidos podem ser enviados *back-to-back* nas aplicações. Alguns sistemas intermediários podem pegar vantagem desta informação, resultando em uma alocação de recurso mais eficiente. Considerando que se

o balde de ficha está cheio, é possível para o fluxo enviar uma série de pacotes *back-to-back* igual ao tamanho do balde de ficha. Se o tamanho do balde de ficha é grande, este *back-to-back* pode ser longo o suficiente para exceder a capacidade do receptor. Para limitar este efeito, a taxa máxima de transmissão salta de acordo com a quantidade de pacotes rápidos sucessivos podem ser colocados na rede.

A latência é o atraso máximo aceitável, entre transmissões de um *bit* pelo emissor e sua transmissão inicial, expressa em microssegundos.

A variação do atraso é a diferença, em microssegundos, entre o máximo e o mínimo possível de atraso que um pacote irá experimentar. Este valor é usado pelas aplicações para determinar a quantia de espaço de *buffer* necessária no lado do receptor, para restaurar a transmissão de dados padrão original. Se uma aplicação receptora requer dados para serem entregues no mesmo padrão que o dado foi transmitido, pode ser necessário para o dispositivo receptor enviar brevemente para o *buffer* de dados, da forma como eles são recebidos no receptor, podendo restaurar a transmissão padrão antiga. Um exemplo disto é um caso onde uma aplicação deseja enviar e transmitir dados como amostras de voz, as quais são geradas e jogadas em intervalos regulares. A quantia de espaço do *buffer* que o dispositivo receptor deixa, determina a quantia de variações em atraso permitida por pacotes individual dentro de um fluxo dado.

2.6. *IP* sobre Bluetooth

O *TCP/IP/PPP* (Protocolo Ponto a Ponto) é usado na *Internet* com *Bluetooth* [1]. O *UDP/IP/PPP* está também disponível como transporte para o Protocolo de Aplicações *Wireless (WAP)*. Na tecnologia *Bluetooth*, o *PPP* [7] é considerado necessário para carregar a informação da *interface* serial sobre o *RFCOMM* (Comunicações *RF*) provendo emulação de cabo serial, usando o subconjunto de padrões *ETSI GSM 07.10* [1]. Porém, a especifica-

ção é aberta e sendo possível configurar *IP* diretamente sobre *L2CAP*. Assim dois cenários podem ser encontrados:

- *TCP/IP* sobre *PPP* sobre *RFCOMM*, o qual está sobre a camada *L2CAP*. Isto permite estabilizar operações e interoperabilidade com várias aplicações. Neste caso, a informação de *framing* disponível para o Controle do Link de Dados de Alto Nível (*HDLC)* no *PPP* passará para a camada *L2CAP* com o objetivo de alertar os limites do pacote *PPP*, permitindo um eficiente *SAR* [8];
- *TCP/IP* diretamente sobre a camada *L2CAP*.

2.7. Considerações Finais

Neste capítulo foram apresentadas as características da tecnologia *Bluetooth*; documentações, arquitetura de protocolos, especificações e os protocolos utilizados na *Internet* com o *Bluetooth*. Estas características são fundamentais para o entendimento dos fatores que influenciam o desempenho dos *links Bluetooth*.

No próximo capítulo, serão apresentados os fatores que influenciam o desempenho do tráfego de dados assíncronos sobre os *links Bluetooth*. Serão sugeridos alguns algoritmos SAR e variantes do TCP para otimizar este tipo de tráfego de dados.

Capítulo 3.0 – Otimização do *Link* Assíncrono

Este capítulo descreve vários fatores que influenciam o desempenho do tráfego de dados assíncronos sobre os *link*s *Bluetooth*, dando ênfase aos algoritmos de Segmentação e Remontagem *(SAR)* e as variantes do *TCP*.

3.1. Visão Geral

Existem vários fatores [3] que influenciam o desempenho do tráfego de dados assíncronos sobre os dispositivos *Bluetooth*, como os esquemas de Segmentação e Remontagem *(SAR)*, as variantes do *TCP*, a otimização do tamanho do *buffer*, os algoritmos de escalonamento, a manipulação de erros, os número de conexões *SCO* e os mecanismos de Qualidade de Serviço *(QoS)*.

Entre eles, detalharemos mais os algoritmos *SAR* e as variantes do *TCP*, procurando um melhor aproveitamento do *link* assíncrono sobre os dispositivos *Bluetooth*.

3.2. Esquemas de Segmentação e Remontagem (*SAR)*

Comparado a outras mídia físicas com fio, o pacote de dados definido pelo Protocolo de Banda Básica é limitado em tamanho [6]. Exportando um *MTU* associado com a maior carga útil da Banda Básica (341 *bytes* para pacotes *DH5*), limita o uso eficiente da largura de banda para os protocolos da camada mais alta, que são desenvolvidos para usar pacotes maiores. Os pacotes grandes, do *L2CAP,* devem ser segmentados em múltiplos pacotes menores da Banda Básica, antes deles serem transmitidos. Similarmente, os pacotes múltiplos da Banda Básica, que são recebidos, podem ser remontados em um único pacote do *L2CAP,* que segue uma simples checagem de integridade. A funcionalidade do *SAR* é absolutamente necessária para suportar protocolos usando pacotes maiores que os suportados pela Banda Básica.

3.2.1. Procedimentos *SAR*

As operações *SAR* são usadas para melhorar a eficiência, suportando um *MTU* com tamanho maior que o pacote da Banda Básica. Isto reduz a

sobre carga, quando são espalhados na rede os pacotes de transporte, usados pelos protocolos da camada mais alta, sobre vários pacotes da Banda Básica. Todos os pacotes do *L2CAP* podem ser segmentados para serem transferidos sobre os pacotes da Banda Básica. O protocolo não executa qualquer operação *SAR*, mas o formato do pacote suporta adaptação para diminuir o tamanho do *frame* físico. Uma implementação do *L2CAP* expõe o *MTU* de saída (do dispositivo receptor remoto) e segmenta os pacotes da camada mais alta ("pedaços grandes"), que podem ser passados ao *LM* via a *Interface* Controladora do Dispositivo *(HCI)*. No lado do receptor, uma implementação do *L2CAP* recebe os "pedaços grandes" do *HCI* e os remonta em pacotes do *L2CAP,* usando a informação do *HCI* e do cabeçalho do pacote.

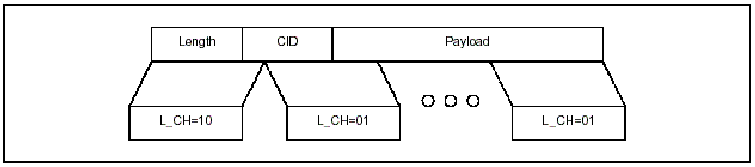


Figura 12: Segmentação L2CAP

O *SAR* é implementado usando pouquíssima sobre carga nos pacotes da Banda Básica. Veja na Figura 12 [1], que os dois *bits* do *L_CH* definidos no primeiro *byte* da carga útil da Banda Básica (também chamado de cabeçalho do *frame*) são usados para sinalizar o começo e a continuação de pacotes do *L2CAP*. O *L_CH* será "10" para o primeiro segmento em um pacote do *L2CAP* e "01" para uma continuação do segmento.

3.2.1.1. Procedimentos de Segmentação

O *MTU* do *L2CAP* será exportado usando uma implementação específica da *interface* do serviço. Isto é responsabilidade do protocolo da camada mais alta, para limitar o tamanho dos pacotes enviados para a camada do *L2CAP* acima do limite do *MTU*. Uma implementação do *L2CAP* segmentará o pacote em *PDUs* para enviar à camada mais baixa. Se o *L2CAP* funciona diretamente sobre o protocolo de Banda Básica, uma implementação pode segmentar o pacote em pacotes da Banda Básica para transmissão. Se o *L2CAP* usa o *HCI* (cenário típico), uma implementação pode enviar "pedaços

grandes" de tamanho do bloco para o controlador do dispositivo onde eles serão convertidos em pacotes da Banda Básica. Todos os segmentos do *L2CAP* associados a um pacote do *L2CAP,* devem ser passados para a Banda Básica antes de qualquer outro pacote do *L2CAP, com* destino à mesma unidade enviada.

3.2.1.2. Procedimentos de Remontagem

O protocolo da Banda Básica entrega os pacotes *ACL* em seqüência e protege a integridade dos dados que usam um *CRC* de 16 *bits*. A Banda Básica também suporta confiabilidade das conexões que usam o mecanismo *ARQ*. Semelhante ao controlador da Banda Básica, ele recebe pacotes *ACL*, sinaliza a camada do *L2CAP* na chegada de cada pacote da Banda Básica ou acumula vários pacotes, antes do *buffer* do receptor encher mais ou expirar o tempo antes da sinalização da camada do *L2CAP*.

Implementações do *L2CAP* têm que usar o campo comprimento no cabeçalho dos pacotes do *L2CAP*, como uma checagem de consistência e descarta qualquer pacote do *L2CAP* que falhe na comparação do campo comprimento. Se a confiabilidade do canal não é necessária, os pacotes com comprimentos impróprios podem ser descartados silenciosamente. Para canais confiáveis, implementações do *L2CAP* têm que indicar à camada superior, que o canal ficou confiável. Os canais são definidos com um valor infinito de *timeout* de esvaziamento.

