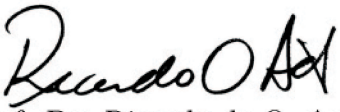
Um Estudo para o Problema de Ordenação Total de Mensagens Aplicado a Redes *Bluetooth* com Restrições Fracas de Tempo Real

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Vicente J.P. de Amorim e aprovada pela Banca Examinadora.

Campinas, 08 de Julho de 2010.



Prof. Dr. Ricardo de O. Anido
Instituto de Computação (UNICAMP)
(Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Fabiana Bezerra Müller – CRB8 / 6162

Amorim, Vicente José Peixoto de

Am68e Um estudo para o problema de ordenação total de mensagens aplicado a redes Bluetooth com restrições fracas de tempo real/Vicente José Peixoto de Amorim-- Campinas, [S.P. : s.n.], 2010.

Orientador : Ricardo de Oliveira Anido

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1.Algoritmos distribuídos. 2.Sistemas de mensagens por difusão. 3.Ordenação de mensagens. 4.Algoritmos de ordenação total. 5.Ordenação causal. 6.Redes Bluetooth. 7.Ambientes de tempo real. 8.Dispositivos móveis. 9.Computação distribuída. I. Anido, Ricardo de Oliveira. II. Universidade Estadual de Campinas. Instituto de Computação . III. Título.

Título em inglês:A study to total order message problem applied to Bluetooth networks using real time weak constraints

Palavras-chave em inglês (Keywords): 1. Distributed algorithms. 2. Message broadcast systems. 3. Message ordering. 4. Total order algorithms. 5. Causal order. 6. Bluetooth networks. 7. Real-time environments. 8. Mobile devices. 9. Distributed computing.

Área de concentração: Computação Distribuída

Titulação: Mestre em Ciência da Computação

Banca examinadora: Prof. Dr. Ricardo Oliveira Anido (IC – UNICAMP)
Prof. Dr. Ricardo A. Rabelo Oliveira (UFOP)
Prof. Dr. Luiz Eduardo Buzato (IC - UNICAMP)

Data da defesa: 08/07/2010

Programa de Pós-Graduação: Mestrado em Ciência da Computação

TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 08 de julho de 2010, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Ricardo Augusto Rabelo Oliveira
ICEB / UFOP

Prof. Dr. Luiz Eduardo Buzato
IC / UNICAMP

Prof. Dr. Ricardo de Oliveira Anido
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Um Estudo para o Problema de Ordenação Total de Mensagens Aplicado a Redes *Bluetooth* com Restrições Fracas de Tempo Real

Vicente J.P. de Amorim[1]

Junho de 2010

Banca Examinadora:

- Prof. Dr. Ricardo de O. Anido
  Instituto de Computação (UNICAMP) (Orientador)
- Prof. Dr. Ricardo Rabelo
  Instituto de Ciências Exatas e Biológicas (UFOP)
- Prof. Dr. Luiz Eduardo Buzato
  Instituto de Computação (UNICAMP)
- Prof. Dr. Edmundo R.M. Madeira
  Instituto de Computação (UNICAMP) (Suplente)

[1]Suporte financeiro de: Bolsa do CNPq (132032/2007-4) 2007–2008

Resumo

O estudo crítico apresentado discute o problema de ordenação de mensagens, típico da área de sistemas distribuídos, contextualizado em um ambiente de comunicação *Bluetooth*. Por ainda serem poucos os trabalhos com tal foco na bibliografia atual, este provê uma visão geral do comportamento de uma classe específica de protocolos distribuídos, quando executados no ambiente citado. Partindo desse contexto, o trabalho utiliza uma análise comparativa de alguns dos diversos algoritmos existentes, como forma de se obter informações sobre determinadas variáveis, e se caracterizar o melhor a ser utilizado em um ambiente de comunicação sem-fio com restrições de tempo real (*real time*).

Ao se demonstrar a viabilidade de utilização deste(s) dentro de um ambiente de comunicação *Bluetooth* (com características *real time*), automaticamente surgem novas oportunidades de aplicações, principalmente para redes móveis onde a topologia predominante é ad-hoc, ou ainda, qualquer outro tipo de aplicação em que seja necessário se garantir a entrega em ordem das informações compartilhadas dentro de um limite de tempo. Como resultado desta análise, propõe-se um protocolo para o problema de ordenação total de mensagens aplicado a redes *Bluetooth*, onde se garante que, no ambiente de comunicação, todas as informações trocadas pelos nós (*sites*) serão enviadas e recebidas na mesma ordem.

Abstract

The presented work discuss the messages ordering problem, a common subject associated to distributed systems area which was here contextualized against Bluetooth network environment. The main target of this work is focused on distributed algorithms not so commonly considered until now, specially when they are applied to this related environment.

As a way to obtain enough information about some systems variables and behavior, a comparative analysis was made between the already proposed protocols and algorithms. It generates a large set of information that makes possible to identify the better approach to be aplied at real time environments.

Once the protocol viability is demonstrated, a large set of new applications can arise, specifically to this case: mobile applications using Bluetooth networks. This is mainly due to the mobile ad-hoc network topology which allows the use of distributed applications. However, it can also bring another class of problems as message ordering, which must ensure that all network shared data will keep a local and global sending order.

Agradecimentos

Primeiramente a Deus, por me permitir concluir mais esta etapa em minha vida. Aos meus pais, Divino e Branca e ao meu irmão João, por sempre me apoiarem nos momentos mais difíceis. Por sempre virem ao meu socorro com uma palavra de estímulo a seguir em frente e não desistir. À minha namorada Juliana, pelo carinho, amor e compreensão durante todo este tempo, em que muitas vezes devido ao trabalho acumulado não pude dar a devida atenção. Ao meu orientador, Prof. Ricardo Anido, que não mediu esforços em minha orientação ao auxiliar-me na chegada a Campinas e em reuniões aos finais de semana e feriados. Aos amigos de Caratinga - em especial ao Paulinho, André, Laura e Ana Paula - que sempre prestaram seu apoio, mesmo quando na impossibilidade de estarem por perto.

Enfim, o meu muito obrigado a todos aqueles que de maneira direta ou indireta me auxiliaram ou indicaram o caminho em algum momento. Desculpas aos que porventura não citei o nome, este trabalho é fruto da contribuição de cada um de vocês.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

O trabalho apresentado nesta dissertação tem seu foco voltado principalmente a redes *Bluetooth*. Além desta característica, numa análise mais teórica pode ser contextualizado também como sendo um ponto de interseção entre duas grandes áreas da computação: Computação distribuída e Redes. Apesar de intrinsecamente relacionadas, até hoje poucos trabalhos consideraram a relação entre as sub-áreas algoritmos de ordenação e redes *Bluetooth*, respectivamente. Aqui também aplicam-se noções básicas de redes multimídia [30], pervasividade, redes ubíquas [21], redes virtuais ou *overlay* [46, 38] e suporte a falhas.

Atualmente, com o advento e a popularização das comunicações sem-fio e da mobilidade, o usuário passou a buscar não somente serviços, mas também diversão e entretenimento em tempo real. Interatividade e a necessidade de estar sempre conectado se tornaram cada vez mais comuns. Com a evolução e o desenvolvimento deste cenário, uma grande variedade de aplicações se tornou possível, principalmente aquelas baseadas na característica móvel presente em determinados dispositivos.

Todavia, para alguns destes protocolos de comunicação sem-fio, a criação de aplicações distribuídas é muito dispendiosa visto que a camada de *software* deve prover as garantias não previstas pelo protocolo físico. Para que tais algoritmos possam funcionar com sucesso, se faz então necessário um estudo das principais características tanto da parte da aplicação distribuída quanto do ambiente no qual a mesma está inserida. Notar algumas peculiaridades deste e adaptar a ele as aplicações pode fazer com que surjam novas soluções para antigos problemas, até agora somente aplicadas a redes cabeadas.

Partindo dos conceitos e tecnologias citadas, este trabalho apresenta uma catalogação e comparação dos principais protocolos de ordenação de mensagens encontrados atualmente na literatura, aplicando-os ao ambiente *Bluetooth*. Além destes, apresenta também um novo protocolo desenvolvido especialmente para ser aplicado a este ambiente quando utilizando restrições de tempo real.

Devido ao grande número de diferentes conceitos e tecnlogias aqui utilizadas, alguns

dos termos e nomenclaturas foram mantidos em seu idioma original como forma de melhor contextualizar e descrever as mesmas.

1.1 Redes multimídia, ambientes pervasivos e redes *overlay*

Muito frequentemente, uma rede sem-fio utilizada por dispositivos móveis emprega uma ou mais das seguintes tecnologias:

- Redes multimídia;
- Ambientes pervasivos; e
- Redes *overlay*.

Segundo a definição apresentada em [30], uma rede multimídia é caracterizada principalmente pela forma como lida com diferentes tipos de mídia quando estas são transportadas através de sua estrutura. O trabalho aqui descrito tem relação direta com este tipo de ambiente e contexto uma vez que, em alto nível, todos os dados transportados pela rede são de alguma forma partes constituintes de conteúdos multimídia de diversos formatos e tipos. Além desta característica, cinco pontos determinam o nível de adequação de um ambiente a conteúdos multimídia:

- Largura de banda;
- Compartilhamento de recursos;
- Garantias de desempenho;
- Escalabilidade;
- Comunicação *multicasting*.

De uma forma geral, analisando estes cinco pontos é possível se notar uma relação direta deste tipo de rede com ambientes tempo real, consequentemente, as mesmas associações podem ser feitas com o trabalho aqui descrito. Entretanto, uma relação direta é percebida no tipo de comunicação exigida pela mesma (*multicasting*)[6]. Este, talvez o principal ponto em comum com o trabalho aqui descrito, acaba por tornar as redes multimídia mais um objeto onde se pode aplicar os mesmos conceitos de computação distribuída descritos no capítulo 3.

Um segundo tópico a ser aqui associado é a noção de pervasividade ou ambientes pervasivos. O conceito de um ambiente pervasivo vem da integração de diversos dispositivos

ou agentes de *software* para que assim possam prover determinado serviço [43]. A característica mais importante de tal ambiente é que todo o compartilhamento de recursos e as trocas de informações, necessárias ao provimento do serviço, sejam feitas sem a necessidade de uma interferência externa (o usuário, por exemplo) ou de um modo menos intrusivo possível [13].

A relação com o trabalho aqui descrito pode ser observada na necessidade de utilização dos dispositivos móveis e redes de comunicação sem-fio, qualquer que seja a arquitetura básica de um sistema pervasivo. Além disso, por empregar agentes inteligentes, na grande maioria dos casos é necessária a utilização de algoritmos distribuídos. Este utiliza noções deste tipo de ambiente para prover determinado serviço aos usuários de uma rede, sem que o mesmo tenha conhecimento dos agentes de software ou das comunicações necessárias para manter o ambiente funcionando.

Outro conceito que também influenciou na implementação deste trabalho foi o tópico redes *overlay*. A definição mais comum para este tipo de rede, descreve-as como sendo estruturas criadas em cima de redes físicas para o provimento de determinados serviços específicos [38]. Ou seja, uma rede virtual que abstrai peculiaridades das redes físicas e implementa diversas outras funcionalidades dependentes do serviço a ser executado. Atualmente, a aplicação mais conhecida deste conceito são as chamadas redes *Peer-to-Peer* (P2P) de compartilhamento de arquivos [4, 2].

Figura 1.1 apresenta uma esquematização da arquitetura mais comumente encontrada de uma rede *overlay*. Nela é possível notar a abstração da camada física por uma segunda rede virtual que pode ser considerada como uma nova rede com características próprias. O trabalho aqui apresentado está diretamente relacionado com o conceito e arquitetura das redes *overlay* ao construir uma estrutura singular onde este tipo de rede é aplicado a um ambiente móvel - união até agora pouco explorada. Entretanto, o ponto de maior relevância se encontra na capacidade de prover comunicações *broadcast/multicast*, consequentemente viabilizando a implementação de sistemas distribuídos que necessitem manter um certo nível de conhecimento em cada nó componente da rede.

1.2 Motivação

Como descrito anteriormente no início deste capítulo, o trabalho aqui apresentado se aproveita do grande crescimento e popularização dos dispositivos móveis, sejam eles, telefones celulares, *handhelds*, *smartphones* ou até mesmo determinados dispositivos com capacidade de processamento um pouco maior (*notebooks, laptops, etc*). Em conjunto com a disseminação destes tipos de dispositivos, um tipo específico de protocolo de comunicação também teve seu uso alavancado. Hoje, ao se tomar os telefones celulares como exemplo, em sua grande maioria é disponibilizada a transferência de dados através de redes *Blue-*

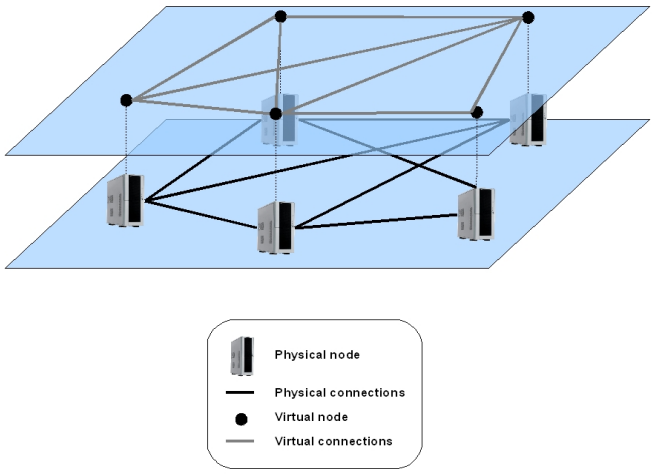


Figura 1.1: Redes *overlay*

tooth. E, aproveitando-se desta abrangência um sem número de aplicações são lançadas e comercializadas todos os anos em diversas áreas, como: *marketing*, notícias, compartilhamento de dados e logicamente entretenimento. Esta última área, foco deste trabalho, é ainda pouco explorada onde os produtos existentes se limitam ao compartilhamento de arquivos e jogos.

Jogos eletrônicos *multiplayer* já não são mais novidade, qualquer que seja a plataforma ou o tipo de conexão utilizada. Um paradoxo a este cenário ocorre quando se contextualiza os mesmos em dispositivos móveis e ambientes de comunicação *Bluetooth*. Um dos principais motivos, senão o principal, para a ocorrência deste cenário é a dificuldade em se criar neste ambiente, aplicações de tempo real com pequenos intervalos de comunicação. Além deste, o ambiente também sofre influência da topologias. As comunicações utilizando redes *Bluetooth*, como será visto no capítulo 2, impõem severas restrições ao modo de formação da rede e a arquitetura da mesma o que torna muito custosa a implementação de sistemas neste contexto.

O trabalho aqui apresentado reúne todos estes conceitos e os aplica na análise e descrição de um ambiente distribuído para a execução, principalmente de jogos, utilizando dispositivos móveis e redes *Bluetooth*.

1.3 Trabalhos relacionados

O assunto redes sem-fio no contexto móvel não é um tema novo. Entretanto, associar redes *Bluetooth*, algoritmos distribuídos e dispositivos móveis ainda não é uma área muito explorada. Em parte, isso se deve a dificuldade de processamento deste tipo de algoritmo, uma vez que normalmente esta classe de dispositivos não dispõem de uma grande capacidade de processamento. Entretanto, alguns trabalhos relacionam diretamente o ambiente de comunicação *Bluetooth* a algoritmos distribuídos sem que haja uma especificação mais formal do tipo de dispositivo onde estes seriam executados.

Os trabalhos apresentados em [51, 14] descrevem algoritmos distribuídos para a formação de *scatternets* em redes *Bluetooth*. O primeiro desenvolve duas aproximações baseadas na estrutura de dados árvore de modo a minimizar o número de mestres presentes na rede, consequentemente também o tempo de comunicação entre os nós. O segundo propõe juntamente com o algoritmo de formação de *scatternets* uma solução para o problema de roteamento de mensagens em topologias deste tipo. O trabalho descreve um esquema de entrega de dados mesmo quando o nó destinatário não se encontra na faixa de cobertura do sinal do nó remetente. Para tanto, utiliza algumas características próprias do protocolo *Bluetooth* descritas no capítulo 2. Um outro trabalho que trata a formação de *scatternets* é o descrito em [45]. Este se diferencia dos anteriores por propor um esquema que leva em conta a capacidade física de processamento de cada um dos dispositivos componentes da rede.

Em [44] é apresentado um algoritmo para formação de *scatternet* baseado nas características da rede e na capacidade física de seus componentes. Para tanto, o mesmo calcula um índice, chamado de *Device Grade* que leva em conta a capacidade de processamento do nó e o nível de carga de sua bateria, o que posteriormente é utilizado na eleição do nó mestre (aquele que possuir maior *device grade* é eleito).

Como citado anteriormente, o trabalho aqui apresentado também tem como embasamento uma parte da área de algoritmos distribuídos. Mais especificamente, o problema de ordenação total de mensagens é contextualizado em um ambiente móvel que utiliza o protocolo de comunicação *Bluetooth*. [20] faz um apanhado sobre o problema de ordenação total, descrevendo a teoria que envolve este tema e ainda faz uma descrição sucinta das diversas propostas existentes para o problema. Neste, é citado o trabalho descrito em [12], que apresenta o protocolo implementado por este trabalho e que serviu de referência para a criação de uma proposta para o problema. Ainda, levando-se em conta as características de ambiente de tempo real que delineiam este trabalho, [5, 7] propõem duas diferentes abordagens para o mesmo. No primeiro, é descrito um algoritmo (*RTCAST*) de ordenação total que leva em conta as limitações que envolvem ambientes deste tipo e provê certo nível de suporte a falhas. O segundo, algoritmo *Totem*, é baseado no conceito

token-ring e utiliza um método probabilístico para suportar determinadas restrições. A principal diferença entre as duas propostas esta no fator nível de restrição do ambiente de tempo real.

Apesar de existirem diversas propostas, nenhuma delas contextualiza as soluções apresentadas em ambientes como o descrito anteriormente. Após ter analisado um conjunto de propostas que preenchem os requisitos tanto de uma área quanto de outra, o trabalho aqui descrito tem como foco principal, preencher a lacuna deixada por estes. Isto é, apresenta-se uma solução para o problema de ordenação total tendo como foco principal aplicações multimídia com restrições de tempo real sendo executadas em dispositivos móveis com poucos recursos físicos que utilizam redes *Bluetooth* como enlace de comunicação e integradas num modelo de ambiente de tempo real.

1.4 Organização do documento

Devido a necessidade de conhecimentos pertencentes a diferentes áreas da computação, o trabalho foi organizado da seguinte maneira de forma a se obter um melhor entendimento geral.

Capítulo 1 apresenta uma introdução a diferentes áreas de uso e trabalhos relacionados. Neste são descritas ainda algumas aplicações que diretamente ou indiretamente justificam a pesquisa apresentada assim como aplicações reais de tal trabalho.