As operações *SAR* transmitem em uma única *PDU* da camada mais alta. Enquanto há um mapeamento uma para um entre uma *PDU* da camada alta e um pacote do *L2CAP*, o tamanho do segmento usado pelas rotinas *SAR* é deixado para a implementação e pode diferir do remetente para o receptor.

3.2.2. Algoritmos *SAR*

Os mecanismos *SAR* são usados para melhorar a eficiência por suportar um *MTU* de tamanho maior que o pacote da Banda Básica. Isto reduz sobrecarga por expandir os pacotes usados pelos protocolos da camada alta sobre vários pacotes da Banda Básica, cobrindo 1, 3 ou 5 *slots*. É importante notar que o tamanho da carga útil [1] (sem *FEC*) para um pacote de 5 *slots* (339 *bytes* em 5 *slots* ou 67,8 *bytes/slot*) é maior que o pacote de 3 *slots* (183 *bytes* em 3 *slots* ou 61 *bytes/slot*) e os pacotes de 1 *slot* (27 *bytes/slot*). Para facilitar o entendimento do algoritmo, definimos como *tamanho,* sendo o tamanho máximo dos *slots* pelo qual um pacote da Banda Básica pode ser enviado. O *tamanho* pode ser menor que 5 devido à presença de conexões *SCO* ou devido a uma alta Taxa de Erros de *Bits* (*BER*) no canal *wireless*. Este parâmetro pode ser transportado pelo *LMP* para o *L2CAP* por um pacote de sinalização.

Como mencionado antes, estudaremos três tipos de algoritmos *SAR* e através de simulações, apontaremos o que obteve melhor performance. Dois algoritmos já existem e foram levemente adaptados para o *Bleutooth*, que são o de Otimização da Utilização do *Slot* (*OSU*) e o do Melhor Ajuste (*BE*) [3]. O algoritmo Randômico não existia e foi idealizada para tornar-se uma alternativa aos outros dois.

A seguir, os três esquemas *SAR* são apresentados:

3.2.2.1. SAR – BF (Best Fit)

Este algoritmo reduz a perda da largura de banda nos pacotes da Banda Básica, ele usa o método do “melhor ajuste” para segmentar os pacotes da camada alta. O algoritmo pode ser resumido nos seguintes passos:

1. Se *tamanho* => 5, envia o pacote *L2CAP* em *5 slots* de pacotes da Banda Básica;

2. Se *tamanho* >= 3 e *tamanho* < 5, envia em *3 slots* de pacotes da Banda Básica;
3. Caso *tamanho* < 3, envia em *1 slot* de pacote da Banda Básica.

3.2.2.2. SAR – OSU (Optimum Slot Utilization)

Este algoritmo diminui o atraso da transmissão dos pacotes da Banda Básica para o *L2CAP* por reduzir o atraso do enfileiramento dos pacotes da Banda Básica. Também maximiza o envio de dados, pois a cada tempo, um escravo recebe o "*polling"* (varredura) para enviar preferencialmente pacotes *multislot*. O algoritmo pode ser resumido nos seguintes passos:

1. Se *tamanho* > 3, envia o pacote *L2CAP* em 5 *slots* de pacotes da Banda Básica;
2. Se *tamanho* <= 3 e *tamanho* > 1, envia o pacote *L2CAP* em 3 *slots* de pacotes da Banda Básica;
3. Caso *tamanho* <=1, envia o pacote *L2CAP* em 1 *slot* de pacote da Banda Básica.

3.2.2.3. *SAR* – Randômico

Este algoritmo escolhe aleatoriamente o tipo de *slot* ao qual os pacotes serão enviados. É utilizada uma função *Random::uniform()*, que gera valores aleatórios entre 0 e 1 uniformemente, ou seja, com a probabilidade de aproximadamente 33,33% para cada tipo de *slot*. Para facilitar o entendimento, definimos como *randomico*, um número gerado pela função randômica. Ele pode ser resumido da seguinte forma:

1. Se *randomico* <= 0,33, envia o pacote *L2CAP* em *5 slots* de pacotes da Banda Básica;
2. Se *randomico* > 0,33 e *randomico* <= 0,66 envia o pacote *L2CAP* em *3 slots* de pacotes da Banda Básica;
3. Se *randomico* > 0,66, envia o pacote *L2CAP* em *1 slot* de pacote da Banda Básica.

3.3 Variantes do *TCP*

O protocolo *TCP* é um protocolo da camada de transporte da arquitetura *TCP/IP*, que fornece um serviço *full duplex*, orientado à conexão, destinado ao transporte confiável de diversas aplicações *(WEB, SMTP, FTP, TELNET)*. Ele é responsável pelo transporte, controle de fluxo, verificação de erros e caso necessário, a retransmissão dos dados do transmissor até o receptor.

As primeiras implementações do *TCP* não possuíam mecanismos de controle de congestionamento, devido o tráfego da *Internet* ainda não ser tão intenso quanto o de hoje. Atualmente, nós temos várias implementações do protocolo, que surgiram com a necessidade de se ter um melhor aproveitamento do *link* em virtude do crescimento exponencial da *Internet*. Tais implementações sofreram várias modificações, inicialmente introduzidas em 1988 por Van Jacobson [12], que proporcionaram grandes melhorias através dos mecanismos de controle de congestionamento. Tempos depois, novas modificações sugeridas por [13] representaram uma melhoria da ordem de 10 vezes no desempenho do *TCP*. No caso dos dispositivos com tecnologia *Bluetooth,* há uma importância maior ainda, visto que a pouca autonomia das baterias e escassez de memória tornam essencial o máximo aproveitamento do link.

Como o protocolo tem as funções de detecção de erros, ordenação de pacotes, controle de fluxo e controle de congestionamento, ele necessita de algoritmos que têm o objetivo de otimizar a vazão, para manter alta a eficiência de utilização do canal e evitar que os diversos fluxos concorrentes causem congestionamento na rede. Pelas características altamente dinâmicas do tráfego de dados em uma rede, faz-se necessário o uso de algoritmos adaptativos para a realização destes objetivos.

3.3.1. Mecanismos de Controle de Congestionamento do *TCP*

Antes de falarmos das variantes do *TCP*, é importante entendermos o funcionamento dos mecanismos de controle de congestionamento que são adotados pelas variantes do *TCP*.

O princípio básico [4] do controle de congestionamento, refere-se ao fluxo de pacotes na rede de forma conservativa, ou seja, um pacote só pode ser colocado na rede depois que outro pacote for retirado. O recebimento do reconhecimento garante que o pacote deixou a rede. Com isto, a taxa de transmissão da rede fica dependente dos reconhecimentos.

Para o *TCP* detectar um congestionamento na rede, ele baseia-se na perda de pacotes. Mesmo havendo outros parâmetros que medem o congestionamento, a baixa taxa de erro presente nas redes atuais, garantem que a perda de pacotes está ligada diretamente ao congestionamento.

O controle de congestionamento do *TCP* é realizado por quatro algoritmos [4]: Começo Lento, Evitar Congestionamento, Retransmissão Rápida e Recuperação Rápida, que são implementados em conjunto.

3.3.1.1. Temporizador

A precisão do temporizador é essencial ao funcionamento adequado das retransmissões, porque indica o tempo de envio de um pacote e o tempo de espera a um reconhecimento, garantindo que o transmissor não tenha que ficar ocioso e efetuando retransmissões desnecessárias. Assim, o transmissor não enviará dados repetidos e transmitidos com sucesso, conseqüentemente aproveitando melhor o *link* de dados.

3.3.1.2. Tempo de Ida e Volta *(RTT)*

O *RTT* é o tempo que um pacote necessita para ir e voltar do transmissor ao receptor dos dados. Ele depende do tempo de propagação do sinal

no meio de transmissão, juntamente com o tempo gasto pelos dados nas filas de espera. Assim, ele varia de acordo com a carga da rede.

A estimativa do *RTT* é feita da seguinte maneira: quando um pacote é enviado, armazena-se o tempo gasto na transmissão dos dados até o seu reconhecimento. Após isto, é feita uma média ponderada dos valores armazenados.

3.3.1.3. Começo Lento (*Slow Start*)

Foi uma solução proposta em [12] para resolver o problema das rajadas de pacotes enviadas a rede sem controle do reconhecimento e o aumento descontrolado do congestionamento.

A idéia do algoritmo, é iniciar a conexão com uma janela menor e gradativamente aumentando-a até o limite da rede, servindo tanto para uma nova conexão, quanto ao reinício após atingir um estouro do temporizador.

O algoritmo é resumido seguinte forma [4]:

1. Adiciona-se a variável de janela de congestionamento *(cwnd)*;
2. Quando começar a conexão, a *cwnd* recebe o valor 1**MSS* (Tamanho Máximo do Segmento);
3. A cada novo reconhecimento que não seja duplicado, a *cwnd* é incrementada;
4. A janela de transmissão é a menor entre a janela anunciada pelo receptor (*awnd*) e a *cwnd*.

Quando se aumenta a *cwnd* de um pacote a cada reconhecimento, isto resulta em um crescimento exponencial da janela, mas ao atingir a capacidade da rede e ocorrer um *timeout,* a *cwnd* retorna ao primeiro passo do algoritmo. Assim, quando a conexão atingir o *timeout* o mecanismo entra em ação para manter um ponto de equilíbrio.

3.3.1.4. Evitar Congestionamento (*Congestion Avoidance*)

Manipula a janela de congestionamento realizando a regra “crescer somando e reduzir multiplicando”, aumentando linearmente e reduzindo exponencialmente a janela.

O algoritmo de Evitar Congestionamento é resumido da seguinte forma [4]:

1. No momento do estouro do temporizador (*timeout*), a *cwnd* passa a ser *cwnd*/2;
2. A cada novo reconhecimento não duplicado, incrementa-se a *cwnd* de 1/*cwnd*;
3. A janela de transmissão é a menor entre a janela do receptor (awnd) e a *cwnd*.

Nas implementações do *TCP*, os algoritmos Começo Lento e Evitar Congestionamento são embutidos conjuntamente, adicionando-se uma variável *ssthresh* (limite do Começo Lento) a qual determinará qual está ativo no momento.

As duas implementação juntas, resultam em [4]:

1. No momento do *timeout*, o *ssthresh* passa a ser *cwnd*/2;
2. Se a *cwnd* < *ssthresh*, a janela é conduzida seguindo o Começo Lento;
3. Se a *cwnd* >= *ssthresh*, a janela é conduzida seguindo o Evitar Congestionamento.

3.3.1.5. Retransmissão Rápida (*Fast Retransmit*)

Seu objetivo é acelerar a retransmissão dos pacotes quando ocorrer congestionamento.

O funcionamento é simples, quando os segmentos são enviados fora de ordem, devido às características do protocolo *IP*, é permitido um máximo de dois reconhecimentos duplicados até o final do processamento destes segmentos e no máximo três reconhecimentos duplicados no total. Caso ultrapasse, é retransmitido o segmento, evitando a perda de tempo ocasionada pelo estouro do temporizador e acelerando a retransmissão.

3.3.1.6. Recuperação Rápida (*Fast Recovery*)

É um melhoramento do algoritmo de Retransmissão Rápida, pois aumenta a eficiência de utilização do *link*.

Ele é resumido da seguinte forma [4]:

1. Quando é atingido o limite de três reconhecimentos duplicados consecutivos, a *ssthresh* passa a ser *cwnd*/2;
2. Então, o segmento é retransmitido;
3. A *cwnd* passa a ser *ssthresh* + 3**MSS*;
4. A cada novo reconhecimento duplicado, incrementa-se a *cwnd* de 1**MSS* e transmite um novo segmento;
5. Ao chegar um novo reconhecimento, a *cwnd* passa a ser a *ssthresh*;

3.3.2. Opção de Reconhecimentos Seletivos

O uso de reconhecimentos seletivos (S*ACK*) por parte do *TCP* proporciona ao transmissor maiores informações sobre os segmentos que foram recebidos de fato, retransmitindo somente o necessário, otimizando a recuperação das perdas.

3.3.2.1. Comportamento do Receptor

Ao gerar um S*ACK*, o receptor deve obedecer às seguintes regras [4]:

1. O primeiro bloco mostra a mudança mais recente no estado do *buffer* do receptor;
2. O receptor deve incluir o maior número possível de blocos;
3. Os blocos de reconhecimento seletivo devem ser preenchidos do mais recente (primeiro bloco) para o mais antigo.

A última regra, garante que no pior caso, ocorrerá uma retransmissão desnecessária dos segmentos de um bloco.

Os segmentos recebidos são mantidos no *buffer* até a aplicação lê-los e em seguida são esvaziados. Se faltar espaço no *buffer* o receptor descarta os segmentos reconhecidos pelo *SACK*.

3.3.2.2. Comportamento do Transmissor

No lado do transmissor é utilizada uma *flag*, indicando que o segmento foi reportado em um bloco. Ao receber um reconhecimento seletivo a *flag* passa a ser 1, assim somente os com a *flag* 0 serão retransmitidos se necessário.

Quando acontecer um *timeout*, as *flags* da fila de transmissão são zeradas, indicando um descarte dos segmentos já reconhecidos. Com isto, é retransmitido o segmento do último reconhecimento independente de *flags*.

O segmento é apagado do *buffer* somente quando for reconhecido.

3.3.3. Implementações do *TCP*

As implementações do TCP foram sofrendo vários ajustes para otimizar o controle do congestionamento da rede ao longo do tempo, daí surgiram as variantes do TCP, cada uma com um melhoramento da anterior.

A seguir, vamos explorar os acréscimos de cada uma delas: *Tahoe*, *Reno*, *New Reno*, *Sack* e *Vegas*.

3.3.3.1. Tahoe

As primeiras implementações do *TCP* não incluíam qualquer mecanismo de controle de congestionamento [5]. Elas usavam a janela do receptor para transmissão e um temporizador de retransmissão. Isto resultava que todos os segmentos após o segmento dado por perdido eram retransmitidos.

O *Tahoe* foi a primeira implementação do *TCP* a incluir o controle de congestionamento [5]. Ele usa o algoritmo de Começo Lento, o Evitar Congestionamento e o Retransmissão Rápida, juntamente com modificações no *RTT*.

Como desvantagem, ele dispara várias vezes o algoritmo de Começo Lento, diminuindo o desempenho da rede.

3.3.3.2. Reno

É um melhoramento do *Tahoe*, adicionando os algoritmos de Retransmissão Rápida e Recuperação Rápida e é utilizado na maioria dos dispositivos ligados à *Internet*.

Tanto o *Reno*, quanto o *Tahoe*, atribuem um segmento para a janela de congestionamento até um *timeout* [5]. No caso do *Reno*, que utiliza o Retransmitir Rápido, a transmissão de um segmento perdido é disparada e executado depois que três reconhecimentos duplicados são recebidos antes do *timeout* ser alcançado.

A Recuperação Rápida faz com que a janela de congestionamento aumente de acordo com o número de reconhecimentos duplicados anteriores a um novo reconhecimento.

Uma desvantagem do *Reno* é que ele não retorna ao algoritmo Começo Lento, quando múltiplos segmentos são perdidos e dispara o algoritmo Evitar Congestionamento. Conseqüentemente, a *cwnd* passa a ser a metade de seu valor anterior várias vezes e utiliza o Começo Lento somente quando o *timeout* expirar.

3.3.3.3. *New Reno*

É uma otimização do *Reno*, quando múltiplas perdas de pacotes acontecem em uma única janela de transmissão [5]. Ele inclui uma modificação no algoritmo de Recuperação Rápida*,* eliminando a necessidade de esperar por um estouro do temporizador no caso dos múltiplos descartes [11].

Na sua modificação, ele deixa o algoritmo de Recuperação Rápida após receber um reconhecimento de todos os segmentos, inclusive o que foi recuperado.

Sua desvantagem é que a retransmissão de mais de um segmento a cada *RTT* provoca atrasos no envio dos próximos segmentos.

Outra novidade é que ele realiza reconhecimentos parciais [5] e se mantém no algoritmo de Retransmissão Rápida, evitando múltiplas reduções na *cwnd*. Quando ocorre perda de um pacote, a próxima informação será enviada após o reconhecimento do pacote retransmitido. Caso aconteça uma única perda, o reconhecimento confirmará a chegada de todos os pacotes, antes do algoritmo de Retransmissão Rápida e se acontecer múltiplas perdas o reconhecimento confirmará os segmento até a próxima perda.

3.3.3.4. SACK

É ideal para redes de alta velocidade e alto atraso, pois o transmissor obtém informações sobre os segmentos recebidos e retransmite em um único *RTT* os múltiplos segmentos perdidos.

Na implementação de reconhecimentos seletivos, não há modificações nos algoritmos de controle de congestionamento, mas difere na política de retransmissões.

3.3.3.5. Vegas

O seu objetivo é antecipar o princípio do congestionamento, acompanhando a taxa real e a esperada. Desta forma, é feito um ajuste na taxa de envio dos dados.

É uma implementação proposta para o *TCP,* que apresenta algumas modificações mais profundas nos mecanismos tradicionais, descrito em [16] e [17]. Tais modificações ocorrem do lado do transmissor, através de um algoritmo para o início de conexão e outro para retransmissões.

Através do acompanhamento da banda disponível, é detectada e evita-se a perda de segmentos. O crescimento exponencial de segmentos a serem enviados ocorre a cada dois *RTT* e durante um deles a janela de congestionamento é fixada. Em seguida é feita uma comparação entre a taxa real e a esperada. Se a taxa real é menor que a esperada, é acionado o algoritmo de Começo Lento.

O algoritmo de Evitar Congestionamento, para esta implementação, faz uma estimativa baseada na vazão. Da mesma forma, a vazão esperada e a real são comparadas e tenta-se mantê-la em uma faixa com um limite mínimo e um máximo, manipulado pela janela de congestionamento (*cwnd*).