Capítulo 2 descreve a principal tecnológia utilizada neste trabalho. O protocolo *Bluetooth* é apresentado, assim como suas subdivisões internas - diferentes camadas -, modos de acesso, aplicações, limitações e modelos de utilização.

Capítulo 3 contextualiza os principais problemas apresentados por este trabalho junto aos conceitos relacionados na área de sistemas distribuídos. Uma breve descrição dos pontos relevantes é apresentada, assim como a maneira que os mesmos podem ser relacionados entre si e o protocolo de comunicação *Bluetooth*. Aqui também são apresentados os conceitos necessários de ordenação de mensagens e ambientes de tempo real necessários ao entendimento do restante do trabalho.

Diante da lacuna de protocolos para ambientes de tempo real encontrados na bibliografia, capítulo 4 apresenta um algoritmo que garante ordem total as mensagens que trafegam dentre entre grupo de processos. Este se baseia e toma vantagem dos conceitos apresentados nos dois capítulos anteriores de forma a maximizar o ganho em tempo, confiabilidade de desempenho dentro de um ambiente com recursos limitados de processamento.

Por fim, no capítulo 5 os resultados de uma comparação entre três diferentes protocolos de ordenação são apresentados. O comportamento dos mesmos é descrito se tomando como base diversos cenários distintos - cada um com o enfoque em um ponto específico do ambiente proposto.

Capítulo 2

Redes *Bluetooth*

O nome *Bluetooth* advém de *Harald Bluetooth*, rei dinamarquês do século X que ficou conhecido por utilizar a diplomacia na restauração do cristianismo e solução de diversos tipos de problemas em seu reino. Do mesmo modo, a tecnologia *Bluetooth* utiliza a troca de informações e acordos para realizar o envio e recebimento de informações dentre os nós conhecidos [41].

Inicialmente desenvolvido pela Ericsson em meados de 1994, a tecnologia *Bluetooth* foi pensada para prover, principalmente, um ambiente de comunicação de baixo custo e baixo consumo de energia que utilizasse ondas de rádio. De 1994 em diante, devido a tais características, o padrão se concretizou e passou a ser utilizado tanto em dispositivos fixos (PCs, Televisões,...) quanto em dispositivos móveis (telefones celulares, *PDAs*, *Handhelds*, *Smartphones*, *Notebooks*, etc) nos mais diversos tipos de aplicações.

Após o passo inicial dado pela Ericsson em 1994, o *Bluetooth* se popularizou cada vez mais e novas empresas se juntaram a esta de modo a padronizar o uso da tecnologia. Desta união, formou-se o *Bluetooth SIG (Special Interest Group)* [47], uma organização privada, sem fins lucrativos que visa a padronizar todas as versões do protocolo. Dentre as mais de 9000 empresas que compõem este grupo estão: *Microsoft, Intel, Ericsson, Nokia, Motorola, Toshiba*. Uma descrição mais detalhada da história e formação do grupo pode ser encontrada em [40, 47].

2.1 Conceitos Básicos

Algumas das razões do sucesso da tecnologia *Bluetooth* ao longo dos últimos anos são: baixo custo, baixo consumo de energia e versatilidade (suporta transmissões de voz e dados). Para se entender o funcionamento do protocolo é necessário que sejam fixados alguns conceitos básicos a respeito da tecnologia.

2.1.1 Espectro de frequência

Bluetooth opera na banda ISM (*Industrial, Scientific and Medical*) utilizando faixas de frequências de 2.40GHz a 2.48GHz. A mesma utilizada em telefones sem-fio comuns, fornos microondas e redes *wireless* (IEEE 802.11), dentre outros. Apesar de ser uma faixa de frequência livre, algumas regras definem como os dispositivos devem operar dentro da mesma.

- O espectro é divido em 79 canais de 1MHz cada;
- Comunicações utilizam *Frequency Hopping Spread Spectrum* (2.1.2);
- Alcance: O protocolo define três classes para potência, consequentemente, área de cobertura:
  - Classe 1: Alcance máximo de 100 metros;
  - Classe 2: Alcance máximo de 10 metros;
  - Classe 3: Alcance máximo de 1 metro.
- Consumo de energia;
- *Bitrate*: De acordo com a versão da especificação, pode suportar de 1Mpbs a 3Mbps.

2.1.2 *Frequency Hopping*

Para a transmissão dos dados, cada protocolo ou rede, utiliza a banda a seu modo. No protocolo *Bluetooth*, as mensagens a serem enviadas são divididas em diversos pacotes e transmitidas. O primeiro pacote é enviado em uma determinada frequência, então, o protocolo seleciona um novo canal para o envio do próximo pacote. Desse modo, todos os dados são enviados utilizando um espalhamento uniforme pelo espectro [41]. A idéia por trás da utilização do FHSS (*Frequency Hopping Spread Spectrum*) é a diminuição da interferência com outros tipos de comunicação, uma vez que as trocas de canais ocorrem com extrema rapidez e a probabilidade de uma colisão com outro pacote é reduzida (Figura 2.1).

Outro aspecto relevante desta aproximação diz respeito à segurança. Como as mudanças de canal (e os padrões destas mudanças) podem ser construídas de diferentes formas, uma terceira parte, interessada em "ouvir" as comunicações que ocorrem pela rede, deve conhecer o padrão de mudança de canal envolvido na troca de mensagens dos outros dois integrantes. Somente através de observações esta é uma variável difícil de determinar, pois, em média, 1600 mudanças de canal são executadas a cada segundo [40]. Entretanto, conforme o número de dispositivos vai aumentando na rede, a probabilidade

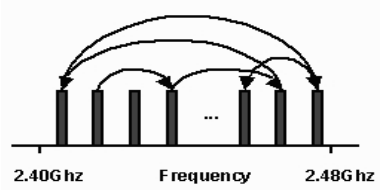


Figura 2.1: Esquema operacional do *FHSS*.

de ocorrência de colisões também aumenta chegando a um ponto em que todas as conexões (e a vazão geral da rede) são degradadas, uma vez que cada vez mais canais estarão sendo ocupados por alguma transmissão num dado momento.

2.2 Topologia básica

Apesar de possuir uma arquitetura e um modelo *ad-hoc*, estes funcionam até certo ponto, de maneira estruturada. Isto porque, o protocolo trabalha em cima do conceito de mestre/escravo, onde, cada mestre fica responsável por um conjunto de no máximo 7 escravos. Dessa forma, em uma comunicação escravo/escravo todo o tráfego de uma rede passa por um ponto central (mestre) antes de chegar ao seu nó destinatário (escravo). Tal característica pode causar um efeito gargalo na rede dependendo do volume de dados a serem compartilhados e o número de nós escravos que devem receber determinadas informações.

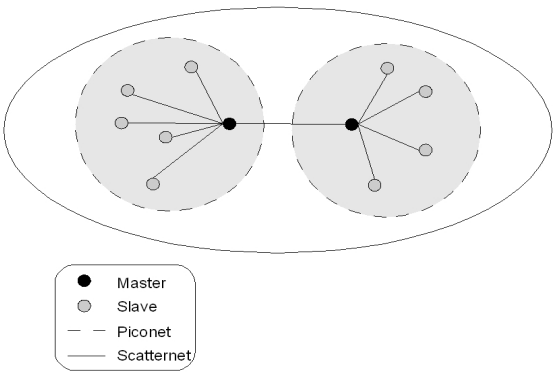


Figura 2.2: Topologia de uma rede *Bluetooth*

A Figura 2.2 exemplifica o modelo descrito anteriormente (mestre/escravo) além de introduzir dois novos conceitos:

- *Piconets*: Rede formada por apenas um mestre, onde este é responsável por todo o tráfego da rede além de controlar as conexões feitas pelos nós escravos;

- *Scatternets*: Cada *piconet* pode possuir um máximo de 8 componentes (1 mestre e 7 escravos). Sendo assim, quando atingido esse limite, é necessário que se forme uma nova rede (*scatternet*), conectada à primeira, de modo que seja possível suportar o novo número de nós.

A formação dinâmica de redes *scatternets* é objeto de estudo de diversas áreas, desde redes de computadores até otimização de algoritmos. [51, 14, 45] propõem diferentes algoritmos, cada um deles com um enfoque, como forma de otimizar a construção deste tipo de topologia. [44] descreve um algoritmo com a mesma finalidade, porém, este leva em conta algumas características do *link* de comunicação e dos dispositivos componentes como forma de otimizar a formação da rede.

Com um enfoque totalmente diferente, [26] relaciona principalmente as implicações refletidas no *hardware* quando da formação de redes deste tipo. O mesmo, descreve ainda características do *hardware* básico utilizado na maioria dos chips *Bluetooth* atuais. Ao final, o mesmo disserta sobre os componentes físicos necessários para se construir um chip que dê suporte a topologia *scatternet*.

2.3 Pilha de protocolos

Como descrito pela Figura 2.3, basicamente, em alto nível, o protocolo *Bluetooth* pode ser dividido em três camadas:

- *Application*: Composta basicamente pelas aplicações que virão a utilizar o ambiente *Bluetooth*.

- *Middleware Protocol*: Responsável por prover interfaces junto a outros tipos de protocolo. Nesta camada, além destes protocolos (PPP, TCP...) incluem-se protocolos especificamente utilizados no ambiente *Bluetooth*, como os descritos em [47].

- *Transport Protocol*: Responsável por toda operação do ambiente *Bluetooth*, gerencia desde conexões com outros nós ao envio e recebimento de mensagens. Subdividido em: *Radio, Baseband, Link Manager, Logical Link and Adaptation, and Host Controller Interface*.

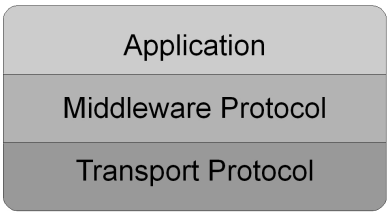


Figura 2.3: Camadas do protocolo *Bluetooth*

2.3.1 Transport Protocol

Este protocolo engloba diversas definições de baixo nível que não cabem ser citadas aqui. Entretanto, são descritas detalhadamente por [40, 47, 29, 24].

Desenvolvido com o objetivo principal de tornar possível à troca de mensagens pelo ambiente *Bluetooth*, o protocolo tem como uma característica marcante o suporte a ambos os modos de comunicação: síncrono ou assíncrono, para, respectivamente voz e dados. Dentro do mesmo, a existência de diversos módulos possibilita além da troca comum de mensagens, um suporte especial quando gerenciando tráfego de "voz".

Figura 2.4 descreve os módulos componentes do protocolo de transporte.

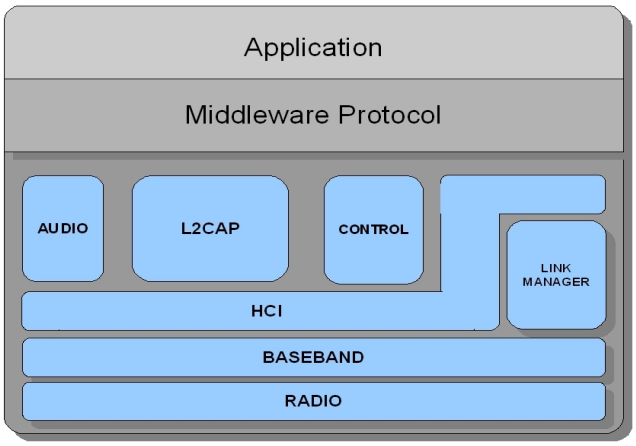


Figura 2.4: Componentes da camada de transporte

Host Controller Interface (HCI)

Devido à grande versatilidade e aos inúmeros tipos de dispositivos que utilizam o protocolo *Bluetooth*, é necessário se padronizar o modo de acesso ao *hardware* de baixo nível. O módulo HCI, definido pelo *SIG* é responsável por receber os diversos tipos de informações (áudio, dados e controle) e repassá-los ao *link manager* e ao *link controller*. Entretanto, dependendo do tipo de dispositivo e de como é composto o *hardware*, o acesso a este módulo pode não ser disponibilizado pelo fabricante, uma vez que a compatibilidade na transmissão das informações pode ser garantida de outras formas, dentro do próprio dispositivo.

L2CAP

Logical Link Control and Adaptation Protocol (L2CAP) atua como uma camada intermediária entre o *middleware* - aplicações que utilizam somente dados - e as camadas mais baixas do protocolo *Bluetooth*. Para que isso possa ocorrer, esta camada utiliza uma espécie de encapsulamento das informações (dados) que são provenientes das aplicações, repassando as mesmas a camada mais baixa. Ao contrário do que ocorre com a camada *HCI*, a implementação do *L2CAP* é obrigatória, uma vez que esta também é responsável por "reformatar" os pacotes de dados que chegam das aplicações, em pacotes que possam ser suportados pelas camadas mais baixas (*baseband*, *radio*).

Nesta espécie de "reempacotamento" dos dados, deve ser levado em conta a variável chamada *MAXIMUM_TRANSMISSION_UNIT* que, como o próprio nome diz, delimita o tamanho máximo dos pacotes que podem ser enviados por cada dispositivo. Além de possibilitar este controle sobre o tamanho dos pacotes que trafegam pelo *link*, a camada *L2CAP* pode prover ainda informações sobre *QOS* de forma a garantir um mínimo de qualidade de serviço na rede. Entretanto, garantia e ordem na entrega de informações não são tratadas aqui e sim nas camadas mais abaixo.

Para se melhor entender o funcionamento desta camada, alguns pontos devem ser descritos mais detalhadamente uma vez que os mesmos tem influência direta no funcionamento geral do protocolo.

- Comunicação: Como descrito anteriormente, as trocas de mensagens são feitas utilizando-se canais lógicos entre os dispositivos. Para que ocorra uma associação dos canais de modo a funcionarem no mesmo *frequency hopping*, um identificador é utilizado para cada canal (*CID*).

- Conexão: Basicamente, utilizando-se os descritores de canais citados anteriormente, a camada *L2CAP* pode enviar/receber dados de duas formas diferentes: "Orientado

a conexão" (*CO*) e "não orientado a conexão" (*CL*). Logicamente, o tratamento para cada um dos tipos é diferenciado.

  - *Connection Oriented - CO*: Trocas de informações ocorrem por meio de uma conexão pré-estabelecida. Leva em conta o padrão de mudança dos canais para que se possa receber e enviar os dados.
  - *Connection Less - CL*: Não é necessário haver uma conexão pré-estabelecida. Normalmente, este conceito é aplicado quando do envio de uma espécie de *broadcast* de dados de controle aos outros dispositivos.

  Inicialmente não existe uma pré-conexão estabelecida. Desse modo, para enviar dados de controle, sinalização e pedidos de conexão, um dispositivo deve utilizar um canal (*CID*) pré-definido. A tabela 2.1 mostra o papel de cada faixa de canal dentro do protocolo [40].

- Sinalização: No caso da utilização de uma troca de dados orientados a conexão, os dispositivos devem seguir um determinado protocolo de estabelecimento de conexão. Este, além de possuir tal funcionalidade, permite ainda acordos sobre diversos pontos, como: Qualidade de serviço (*QoS*, tamanho do *MTU*, etc).

  Como pode ser observado na tabela 2.1, o canal de *CID* 0x0001 é utilizado para o envio e recebimento de requisições de conexão. Nele, a garantia de entrega das mensagens é feita através de retransmissões após um determinado tempo (*timeout*), normalmente menos de $1(s)$ por cada rodada de pedido/resposta. A última fase do mesmo engloba um *timeout* maior (cerca de 300s) pois pode envolver uma autenticação do usuário. De maneira geral, a formação de uma conexão entre dois dispositivos ocorre através das seguintes rodadas:

  - 1ª: Nó local - quem inicia o pedido de pedido de conexão - envia uma mensagem requisitando a criação de um canal (utilizando inicialmente o canal de *CID* 0x0001) ao dispositivo remoto - quem recebe o pedido de conexão. Este, por sua vez, responde enviando uma mensagem aceitando ou não a requisição.
  - 2ª: Caso a conexão tenha sido aceita no passo anterior, o dispositivo local envia a configuração de diversas variáveis que determinam as características da conexão. Também para esta fase o dispositivo remoto deve aceitar ou rejeitar os valores de cada variável. Entretanto, caso haja uma rejeição aos parâmetros, os dois componentes entram em uma fase de negociação dos mesmos, até que cheguem a um acordo.
  - 3ª: Com o sucesso da fase anterior, os dispositivos estão aptos a trocarem os dados que são encaminhados pelas camadas superiores do protocolo.

CID	Funcionalidade
0x0000	Nulo
0x0001	Troca de informações de controle e sinalização
0x0002	Troca de informações no modo *connection less*
0x0003 a 0x003F	Reservado
0x0040 a 0xFFFF	Alocados sob demanda conforme solicitação do dispositivo. Pode ser utilizado por canais *CL* e *CO*

Tabela 2.1: Identificadores de canal (*CID*) e suas funcionalidades.

– 4ª: Após a troca das informações, a conexão entre os dois dispositivos pode ser encerrada explicitamente através de um pedido/resposta.

Em [40, 41, 47] podem ser encontradas informações mais detalhadas sobre todas as fases acima, assim como uma descrição do formato de cada pacote em cada uma das rodadas de requisição/resposta.

Audio

A transmissão de áudio pelo protocolo *Bluetooth* é feita de modo muito particular. O ambiente adota uma característica própria na transmissão e recepção deste tipo de informação utilizando 64Kbps na transmissão e recepção de voz, banda essa, suficiente para suportar uma conversação humana. Basicamente, a especificação do protocolo *Bluetooth* permite dois tipos de codificação de áudio no sistema:

- *PCM (Pulse Coded Modulation)*: Utilizado principalmente em tráfego de rajadas (dados de voz de curta duração).
- *CVSD (Continuous Variable Slope Data)*: Levando-se em consideração as características da voz humana, é possível se utilizar um certo nível de predição. É neste ponto que trabalha *CVSD* suportando uma porcentagem maior de pacotes com erro.

Link Manager

Utilizando o *Link Manager Protocol - protocolo de troca de mensagens*, esta camada é responsável pelo controle, manutenção e gerenciamento do *link* de troca de dados entre dois dispositivos comunicantes. Além desta, são também funções associadas ao *Link Manager*:

- Segurança: Autenticação e criptografia das informações trocadas são as duas medidas de segurança adotadas pela especificação *Bluetooth*. A primeira, baseada na

idéia de compartilhamento de chaves, prevê que os dois dispositivos façam uso de uma mesma chave de autenticação e utiliza o esquema de desafio-resposta para efetuar o processo de autenticação. A segunda, apesar de não ser mandatória na implementação, utiliza um algoritmo de encriptação onde a chave é uma espécie de derivação da chave compartilhada utilizada na autenticação. Por esse motivo, a autenticação passa a ser obrigatória no caso da utilização de um *link* encriptado pelos dispositivos.