Após a comparação entre as vazões, a implementação obtém um parâmetro para decidir se a janela de congestionamento aumenta ou diminui.

Em resumo, a quantidade de *bytes* na rede é diretamente proporcional à vazão esperada. Como a janela aumenta a cada *RTT*, a quantidade de bytes na rede aumenta. Isto garante um monitoramento e controle da quantidade de dados não enviados.

Outra modificação acrescida no *Vegas*, está relacionada à precisão do relógio nos temporizadores, eliminando a dependência neste tipo de situação [16].

3.4. Outros Fatores que Influenciam o Desempenho do *Link*

3.4.1. Otimização do Tamanho de *Buffer*

Na maioria dos dispositivos, os recursos de memória são escassos, como é o caso em pequenos dispositivos tais como *PDAs*. Para estes dispositivos, o tamanho do *buffer* é mantido pequeno. Os *buffers* de dados no *Bluetooth* são para o *L2CAP* e a Banda Básica.

3.4.2. Algoritmos de Escalonamento

Múltiplas sessões da camada de transporte compartilham o *link wireless* na *piconet* quando múltiplos escravos estão ativos ou quando um escravo tem múltiplas conexões de dados. O dispositivo mestre usando um escalonamento *Round Robin (RR)*, alcança justo compartilhamento da largura de banda e alta utilização do *link* quando cada conexão tem igual fluxo de dados. Em uma situação típica, porém, cada escravo na *piconet* tem variação de dados na taxa de entrada. Conseqüentemente, numerosos *slots* da Banda Básica são perdidos pela fonte do *polling* com baixa taxa de entrada, desse modo diminui a utilização do *link*, aumenta o atraso do enfileiramento e principalmente torna o compartilhamento injusto na largura de banda. Em [3], três algoritmos de escalonamento são apresentados: o Fluxo Adaptativo baseado na Varredura *(AFP),* o *Sticky* e o Fluxo Adaptativo *Sticky* baseado na Varredura *(StickyAFP).*

3.4.3. Manipulação de Erros

Para o esquema *ARQ*, o *timeout* é quantificado por um número máximo de retransmissões. O propósito do esquema *FEC* na carga útil dos dados é para reduzir o número de retransmissões. Porém, em um ambiente razoavelmente livre de erros, o *FEC* resulta em uma sobrecarga desnecessária, que reduz a vazão. Usando um modelo de canal [9] Dois Estados *Markov*, temos como estudar o efeito do uso do *FEC* na carga útil da Banda Básica e da variação do número máximo de retransmissões no esquema *ARQ* na vazão de dados.

Há uma versão de algoritmo de escalonamento [10], o Pacote Dependente do Estado do Canal *(CSDP)* para melhorar a vazão de dados sobre *links wireless*. Ele encontra um pacote perdido (indicado por receber um reconhecimento negativo) e as políticas do *CSDP* adiam a retransmissão para aquele escravo até o próximo *polling*. Se o comprimento do período adiado for maior que o período de *timeout* do *TCP*, a origem receberá o *timeout* e retransmitirá uma cópia do pacote com atraso, deste modo, desnecessariamente aumenta a carga no sistema.

3.4.4. Número de Conexões *SCO*

O máximo de conexões *SCO* [1] é de até três *links* simultâneos, para suportar tráfego do tipo tempo real (tal como voz) pelo mestre. O mestre irá enviar pacotes *SCO* em intervalos regulares. Visto que os *slots* são reservados nos *links SCO*, os pacotes *ACL* não são suportados na presença de três *links SCO*. Na presença de dois *links SCO*, somente pacotes *ACL* de 1 *slot* podem ser enviados. Na presença de um *link SCO*, pacotes *ACL* de 1 e 3 *slots* podem ser suportados.

3.4.5. Mecanismos de *QoS*

O *L2CAP* permite aplicações que demandam *QoS* com certos parâmetros [1], que são: largura de banda, latência e variação de atraso. O *L2CAP* checa se o *link* prove isto, caso positivo o *L2CAP* ainda checa se é possível.

O mestre então envia um comando de configuração do *QoS* com os parâmetros que dependem da aplicação para a qual a conexão é requerida.

O *LMP* passa os parâmetros de *QoS* para o escalonador (baseado no *RR*) que descobre se a conexão pode ser aceita ou não.

Depois que a negociação de *QoS* no *LMP* acaba, ele envia um sinal para o dispositivo. O dispositivo então envia um sinal de reconhecimento para o *L2CAP*. A sinalização é realizada pelo *L2CAP*.

O mapeamento e as negociações de *QoS* são feitos apenas uma vez por conexão *ACL*. Os requisitos de *QoS* para aplicações padrão (*Telnet* e *ftp*) são levados ao *MAC* (escalonador) através de especificações de fluxo.

3.5. Considerações Finais

Neste capítulo foram apresentados os fatores que influenciam no desempenho do tráfego de dados assíncronos sobre o *link Bluetooth*. Foi destacado dois destes fatores (*SAR* e variantes do *TCP*), que foram simulados para observar o comportamento dos seus algoritmos, dando condições para apontar, baseado nos resultados, qual a melhor opção para a otimização do *link*.

No próximo capítulo são apresentados os resultados das simulações realizadas com os dois fatores citados acima. Estas simulações foram basea-

das em dois cenários envolvendo tráfego de dados com o protocolo *TCP* e tráfego gerados pelo *CBR*.

Capítulo 4.0 – Análise de Desempenho do Tráfego de Dados Assíncronos

Este capítulo faz uma avaliação do desempenho do tráfego de dados assíncronos, sob a influência de dois fatores (variantes de *TCP* e *SAR*) que interferem na otimização do uso de seus *link*s.

4.1. Técnica de Avaliação de Desempenho

A técnica de avaliação utilizada foi a simulação, que se baseia na execução e construção de programas que se comportam como o sistema analisado. O simulador *Network Simulator (NS)* [18] foi escolhido pela larga quantidade de trabalhos acadêmicos desenvolvidos nele, por possuir código fonte aberto e ser *free*. Para obter a pilha de protocolos do *Bluetooth*, foi adicionado ao *NS* a extensão *BlueHoc* [19].

Foi necessário o desenvolvimento de trechos de código para implementar os algoritmos *SAR* (ver anexos) e o acréscimo da rotina do tráfego *CBR,* que na versão 3.0 do *BlueHoc* não existe.

4.2. Ambiente de Simulação

O *hardware* e *software* utilizados nas simulações foram:

- Micro computador *Pentium III* 1 Ghz;
- Memória *RAM* de 256 *MB*;
- Espaço em disco de 500 *MB* para os resultados da simulação;
- Sistema Operacional Linux (Conectiva) versão 8.0;
- *Network Simulator* versão 2.1b8.
- *BlueHoc* versão 3.0

4.3. Métricas

Para obter uma análise mais detalhada e termos condições de apontar o melhor resultado obtido nas simulações, foi utilizado quatro métricas, nas quais foram escolhidas devidas as características importantes para medirmos a transferências dos dados sobre os *links* assíncronos.

As métricas para avaliar o desempenho do tráfego de dados assíncronos sobre *links Bluetooth* foram:

- Atraso: atraso fim a fim entre origem e destino dos pacotes;
- Variação de atraso: diferença entre o atraso máximo e o mínimo;
- Vazão: vazão dos *bytes* em um determinado intervalo de tempo;
- Perda de pacotes: porcentagem de perda de pacotes nas transmissões de dados.

4.4. Topologia e Parâmetros da Simulação

As simulações foram baseadas em dois cenários. O primeiro, com seis dispositivos, sendo um mestre e cinco escravos (Veja a Tabela 6). Cada escravo representa uma das cinco variantes do *TCP* (*Reno, New Reno , Tahoe, Sack* e *Vegas*) que são estudadas anteriormente.

Dispositivos	Aplicação	Protocolo	*StartTime*
Mestre	-	-	0
Escravo1	*FTP*	*TCP Reno*	0,1
Escravo2	*FTP*	*TCP New Reno*	0,2
Escravo3	*FTP*	*TCP Tahoe*	0,3
Escravo4	*FTP*	*TCP Sack*	0,4
Escravo5	*FTP*	*TCP Vegas*	0,5

Tabela 6: Parâmetros do Cenário 1 – Variantes do *TCP*

O segundo cenário tem três dispositivos, sendo um mestre e dois escravos (Veja a Tabela 7). Neste caso, os escravos com tráfego *CBR* foram utilizados para testarmos os algoritmos *SAR* (*BF*, *OSU* e Randômico) implementados, pois como o tráfego de dados *CBR* resulta em uma constante, da mesma forma, os algoritmos implementados têm que obter o mesmo comportamento.

Dispositivo	Aplicação	Protocolo	*StartTime*
Mestre	-	-	0,0
Escravo1	*CBR*	*UDP*	0,1
Escravo2	*CBR*	*UDP*	0,2

Tabela 7: Parâmetros do Cenário 2 – Tráfego *CBR*

Nos dois cenários, o tempo simulado (*SimulationTime*) foi de 50 segundos, tempo máximo possível com a versão do *BlueHoc* existente. O *time-out* do procedimento de Investigação (*InquiryTimeout)* foi de 10,24 segundos, tempo ideal e sugerido pela documentação do *BlueHoc* [19]. O momento do procedimento de Busca de Investigação (*InquiryScanOffset)* de todos os escravos foi de 1 segundo, o mesmo para todos os escravos para não alterar os resultados da simulação, visto que são cinco tipos diferentes de TCP sendo submetido às mesmas condições na simulação.

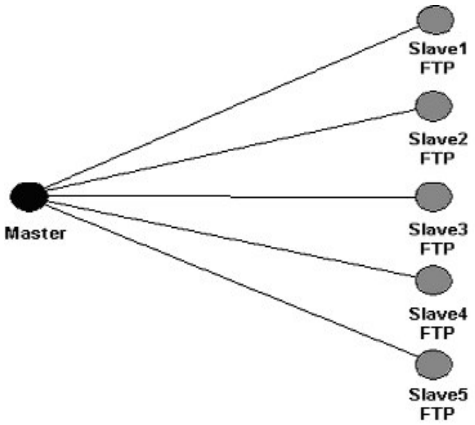


Figura 13: Primeiro Cenário da Simulação (Variantes do *TCP*)

No primeiro cenário (Figura 13), para os cinco *links* foram utilizadas a mesma aplicação *(Application) FTP*, que faz uso do protocolo *TCP* da camada de transporte da arquitetura *TCP/IP*. Assim, como citado anteriormente, na simulação foram mantidas as mesmas condições para os cinco escravos, permanecendo no mesmo cenário todas as variantes do *TCP* com parâmetros iguais. As transmissões foram iniciadas *(StartTime)* no mestre a 0 segundos e em cada escravo a 0,1, 0,2, 0,3, 0,4 e 0,5 segundos respectivamente (Tabela 6). Cada escravo teve início em tempos diferentes, mas com pouca diferença entre eles, para não termos uma situação em que o mestre, nos

primeiros momentos, fosse sobrecarregado. À distância entre o mestre e todos os escravos é menor que 10 metros (alcance máximo para dispositivos sem amplificador de sinal). Com isto, o cenário foi idealizado dentro dos limites de distância para não afetar o desempenho dos escravos.

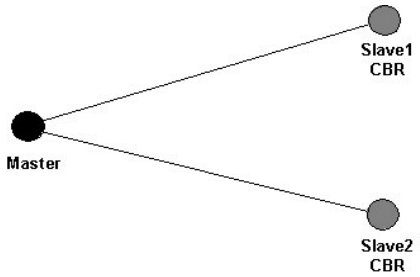


Figura 14: Segundo Cenário da Simulação (Tráfego *CBR*)

No segundo cenário (Figura 14), para os dois *links* foi utilizada a mesma aplicação *(Application),* gerando um tráfego *CBR* que faz uso do protocolo *UDP* da camada de transporte da arquitetura *TCP/IP*. O tráfego *CBR* gera uma constante, garantindo o funcionamento correto dos algoritmos implementados após os resultados das suas simulações. Como a análise do tráfego proposta na dissertação está relacionada com variantes do *TCP*, o resultado deste cenário será somente para efeito de verificação destes algoritmos *SAR*. As transmissões foram iniciadas *(StartTime)* no mestre a 0 segundos e em cada escravo a 0,1 e 0,2 segundos respectivamente (Tabela 7). Como no cenário anterior, os tempos são diferentes, mas próximos para não causar uma sobrecarga no mestre. Da mesma forma que o cenário 1, à distância entre o mestre e todos os escravos é menor que 10 metros, dentro dos limites da tecnologia para não interferir no resultado.

Os parâmetros *QoS* utilizados nas duas topologias podem ser encontrados na Tabela 8. Os valores escolhidos são o mínimo estabelecido para as aplicações mencionadas, permitindo avaliar o desempenho do tráfego de dados. Estes valores foram obtidos na própria implementação do *BlueHoc* [19]. Para os parâmetros com valor -1, significa o valor máximo para aquele parâmetro.

Parâmetros	*CBR*	*FTP*
MTU	1000 *bytes*	1000 *bytes*
Taxa da Ficha	64	-1
Balde de Ficha	-1	-1
Comprimento da fila de entrada de dados	30	-1
Pico da Largura de Banda	100000	-1
Latência	-1	0
Sensibilidade à perda	1	0

Nota: 0 (desligado), 1 (ligado) e –1 (máximo)

Tabela 8: Parâmetros *QoS* da Simulação

4.5. Resultados

Após várias simulações repetidas, exaustivamente, para garantir a confiabilidade dos resultados, veremos a seguir os valores obtidos nas simulações dos cenários citados.

Para os valores de atraso obtidos (Tabela 9, Tabela 10 e Tabela 11), através das simulações dos algoritmos *SAR* (Melhor Ajuste-*BF,* Otimização da Utilização do *Slot-OSU* e Randômico), com as variantes do *TCP* (*Reno, New Reno, Tahoe, Sack* e *Vegas*), podemos observar que o menor atraso médio (0,3374 segundos) e o menor atraso máximo (0,6969 segundos) é alcançado com o algoritmo *SAR-BF* e com o *TCP Vegas*. Da mesma forma, a variação de atraso menor (0,5488 segundos) foi alcançada pelo algoritmo *SAR-BF* com o *TCP Vegas*. Isto se deve a otimização dos algoritmos de congestionamento aplicados no *TCP Vegas* e a política de melhor aproveitamento no algoritmo *SAR-BF*.

Apesar do melhor resultado ser obtido utilizando o *TCP Vegas*, mas devido não existir implementações em funcionamento desta variante, o *TCP Sack*, juntamente com o algoritmo *SAR-BF*, obtiveram resultados melhores que as outras variantes. O seu atraso médio é de 0,6449 segundos e o atraso máximo de 0,8856 segundos. A sua variação de atraso também foi a menor, 0,8838 segundos. No caso do *TCP Sack*, o reconhecimento seletivo ajuda a

diminuir o atraso, permitindo a transmissão de dados com uma janela mais flexível que as outras variantes.

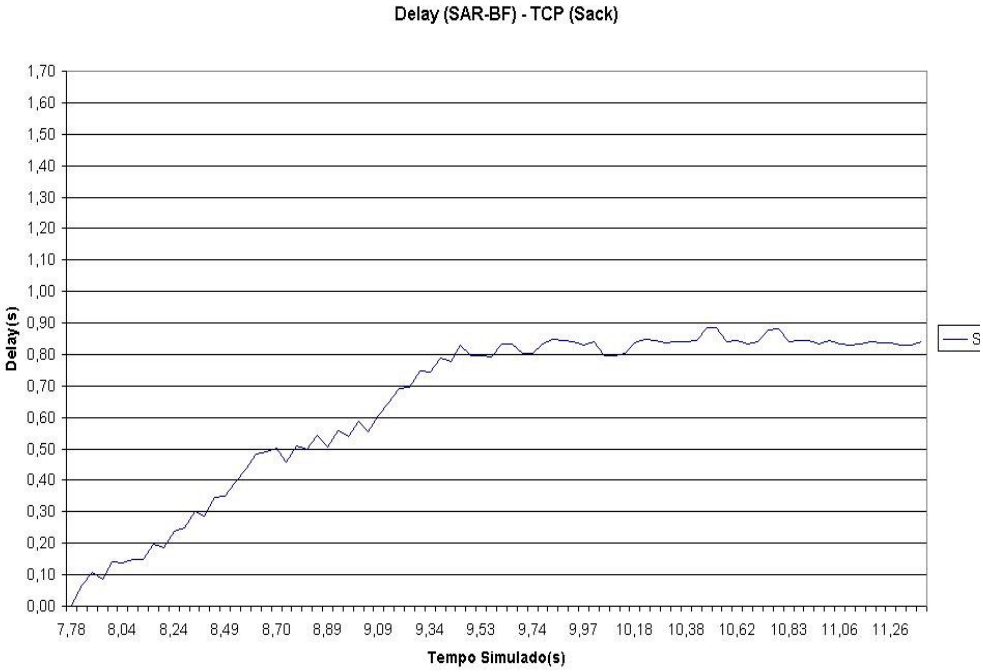


Figura 15: Atraso com *SAR-BF* e *TCP New Reno*

Observe na Figura 15, o comportamento do atraso no algoritmo *SAR-BF* com o *TCP Sack*. Até os 9,5 segundos o atraso cresce, mas logo se mantêm quase constante e com a segunda menor média entre as outras variantes, atrás somente da variante do *TCP Vegas*.