- Gerenciamento de energia: A especificação do protocolo *Bluetooth* prevê a capacidade de se adequar o dispositivo a diferentes estados de consumo de energia dependendo do modo de operação do mesmo. Basicamente, três estados de baixo-consumo de energia são definidos:

  - *Sniff*: No modo normal de troca de informações, o nó escravo escuta por uma comunicação do mestre a cada *slot* de número par. Entretanto, no modo *sniff* este ciclo passa a ter um intervalo maior de modo a se poupar energia. Normalmente, o intervalo de tempo de *sniff* é negociado entre o dispositivo mestre e o escravo.

  - *Hold*: Em modo *hold* um dispositivo escravo tem sua comunicação com o mestre interrompida por um determinado período de tempo (negociado previamente com o mestre).

  - *Park*: Diferentemente dos dois modos anteriores, quando no estado *park* um dispositivo não é mais considerado um membro da rede. Entretanto, a especificação do protocolo prevê um método que possibilita uma rápida readmissão do dispositivo em modo *park*. Para que isto seja possível, o dispositivo neste modo continua a receber informações de *broadcast* vindas do mestre através de um canal de comunicação especial (*beacon channel*) por onde o mesmo pode ser "acordado" e chamado de volta à integrar uma determinada *piconet* ativa.

- *Qos*: De modo a garantir um mínino de qualidade de serviço numa *piconet*, o protocolo *Bluetooth* permite que determinados valores de algumas variáveis - como largura de banda e atraso - possam ser ajustados entre os dispositivos mestre e escravo. Para tanto, ao dispositivo mestre é permitido que ajuste os valores destas variáveis diretamente nos nós escravos. Entretanto, para que um escravo consiga executar esta mesma ação é necessário que este, previamente, negocie tais valores com o mestre de sua *piconet*.

Baseband

Este módulo é na verdade um encapsulamento ao módulo *link manager* e trabalha principalmente como uma interface de baixo nível de modo a prover determinadas funcionalidades a esta. Para tanto, o mesmo gerencia e manipula uma grande variedade de diferentes tipos de informação, como: dados síncronos / assíncronos, dados de controle do *link*, etc. Anteriormente a criação de uma rede *piconet* propriamente dita (e a posterior troca de informações entre seus integrantes), determinados passos devem ser seguidos para que esta seja formada. O módulo *baseband* é o responsável pelo acesso de baixo nível e pelo gerenciamento das *piconets*, definição dos papéis (mestre / escravo) de cada dispositivo e método de compartilhamento do meio por informações síncronas ou assíncronas. Além das funcionalidades descritas pelo *link manager*, são funções desde módulo:

- Formação de *piconets*: Como descrito anteriormente na subseção 2.2, uma *piconet* é a topologia de rede mais simples possível de ser construída em um ambiente *Bluetooth*. Os nós que virão a compor uma *piconet* sempre estarão em um dos seguintes estados:

  - *Standby*: Estado de baixo consumo de energia. Um nó permanece nele até que faça uma busca por vizinhos ou receba uma solicitação de conexão de outro dispositivo;
  - *Inquiry*: Dispositivo executa uma busca e localiza nós próximos através das respostas recebidas ao seu *inquiry*;
  - *Page*: Nó que deseja iniciar a criação de uma *piconet* utiliza este estado para enviar convites aos nós que foram descobertos pelo estado anterior.
  - *Connected*: Dispositivos que aceitaram o convite, agora são explicitamente conectados.

  Apesar de já neste último estado (*Connected*) estarem conectados, é necessário que todos os componentes da rede tenham conhecimento da ordem e frequência dos *hops* a serem utilizados como forma de viabilizar o envio e recepção dos dados aos vizinhos. A determinação deste parâmetro envolve a utilização de dois elementos distintos:

  - Endereço (*BD_ADDR*): Definido pela IEEE, é uma sequência única de 48 bits que identifica de forma única um dispositivo.
  - Relógio: Cada dispositivo possui internamente implementado um relógio físico com frequência aproximada de 3.2kHz (duas vezes a frequência nominal de *hops* - 1.600/s - do padrão).

O endereço único do dispositivo e o relógio interno são então utilizados pelo *FSM* (*Frequency Selection Module*) para a geração da frequência de transmissão/recepção. A Figura 2.5 apresenta um diagrama de blocos exemplificando os dados de entrada e saída utilizados pelo módulo *FSM*.

- Controle de acesso ao meio: As informações trocadas neste tipo de rede requerem, além de conexão, certo nível de sincronização dos *hops* de ambos os dispositivos comunicantes. A decisão de qual será a sequência de *hops* pelos canais e feita pelo mestre da rede, assim como a definição de qual escravo está apto a enviar/receber dados em cada *hop* (canal utilizado no momento).

- Controle de falhas: Durante o tempo de estadia em cada frequência (entre cada *hop*), o módulo *baseband* pode enviar o pacote de dados chamado BB_PDU (*Baseband packet data unit*). O formato deste pacote é descrito pela Figura 2.6 e está intrinsicamente relacionado com o esquema de correção de falhas implementado pelo protocolo.

  - *Access code*: Utilizado para identificação da *piconet* a qual pertence o nó. Através deste identificador, mesmo que dados de outras *piconets* sejam recebidos, estes não serão considerados. Pode conter 68 ou 72 *bits* de acordo com a finalidade do pacote. Isto é, se o pacote serve exclusivamente a um *inquiry* ou *page* (não possui *header* e *payload*), utiliza-se 68 *bits*, caso contrário, 72 *bits*.
  - *Header*: Contém 18 bits para correção de erro, entretanto, o tamanho do campo chega a 54 bits pois cada *bit* é representado três vezes como forma de se gerar redundância e suportar possíveis falhas na transmissão do pacote. A especificação *Bluetooth* utiliza código de *Hamming* no esquema de correção de erros.
  - *Payload*: Carrega os dados de aplicação que foram entregues pelas camadas superiores.

Radio

O módulo *Radio* controla, basicamente, as operações de multiplexação e demultiplexação dos dados enviados através das camadas superiores. Como citado anteriormente, o protocolo *Bluetooth* trabalha na frequência de 2.4GHz da banda *ISM* (a mesma utilizada por fornos microondas, telefones sem fio, etc.) utilizando o esquema descrito anteriormente, chamado *FHSS - Frequency Hopping Spread Spectrum*. Este opera num conjunto de 79 canais (ou 23, devido a restrições em determinados países). Além disso, determinados detalhes devem ser observados no funcionamento do *Radio Bluetooth*:

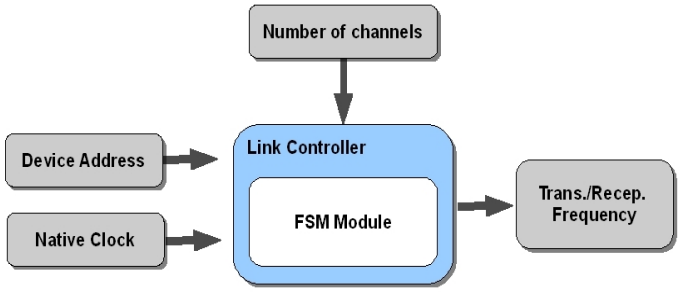


Figura 2.5: Arquitetura de suporte ao módulo *FSM*

Figura 2.6: Formato do pacote BB_PDU

- Pico máximo de frequência não pode exceder 1 watt (30 dBm); e
- Tempo de permanência em cada canal não pode exceder a 0.4 segundos.

O módulo *Radio* assume, ao mesmo tempo, dois diferentes papéis dentro do protocolo dependendo do sentido do tráfego de dados:

- Envio: Recepção dos dados através do *Link Controller*, seleção do canal de início e correta potência para a transmissão de determinado *stream* de dados.
- Recepção: Seleção da frequência inicial correta. Após recepção dos dados, encaminhamento dos mesmos às camadas superiores do protocolo.

De acordo com a especificação em [47], não existe uma arquitetura ou um conjunto de *interfaces* obrigatórias para os componentes do protocolo. Desta forma, cada fabricante pode dispor os módulos de acordo com sua arquitetura própria, priorizando o baixo custo e o baixo consumo de energia.

2.3.2 *Middleware Protocol*

Ao se observar a localização desta camada dentro da arquitetura do protocolo *Bluetooth* é possível se notar o papel desenvolvido pela mesma. A camada de *middleware* é responsável

por abstrair as funcionalidades de baixo nível e prover às aplicações interfaces mais simples de forma a permitir um uso mais simplificado do protocolo.

Basicamente, esta camada é composta por diversos módulos que podem ser agrupados em três funcionalidades básicas:

- Uma abstração simples da camada de transporte (descrita na subseção 2.3.1);
- Busca/descoberta de serviços disponíveis dentro do alcance do dispositivo; e
- Acesso e controle a dados/tráfego específicos de áudio e telefonia.

Figura 2.7 descreve os módulos componentes da camada *middleware*.

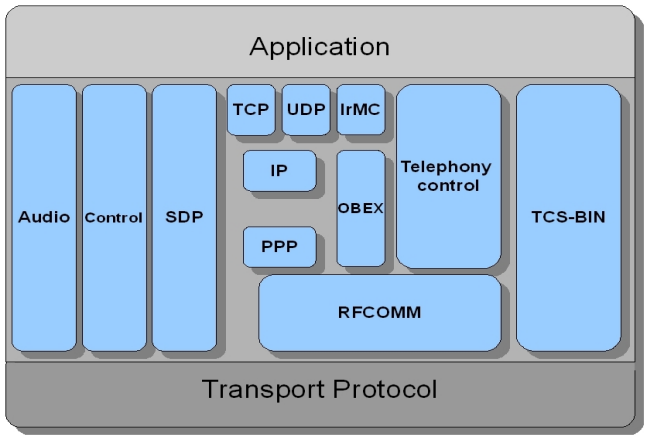


Figura 2.7: Componentes da camada *middleware*

Audio

Como citado anteriormente, o protocolo *Bluetooth* provê um tratamento especial à parte de telefonia. Nesta, está incluso a parte de tráfego de voz. Uma vez que este tipo de tráfego é estabelecido pelo módulo relacionado (*TCS-BIN* e *Telephony Control*), todo os dados de voz/áudio ficam a cargo do módulo *Audio*, que por sua vez, tem a capacidade de criar uma espécie de túnel da camada de aplicação diretamente para a camada mais baixa (*baseband*).

Esta diferenciação de tratamento no tráfego de áudio, levou o protocolo *Bluetooth* a necessidade de suportar um encapsulamento de transporte particular e um formato de pacote diferente daqueles empregados em tráfego de dados. Os canais síncronos (*SCO* -

Synchronous Connections Oriented) estabelecidos entre mestres e escravos possuem uma largura de banda maior, se comparados aos canais assíncronos, de modo a não degradar a qualidade neste ambiente com características de tempo real. Entretanto, uma *piconet* suporta um número máximo de três canais *SCO* sendo que, para a existência destes, faz-se necessário um canal assíncrono normal para o tráfego de dados de controle.

A subseção 2.3.1 descreve os esquemas de modulação utilizados assim como a banda reservada especialmente para o tráfego de voz pela especificação do protocolo *Bluetooth*.

Protocolos *IrDA*

Uma das principais características do ambiente *Bluetooth* é o suporte a comunicação entre diversos tipos de dispositivos, de classes diferentes e com capacidades de processamento distintas. Essa característica de interoperabilidade se mantém no baixo nível através da utilização de módulos de suporte ao protocolo de comunicação via infravermelho.

Ao implementar estes módulos, a idéia principal do protocolo *Bluetooth* não é prover meios de se criar canais de comunicação utilizando a interface infravermelho, mas sim, proporcionar suporte a aplicações de modo que estas utilizem um mesmo modo de acesso, recebimento e envio dos dados. Dessa forma, uma aplicação inicialmente escrita para o ambiente infravermelho pode ser facilmente portada para *Bluetooth* sem que para isso haja necessidade de grandes modificações. Caracteriza-se assim um suporte a interoperabilidade somente no nível de aplicação sem aprofundamento até o nível físico do protocolo.

O suporte a aplicações que fazem uso do infravermelho se dá principalmente devido a utilização de um protocolo à parte que provê uma interface comum aos dois ambientes. Este protocolo, *OBEX*, é descrito com mais detalhes na próxima subseção.

OBEX

Além dos protocolos de transporte nativos do ambiente *Bluetooth*, existe o suporte a um conjunto de protocolos utilizados por aplicações infra-vermelho (IrDA) de modo a prover interoperabilidade entre estas duas classes de aplicações. A base deste suporte se apoia sobre o conceito de compartilhamento de objetos, conhecido como *object exchange* ou somente, *OBEX*.

A maior parte do tráfego das aplicações dentro do protocolo *Bluetooth* é feita através de *streams* de dados utilizando-se os canais pré-estabelecidos entre mestres e escravos. Entretanto a especificação provê um meio de se abstrair os dados trocados em forma de objetos de pré-definidos (como padrões de mensagens, cartões, etc.), facilitando assim a implementação e operação de determinados tipos de aplicações.

Dentro da camada *middleware* é implementado o módulo *OBEX* [8], que como próprio

nome diz, fica encarregado pela recepção dos objetos (vindos do módulo *IrMC* e que por sua vez foram encaminhados pelas aplicações), desempacotamento e repasse dos dados às camadas inferiores (*RFCOMM* e opcionalmente *TCP/IP*). Uma vez que o módulo *TCP/IP* (ainda) não é utilizado pelo padrão *Bluetooth*, uma correta operação conjunta do módulo *OBEX* com a camada de transporte *RFCOMM* determinados pontos devem ser levados em consideração:

- Cliente/Servidor: Por utilizar na transferência dos objetos o esquema *push/pull*, é necessário que um mesmo nó implemente tanto a funcionalidade de cliente (*push* do objeto) como de servidor (*pull* do objeto). Uma característica da operação normal do protocolo é a utilização do conceito de sessões de comunicação para permitir o compartilhamento de informações entre dois dispositivos. Além disso, frequentemente é associado o papel de cliente ao nó que inicia a conexão e o papel de servidor ao nó que responde às requisições feitas pelo primeiro.

- Multiplexação na camada de transporte: Todos os objetos e dados compartilhados pelo protocolo *OBEX* trafegam sobre uma camada padrão do ambiente *Bluetooth*: *RFCOMM*, descrito em mais detalhes posteriormente dentro desta subseção. Dessa forma, controles sobre a existência de uma sessão *OBEX* devem ser fornecidos de modo a poder se determinar se o canal continua em uso ou se o mesmo já pode ser fechado (para o caso de término da sessão de comunicação).

- Suporte à *SDP (Service Discovery Protocol)*: A utilização do protocolo de descoberta pelo *OBEX* segue o mesmo princípio de raciocínio que para o ambiente nativo. Ou seja, é necessário uma definição dos nós vizinhos (que podem vir a operar dentro do alcance), além de um acordo sobre a ordem do *frequency hopping* para um dado canal de comunicação.

Em uma descrição de alto nível, basicamente, o módulo *OBEX* é utilizado como uma ponte de modo a prover interoperabilidade com outros tipos de aplicação através de uma interface comum. Entretanto, uma descrição mais aprofundada do protocolo *OBEX* foge ao escopo do trabalho aqui apresentado. A especificação, assim como dados detalhados da implementação e características de operação do mesmo podem ser encontrados em [8].

SDP

Uma característica ímpar do protocolo *Bluetooth* é a dinamicidade na composição de sua topologia. Esta característica não é encontrada na maioria das outras redes *LAN* ou até mesmo outros tipos de rede sem fio como IEEE 802.11 [3]. Enquanto nesse tipo de rede os nós e os serviços disponibilizados são, de certa forma, estáticos, o ambiente *Bluetooth*

está frequentemente suscetível a modificações seja no conjunto de serviços disponíveis ou arquitetura/topologia da mesma.

Além destas características, outro ponto importante é o modo de operação de uma rede *Bluetooth*. Apesar de possuir um nó principal que centraliza todo o tipo de tráfego em uma *piconet*, uma rede deste tipo pode ser considerada como *ad-hoc* uma vez que não é necessário se determinar previamente o mestre e, qualquer um dos nós componentes está apto a se tornar um mestre. Considerando tais pontos, fica claro a necessidade de um protocolo que dinamicamente e automatizadamente faça uma busca por serviços presentes no ambiente e seja capaz de reconhecer uma topologia básica da rede na qual o dispositivo está inserido. Devido a esta necessidade, o ambiente *Bluetooth* implementa protocolo próprio para a descoberta de serviços.

O *SDP - Service Discovery Protocol* está inserido dentro da camada *middleware* e é o módulo responsável por lidar com a característica discutida nesta subseção até aqui. O modo de operação do *SDP* se baseia nos conceitos de publicação de serviços e cliente/servidor, onde um mesmo nó pode assumir ambos os papéis, já que qualquer dispositivo presente na rede pode publicar serviços a serem utilizados por outros componentes.

De modo a tornar possível esse compartilhamento de serviços, o *SDP* implementa localmente em cada nó uma lista de registros (*service registry*) com os serviços disponibilizados pelo dispositivo. Cada uma das entradas desta lista, que consequentemente possui um serviço associado, é chamada de *service record* que por sua vez, armazena os dados mandatórios para a descrição do serviço (textual e binária), além de fornecer as informações necessárias aos clientes que venham a interagir com o mesmo.

Figura 2.8 descreve a estrutura básica de um *service registry* de forma a organizar os registros de serviços, descrições de interfaces e atributos dos mesmos. Como definido pela própria especificação do protocolo *Bluetooth*, cada serviço pode ser descrito à partir de dois conjuntos de dados:

- Atributos universais: Independentemente do tipo de serviço, tais atributos sempre constam no *service record*; e

- Atributos específicos: Aplicados especificamente a determinados tipos de serviços, estes servem para caracterizar a natureza e aplicação dos mesmos.

O processo de busca e descobrimento de serviços é baseado principalmente em um determinado atributo chamado *UUID*. O *UUID* (*Universally Unique Identifier*) é um identificador de 128 *bits* proposto pela *ISO* e gerado através de um determinado algoritmo com o objetivo de identificar unicamente cada serviço disponibilizado pelos dispositivos sem que haja a necessidade do registro deste valor em qualquer organização de padronização internacional.