Atraso *(SAR-BF)* - Variantes do *TCP*					
	Escravo1 ***TCP Reno***	**Escravo2** ***TCP New Reno***	**Escravo3** ***TCP Tahoe***	**Escravo4** ***TCP Sack***	**Escravo5** ***TCP Vegas***
	Atraso (s)	**Atraso (s)**	**Atraso (s)**	**Atraso (s)**	**Atraso (s)**
Média	0,6708	0,8841	0,6649	0,6449	0,3374
Máximo	0,9944	1,5756	0,9931	0,8856	0,6969
Mínimo	0,0019	0,0019	0,0019	0,0019	0,1481
Variação	0,9925	1,5738	0,9913	0,8838	0,5488

Tabela 9: Distribuição do Atraso com *SAR-BF* e Variantes do *TCP*

Atraso *(SAR-OSU)* - Variantes do *TCP*					
	Escravo1 ***TCP Reno***	**Escravo2** ***TCP New Reno***	**Escravo3** ***TCP Tahoe***	**Escravo4** ***TCP Sack***	**Escravo5** ***TCP Vegas***
	Atraso (s)	**Atraso (s)**	**Atraso (s)**	**Atraso (s)**	**Atraso (s)**
Média	1,3374	1,9140	1,0999	1,0848	0,4695
Máximo	1,6794	2,3231	1,3044	1,2656	0,8569
Mínimo	0,0019	0,0019	0,0019	0,0019	0,1856
Variação	1,6775	2,3213	1,3025	1,2638	0,6713

Tabela 10: Distribuição do Atraso com *SAR-OSU* e Variantes do *TCP*

Atraso (*SAR*-Randômico) - Variantes do *TCP*					
	Escravo1 ***TCP Reno***	**Escravo2** ***TCP New Reno***	**Escravo3** ***TCP Tahoe***	**Escravo4** ***TCP Sack***	**Escravo5** ***TCP Vegas***
	Atraso (s)	**Atraso (s)**	**Atraso (s)**	**Atraso (s)**	**Atraso (s)**
Média	2,1906	9,5198	5,8574	1,6693	2,9966
Máximo	2,6119	14,1181	7,4031	1,9494	4,7219
Mínimo	0,0031	0,0019	0,0019	0,0019	0,9694
Variação	2,6088	14,1162	7,4013	1,9475	3,7525

Tabela 11: Distribuição de Atraso com *SAR*-Randômico e Variantes do *TCP*

Nos valores de atraso obtidos, através das simulações dos algoritmos *SAR* (*BF, OSU* e Randômico) com tráfego *CBR*, foram constados os valores constantes, com pouquíssima variação entre eles, comprovando a validade das implementações dos mesmos.

Para os valores de vazão obtidos (Tabela 12, Tabela 13 e Tabela 14), através das simulações dos algoritmos *SAR* (*BF, OSU* e Randômico), com as variantes do *TCP*, podemos observar que a maior média de vazão (259.089 bytes) em um intervalo médio de tempo de 9,66 segundos e a maior vazão máxima (800.000 bytes) é alcançado com o algoritmo *SAR-BF* e o *TCP Vegas*. Isto se deve ao melhor desempenho alcançado pela otimização dos algoritmos de congestionamento do *TCP Vegas* e a redução da perda de largura de banda nos pacotes da Banda Básica suportado pelo algoritmo *SAR-BF*.

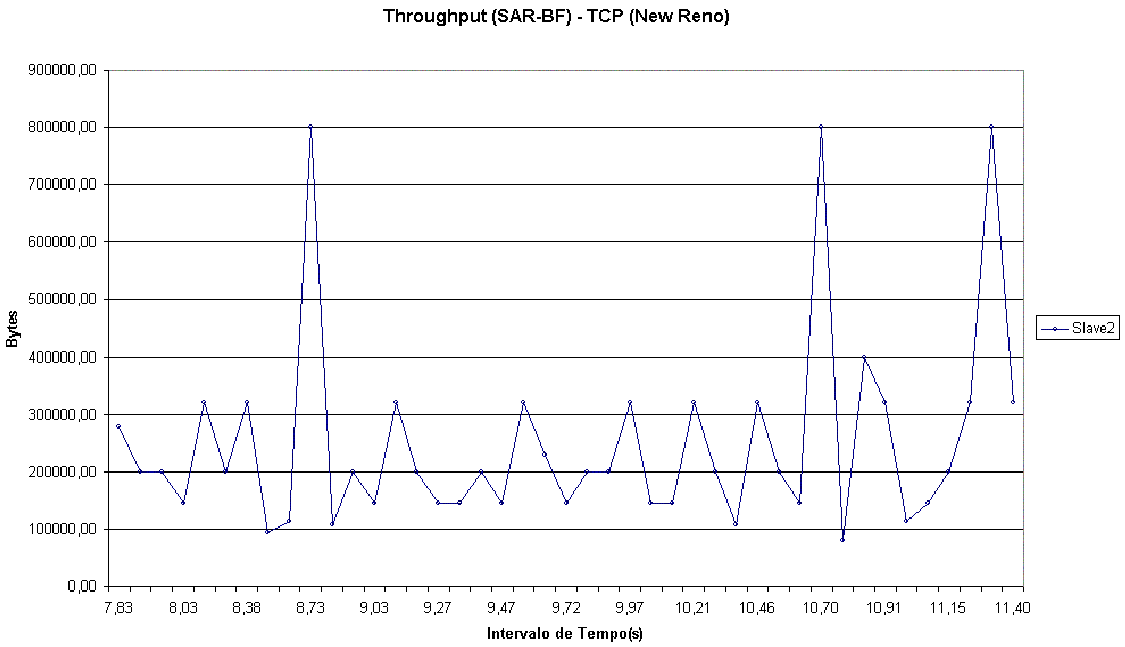


Figura 16: Vazão com *SAR-BF* e *TCP New Reno*

Na Figura 16, acompanhe o comportamento da vazão no algoritmo *SAR-BF* com o *TCP New Reno*. A vazão, neste caso, fica variando entre 100.000 e aproximadamente 320.000 *bytes* e com picos de 800.000 *bytes* em alguns intervalos de tempo, pois os algoritmos de controle de congestionamento no *TCP New Reno* permitem que em alguns momentos a vazão chegue no seu valor máximo.

Apesar do melhor resultado ser obtido utilizando novamente o *TCP Vegas,* mas devido não existir implementações em funcionamento desta variante, o *TCP New Reno*, juntamente com o algoritmo *SAR-BF*, obtiveram resultados melhores que as outras variantes. A sua vazão média é de 250.616 *bytes* em um intervalo de tempo de 9,68 segundos e também com uma vazão máxima de 800.000 *bytes*.

Nos valores de vazão obtidos, através das simulações dos algoritmos *SAR* (*BF, OSU* e Randômico), com tráfego *CBR*, foram constados os valores

constantes, com pouquíssima variação entre eles, comprovando novamente a validade das implementações dos mesmos.

Vazão *(SAR-BF)* - Variantes do *TCP*										
	Escravo1 ***TCP Reno***		**Escravo2** ***TCP New Reno***		**Escravo3** ***TCP Tahoe***		**Escravo4** ***TCP Sack***		**Escravo5** ***TCP Vegas***	
	T(s)	***Bytes***	**T(s)**	***Bytes***	**T(s)**	***Bytes***	**T(s)**	***Bytes***	**T(s)**	***Bytes***
Média	9,64	213.136	9,68	250.616	9,54	213.658	9,61	230.522	9,66	259.089
Máximo	11,38	800.000	11,40	800.000	11,39	800.000	11.41	800.000	11,30	800.000
Mínimo	7,80	4.923	7,83	80.000	7,82	64.000	7,78	5	7,94	94.118

Tabela 12: Vazão com *SAR-BF* e Variantes do *TCP*

Vazão *(SAR-OSU)* – Variantes do *TCP*										
	Escravo1 ***TCP Reno***		**Escravo2** ***TCP New Reno***		**Escravo3** ***TCP Tahoe***		**Escravo4** ***TCP Sack***		**Escravo5** ***TCP Vegas***	
	T(s)	***Bytes***	**T(s)**	***Bytes***	**T(s)**	***Bytes***	**T(s)**	***Bytes***	**T(s)**	***Bytes***
Média	28,79	165.020	28,94	194.856	28,80	154.919	28,84	158.566	28,83	240.981
Máximo	49,99	400.000	49,97	400.000	49,96	400.000	49,93	400.000	49,63	400.000
Mínimo	7,80	4.571	7,83	6.400	7,81	6.400	7,78	5	8,12	42.105

Tabela 13: Vazão com *SAR-OSU* e Variantes do *TCP*

Vazão (*SAR*-Randômico) - Variantes do *TCP*										
	Escravo1 ***TCP Reno***		**Escravo2** ***TCP New Reno***		**Escravo3** ***TCP Tahoe***		**Escravo4** ***TCP Sack***		**Escravo5** ***TCP Vegas***	
	T(s)	***Bytes***	**T(s)**	***Bytes***	**T(s)**	***Bytes***	**T(s)**	***Bytes***	**T(s)**	***Bytes***
Média	29,06	74.693	28,76	102.214	28,81	132.606	28,80	52.674	28,84	80.494
Máximo	49,92	533.333	49,83	400.000	49,93	800.000	49,98	400.000	48,92	200.000
Mínimo	7,85	23.881	7,88	6.400	7,87	4.923	7,80	5	8,75	23.529

Tabela 14: Vazão com *SAR*-Randômico e Variantes do *TCP*

Quanto às porcentagens de perda de pacotes obtidas (Tabela 15), através das simulações dos algoritmos *SAR* (*BF, OSU* e Randômico), com as variantes do *TCP*, pode-se observar que a menor porcentagem (0,01) é alcançada com o algoritmo *SAR-BF* e a variante do *TCP Vegas*. Como já mencionado anteriormente, devido as característica de melhor aproveitamento no

SAR-BF e a otimização dos algoritmos de congestionamento do *Vegas*, fazem com que os dois juntos, resultem em melhor desempenho.