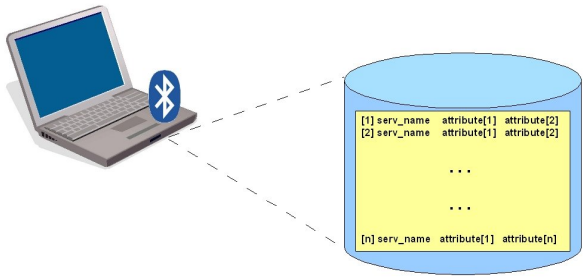


Figura 2.8: Arquitetura dos descritores de serviços

Entretanto, para que a descoberta de tal identificador e consequentemente a caracterização do serviço seja possível, é necessário que o protocolo *SDP* execute uma série de trocas de mensagens (conhecidas como *SDP_PDU (SDP Protocol Data Unit)* e características do protocolo *SDP*) com o nó servidor:

- O dispositivo cliente define o serviço pelo qual está interessado e o seu módulo *SDP* envia uma mensagem *SDP_PDU*, ao nó que está atuando como servidor;
- Este, por sua vez, compila outra mensagem *SDP_PDU* contendo as respostas, positivas ou não, em relação ao serviço que foi requisitado pelo cliente;
- De posse de uma lista de serviços que preenchem seu critério de busca (e consequentemente do *UUID* de cada um deles), o nó cliente requisita especificamente mais detalhes sobre os atributos de um determinado serviço.
- O servidor envia de volta ao cliente uma mensagem *SDP_PDU* contendo o detalhamento dos atributos daquele serviço requisitado no passo anterior.

O processo de descoberta de serviços pode ser executado de duas formas diferentes. O descrito anteriormente se baseia em transações únicas de requisição/resposta a cada tipo de dado necessitado. Entretanto, outro modelo de requisições compostas pode ser utilizado. O esquema descrito pelos passos anteriores está relativamente resumido, todavia, em [40, 47] é possível encontrar uma definição mais completa de ambos os processos de busca e descobrimento de serviços pelo protocolo *SDP*.

RFCOMM

Posicionado bem acima do módulo *L2CAP* (Figura 2.9), o módulo *RFCOMM* é o responsável pela abstração e emulação de uma espécie de interface serial para o envio e recepção dos dados vindos da aplicação ou da própria camada *middleware*, se tornando, de

certa forma um protocolo comum às camadas posicionadas acima do módulo de transporte 2.3.1, incluindo aí, neste conjunto o módulo *OBEX*, responsável pela interoperabilidade com aplicações *IrDA*.

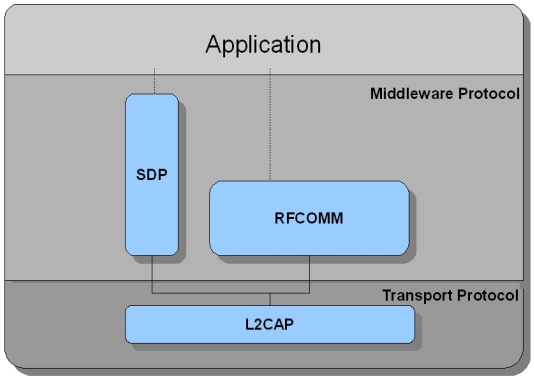


Figura 2.9: Localização relativa dos protocolos *RFCOMM* e *SDP*

O principal motivo de se especificar esta camada dentro do protocolo *Bluetooth* foi prover compatibilidade com aplicações que utilizam o conceito de comunicação serial, tornando possível a troca dos cabos seriais por uma interface de comunicação sem-fio (RF). [40] apresenta uma lista de funcionalidades que devem ser suportadas pela camada *RFCOMM* de modo a se prover um serviço similar às interfaces seriais cabeadas.

Como pode ser observado em Figura 2.9, *RFCOMM* serve de protocolo de transporte a aplicações de alto nível, entretanto, utiliza o módulo *L2CAP* no enlace de transporte. O funcionamento do protocolo é totalmente orientado a conexões (estabelecidas por esta camada mais baixa) onde cada uma permite a utilização de somente uma conexão entre determinado par de dispositivos. Entretanto, utilizando-se a capacidade de multiplexação do protocolo, é possível trabalhar com um número maior de canais (até um máximo de 60) sobre um mesmo *link*. A separação de qual canal está associado a que aplicação é feita através de um identificador numérico para cada uma destas (*DLCI - Data Link Connection Identifier*), possibilitando assim que todos os dados possam trafegar separadamente sobre um mesmo *link* de conexão.

O estabelecimento da conexão prossegue da seguinte maneira:

- A primeira aplicação a fazer uso de determinado *link* deve, primeiramente, executar todo o processo de conexão entre os dois nós comunicantes; e
- Após o estabelecimento da conexão, os canais podem ser utilizados para o envio/recebimento dos dados;

- Ao término da última aplicação, esta deve fechar a conexão, e consequentemente, encerrar o *link* existente.

Para se manter toda esta arquitetura funcionando é necessário que os dispositivos troquem informações de controle de forma a gerenciar e manter todo o conjunto de canais e *links*. Este tipo de controle, de acordo com a especificação em [47], transita dentro de um canal especial, previamente reservado pelo protocolo para tal finalidade.

A descrição completa do protocolo *RFCOMM* foge ao escopo do trabalho aqui apresentado. Entretanto, uma definição mais detalhada pode ser encontrada no documento de especificação do protocolo em [47].

Controle de dados de voz

De modo a suportar o tráfego específico de voz, o protocolo *Bluetooth* precisa tratar de uma maneira especial este tipo de informação. Para tanto, a camada *middleware* implementa os módulos *Telephony Control* e *TCS-BIN* (*Telephony Control Specification* - o sufixo *BIN* provém da codificação binária utilizada pelo mesmo) possibilitando assim o estabelecimento, controle e gerenciamento de chamadas telefônicas pelo protocolo *Bluetooth*.

- *TCS-BIN*: Quando o ambiente *Bluetooth* reconhece a utilização da funcionalidade de voz, este é ativado automaticamente para que possa gerenciar o envio e recebimento dos dados, utilizando diretamente a camada de transporte *L2CAP* (2.3.1). Dois pontos importantes caracterizam o módulo:
  - Gerenciamento de grupos: Permite o compartilhamento de uma conexão telefônica com os nós vizinhos participantes de um mesmo grupo desde que estes também implementem o módulo *TCS-BIN*.
  - Não orientado a conexão: Uma característica particular do protocolo é a implementação de um módulo de troca de informações de controle das chamadas entre dois componentes sem que haja necessidade de se estabelecer uma conexão física entre os dois. Entretanto, apesar de existir uma banda reservada para o compartilhamento deste tipo de informação, somente mensagens *CL_Info* são permitidas pela especificação. Estas carregam dados de controle sobre o enlace de áudio (utilizado como parâmetro para diversos ajustes direta e indiretamente relacionados à qualidade deste) e sobre a empresa que provê o serviço de telefonia.
  - Controle das chamadas: Componente que atua como um controlador de todas as chamadas telefônicas suportadas pelo dispositivo. Entretanto, este é limitado somente aos processos de recebimento e discagem, sendo que seu escopo

não compreende o gerenciamento do canal criado entre os dispositivos para a troca de informações. Como forma de gerenciar o esquema de chamadas o módulo implementa um esquema de máquina de estados que prevê todos os passos a serem executados durante o procedimento de discagem e recebimento de chamadas.

- *Telephony Control*: Tem sua utilização voltada principalmente a aplicações que para se comunicar necessitam ter controle sobre algum tipo de *modem*, possibilitando assim que este seja controlado via *software*.

Protocolos de rede

Uma das principais idéias da especificação do ambiente *Bluetooth* é prover compatibilidade e interoperabilidade com diversos tipos de outros protocolos. Visando tal possibilidade, o protocolo *Bluetooth* prevê a implementação de um modo de comunicação com diferentes protocolos típicos de redes LAN, como: TCP, IP, UDP, etc.

Através deste suporte, uma aplicação *Bluetooth* pode, utilizando esta camada de *software* se comunicar com outras aplicações que possuam uma interface nos protocolos anteriormente citados. Este comportamento é utilizado para a comunicação com pontos de acesso através do protocolo PPP (*point-to-point*). Uma descrição mais detalhada do modo de operação e implementação destes protocolos foge ao escopo do trabalho aqui apresentado, e, uma vez que atualmente estas interfaces já são largamente utilizadas, o detalhamento de cada uma delas pode ser encontrado em suas respectivas especificações.

2.3.3 *Application Protocol*

Assim como nas camadas discutidas em 2.3.1 e 2.3.2 a idéia de abstração se mantém aqui. Basicamente, a camada de aplicação serve como interface de comunicação e acesso às funcionalidades contidas nos níveis mais baixos do ambiente *Bluetooth*. A especificação do protocolo prevê que nesta camada sejam alocadas tanto aplicações que venham a atuar como *device drivers*, consequentemente dependentes de plataforma, assim como aplicações independentes que fazem uso das primeiras de modo a ter acesso indireto ao *hardware* e utilizar as funcionalidades associadas ao ambiente.

Sob um olhar mais detalhado é possível se notar que a camada de aplicação na verdade é dividida em duas outras subcamadas específicas para as funcionalidades relacionadas pelas mesmas:

- Perfis de aplicações (*Application profiles*);
- *APIs* dependentes de plataforma.

Este conceito de separação interna é útil quando se pensa que aplicações de determinados tipos necessitam possuir uma interface comum de modo a prover interoperabilidade entre diversas plataformas, por outro lado a especificação permite também ao desenvolvedor criar aplicações adaptáveis à suas necessidades e que possuam funcionalidades específicas. Apesar de não padronizar todos os tipos de aplicações, a especificação do protocolo define um padrão básico para as ações obrigatórias a serem executadas por qualquer tipo de aplicação, como: protocolo de descoberta de serviço (2.3.2), gerenciamento de conexões (2.3.1), etc.

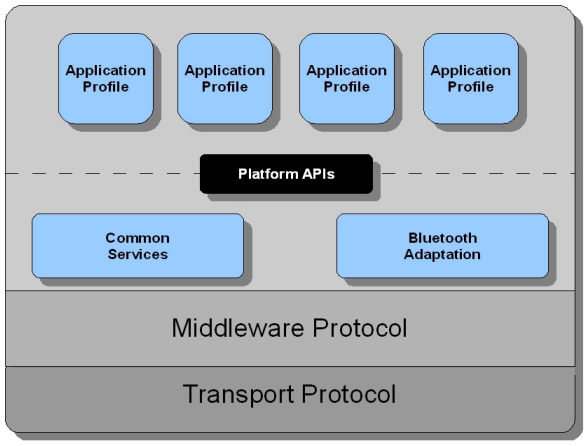


Figura 2.10: Componentes da camada *application*

A Figura 2.10 apresenta com mais detalhes a arquitetura componente da camada de aplicação proposta pela especificação *Bluetooth* e a separação interna contida na mesma.

Perfis de aplicações

Além de possibilitar o uso da pilha de protocolos diretamente pela aplicação, a especificação *Bluetooth* permite ainda um esquema de padronização a ser utilizado em determinados tipos de aplicações e funcionalidades. Para tanto, estas classes de aplicações são agrupadas pelo protocolos em diferentes perfis que possuem uma espécie de interface interna obrigatória para aplicação que implemente o mesmo.

Ao especificar funcionalidades e não diretamente interfaces, o protocolo *Bluetooth* permite manter a interoperabilidade entre aplicações em diferentes plataformas além de uma maior maleabilidade na implementação dos perfis, uma vez que estes podem ser facilmente portados para diferentes plataformas desde que venham a prover as mesmas funções às aplicações.

Apesar da existência de uma grande gama de diferentes perfis, aqui, por limitação de escopo é descrito somente como estes estão separados e organizados internamente segundo a especificação.

Como forma de facilitar o entendimento e o uso dos mesmos, os perfis aptos a serem utilizados pelas aplicações *Bluetooth* apresentam-se relacionados e divididos em diferentes grupos de acordo com suas funcionalidades, serviços providos e aplicabilidade dentro do protocolo.

- Perfis *Serial* e de rede: Estão englobados aqui os perfis referentes à comunicação serial e de rede. Internamente estes dois podem ser claramente separados já que os perfis de rede operam sobre interfaces de protocolos de comunicação via rede específicos, como: *dial-up*, fax, *LAN*, etc;

  Por outro lado, como o próprio nome descreve, o perfil *Serial* fica encarregado de prover interfaces para a comunicação serial entre os dispositivos. Internamente descrito como o nome de *SPP (Serial Port Profile)*, este utiliza diretamente a camada *RFCOMM*.

- Telefonia: Perfis de aplicações que suportam a utilizam de tráfego de voz através do ambiente *Bluetooth*;

- Genérico: Os perfis deste tipo são necessários para a criação de um canal de comunicação entre os dispositivos comunicantes. Dessa forma, os perfis que se encaixam neste grupo estão diretamente relacionados com a camada de transporte do protocolo 2.3.1. Co-existem neste grupo dois diferentes perfis:

  – *SDAP* (*Service Discovery Application Profile*): Responsável por prover uma interface de comunicação às aplicações que desejam utilizar o módulo de descobrimento de serviços (*SDP*). Basicamente, todo dispositivo que participa do processo de descoberta, pode ser encaixado dentro dos seguintes papéis (em somente um ou em ambos):

    * **Dispositivo local**: Considera-se como dispositivo local aquele que inicia o processo de descoberta ao tentar localizar nós dentro do alcance de seu sinal;
    * **Dispositivo remoto**: Por sua vez, como dispositivo remoto considera-se aquele que recebe o *inquiry* e responde à esta solicitação de descoberta.

  Apesar do protocolo *Bluetooth* não especificar *APIs* obrigatórias para lidar com a funcionalidade de descoberta de serviços, determinadas funções (primitivas) devem obrigatoriamente ser implementadas como uma maneira de prover interoperabilidade entre os dispositivos.

  - ***serviceBrowse***: Utilizado quando o nó executa uma busca geral por serviços disponíveis ou por determinado serviço específico;
  - ***serviceSearch***: Similar ao anterior, esta primitiva permite ainda que os serviços encontrados sejam filtrados de acordo com seus atributos;
  - ***enumeratedRemDev***: Utilizado para se executar uma busca dos nós vizinhos dentro da área de alcance do mesmo. Uma característica adicional desta primitiva é a capacidade de se filtrar as respostas de acordo com o tipo do dispositivo;
  - ***terminatePrimitive***: Após sua invocação, qualquer uma das primitivas anteriores tem sua execução finalizada.

- *GAP* (*Generic Access Profile*): Pode ser definido como uma base comum a todos os perfis do protocolo *Bluetooth* uma vez que este é utilizado por toda aplicação que deseja o estabelecimento de um canal de comunicação com outro dispositivo. A verdadeira importância do perfil reside no fator interoperabilidade que é característica deste ambiente, sendo que o perfil *GAP* define toda a máquina de estados pelo qual passa o processo que antecede a conexão de dois nós, como: descoberta de dispositivos, *inquiry*, pareamento e gerenciamento do canal. Além destas definições, o perfil suporta ainda diversos modos de operação, de acordo com cada característica implementada pelo mesmo:
  - **Modos de conectividade**: Para que o dispositivo esteja apto a estabelecer um canal de conexão, este deve possuir interfaces que possibilitem criar e gerenciar interfaces de comunicação com outros componentes da rede. Os diferentes modos de conectividade permitem diretamente às aplicações gerenciar esta característica do dispositivo.

  - **Modos de descobrimento**: Além de possibilitar que o dispositivo seja "conectável", o perfil *GAP* gerencia ainda o método de descobrimento e visibilidade. Três destes níveis são definidos:
    - **Modo geral**: Dispositivo responde a todo *inquiry* que também utilize o mesmo modo, podendo assim ser descoberto;
    - **Modo limitado**: Dispositivo só responde a *inquiries* que também utilizem o modo limitado em sua busca;
    - **Modo de não descoberta**: Dispositivo não responde a qualquer tipo de *inquiry*, estando assim em um estado de impossibilidade de ser descoberto por outros dispositivos.

* **Modos de conectabilidade**: Durante o ciclo de vida de uma aplicação, alguns dos passos iniciais da mesma é executar o descobrimento de nós vizinhos ou ser descoberta por outros dispositivos. Para que este segundo caso possa acontecer, o nó deve estar apto a responder tais solicitações e *inquiries*, fatores estes que garantem que o dispositivo seja "conectável" e possa ser utilizado diretamente pela aplicação através do perfil *GAP*.

* **Modos de pareamento**: O perfil *GAP* possibilita à aplicação a utilização de esquemas de pareamento em suas comunicações. A funcionalidade de pareamento é executada todas as vezes que determinados dispositivos desejam criar um canal seguro de comunicação entre os mesmos. Para a criação deste canal, utiliza-se ainda protocolos específicos para autenticação.

* **Modos de segurança**: Assim como as características citadas anteriormente, o perfil *GAP* também oferece suporte a determinados níveis de segurança à serem utilizados pelas aplicações.
  - Modo de segurança 1: Questões relacionadas a segurança não são consideradas neste modo;
  - Modo de segurança 2: Considerações a respeito da segurança são feitas na camada *L2CAP* de modo a ser possível uma associação entre o protocolo utilizado, os dispositivos comunicantes e as aplicações que fazem uso do canal;
  - Modo de segurança 3: No nível 3 de segurança, as considerações a respeito da segurança são feitas na camada *link manager* (Seção 2.3.1) onde a autenticação é utilizada para todas as conexões criadas entre os dispositivos.

Capítulo 3

Ambientes de Tempo Real e Ordenação Total de Mensagens

3.1 Introdução

O protocolo proposto pelo trabalho aqui apresentado pode ser contextualizado, ao mesmo tempo, dentro destes dois assuntos distintos: Ambientes de Tempo Real e Ordenação Total. Além da dificuldade em se encontrar um ponto comum entre estas duas áreas, associam-se as mesmas, aplicações para ambientes *Bluetooth* (apresentado no capítulo 2), fazendo com que se tenha três variáveis a serem consideradas nas implementações propostas.

Observar uma relação entre as três áreas descritas anteriormente só é possível quando estão localizadas dentro de um mesmo contexto. Esta é a idéia aqui apresentada quando se propõe um estudo sobre algoritmos de ordenação total para aplicações que possuem constantes delimitadoras do tempo de comunicação entre determinadas entidades de *software* e/ou componentes de uma rede.

Apesar de serem claras as definições tanto para ambientes de tempo real quanto para o problema de ordenação total de mensagens, até hoje, na literatura uma contextualização destas duas áreas juntamente à redes *Bluetooth* não é facilmente encontrada. O ineditismo do trabalho aqui apresentado se baseia principalmente neste fator. Entretanto, para entender como estas grandes áreas estão relacionadas, é necessário entender o funcionamento das mesmas quando não intimamente ligadas umas às outras.

3.2 Ambientes de Tempo Real

O conceito de tempo real está diretamente associado a eventos que possuem um limite de tempo para desencadear uma ação. Dessa forma, é uma unanimidade a descrição da variável "tempo de resposta" como o principal componente de um ambiente deste tipo [15, 36].