Novamente, devido a não implementação do *TCP Vegas*, a outra menor porcentagem, também de 0,01 é alcançada pela variante do *TCP New Reno*.

Mestre	**Escravo1** ***TCP Reno***	**Escravo2** ***TCP New Reno***	**Escravo3** ***TCP Tahoe***	**Escravo4** ***TCP Sack***	**Escravo5** ***TCP Vegas***
% Perda de Pacotes (*SAR-BF*) - Variantes do *TCP*					
0,06	0,07	0,01	0,03	0,04	0,01
% Perda de Pacotes (*SAR-OSU*) - Variantes do *TCP*					
0,21	0,33	0,09	0,17	0,15	0,09
% Perda de Pacotes (*SAR*-Randômico) - Variantes do *TCP*					
0,05	0,20	0,02	0,05	0,20	0,03

Tabela 15: Perda de pacotes (%) em todos os algoritmos *SAR* e Variantes do *TCP*

Capítulo 5.0 – Conclusão

Este capítulo apresenta o resultado final da análise e testes realizados neste trabalho. São apontados, também, trabalhos futuros e referências bibliográficas utilizadas.

5.1. Considerações Finais

As aplicações de dados, que funcionam sobre o *Bluetooth*, tais como *HTTP*, *FTP* e *real audio,* necessitam de protocolos da camada de transporte da arquitetura *TCP/IP* (*TCP* e *UDP*) para enviar pacotes sobre os *links wireless*. Com base nisso, foi necessário fazer uma análise para aumentar o desempenho destas aplicações na tentativa de otimizar os recursos existentes, que na maioria dos dispositivos para este tipo de tecnologia são escassos.

Alguns fatores influenciam o desempenho do tráfego de dados sobre estes dispositivos, normalmente, de tamanho reduzido e pouca autonomia de uso. Portanto, há uma crescente busca de torná-los mais eficientes, através do melhor aproveitamento de seus recursos.

Três algoritmos de Segmentação e Remontagem *(SAR)*: o de Melhor Ajuste *(BF)*, que tem como característica diminuir a perda da largura de banda; o de Otimização da Utilização do *Slot (OSU)*, que tem como característica a diminuição do atraso e o Randômico, que decide qual o tamanho do *slot* a ser utilizado baseado em uma função randômica uniforme e as cinco variantes do *TCP: Reno, New Reno, Tahoe, Sack* e *Vegas* com seus algoritmos de controle de congestionamento, foram os dois fatores simulados no *NS* [18] e sua extensão para *Bluetooth*, o *BlueHoc* [19]. Os parâmetros as quais foram analisados os desempenhos do tráfego de dados assíncronos são o atraso, a variação de atraso, a porcentagem de perda de pacotes e a vazão.

Concluiu-se, através dos resultados, que o melhor algoritmo de Segmentação e Remontagem *(SAR)* é o de Melhor Ajuste *(BF)*, que apresenta o menor atraso médio, menor variação de atraso, menor porcentagem de perda de pacotes e um melhor desempenho da vazão média.

Para as variantes do *TCP,* o que obteve um menor atraso médio, menor variação de atraso, menor porcentagem de perda de pacotes e um melhor desempenho da vazão média é o *TCP Vegas*, que ainda não possui

uma implementação. Com isto, o *TCP New Reno* é a melhor implementação em uso, que apesar de ter um atraso médio e uma variação de atraso um pouco maior que os demais *TCP*, mas proporciona uma vazão média e porcentagem de perda de pacotes consideravelmente melhores que os outros.

Assim, a melhor escolha para otimizar o *link* de dados assíncronos sobre *Bluetooth* é o algoritmo *SAR* de Melhor Ajuste *(BF),* juntamente com o *TCP New Reno*.

5.2. Contribuições

As contribuições proporcionadas por este trabalho estão relacionadas principalmente com os dois fatores analisados, que são:

- Variantes do protocolo *TCP*, que faz uma comparação entre os algoritmos de controle de congestionamento utilizado, buscando um melhor desempenho nas transmissões envolvendo tráfego *Internet* nos dispositivos *Bluetooth*;
- Desenvolvimento de algoritmos *SAR* para o simulador *BlueHoc*, fazendo uma comparação entre eles, buscando uma solução mais adequada aos dispositivos *Bluetooth*;
- Melhor aproveitamento dos recursos de dispositivos *Bluetooth*, através da escolha do algoritmo *SAR* e da variante do *TCP*, que juntos apresentaram um melhor desempenho;
- Orientar, através dos cenários e resultados obtidos, outros trabalhos relacionados com a tecnologia *Bluetooth* e o simulador *BlueHoc*.

5.3. Trabalhos Futuros

Propõe-se uma análise mais aprofundada nos outros fatores que influenciam o desempenho do tráfego de dados assíncronos em *links Bluetooth*.

Estes fatores, quando analisados em conjunto, podem apresentar uma solução mais promissora com relação ao desempenho do *link* de dados, complementando os objetivos deste trabalho.

Para isso, será necessário avaliar o desempenho dos seguintes fatores:

- Tamanho do *buffer*;
- Algoritmos de escalonamento;
- Algoritmos de manipulação de erros;
- Número de conexões *SCO*;
- Qualidade de Serviço *(QoS)*.

5.4. Referências Bibliográficas

[1] Bluetooth SIG, "Bluetooth Specification", www.Bluetooth.com, Jan. 2003.

[2] J. Haartsen, "The Bluetooth Radio System", IEEE Personal Communications, Vol. 7, No. 1, pp.28-36, Feb. 2000.

[3] Abhishek Das, Abhishek Ghose, Ashu Razdan, Huzur Saran & Rajeev Shorey, " Enhancing Performance of Asynchronous Data Traffic over the Bluetooth Wireless Ad-hoc Network", IBM India Research Laboratory, Aug. 2001.

[4] Guilherme P. T. Santos e Saulo V. de Vasconcellos. Avaliação por Simulação dos Mecanismos de Controle de Congestionamento do TCP. Escola de Engenharia-UFRJ. Rio de Janeiro-RJ, Agosto 2000.

[5] Jorge Luiz de C. e Silva, Marcília A. Campos, José N. de Souza e Paulo Roberto F. Cunha. Análise Estatística de Descarte de Pacotes em Redes TCP. UFPE e UFC. SBRC2002.

[6] W. Stallings, "Wireless Communications and Networks", First Edition, 2002, Prentice Hall.

[7] W. Simpson, "The Point-to-Point Protocol (PPP)", RFC 1661, July 1994.

[8] P. Bhagwat, I. Korpeoglu, C. Bisdikian, M. Naghshineh and S. K. Tripathi, "Bluesky: A Cordless Networking Solution for Palmtop Computers", Mobicom´99, Seattle, Washington, Aug. 1999.

[9] H. S. Wang and N. Moayeri, "Finite-state Markov channel – a useful model for radio communication channels", IEEE Trans. Veh. Technol., Vol. 44, pp. 163-171, Feb. 1995.

[10] P. Bhattacharya, A. Krishna and K. Tripathi, "Using channel state dependent packet scheduling to improve TCP throughput over wireless LANs", Wireless Networks, pp. 91-102, 1997.

[11] S. Floyd e K. Fall, "Simulation-based Comparisons of *Tahoe*, *Reno*, and S*ACK TCP*", ACM Computer Communications Review, vol. 26, no. 3, pp. 42-48, julho 1996.

[12] V. Jacobson e M. J. Karels, "Congestion Avoidance and Control", em Proceedings of ACM SIGCOMM, Symposium on Communication Architectures and Protocols, pp. 314-329, 1988.

[13] V. Jacobson, "Modified TCP Congestion Avoidance Algorithm", tech. rep., E-mail to the end2end-interest mailing list, 30th. Apr. 1990. URL: ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt.

[14] J. Postel, "Transmission Control Protocol", Internet RFC 793, setembro 1981.

[15] S. Floyd e T. Henderson, "The New Reno Modification to TCP's Fast Recovery Algorithm", Internet RFC 2582, abril 1999.

[16] L. S. Brakmo e L. L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a global Internet", IEEE Journal on Selected Areas in Communications, vol. 13, no. 8, pp. 1465-1480, outubro 1995.

[17] J. S. Ahn, P. Danzig, Z. Liu e L. Yan, "Evaluation of TCP Vegas: Emulation and Experiment", em Proceedings of ACM SIGCOMM, Symposium on Communication Architectures and Protocols, 1995.

[18] Network Simulator (versão 2.1b8). Endereço eletrônico: *http://www.isi.edu/nsnam/ns/*.

[19] Pilha de protocolos *Bluetooth* - *BlueHoc* (versão 3.0). Endereço eletrônico: *http://www-124.ibm.com/developerworks/projects/bluehoc*, Jun. 2001.

Anexos

Essa parte contém *scripts* utilizados nas simulações e trechos de código que foram alterados na pilha de protocolos *Bluetooth* no *BlueHoc* [19].