Entretanto, é comum, apesar de errônea, a idéia de que sistemas de tempo real necessitem sempre de limites de tempo muito curtos para executarem com sucesso. Na verdade, o tempo entre a geração de um evento e a reação ao mesmo varia conforme o tipo de aplicação, podendo ser de poucos microssegundos (no caso de sistemas de controle de vôo) até horas (no caso de uma aplicação que controla uma reação química). Baseando-se em tais premissas, o conceito acaba por ter uma divisão em dois grupos distintos de aplicações:

- *Hard Timing Constraints*: Sistemas onde a ausência de resposta a um evento, ou seja, uma falha, pode ter uma consequência mais séria no ambiente;
- *Soft Timing Constraints*: Sistemas que prevêem um certo tipo de tolerância para o tratamento de uma falha ocorrida.

Em [15], além da contextualização das aplicações nos dois grupos descritos anteriormente, é apresentada uma descrição genérica da composição de um ambiente de tempo real em duas partes:

- **Sistema controlador**: Dispara determinadas ações após basear-se em dados recebidos dos processos controlados; e
- **Processo controlado**: Entidade responsável por analisar e enviar dados relacionados às medições do sistema, de modo a permitir que o sistema controlador intervenha através de determinadas ações.

A Figura 3.1 apresenta uma descrição visual dos papéis relacionados a cada um dos grupos citados acima. Nela, é possível se observar que após um determinado período de análises, o sistema controlador dispara uma ação de modo a tentar corrigir ou manter equilibrado o ambiente como um todo.

Além da descrição lógica (Seção 3.2.1) das aplicações de tempo real, tem-se ainda uma arquitetura física que deve ser capaz de suportar todas as medições e consequentes ações a serem tomadas pelo processo controlador do ambiente. A junção do modelo lógico com o modelo físico depende principalmente da necessidade de se aliar tarefas utilizadas para mapear um no outro além do desenvolvimento e implementação da aplicação em si. Tais tarefas podem ser agrupadas dentro dos seguintes grupos:

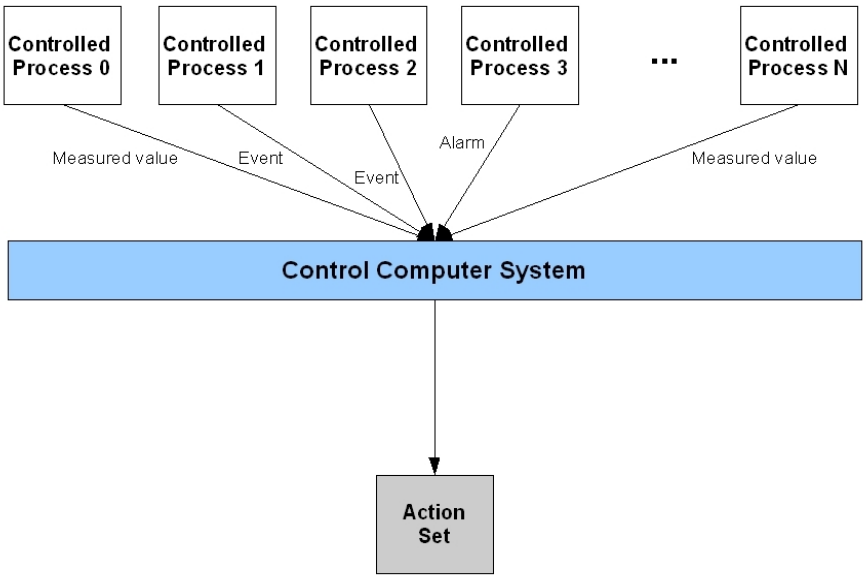


Figura 3.1: Arquitetura básica de uma aplicação de tempo real (Adaptada de [15]).

- **Análise de requisitos**: Como em todo processo de desenvolvimento, uma análise de requisitos é necessária como forma de se mapear as necessidades do sistema. Em especial, limitações de tempo de resposta para este tipo de ambiente.

- **Desenho preliminar do sistema**: Utilizado como forma de se mapear as necessidades levantadas no item anterior em pontos de implementação. Define como cada componente/módulo será desenvolvido para compor o sistema.

- **Desenvolvimento de componentes específicos**: Desenvolvimento ou reaproveitamento e integração tanto dos componentes de *software* quanto de *hardware*.

- **Testes de integração**: Parte-se das premissas pressupostas na fase de análise e então, é executada uma bateria de testes que tendem a validar o desenvolvimento feito na fase anterior.

- **Validação do usuário**: Executado antes da aceitação do sistema visa validar todas as variáveis que englobam e definem o mesmo. Dessa forma, o usuário pode testar e verificar se o sistema funciona de acordo com as especificações coletadas no primeiro passo do processo.

3.2.1 Arquitetura física e lógica

- A questão "arquitetura física" frequentemente remete a uma associação direta com o *hardware*. No caso de ambientes de tempo real esta associação também permanece verdadeira, uma vez que toda e qualquer aplicação de tempo real, senão totalmente, deve ser em partes suportada por uma plataforma de *hardware* adequada.

- Por outro lado, a arquitetura lógica se baseia principalmente em componentes de *software* para a sua execução. Dentro destes componentes a classe que talvez ganhe mais destaque sejam os sistemas operacionais. [15] classifica genericamente os sistemas de computação em geral em três diferentes grupos:

  – Sistemas transformacionais: Baseados principalmente no aspecto de mesmo nome, os resultados gerados pelo sistema são computados através de dados coletados durante todo o ciclo de vida da aplicação até um dado momento. Isto é, dados coletados no início da execução do sistema podem ser utilizados a qualquer momento, dependendo da necessidade apresentada pelo mesmo, sem que haja uma limitação de tempo para o uso dos mesmos.

  – Sistemas interativos: Além de se basear no aspecto transformacional, utiliza também o fator "relacionamento entre entidades de *software*". A característica principal é sua utilização por determinado sistema de computação de dados que foram gerados por um segundo sistema.

  – Sistemas reativos: Além das duas características listadas nos pontos anteriores, este faz uso ainda de limitantes de tempo pré-determinadas pelo processo controlador.

  A Figura 3.2 apresenta uma separação visual para o conceito de aspectos e sistemas computacionais.

3.2.2 Sistemas Operacionais e Escalonamento de Tarefas em Tempo Real

Atualmente, o modelo arquitetural mais comum para um ambiente tempo real prevê a utilização de um sistema centralizador de todas as operações do sistema. Da mesma forma, é comum também o conceito de alocação estática dos recursos administrados por este sistema operacional, de modo que todas as associações dos recursos são feitas antes do início da execução do sistema.

Entretanto, em determinados modelos de ambientes de tempo real não é possível se aplicar este conceito uma vez que os mesmos podem necessitar utilizar em quantidade

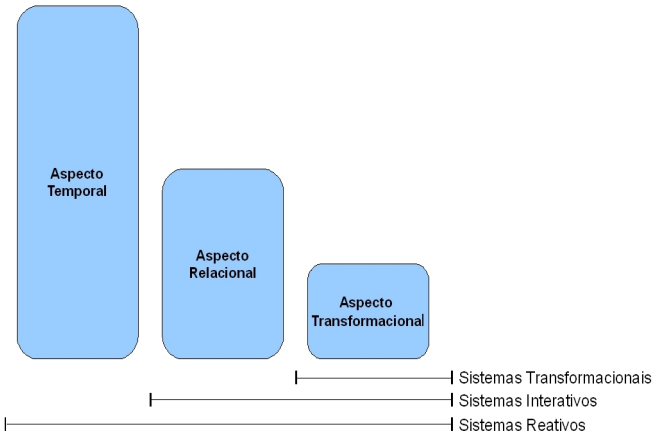


Figura 3.2: Relacionamento entre aspectos e sistemas computacionais.

elevada determinado recurso, como por exemplo: memória de armazenamento ou uma gama heterogênea de *softwares* controladores. Como forma de se considerar ambos os casos, sistemas operacionais híbridos foram desenvolvidos para se prover características de um sistema comum, mas que possuísse um *kernel* não sujeito a interrupções. A partir disso, é possível se concluir de forma clara que o principal gargalo do sistema se encontra no fator **escalonamento de tarefas** a serem executadas pelo processador.

Por definição, um sistema operacional de tempo real irá executar tarefas deste mesmo tipo que, por sua vez, são agrupadas e escalonadas de acordo com suas características principais:

- **Períodos**:
  - Periódicas: São executadas a cada determinado período de tempo fixo;
  - Aperiódicas: Não possuem claramente um período de execução. Ou seja, podem ser escalonadas a qualquer momento pelo sistema operacional. [15][31][53] apresentam uma lista de parâmetros que regem e delimitam o conceito de tarefas periódicas e aperiódicas.
- **Restrições**:
  - Fortes: A não completude de determinada tarefa dentro de um determinado limiar de tempo pode levar o sistema como um todo a uma falha grave. Quando uma tarefa possui uma restrição forte de tempo real, utiliza-se a seguinte expressão para o cálculo do tempo limite para a execução da mesma: $d = r + D$, onde:

d: Tempo limite de término da execução da tarefa;

r: Tempo normal de término da execução da tarefa;

D: Atraso máximo permitido quando a tarefa não se encerra em r.

- Fracas: A não completude de determinada tarefa dentro de um determinado limiar de tempo pode causar uma degradação no tempo de resposta, mas não uma falha grave no sistema;
- Firmes: A não completude de N limites de tempo não levará a uma falha total do sistema. Entretanto, o não respeito a um número maior que N pode levar a uma falha completa do sistema.

- **Preemptividade**:
  - Preemptivas: Uma vez iniciada, este tipo de tarefa pode ser escalonada pelo *kernel* de volta ao conjunto de tarefas prontas para serem executadas;
  - Não-preemptivas: Uma vez iniciada, este tipo de tarefa deve executar por completo. Isto é, uma vez de posse de determinado recurso, esta deve executar até o seu final sem que seja escalonada pelo *kernel* de volta ao conjunto de tarefas prontas para serem executadas.

- **Dependência**: Como em qualquer tipo de sistema operacional, as tarefas não executam isoladamente. A interação e dependência de uma tarefa por outra também ocorre em ambientes de tempo real. Para tratar tal característica (que com o tempo acaba por tornar-se um problema) é necessária a definição de uma espécie de "precedência relacional" entre as tarefas, que por sua vez leva em conta o tipo de relacionamento entre as mesmas de modo a definir uma ordem de execução. A precedência é definida de modo estático antes da execução sendo que se aplicam as estas também os conceitos de exclusividade, região crítica e alocação de recursos.

- ***Jitter***: Em ambientes de tempo real, o conceito de *Jitter* pode ser considerado como sendo o tempo decorrido entre o início de duas tarefas consecutivas;

- **Urgência**: Este conceito está diretamente relacionado com o tempo máximo de término de uma tarefa de modo que o nível de urgência pode ser definido como o tempo máximo para a execução da mesma quando contextualizada com o conjunto total a ser executado;

- **Nível de importância**: Quando o nível de urgência de duas tarefas é o mesmo, utiliza-se o nível de importância como um fator de desempate na criação do grafo de prioridades de execução de tarefas do sistema;

- **Prioridade externa**: A definição de prioridades de um sistema operacional pode ser feita tanto pelo próprio sistema ou por algum ator externo que por sua vez se baseia em outros fatores para determinar a ordem de execução das mesmas.

A partir dos conceitos anteriormente apresentados nota-se que o escalonamento das tarefas em um sistema operacional está diretamente relacionado a uma série de fatores que delimitam a necessidade de tempo e urgência para a execução e término das mesmas. Isto torna possível ordená-las, além de principalmente garantir que sejam executadas dentro dos limites de tempo impostos pelo ambiente de tempo real.

De um modo geral, este tipo de escalonamento pode ser associado a dois diferentes grupos de tarefas:

- Tarefas relacionadas a determinada(s) ação/ações que visam permitir a execução da(s) mesma(s) num ambiente de operação normal; ou
- Conjunto de tarefas de maior importância quando em um ambiente de operação anormal (após a ocorrência de determinadas falhas ou eventos inesperados).

Dessa forma, em qualquer um destes ambientes é possível se obter uma lista ordenada das tarefas à serem executadas. Entretanto, a geração desta lista requer a aplicação de um algoritmo específico de escalonamento que decidirá de acordo com determinada política a tarefa a ser executada num dado momento. Os algoritmos mais comumente considerados são:

- *FIFO - First In First Out*: Como o próprio nome descreve, o algoritmo se baseia no conceito de fila onde as requisições mais velhas tem prioridade de processamento sobre as requisições mais novas;
- Mais curtas primeiro: Tarefas que demandam um menor tempo de execução são processadas com maior prioridade (e sem preempção) em detrimento das mais demoradas. O algoritmo apresenta determinadas dificuldades de implementação uma vez que estaticamente, a determinação do tempo de execução de determinada tarefa apresenta-se muito custosa;
- *Round-robin*: Utiliza o conceito de tempo compartilhado onde o tempo de uso de determinado recurso é dividido em *slots*. Cada tarefa, para o processamento de seus dados, pode fazer uso de um ou mais *slots* de tempo. Caso esta termine o processamento antes do final do tempo a ela delegado, o recurso é automaticamente liberado para a próxima tarefa. Caso a tarefa não consiga finalizar seu processamento dentro de seu espaço de tempo, esta é escalonada, entra novamente na fila de execução para que posteriormente possa voltar a utilizar o recurso.

Área	Aplicação
Engenharia Aeroespacial	Controle dos sistemas de navegação de aeronaves
Sistemas críticos	Controles de usinas nucleares
Indústria automobilística	Sistemas de freios, *airbags* e segurança em geral
Multimídia	Jogos e vídeo/áudio *on demand*

Tabela 3.1: Exemplos de utilização de sistemas/ambientes de tempo real.

- Prioridades relacionadas às restrições: Para cada tarefa é associada uma prioridade e a tarefa em execução é sempre aquela com maior nível de prioridade. O algoritmo possui duas versões, sendo que uma delas utiliza o modelo preemptivo e o outro não. Entretanto, o algoritmo fica sujeito à ocorrência de *starvation* já que tarefas com baixa prioridade podem nunca ser executadas.

- Prioridades de múltiplos níveis: As tarefas que desejam utilizar determinado recurso são agrupadas em diferentes filas onde cada uma representa uma diferente prioridade. A partir desta separação inicial, a execução das tarefas de cada fila pode seguir um dos algoritmos descritos anteriormente (*Round-robin* ou *FIFO* sem preempção) sem que necessariamente o tamanho dos *slots* de tempo seja o mesmo.

3.2.3 Exemplos e Aplicabilidade

Como descrito nas seções anteriores, os conceitos de tempo real estão relacionados principalmente a aplicações com alguma restrição de tempo de resposta. Para tanto, existe uma gama de premissas e conceitos que delimitam o assunto e, consequentemente caracterizando também as aplicações.

A contextualização destes conceitos com o presente trabalho se dá na análise da classe de aplicações descritas pelo mesmo. Esta engloba principalmente aplicações que demandam restrições mínimas da variável "tempo de resposta" de modo que seja possível se proporcionar ao usuário uma experiência interativa com determinado nível de qualidade. Além disso, a caracterização destas atividades como tal, faz com que apareça a necessidade de utilização de outra classe de conceitos/problemas que serão discutidos na seção 3.3.

Tabela 3.1 exemplifica alguns dos conjuntos de aplicações dos sistemas/ambientes de tempo real mais conhecidos atualmente.

3.3 Ordenação Total de Mensagens

A ordenação de mensagens é um problema básico da área de computação distribuída e está intimamente ligada ao conceito de distribuição e execução simultânea de diferentes processos.

Em uma contextualização mais ampla deste problema, nota-se que as mensagens são enviadas a destinatários que nem sempre estão presentes (ou próximos) fisicamente, e, consequentemente, suscetíveis a determinados grupos de problemas. É interessante ressaltar o fato de que nem sempre o problema de ordenação de mensagens é tratado da mesma forma, podendo variar de acordo com o modelo, escopo e tipo de aplicação ao qual irá servir.

Os algoritmos descritos por este trabalho (em especial o apresentado no capítulo 4) utilizam diversos conceitos que estão diretamente associados à teoria dos sistemas distribuídos, mais especificamente algoritmos distribuídos [37, 9, 22].

- Algoritmos distribuídos:
  - Conceitos básicos:
    - Algoritmos síncronos / assíncronos;
    - Eleição;
    - Falhas;
    - Oráculos;
  - Consenso distribuído:
    - Grupos de processos;
    - Ordenação de mensagens.

É possível se notar ainda certa interseção com conceitos típicos de outras áreas, como: protocolos de rede, *broadcast/multicast*, computação pervasiva e ambientes de tempo real (3.2).

3.3.1 Algoritmos Distribuídos

[48], numa descrição resumida, apresenta um sistema distribuído como sendo:

> "Uma coleção de computadores independentes que, para o usuário são apresentados como sendo a abstração de somente um."

Dessa forma, baseando-se nesta descrição e generalizando a idéia de algoritmos distribuídos, é possível concluir que os mesmos são um conjunto de processos independentes

que executam em diferentes nós, fisicamente próximos (ou não). Utilizam canais de comunicação para a troca de informações de modo a manter a coerência global na execução de determinada tarefa. Com o objetivo de maximizar o processamento e execução destas, atualmente, os algoritmos distribuídos são utilizados em diversos tipos de aplicações como: telecomunicações, indústria bélica, cálculos científicos, robótica, etc.

No trabalho aqui apresentado, os algoritmos distribuídos são representados principalmente através de uma classe particular de problemas, chamada "ordenação de mensagens em grupos de processos". A proposta apresentada no capítulo 4 faz uso direto destes conceitos para implementar um protocolo que permite o compartilhamento de informações de forma ordenada utilizando para tanto diversos canais de comunicação em um ambiente de rede *Bluetooth*.

Dentre as diversas variáveis existentes em algoritmos distribuídos algumas merecem especial atenção por influenciarem diretamente ou indiretamente o comportamento do sistema como um todo. Em sua grande maioria, tais variáveis se aplicam diretamente aos processos. As seguintes características normalmente são relevantes em ambientes de comunicação como o descrito.

Para uma melhor compreensão do conceito geral deste tipo de protocolo, faz-se necessário a introdução a conceitos básicos aplicados de modo geral a computação distribuída.

Conceitos básicos

Algoritmos síncronos / assíncronos

A caracterização de um algoritmo como síncrono ou assíncrono [16, 50, 20] está diretamente relacionado a duas variáveis distintas:

- Diferença de velocidade entre o processo mais rápido e o mais lento; e
- Atraso na comunicação.

Para cada uma destas existe a delimitação de valores máximos e mínimos que servem como medida do nível de sincronia alcançado pelo sistema. Assim sendo, em relação a sincronia e baseando-se nestas variáveis, [20] classifica os algoritmos distribuídos em:

- Síncronos: Os valores de tempo estabelecidos pelas variáveis são sempre respeitados;
- Assíncronos: Não existe um valor conhecido para ambas as variáveis;
- Parcialmente síncronos: Meio-termo entre os dois conceitos anteriores, nos algoritmos parcialmente síncronos os limites definidos para as duas variáveis são eventualmente respeitados (quando existem).