Scripts da Simulação

Variantes do *TCP*

```
set Sim(Trace) OFF
set Sim(AgentTrace) OFF
set Sim(NumDevices) 6
set Sim(Transport) [list TCP/Reno TCP/NewReno TCP TCP/SACK1 TCP/Vegas]

set Sim(xpos_) [list 0.42 7.1 7.16 7.2 7.3 7.16]
set Sim(ypos_) [list 3.62 0.66 1.88 3.14 4.34 5.6]

set Sim(Application) [list FTP FTP FTP FTP FTP]
set InqTimeout 32768
set NumResponses 5

set SimulationTime 50.0
set StaRTTime [list 0 0.1 0.2 0.3 0.4 0.5]

set InqScanOffset [list 3200 3200 3200 3200 3200]

source ./proc.tcl
source ./run.tcl
```

CBR

```
set Sim(Trace) OFF
set Sim(AgentTrace) OFF
set Sim(NumDevices) 3
set Sim(Transport) [list UDP UDP]

set Sim(xpos_) [list 1.14 6.7 6.78]
set Sim(ypos_) [list 3.56 1.38 5.0]

set Sim(Application) [list Traffic/CBR Traffic/CBR]
set InqTimeout 32768
set NumResponses 2

set SimulationTime 50.0
```

```
set StartTime [list 0 0.1 0.2]

set InqScanOffset [list 3200 3200]

source ./proc.tcl
source ./run.tcl
```

Código Alterado

Arquivo *bt-drr.cc* (Escalonador *Deficit Round Robin*)

SAR-BF

```
...

uchar
BT_DRRScheduler::mapQoS (uchar ch, flowSpec* qos, con_attr* qoslm_)
{
        bt_packet_type type;
        int mtu = qos->mtu;
        double rate = qos->token_rate*1000;

        // If application is loss sensitive, use FEC
        if (qos->loss_sensitivity) {
                // select appropriate packet type based on MTU
                // BF
                if (mtu >= Payload[14])
                        type = BT_DM5;
                // BF
                else if (mtu < Payload[14] && mtu >= Payload[10])
                        type = BT_DM3;
                else
                        type = BT_DM1;
        }
        else {
                // select appropriate packet type based on MTU
                // BF
                if (mtu >= Payload[15])
                        type = BT_DH5;
```

```
                // BF
                else if (mtu < Payload[15] && mtu >= Payload[11])
                //
                        type = BT_DH3;
                else
                        type = BT_DH1;
        }
        int pi = (int)(Payload[type]*8/(rate*SlotTime));
        uchar result = updateQuanta(ch, qos->token_rate, type);
        if (result) {
                qoslm_->packet_type = type;
                qoslm_->polling_interval = pi;    // To be used when QoS is reserved
                                        // in the reverse direcetion as well
        }
        return result;
}

…
```

SAR-OSU

```
…

uchar
BT_DRRScheduler::mapQoS (uchar ch, flowSpec* qos, con_attr* qoslm_)
{
        bt_packet_type type;
        int mtu = qos->mtu;
        double rate = qos->token_rate*1000;

        // If application is loss sensitive, use FEC
        if (qos->loss_sensitivity) {
                // select appropriate packet type based on MTU
                // OSU
                if ((mtu > Payload[14]) || ((mtu <= Payload[14]) && (mtu > Payload[10])))
                        type = BT_DM5;
                // OSU
                else if (mtu <= Payload[10] && mtu > Payload[3])
                        type = BT_DM3;
                else
```

```
                type = BT_DM1;
        }
        else {
                // select appropriate packet type based on MTU
                // OSU
                if ((mtu > Payload[15]) || ((mtu <= Payload[15]) && (mtu > Payload[11])))
                        type = BT_DH5;
                // OSU
                else if (mtu <= Payload[11] && mtu > Payload[4])
                        type = BT_DH3;
                else
                        type = BT_DH1;
        }
        int pi = (int)(Payload[type]*8/(rate*SlotTime));
        uchar result = updateQuanta(ch, qos->token_rate, type);
        if (result) {
                qoslm_->packet_type = type;
                qoslm_->polling_interval = pi;    // To be used when QoS is reserved
                                        // in the reverse direcetion as well
        }
        return result;
}

…
```

SAR-Randômico

```
…

uchar
BT_DRRScheduler::mapQoS (uchar ch, flowSpec* qos, con_attr* qoslm_)
{
        bt_packet_type type;
        //int mtu = qos->mtu;
        double rate = qos->token_rate*1000;
        // Randomico
        float randomico = Random::uniform();
        //
        // If application is loss sensitive, use FEC
        if (qos->loss_sensitivity) {
```

```
                // Randomico
                if (randomico <= 0,33)
                        type = BT_DM5;
                else if (randomico <= 0,66)
                        type = BT_DM3;
                else
                        // randomico > 0,66 e < 1
                        type = BT_DM1;
        }
        else {
                // Randomico
                if (randomico <= 0,33)
                        type = BT_DH5;
                else if (randomico <= 0,66)
                        type = BT_DH3;
                else
                        // randomico > 0,66 e < 1
                        type = BT_DH1;
        }
        int pi = (int)(Payload[type]*8/(rate*SlotTime));
        uchar result = updateQuanta(ch, qos->token_rate, type);
        if (result) {
                qoslm_->packet_type = type;
                qoslm_->polling_interval = pi;    // To be used when QoS is reserved
                                        // in the reverse direcetion as well
        }
        return result;
}

…
```

Arquivo *L2CAP.cc* (Camada *L2CAP*)

SAR-BF

```
...

// get the packet type to be used
bt_packet_type
L2CAP::getPacketType(Packet* p, uchar ch)
```

```
{
        hdr_cmn* hdr = HDR_CMN(p);
        int size = hdr->size();
        uchar FEC = 0;
        // Get the largest packet type for the Connection
        bt_packet_type largest = (packetType_[ch]) ? packetType_[ch] : BT_DH5;
        bt_packet_type type;
        if (largest == BT_DM1 || largest == BT_DM3 || largest == BT_DM5)
                FEC = 1;

        if (FEC) {
                // FEC required
                // Best Fit - BF
                if (size >= Payload[14])
                        type = BT_DM5;
                // BF
                else if (size < Payload[14] && size >= Payload[10])
                        type = BT_DM3;
                else
                        type = BT_DM1;
        }
        else {
                // No FEC
                // Best Fit - BF
                if (size >= Payload[15])
                        type = BT_DH5;
                // BF
                else if (size < Payload[15] && size >= Payload[11])
                        type = BT_DH3;
                else
                        type = BT_DH1;
        }
        type = (type > largest) ? largest: type;

        return type;
}

...
```

SAR-OSU

```
...

// get the packet type to be used
bt_packet_type
L2CAP::getPacketType(Packet* p, uchar ch)
{
        hdr_cmn* hdr = HDR_CMN(p);
        int size = hdr->size();
        uchar FEC = 0;
        // Get the largest packet type for the Connection
        bt_packet_type largest = (packetType_[ch]) ? packetType_[ch] : BT_DH5;
        bt_packet_type type;
        if (largest == BT_DM1 || largest == BT_DM3 || largest == BT_DM5)
                FEC = 1;

        if (FEC) {
                // FEC required
                // Optimum Slot Utilization - OSU
                if ((size > Payload[14]) || ((size <= Payload[14]) && (size > Payload[10])))
                        type = BT_DM5;
                // OSU
                else if (size <= Payload[10] && size > Payload[3])
                        type = BT_DM3;
                else
                        type = BT_DM1;
        }
        else {
                // No FEC
                // Optimum Slot Utilization - OSU
                if ((size > Payload[15]) || ((size <= Payload[15]) && (size > Payload[11])))
                        type = BT_DH5;
                // OSU
                else if (size <= Payload[11] && size > Payload[4])
                        type = BT_DH3;
                else
                        type = BT_DH1;
        }
        type = (type > largest) ? largest: type;
```

```
        return type;
}

...
```

SAR-Randômico

```
...

// get the packet type to be used
bt_packet_type
L2CAP::getPacketType(Packet* p, uchar ch)
{
        hdr_cmn* hdr = HDR_CMN(p);
        uchar FEC = 0;
        // Randomico
        float randomico = Random::uniform();
        //
        // Get the largest packet type for the Connection
        bt_packet_type largest = (packetType_[ch]) ? packetType_[ch] : BT_DH5;
        bt_packet_type type;
        if (largest == BT_DM1 || largest == BT_DM3 || largest == BT_DM5)
                FEC = 1;

        if (FEC) {
                // FEC required
                // Randomico
                if (randomico <= 0,33)
                        type = BT_DM5;
                else if (randomico <= 0,66)
                        type = BT_DM3;
                else
                        // randomico > 0,66 e < 1
                        type = BT_DM1;
        }
        else {
                // No FEC
                // Randomico
                if (randomico <= 0,33)
```

```
                    type = BT_DH5;
                else if (randomico <= 0,66)
                    type = BT_DH3;
                else
                    // randomico > 0,66 e < 1
                    type = BT_DH1;
        }
        type = (type > largest) ? largest: type;

        return type;
}

…
```