Eleição

Outro conceito de algoritmos distribuídos também utilizado pelo protocolo apresentado no capítulo 4 é, apesar de não ser utilizada somente para tal finalidade, a idéia de eleição de um líder dentro de um determinado conjunto de processos. O procedimento básico adotado por este tipo de algoritmo é: dado um grupo de processos e algumas rodadas de execução, todos os componentes chegam a um consenso sobre um único nó, que virá a ser eleito. A função do nó eleito varia de acordo com o contexto e necessidades da aplicação. Atualmente, na bibliografia constam diversos algoritmos que podem ser aplicados a eleição em grupos de processos. [37] apresenta alguns destes, inicialmente aplicados à redes em anel, entretanto, aqui limita-se a descrever um dos mais famosos e utilizados:

- Algoritmo LCR (*Le Lann, Chang, Roberts*): Considera sempre que cada processo do grupo possui um identificador único, conhecido como *UID* e se baseia em comparações simples deste valor para que um único processo possa ser eleito. O LCR é executado em no máximo N rodadas, onde N é o número de processos no anel. Em cada rodada:
  - Cada um dos processos envia seu identificador para o próximo processo do anel;
  - Quando um processo recebe um identificador este é comparado ao seu próprio *UID*;
  - Se o *UID* recebido é maior que o seu próprio, o *UID* recebido é então repassado em diante nas próximas rodadas;
  - Se o *UID* recebido é menor que o seu próprio, então este é repassado em diante nas próximas rodadas;
  - Se o *UID* recebido é igual ao seu próprio, então, o nó se declara como líder eleito.

  Figura 3.3 apresenta uma visão esquemática do algoritmo *LCR* no momento do encaminhamento dos identificadores (*UID*s) para os outros componentes da rede.

- Algoritmo de eleição para o protocolo *Bluecast*: O protocolo de ordenação de mensagens apresentado no capítulo 4 utiliza internamente um mecanismo de eleição especialmente desenvolvido para ambientes *Bluetooth*. O mesmo, para a execução da eleição leva em conta o formato específico dos endereços de cada um dos nós que pertencem a este tipo de ambiente.

Além dos dois algoritmos de eleição descritos anteriormente, [37] descreve outros algoritmos síncronos e assíncronos para o problema de eleição tanto para redes em anel quanto para redes gerais.

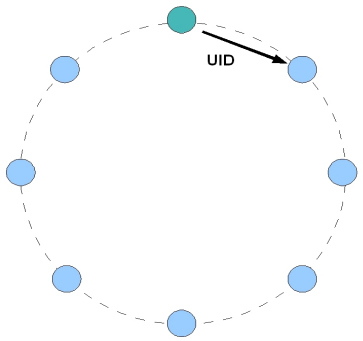


Figura 3.3: Esquema de operação do algoritmo *LCR*

Falhas

Atualmente, quase todas as propostas de protocolos confiáveis para difusão de mensagens envolvem determinados níveis de suporte a falhas, desde o mais básico até os mais complexos - devido à exigências que muitas vezes são impostas pelo ambiente de operação dos mesmos. Assim também ocorre com a idéia apresentada por este trabalho (capítulo 4) e com determinados protocolos apresentados por [5, 37, 12, 49].

Apesar de sua nomenclatura, por si só, a apresentação de uma definição completa sobre o tema falhas (ou o suporte a estas) ([25, 23]) pode ser dividida em diversas e diferentes categorias de acordo com sua ocorrência, ambiente, tipo, etc. Desta forma, visando um melhor entendimento da aplicabilidade de cada um dos tipos de suporte a falhas, faz-se necessária uma descrição mais detalhada das mesmas. Geralmente, as falhas são classificadas de acordo com sua natureza e o ambiente nas quais estas se apresentam. Assim sendo, o tipo e o modo de detecção podem ser aplicados tanto no ambiente síncrono quanto no ambiente assíncrono.

- **Tipos de falhas**: No ambiente descrito aqui neste trabalho, se considera, basicamente a ocorrência de dois tipos de falhas:
  - **Falhas de processos**: Como o próprio nome diz, este tipo de falha está diretamente associado ao processo (nó presente na rede de difusão). Os items abaixo descrevem a natureza de cada uma das falhas, assim como as condições sob as quais estas podem ocorrer:
    - **Falhas do tipo falha-e-pára**: Acontece uma interrupção permanente do processo, entretanto uma terceira parte consegue detectar a falha ocorrida;
    - **Falhas por *crash***: São classificados aqui os processos que, por algum motivo, param seu funcionamento sem que haja possibilidade de uma recu-

peração e retorno ao estado anterior (prévio à ocorrência do erro). Diferentemente do caso anterior, neste não é possível se detectar o acontecimento da falha no processo;

  - ∗ **Falhas por omissão**: Na ocorrência deste tipo de falha, o processo passa a não responder a todas as requisições, isto é, o mesmo passa a se omitir no envio de determinadas respostas (mensagens);
  - ∗ **Falhas de tempo**: Aplicável a ambientes síncronos (subseção 3.3.1), acontece quando um processo viola determinadas regras de tempo. Ou seja, a resposta à determinada requisição não pode levar mais que um limiar limite de tempo acordado para que o ambiente permaneça sendo caracterizado como síncrono [52];
  - ∗ **Falhas Bizantinas**: A caracterização deste tipo de falha se torna de certo modo genérico devido à natureza das mesmas. Teoricamente, toda e qualquer falha que não se enquadre dentro dos grupos anteriores pode ser caracterizada como pertencente a este grupo. Um processo que apresente um comportamento típico destas falhas torna-se apto a executar determinadas ações não-desejáveis com o intento de paralisar o funcionamento de todo o sistema:
    - · Duplicação de mensagens;
    - · Alteração do conteúdo de mensagens;
    - · Geração de mensagens espúrias.

- – **Falhas de *links***: Uma vez que em qualquer sistema distribuído a consistência da execução é feita através de troca de mensagens, o canal pelo qual estas trafegam também está sujeito a erros. Desse modo, as falhas de *links* são frequemente associadas ao enlace por onde informações/mensagens são trocadas. A literatura apresenta hoje a classificação dos canais de comunicação em dois tipos [20, 37, 10]:
  - ∗ **Canais confiáveis (*Reliable Channels*)**: Como o próprio nome descreve, os enlaces confiáveis de comunicação garantem que qualquer mensagem enviada por determinado remetente, eventualmente será entregue ao seu destinatário;
  - ∗ **Canais não-confiáveis (*Lossy Channels*)**: Em contrapartida aos canais confiáveis, este meio de comunicação não oferece garantias de entrega das mensagens enviadas devido a ocorrência dos mais diversos tipos de erros, como: colisões, roteamento, queda da rede, etc. Em uma analogia direta, os canais confiáveis podem ser comparados com o protocolo TCP enquanto que este tipo de canal pode ser associado ao protocolo UDP.

Oráculos

Genericamente, oráculos podem ser classificados como entidades (*software* ou *hardware*) que provêem informações a respeito do sistema como um todo ([32, 42]), ou especificamente sobre um determinado processo. Desse modo, à partir das informações providas por um oráculo, o protocolo distribuído utilizado está apto a tomar as ações relacionadas de forma a corrigir a execução do sistema, caso necessário. Oráculos podem ser agrupados dentro de três diferentes classes:

- **Relógios Físicos**: Oráculos baseados em relógios provêem basicamente informações sobre o tempo físico. Normalmente, assim como os relógios reais, os oráculos baseados em relógios iniciam em determinado valor e vão sendo incrementados passo-a-passo onde cada um destes incrementos é contado como uma unidade de tempo do mesmo. Quanto ao sincronismo, estes são caracterizados de três diferentes formas:
  - **Parcialmente síncronos**: Em um mesmo dado momento, a diferença entre os valores retornados por dois diferentes relógios variam em no máximo ϵ;
  - **Síncronos**: Em um mesmo dado momento, a diferença entre os valores retornados por dois diferentes relógios não possui variação $\epsilon = 0$;
  - **Assíncronos**: Não existe um limite máximo (ϵ) para a diferença de variação dos valores retornados por dois relógios num mesmo dado momento.

- **Detectores de falhas**: Talvez o mais conhecido dos oráculos, os detectores de falha, como leva a entender o próprio nome, tem por característica a detecção falhas dentro do ambiente no qual está sendo executado. Segundo o trabalho apresentado por [11], os detectores de falha podem ser modelados como sendo um conjunto de módulos, onde cada um destes está associado a um diferente processo. Dessa forma, um dado processo pode perguntar ao seu detector de falhas o *status* de qualquer outro processo do grupo.

  Entretanto, assim como em outras áreas da computação, a detecção do estado corrente de um processo ou sistema pode não ser tão simples. Na prática, os detectores de falhas não são 100% confiáveis uma vez que o provimento das informações relativas aos outros processos pode não corresponder ao estado atual do processo. Baseando-se nesta característica, [11] introduz o conceito de suspeita, onde um determinado processo pode suspeitar da ocorrência de uma falha num outro dado processo.

  A classe de um detector é dada a partir do nível de incerteza / "desconfiança" provida por este sobre as falhas de um processo. Informalmente, estas classes são

regidas diretamente por dois conceitos que vem a gerar oito grupos de classes diferentes:

- **Completude (*Completeness*)**:
  - **Completude fraca (*Weak Completeness*)**: Descreve um limite de tempo após o qual todos os processos que falharam são permanentemente colocados sob suspeita por algum processo cuja execução continua correta;
  - **Completude forte (*Strong Completeness*)**: Descreve um limite de tempo após o qual todos os processos que falharam são permanentemente colocados sob suspeita por todos os processos cujas execuções continuam corretas.
- **Precisão (*Accuracy*)**:
  - **Precisão fraca (*Weak Accuracy*)**: Algum processo não é suspeito de falha até que a mesma ocorra;
  - **Precisão forte (*Strong Accuracy*)**: Nenhum processo é suspeito de falha até que a mesma ocorra;
  - **Precisão eventual fraca (*Eventual Weak Accuracy*)**: Existe um tempo após o qual algum processo com execução correta não é suspeito de falha por um outro processo de execução correta;
  - **Precisão eventual forte (*Eventual Strong Accuracy*)**: Existe um tempo após o qual todos os processos com execução correta não são suspeitos de falha por um outro processo de execução correta.

Consenso Distribuído

O conceito de consenso dentro da área de sistemas distribuídos descreve uma classe de problemas onde todos os processos componentes de um grupo devem chegar a um acordo para a execução de determinadas atividades ou tomada de decisões.

Os problemas de consenso podem ser subdivididos em diversas classes, de acordo com a aplicabilidade e o ambiente no qual cada um está inserido. Dessa forma, a classe de problemas aqui apresentada se encaixa dentro de uma destas diferentes taxonomias do problema. Basicamente, num agrupamento resumido, tem-se as seguintes classes:

- ***Consenso Bizantino***: Diretamente associado com as falhas bizantinas, este item foi inicialmente descrito e contextualizado junto ao problema dos **generais Bizantinos** [37, 35, 18]:
  - Em 1453, todas as divisões do exército Bizantino estavam acampadas ao redor de uma cidade inimiga. Cada uma destas divisões era comandada por

seu próprio general sendo que uma estava apta a se comunicar com as outras através da utilização de um mensageiro. O problema de acordo se faz da seguinte maneira: De forma a atacar mais fortemente a cidade (visando ter uma menor possibilidade de derrota) os generais necessitavam entrar em um acordo sobre o momento certo para realizar o ataque. Entretanto, alguns dos generais poderiam ser traidores e agir de forma a tentar prevenir o ataque. Em [35] é descrito um algoritmo que trata este problema de forma que:

  * Todos os generais leais decidam pelo mesmo momento de ataque;
  * Um pequeno número de generais traidores não possa utilizar os generais leais para escolher um mal momento de ataque.

- ***Difusão Confiável***: Como descrito por [20, 37, 12, 27], o problema de difusão confiável pode ser tratado com um subconjunto do problema de difusão com ordenação total. Basicamente, difusão confiável consiste em:

  – Dado um grupo de processos que se comuniquem através da troca de mensagens, o *broadcast* confiável ocorre quando, um destes processos do grupo envia uma mensagem m e esta:

    * Garantidamente é entregue a todos os processos operando corretamente; ou
    * A entrega não ocorre a nenhum dos processos do grupo.

  Dessa forma, garante-se que todos os processos operando corretamente sempre terão o mesmo nível de conhecimento sobre os outros processos do grupo ao qual pertence.

- ***Difusão com Ordenação Total***: O problema de ordenação total para comunicações por difusão pode ser resumidamente definido como o problema de difusão confiável acrescido de algumas outras restrições e variáveis. Isto porque, além das características deste último, como o próprio nome descreve, o conceito de ordenação de mensagens deve ser empregado. Além de prover a entrega destas para todos o processos (ou para nenhum deles), as mensagens devem ter sua ordem de envio mantida na entrega.

  Além desta característica macro, um sistema que implementa o conceito de ordenação total deve levar em conta diversos outros parâmetros como: falhas, modo de ordenação das mensagens e sincronismo. Entretanto, para se delimitar este campo de atuação é necessário um entendimento sobre determinados conceitos básicos. De acordo com [11], estes conceitos podem ser classificados dentro das seguintes classes:

- **Validade**: Se um processo válido envia uma mensagem m, então, eventualmente esta mensagem será entregue a todos os outros processos válidos do grupo;
- **Acordo uniforme**: Se uma mensagem m é entregue a um processo, então, esta mesma mensagem m será entregue a todos os outros processos válidos do grupo;
- **Integridade uniforme**: Para uma dada mensagem m, todos os processos válidos devem receber a mesma pelo menos uma vez, desde que esta tenha sido enviada por algum processo válido pertencente ao grupo;
- **Ordem total uniforme**: Se a dois processos são entregues as mensagens m e m', então m só deve ser entregue antes de m' se assim for feito em ambos os processos.

Os conceitos acima estão diretamente relacionados ao problema de ordenação total. Entretanto, os algoritmos empregados para a solução do mesmo, também apresentam determinadas características e propriedades em comum, e que devem ser guardadas.

- **Uniformidade**: Determinados tipos de aplicações requerem certas características de uniformidade. Isto é, ações devem ser tomadas em relação a todos os tipos de processos pertencentes ao grupo (faltosos ou não). Entretanto, como descrito por [20], as características de uniformidade muitas vezes não são levadas em conta se os protocolos desenvolvem meios de corrigir as inconsistências e suportar as falhas que podem vir a acontecer devido ao desrespeito às premissas de uniformidade.

  A Figura 3.4 apresenta um exemplo do problema de violação da propriedade de uniformidade. Nela, o processo 1 faz o *broadcast* de uma mensagem m, confirma o recebimento da mesma para logo em seguida entrar em um estado de erro. Entretanto, a falha não é enxergada pelos outros processos do grupo, que por sua vez recebem as mensagens corretamente e só notam a anomalia quando vão enviar as mensagens de confirmação.

  O desrespeito às características de uniformidade leva à necessidade de se definir dois novos conceitos:

  * **Acordo (*Agreement*)**: Se uma mensagem m for entregue a um processo **válido**, então, eventualmente a mesma deve ser entregue a todos os processos válidos pertencentes ao grupo;

    * **Ordem Total (*Total Order*)**: Se a dois processos **válidos** são entregues as mensagens *m* e *m'*, então *m* só deve ser entregue antes de *m'* se assim for feito em ambos os processos.

– **Contaminação**: O conceito de contaminação explicita um comportamento plausível de se acontecer mesmo quando as duas premissas anteriores (acordo e ordem total) são guardadas. Um processo que no futuro vai se tornar incorreto pode, anteriormente a isso, fazer o *broadcast* de mensagens naturalmente, uma vez que seu estado corrente é válido.

  A Figura 3.5 apresenta um exemplo de contaminação. No cenário descrito, todos os processos do grupo possuem uma execução correta até o momento posterior ao envio de notificação de recebimento da mensagem *m1*. A partir deste ponto, o processo 1 não mais envia a mensagem de confirmação para *m2*, fato este que o coloca como tendo uma execução incorreta à partir do momento no qual envia a mensagem de confirmação para *m3* (ferindo a ordem de entrega das mensagens). Este comportamento inválido acaba por contaminar os outros processos do grupo quando o processo 1, erroneamente envia a mensagem *m4* e não mais consegue receber mensagens de confirmação dos processos 2 e 3.

– **Ordenação**: Além dos conceitos básicos apresentados previamente, a escolha de um método de ordenação para as mensagens se faz necessário para que a especificação dos problemas de ordenação total esteja completa. [20] divide os modos de ordenação de mensagens, basicamente, em três categorias diferentes:

    * **Ordenação FIFO *First-in / First-out***: Muito aplicado em métodos de ordenação convencionais pode servir também para definir um comportamento aplicável a ambientes distribuídos onde: Para duas mensagens *m* e *m'*, *m'* só pode ser entregue antes de *m* caso seu *broadcast* também tenha ocorrido antes;

    * **Ordenação causal**: Dirigido basicamente pelo conceito de causalidade, a ordem das mensagens é definida através do acontecimento de eventos sucessivos. Inicialmente introduzido por Lamport em [34], todo este conceito é ancorado na idéia de precedência de eventos. [23] classifica o conceito geral de ordenação causal dentro de duas propriedades:

        · **Ordenação causal**: Se, através de algum evento, o envio de uma mensagem *m* precede o envio de uma mensagem *m'*, então, nenhum processo válido enviará a confirmação de *m'* sem que tenha enviado previamente a confirmação de recebimento de *m*;

        · **Ordenação local**: Se um processo correto envia a mensagem *m* e

só envia *m'* após a confirmação de recebimento de *m*, então, nenhum processo válido confirmará a entrega de *m'* antes de *m*.

Em [39] é proposto um protocolo de ordenação total utilizando os conceitos de causalidade.

* **Ordenação no envio**: Outro importante conceito a ser considerado no problema de ordenação de mensagens é o número de processos aptos a enviarem as mensagens de *broadcast* ao mesmo tempo. Basicamente, consideram-se duas possibilidades específicas: um processo ou vários. Em ambos os casos soluções previamente apresentadas podem ser aplicadas ao problema de ordenação de mensagens. Entretanto, resumidamente todas as propostas já apresentadas baseiam-se na utilização de um número de sequência para as mensagens e um processo sequenciador central que dita a ordem local (em cada processo) em que as mesmas devem ser entregues. A implementação do protocolo de ordenação total apresentado neste trabalho no capítulo 4 lança mão especificamente destas duas soluções para prover ordenação total a grupos de processos.

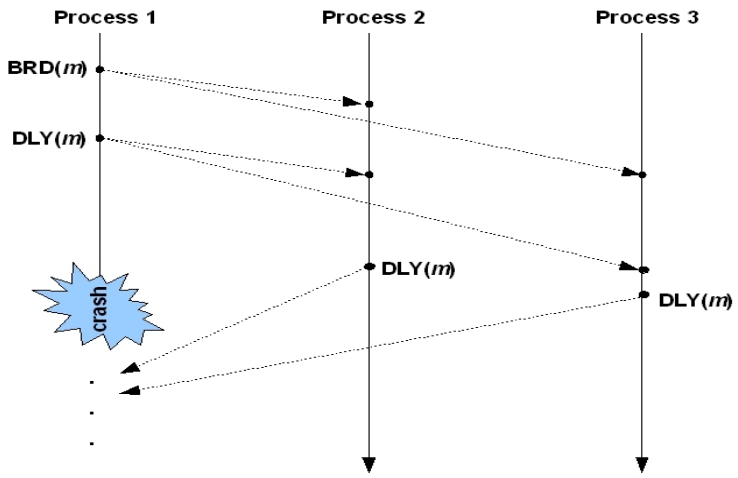


Figura 3.4: Exemplo de violação da propriedade de uniformidade

Resumidamente, de forma a faciliar o entendimento dos relacionamentos entre os conceitos previamente apresentados, toda a idéia pode ser mapeada em forma de um grafo direcionado como apresentado na Figura 5.17.

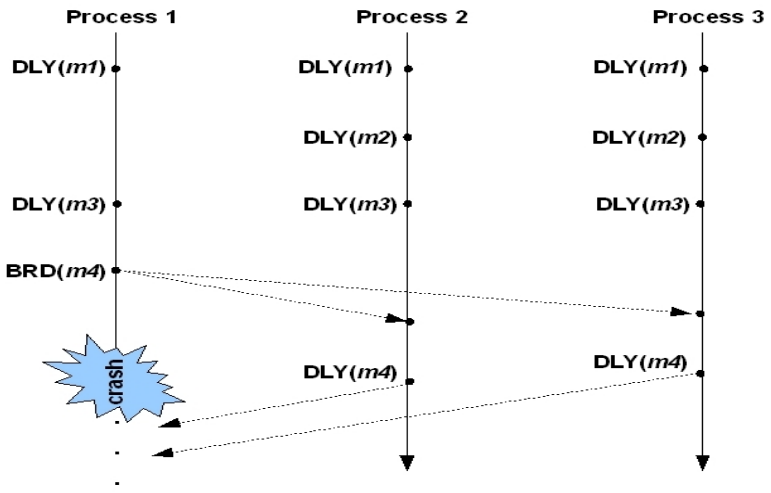


Figura 3.5: Exemplo da ocorrência de contaminação em comunicação *broadcast*

As propriedades apresentadas anteriormente não englobam todo o conjunto de conceitos aplicáveis ao problema de ordenação total, entretanto, uma especificação mais detalhada das variáveis contidas deste problema foge ao escopo do trabalho apresentado. Dessa forma, é prudente não se atentar a determinados detalhes que não foram utilizados na prática deste referido trabalho, deixando estas descrições para um documento onde um delineamento do escopo seja completamente factível, necessário e de maior utilidade.

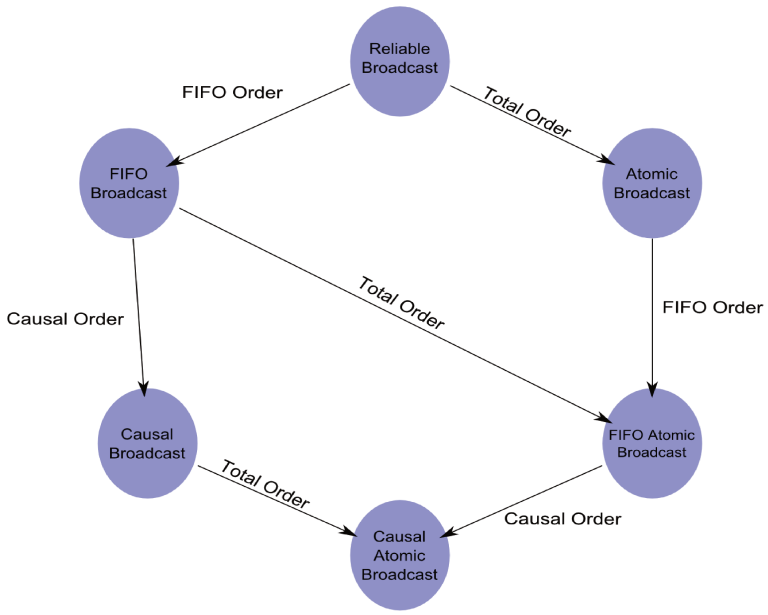


Figura 3.6: Relacionamentos entre os conceitos de difusão confiável e ordenação.

Capítulo 4

O Protocolo *Bluecast*

4.1 Introdução

Com o foco voltado principalmente para aplicações que necessitam garantir execução em tempo real, o algoritmo de difusão confiável *Bluecast* é um protocolo síncrono que garante as características necessárias a execução de uma aplicação em um ambiente de comunicação não-confiável. Para tanto, o algoritmo se aproveita de particularidades exclusivas de uma rede *Bluetooth* e oferece, além de ordem, suporte a falhas e atomicidade para grupos de processos.

4.1.1 Visão Geral

O trabalho aqui apresentado propõe a implementação de um protocolo leve para a troca de mensagens em ambientes *real time*. O conceito de "leve" vem do fato da não-utilização de mensagens de confirmação (*"ACKS"*) para cada mensagem entregue. O algoritmo, garante ainda uma resiliência **N** mínima para a perda de mensagens.

Além das características básicas da grande maioria dos algoritmos de ordenação total, o protocolo *Bluecast* leva em conta algumas características particulares da topologia das redes *Bluetooth*, detalhada no capítulo 2. O algoritmo *Bluecast* implementa um esquema implícito para a eleição de um mestre que irá gerenciar todas as conexões e tráfego dos dados na rede.

Figura 4.1 apresenta uma descrição do posicionamento do protocolo em relação às outras camadas em um ambiente de comunicação *Bluetooth*.

Em resumo, a execução do algoritmo se baseia em 4 fases distintas:

- *Fase de Eleição*;
- *Fase de Reforma*;

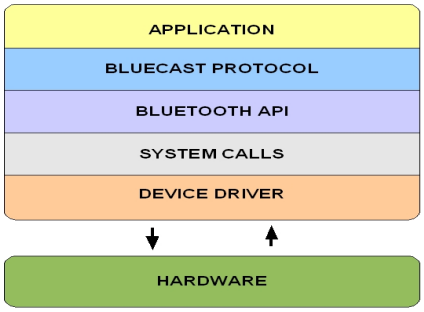


Figura 4.1: Arquitetura de camadas para o protocolo Bluecast

- *Fase de Difusão*;
- *Fase de Aceitação*.

4.1.2 Fase de Eleição

O protocolo aqui descrito tem suas bases fundamentadas no conceito de mestre/escravo. Assim sendo, é necessário que todos os nós da rede entrem em um consenso quanto a quem será o mestre. Inicialmente, todos os nós participantes desta fase têm chances iguais de serem eleitos como mestre ou escravo, isto é, nenhuma característica particular de um determinado dispositivo é levada em conta no momento da eleição.

Figura 4.2 exemplifica a configuração inicial de uma rede *Bluetooth*. A partir deste estado o protocolo inicia sua execução. Para que a eleição possa ocorrer é utilizada uma característica particular das redes *Bluetooth*.

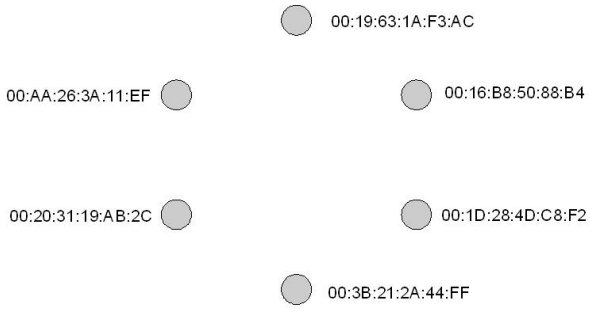


Figura 4.2: Exemplo de uma topologia inicial de uma rede *Bluetooth*

Uma peculiaridade das redes *Bluetooth* é a necessidade de, antes de se fazer uma

conexão física, se obter uma lista dos dispositivos vizinhos. Como descrito na seção 2.3.1, a fase de *inquiry* (própria da especificação *Bluetooth*) é quem, na camada física, executa tal tarefa.

Desse modo, ao conhecer o endereço físico de todos os vizinhos, um determinado nó tem condições de aplicar uma função matemática neste endereço e determinar um identificador (UID) para seu próprio endereço e para cada um de seus vizinhos.

A Figura 4.3 exemplifica a composição dos endereços de dispositivos *Bluetooth*, formados por um conjunto de 6 *bytes* sequenciais que, pela eleição do protocolo *Bluecast* são tratados como sendo um *array* de 12 posições (Figura 4.4) de modo a determinar um UID único para cada um dos nós presentes na rede. Assim sendo, à partir deste vetor de 12 posições, é possível se aplicar a seguinte equação tendo em vista o cálculo de um UID único que identificará o nó durante todo o processo de execução do protocolo:

Cálculo do UID:

$$UID_{node} = \sum_{i=1}^{12} address[i] * i \tag{4.1}$$

Figura 4.3: Exemplo do formato de endereços para dispositivos *Bluetooth*

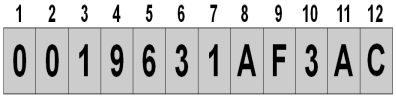


Figura 4.4: Tratamento do endereço *Bluetooth* pelo protocolo *Bluecast*

Após o cálculo do seu próprio UID, o algoritmo de eleição proposto segue através das seguintes fases:

- Cada nó pertencente à rede, utilizando-se do processo de *inquiry* provido pelo protocolo *Bluetooth* (capítulo 2) descobre, além do endereço físico, diversas informações sobre os nós presentes dentro de seu raio de alcance;
- Para cada um destes endereços descobertos, o UID é calculado aplicando-se a equação 4.1;

- Caso o nó chegue a conclusão de que possui o maior UID dentre todos os elementos da rede, este declara-se como eleito.

A Figura 4.5 exemplifica a tarefa de descoberta dos endereços físicos, assim como a geração dos UIDs à partir destes. O nó mestre será aquele que possuir o maior UID dentre todos os elementos da rede. Automaticamente, se um dado elemento descobre não ser ele o mestre, este se declara como um nó escravo (Figura 4.6). Ao final da fase de eleição todos os componentes devem saber o endereço físico, e consequentemente o UID do nó que virá a se tornar o mestre da futura rede de difusão. Dessa forma, todo o processo de eleição é feito implicitamente, sem que o protocolo utilize uma troca explícita de mensagens.

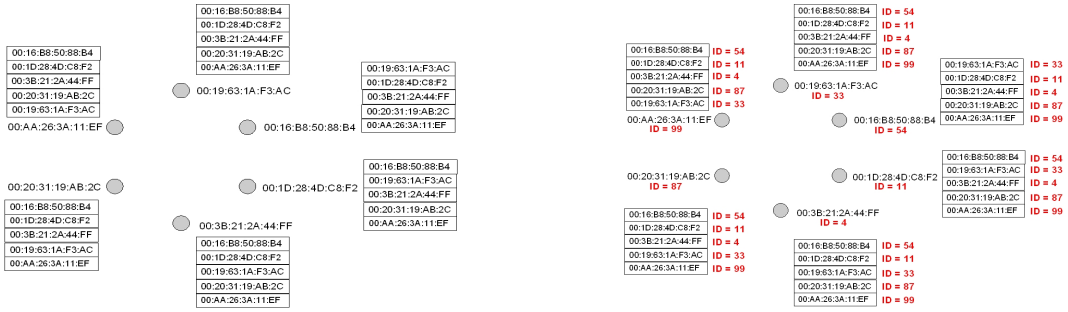


Figura 4.5: Descoberta dos endereços e geração dos UIDs

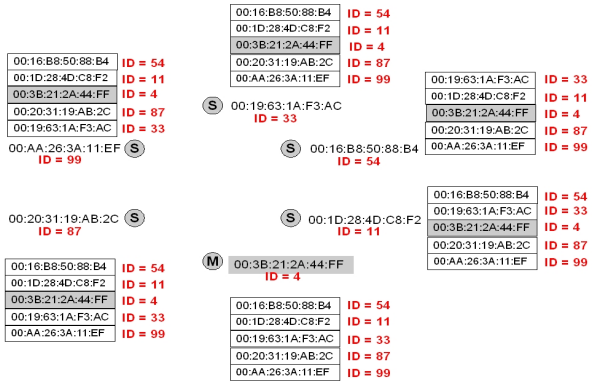


Figura 4.6: Eleição do nó Mestre

O Código 4.1.1 apresenta uma descrição de alto-nível da implementação para a fase de eleição do protocolo de comunicação *Bluecast*.

```
1   INICIO ElectionPhase()
2
3           INTEIRO localId, idEleito;
4
5           localId = ObtemEndereco(enderecoBluetooth);
6           idEleito = localId;
7
8           PARA i = 0 ATE numero_dispositivos_remotos
9                   SE idDispositivoRemoto[i] < idEleito
10                          idEleito = idDispositivoRemoto[i]
11                  FIMSE
12          FIMPARA
13  FIM
```

Código 4.1.1: *Fase de eleição*

4.1.3 Fase de Reforma

Após a determinação do nó mestre, o protocolo passa automaticamente à fase de reforma. Esta é utilizada para que a rede seja explicitamente formada.

O ambiente *Bluetooth* possui uma característica curiosa em relação a outros protocolos de rede existentes. Na formação de uma rede do tipo *Piconet*, não são os nós escravos que se conectam ao mestre. Mas, este é quem cumpre a tarefa de buscar e conectar cada nó escravo. Dessa forma, considerando-se os resultados da etapa anterior, as seguintes ações são tomadas na fase de reforma:

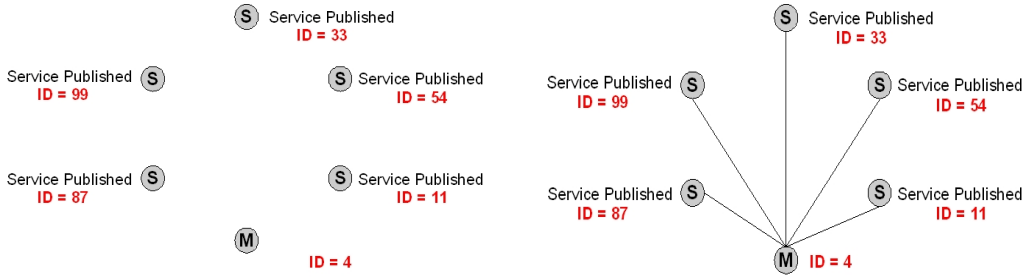


Figura 4.7: Publicação do serviço e formação da rede

- Se ao final da fase de eleição, o nó chega à conclusão de que atuará como mestre, o mesmo deve:
  - Para cada nó conhecido (e participante da eleição), enviar uma solicitação de conexão;
  - Uma vez que todas as conexões foram aceitas (Figura 4.7), o protocolo segue para a próxima fase.

- Se ao final da fase de eleição, o nó chega à conclusão de que atuará como escravo, ele deve:
  - Publicar o serviço que será utilizado para a comunicação (Figura 4.7);
  - Aguardar que receba a solicitação de conexão vinda do nó mestre;
  - Uma vez recebida (e aceita) a solicitação de conexão, o protocolo segue para a próxima fase.

O código apresentado em 4.1.2 apresenta uma descrição de alto-nível da implementação para a fase de reforma do protocolo de comunicação *Bluecast*.

```
1  INICIO ReformPhase()
2
3          INTEIRO localId, idEleito;
4
5          SE localId == idEleito
6                  PARA i = 0 ATE numero_dispositivos_remotos
7                          endereco = obtemEnderecoServico(dispositivoRemoto[i])
8                          conectaServico(endereco)
9                  FIMPARA
10         FIMSE
11
12         SE localId != idEleito
13                 publicaServico()
14                 aguardaConexaoMestre()
15         FIMSE
16
17 FIM
```

Código 4.1.2: *Fase de reforma*

4.1.4 Fase de Difusão

Todo e qualquer algoritmo de ordenação total em redes *broadcast* deve ter em seu foco principal a funcionalidade de troca de mensagens. É dentro desta fase que o protocolo deve garantir que todas as informações enviadas pelos integrantes sejam sempre entregues na ordem de envio. E, no caso de ordem total, que as mensagens sejam entregues a todos os integrantes na mesma ordem.

Devido ao aumento da preocupação com a questão da interatividade, muitos dos protocolos de ordenação total encontrados hoje na literatura são integrados a ambientes de comunicação com características de tempo real (descritas previamente no capítulo 3). Dessa forma, garantir que as mensagens sejam entregues dentro de um limiar de tempo (N) caracteriza um protocolo como sendo tempo real. Para o protocolo aqui proposto, todos os pontos relevantes (garantia de entrega, limiares de tempo real, etc) descritos anteriormente são levados em conta nesta fase de difusão.

```
1  INICIO BroadcastPhase()
2
3          INTEIRO localId, idEleito, MESTREID, GID;
4          MESSAGE msg;
5
6          msg = constroiBRDMsg()
7
8          SE localId == idEleito
9                  adicionaFilaRecebidos(msg)
10
11                 MESSAGE msg_t = retiraFilaRecebidos()
12
13                 msg_t.id = GID
14                 GID = GID + 1
15
16                 enviaMensagem(msg_t)
17                 adicionaFilaACK(msg_t)
18         FIMSE
19
20         SE localId != idEleito
21                 enviaMenssagem(msg, MESTREID)
22                 adicionaFilaEnviados(msg)
23                 ENQUANTO !recebeACK(msg)
24                         ack = aguardaACK(MESTREID)
25                 FIMENQUANTO
26
27
28                 SE ack.id != GID
29                         PARA i = GID ATE (ack.id-1)
30                                 msgPerdida = solicitaMensagem(i)
31                                 adicionaFilaACK(msgPerdida)
32                                 GID = GID + 1
33                         FIMPARA
34                 FIMSE
35
36                 adicionaFilaACK(retiraFilaEnviados(msg))
37         FIMSE
38
39 FIM
```

Código 4.1.3: *Fase de Broadcast*

Após passar pela fase de reforma, todos os nós da rede estão aptos a iniciar as trocas de mensagens (descrita em 4.1.3). Para tanto, determinados passos devem ser seguidos e algumas ações devem ser tomadas.

Execução correta do protocolo

Cada elemento presente na rede de *broadcast* que desejar enviar uma mensagem, deve:

- Construir uma mensagem do tipo BRD (*broadcast*);
- Enviá-la ao mestre, eleito na primeira fase do algoritmo (Figura 4.8);
- Esperar pelo ACK implícito do mestre. Tal mensagem de ACK carrega:

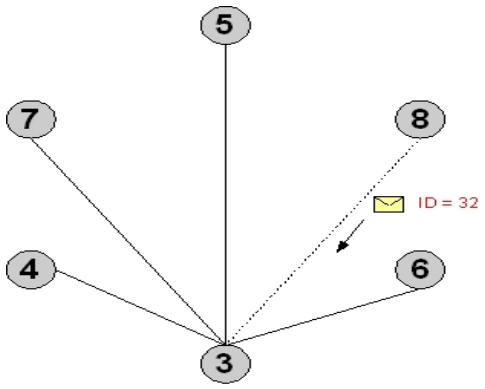


Figura 4.8: Envio da mensagem Escravo / Mestre

– O ACK da mensagem enviada anteriormente pelo escravo, através de uma outra mensagem do tipo BRD;

– Um inteiro identificador, indicando que a mensagem foi sequenciada globalmente pelo mestre;

Ao nó mestre é delegado o dever de sequenciar as mensagens recebidas antes de encaminhá-las aos nós escravos. Ao enviar tais mensagens sequenciadas aos nós Escravos, este também coloca uma cópia da mesma em sua fila de mensagens recebidas para posterior tratamento. Dessa forma, ao receber um BRD contendo um identificador global (GID), cada integrante da rede assume que tal mensagem automaticamente é a confirmação de entrega da mensagem anterior (GID - 1) e esta pode ser encaminhada a aplicação.

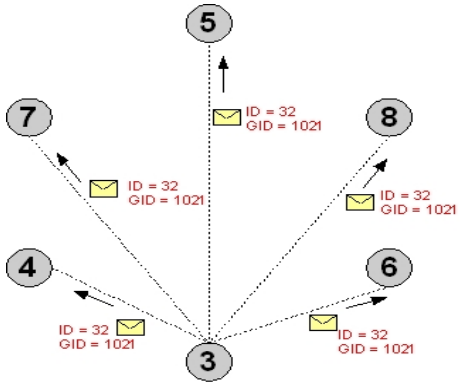


Figura 4.9: Mestre: Sequenciamento e envio das mensagens

Caso o mestre necessite enviar uma mensagem, este, por questões de coerência deve colocar esta mensagem em sua própria fila de chegada. Quando do seu processamento no próprio nó, o mestre a tratará como sendo uma mensagem vinda de algum dos escravos, literalmente atuando como outro elemento escravo. Tal atitude, apesar de fazer necessária uma ação aparentemente descartável, faz com que se mantenha justa a ordem de chegada das mensagens ao processo mestre. Além disso, como a mensagem não é fisicamente enviada, acaba não gerando tráfego adicional na rede.

Parâmetros de execução

Em qualquer protocolo síncrono é necessário que se definam limites/valores para determinadas variáveis do sistema. Assim também acontece com o protocolo aqui apresentado.

- **Resiliência:** Capacidade que o sistema possui de suportar determinados tipos falhas. O protocolo *Bluecast* utiliza tal parâmetro para indicar o número de mensagens sequenciais que um nó pode não receber e que se necessárias podem ser recuperadas pelo processo incorreto.

- **Tempo limite:** No caso de aplicações baseadas principalmente no tráfego em rajadas, os intervalos de comunicação podem ser exageradamente grandes. Logo, um intervalo de comunicação muito longo pode ser confundido com uma queda do nó. O parâmetro **tempo limite** define o intervalo de tempo máximo em que um determinado processo pode ficar sem enviar mensagem alguma. Após tal intervalo considera-se que o nó caiu. De modo a se evitar tal problema, o protocolo *Bluecast* envia uma mensagem sinalizadora, indicando que ainda está ativo.

- **Mensagem de sinal (*Heartbeat*):** Visa informar aos outros processos que este ainda participa do grupo de *broadcast*. Este tipo de mensagem só é utilizado como uma sinalização quando não existem mensagens de BRD a serem enviadas pelo nó.

Condições e tratamento de falhas

De modo a tornar o protocolo robusto o suficiente para o ambiente aplicado, determinados tipos de falhas devem ser suportados de modo que o grupo de processos se mantenha e ainda, garanta a entrega de todas as mensagens.

- **Perdas de mensagens:** Em determinados cenários específicos, um nó pode deixar de receber alguma mensagem (falhas no próprio nó ou ainda interferências no ambiente de comunicação). Para tanto, cada nó deve manter um indicador da próxima

mensagem esperada (NXT_GID). Ao receber uma mensagem e notar que a mesma não é a esperada ($NXT\_GID \neq GID$), o nó deve solicitar ao mestre a retransmissão da(s) mesma(s).

- **Saída voluntária de nós:** Como descrito na subseção 4.1.4, antes de pedir a retransmissão de determinado conjunto de mensagens, o nó deve levar em conta se $GID - NXT\_GID <= N$. Isto é, se o número de mensagens perdidas se encontra dentro do limite suportado pelo sistema. Caso esta assertiva não seja verdadeira, o nó deve se retirar voluntariamente do grupo de processos enviando ao mestre uma mensagem de saída. Este, por sua vez, finaliza a conexão com o elemento (e notifica os outros processos desta saída) sem que o protocolo tenha necessidade de voltar à fase de reforma.

- **Falha de nós:** O item anterior descreve a saída voluntária de nós. Entretanto, há de se considerar o caso onde alguma falha cause a queda do processo. Para uma correta detecção de queda de processos, o protocolo aqui descrito se baseia fortemente no conceito de mensagens de sinal (subseção 4.1.4). Dessa forma, neste cenário devem ser levados em conta dois casos:

  - **Processo escravo**: O processo mestre nota que um escravo caiu se, após um limiar de tempo T (*timeout*), o mesmo não recebe nenhuma mensagem do escravo. Neste caso, o processo mestre, encerra sua conexão, e notifica todos os outros componentes do grupo sobre a saída do mesmo.

  - **Processo mestre**: A queda de um processo mestre é, logicamente, notada pelos processos escravos levando-se em conta o mesmo princípio do item anterior. Os escravos não recebem mensagens do mestre dentro de um limite de tempo T (*timeout*). Entretanto, neste caso é necessário que, após a detecção da queda do mestre o protocolo volte à fase de eleição para que um novo mestre seja escolhido. Após esta nova eleição, os nós devem ainda, entrar em um consenso quanto ao último *GID* recebido por todos os processos.

4.1.5 Fase de Aceitação

As três fases anteriores são suficientes para garantir todas as premissas de um protocolo de ordenação total. Entretanto, é necessário que se delineie um modo para que processos externos (fora do grupo) possam, em determinado momento, no futuro, se juntar ao mesmo.

A forma de troca de mensagens utilizada pelo ambiente *Bluetooth* é totalmente orientada a conexões. Enquanto estas durarem, um nó está apto a enviar e receber dados a outra parte. Esta característica traz certas dificuldades a tarefa de inclusão de novos processos a um grupo já formado. Isto porque, faz-se necessário que o nó mestre (centralizador de todas as conexões) tenha conhecimento do desejo de um determinado processo em se tornar parte da rede de difusão. Para que isto seja possível é necessário se possuir duas visões distintas da execução do protocolo neste dado momento:

- **Escravo**: Todo e qualquer novo nó que deseja ser incluso no grupo de *broadcast*, necessita:
  - Fazer um *inquiry* e descobrir seus vizinhos;
  - Nestes, fazer uma busca específica pelo serviço do protocolo *Bluecast*;
  - Caso o serviço seja encontrado:
    - O novo elemento chega a conclusão que já existe um grupo de processos formado;
    - Publica um serviço de mesmo ID; e
    - Espera que o mestre do grupo existente faça sua inclusão.
  - Caso o serviço não seja encontrado:
    - O novo elemento chega a conclusão que ainda não existe um grupo de processos formado;
    - O mesmo utiliza a lista de nós vizinhos para executar a fase de eleição; e
    - O protocolo segue como descrito na subseção 4.1.2

- **Mestre**: O protocolo *Bluecast* separa a fase de difusão em dois tipos distintos. Nesta, ambas são executadas, entretanto, o tempo de execução de cada uma pode variar conforme o foco da aplicação:
  - Na primeira delas ocorre a troca de mensagens propriamente dita;
  - A segunda parte engloba a inclusão de novos elementos, onde:
    - O processo mestre executa um *inquiry* visando descobrir a existência de novos processos. Isto se torna possível através da declaração explícita do novo processo em integrar o grupo ao publicar o serviço associado ao algoritmo *Bluecast*;
    - Caso um novo processo seja encontrado, é função do mestre conectá-lo a rede já formada e continuar a fase de difusão juntamente com os processos remanescentes.

4.1.6 Conclusão

A principal motivação do protocolo aqui apresentado está diretamente relacionada com o ambiente em que o mesmo é contextualizado. Devido a escassez de propostas específicas para um ambiente com alto nível de restrições, o algoritmo demonstrado neste capítulo preenche uma lacuna ainda pouco explorada na literatura atual. Esta, até agora apenas explorava os dois assuntos separadamente sem que houvesse um relacionamento direto e efetivo entre ambientes de tempo real e métodos de ordenação de mensagens.

Quanto a aplicabilidade do mesmo, é inevitável não notar a atual tendência do mercado de dispositivos móveis, que, mais do que nunca presa pela interatividade, mobilidade e imersão do usuário em conteúdos contextualizados com sua realidade e ambiente.

O sistema aqui apresentado atinge seu objetivo ao minimizar o tempo de interação entre processos componentes de uma rede e consequentemente maximizar a interativade entre os mesmos, uma vez que possuem capacidade limitada de comunicação e processamento. Os resultados apresentados no capítulo 5 pretendem mensurar o ganho na utilização deste protocolo em comparação com outros algoritmos previamente apresentados na literatura.

Capítulo 5

Protocolos de Ordenação: Análises e Comparações

Ao se propor um estudo e comparação de diversos tipos de trabalhos, não basta que tal tarefa seja executada analiticamente. Dados empíricos e práticos são extremamente importantes para validar propostas e soluções, assim como desmistificar conceitos errôneos formados a partir apenas de teorias. Para tanto, o capítulo corrente apresenta não só tal parte analítica, como também resultados práticos obtidos da experimentação em ambientes e dispositivos reais.

Atualmente são muitos os trabalhos com foco em algoritmos e protocolos que visam solucionar ou melhorar problemas típicos da área de computação distribuída, como: atomicidade, ordem, ordem total, etc. Alguns destes trabalhos mais relevantes são citados em 3.3, entretanto nem todas as referências bibliográficas da área podem ser consideradas aptas a serem aplicadas ao ambiente no qual se baseou este trabalho. Para tanto, além de uma pesquisa dos trabalhos previamente existentes fez-se necessária uma catalogação dos protocolos possíveis de serem implementados e/ou importados para o ambiente de tempo real em redes *Bluetooth*.

5.1 Protocolos Considerados

A corrente seção apresenta uma lista de protocolos/algoritmos considerados para análise e implementação, assim como uma discussão sobre a adequabilidade ao ambiente apresentado através dos capítulos 2 e 3.

- *Amoeba*: Os protocolos que garantem ordenação total/atômica normalmente podem ser agrupados através de diferentes classes. Duas dessas se baseiam no nível de mobilidade dos sequenciadores de mensagens. A proposta apresentada por [28] - *Amoeba* - se encaixa no grupo de sequenciadores fixos, onde, durante toda a execução do algoritmo o processo responsável pelo sequenciamento das mensagens não é alterado. As principais características apresentadas pelo algoritmo são:

  - Conceitos básicos: Ordenação Total atingida através da utilização em conjunto dos conceitos de sequenciamento de mensagens e *ACKs* negativos;
  - Tolerância a falhas: Usuário pode selecionar o nível desejado de tolerância a falhas.

  Por assumir que as mensagens transitam num meio de comunicação não confiável, o protocolo implementa um esquema de recuperação de erros. Além disso, agrega as funcionalidades de grupos de processos, confiabilidade e ordenação total.

  - **Avaliação:** Um dos mais completos algoritmos encontrados na literatura, esta proposta tenta tratar diversos fatores presentes na comunicação de grupos de processos (como controle do fluxo de mensagens, e minimização dos custos de comunicação). Tais fatores fazem que com o protocolo ganhe em complexidade e necessidade de processamento, consequentemente, tornando-o inapto a ser aplicado no ambiente descrito por este trabalho. Além disso, outra característica do protocolo é a entrega de mensagens antes da estabilização do grupo de processos. Isto faz com que a propriedade de "uniformidade" possa não ser respeitada em determinados casos, violando assim o conceito de ordenação total.

- *Lamport*: Em [34], Lamport descreve um protocolo baseado no histórico de comunicação do grupo de processos. O algoritmo proposto parte de uma pré-determinada ordem causal para chegar à ordenação total. Para tanto, a utilização de relógios lógicos síncronos é imprescindível.

  - **Avaliação:** Para o ambiente de comunicação de tempo real, partir de uma ordenação causal pode ser uma estratégia interessante no desenvolvimento de protocolos de ordenação total. Entretanto, a utilização de um relógio lógico global obrigatoriamente síncrono torna impossível a aplicação do mesmo neste ambiente real. Além disso, a proposta apresentada por Lamport não prevê tolerância a falhas, fato esse que acaba por definir uma "linha de corte" na seleção dos algoritmos que melhor se aplicam ao ambiente descrito.

- *Train*: [17] apresenta um algoritmo de ordenação atômica assíncrono. O conceito básico do mesmo é uma analogia a um trem de carga que transporta as mensagens de difusão entre os integrantes do ambiente. Analogamente, quando um processo recebe a passagem do trem (com as mensagens), entrega a este a confirmação das que foram recebidas enquanto deposita ao mesmo as suas próprias mensagens para que possam ser levadas aos outros integrantes do grupo. Dessa forma, o trem faz o papel de um sequenciador garantindo que as mensagens sejam entregues em ordem.

  - **Avaliação:** Numa análise inicial é possível se notar uma deficiência do mesmo para que possa ser aplicado ao ambiente de tempo real. Quanto maior o número de processos na rede, maior será o número de mensagens circulando pelo trem ao mesmo tempo. Fato que, com certeza, devido às características de escassez de recursos do ambiente, pode impossibilitar a utilização do mesmo ao se gerar uma sobrecarga muito grande e desnecessária na comunicação entre os integrantes do grupo.

- *Pinwheel*: Do mesmo autor que o algoritmo anterior, o protocolo *Pinwheel* [19] propõe a implementação do conceito de um sequenciador dinâmico que é identificado através da posse (ou não) de um *token*. O protocolo proposto assume que tal *token* vai ser passado entre todos os componentes da rede com uma velocidade variante - de acordo com o volume de mensagens sendo trocadas pelo grupo. E, assim como em outros algoritmos baseados em passagem de *token*, neste também, a quem tem a posse do mesmo é delegada a função de sequenciador das mensagens num dado momento.

  - **Avaliação:** Para uma correta execução, o protocolo *Pinwheel* considera que pelo menos a maioria dos processos se mantém conectados durante todo o tempo de execução. A falta de suporte a tratamento e correção de erros deste tipo dificulta a aplicação do algoritmo em ambientes de tempo real. Ainda, tal fato se torna mais agravante pelas características do enlace de comunicação sem fio onde a ocorrência de erros é ainda maior. Dessa forma, apesar de prover ordenação total uniforme, a aplicação do mesmo a este trabalho não foi priorizada.

- *Chang e Maxemchuk*: Uma das mais difundidas propostas para algoritmos de ordenação total, a proposta de Chang & Maxemchuk em [12], é até hoje um dos protocolos de comunicação confiável mais estudado e implementado. O algoritmo desenvolve a noção de um sequenciador móvel - que é trocado a cada N mensagens enviadas. Além disso, a tolerância a falhas é garantida através da utilização de *ACKs* para as mensagens (que indiretamente também sinalizam que um processo não falhou e permanece no grupo de comunicação). A partir de tais funcionalidades o protocolo consegue prover suporte a:

  - Falhas nos processos;
  - Falhas no envio/recebimento de mensagens; e
  - Ordenação total.

  - **Avaliação:** Apesar de possuir um grande *overhead* na parte de comunicação, o protocolo é consistente e suporta tanto falhas de processos quanto de entrega de mensagens (primordiais para o ambiente de comunicação de tempo real). Além disso, o mesmo implementa um esquema de formação do grupo de processos chamado de fase de reforma, que visa manter um grupo mínimo de processos se comunicando. Por tais características este protocolo foi utilizado neste trabalho como métrica de comparação com outras propostas. Uma descrição mais detalhada do mesmo pode ser encontrada no Apêndice A.

- *RTCAST*: O trabalho apresentado em [5] é um dos poucos encontrados na literatura atual que propõe um protocolo de difusão confiável exclusivamente para o ambiente de comunicação de tempo real. No trabalho são descritos esquemas de suporte a falhas de canais e processos. Além disso, as seguintes características podem ser identificadas no mesmo:

  - Baseado no conceito de passagem de *token*;
  - Entregas de mensagens em no máximo uma rodada após o envio das mesmas;
  - Garante ordem e atomicidade.

  O que diferencia a proposta das anteriormente apresentadas é o fato do mesmo, segundo os autores, ser um protocolo "leve". Propriedade esta alcançada através da não-utilização de mensagens de confirmação para cada mensagem de difusão enviada.

  - **Avaliação:** Apesar de apresentar uma proposta inovadora, a implementação do algoritmo esbarra em dois pontos principais:
    * Necessidade de um relógio físico sincronizado globalmente entre os processos, de modo que os mesmos tenham a capacidade de notar o *timeout* no envio/recebimento das mensagens, assim como na passagem do *token*;
    * O protocolo não propõe um algoritmo de eleição. Dessa forma, o *token* já inicia com determinado processo. Este fato dificulta a aplicação do protocolo num ambiente *Bluetooth*, uma vez que os processos deste tipo de grupo não se conhecem previamente, impossibilitando a escolha de um líder inicial sem que se chegue a um problema de consenso distribuído.

5.2 Metodologia

A partir desta análise inicial, dois diferentes algoritmos foram implementados e os desempenhos testados a fim de determinar o mais propício para ser utilizado junto a aplicações com características de tempo real.

As outras propostas foram descartadas devido ao alto custo de implementação e adaptação ao ambiente proposto por este trabalho. Tal decisão foi tomada tendo-se como base principalmente o fato de que analiticamente, analisando-se as características de cada trabalho é possível se concluir a inviabilidade de aplicação dos mesmos .

- **Protocolo de Chang & Maxemchuk** [12](Apêndice A): Como citado anteriormente baseia-se no conceito de passagem de *token* e sequenciador dinâmico, onde a cada mensagem de *broadcast* enviada, uma outra de confirmação deve segui-la;

- **Protocolo *Bluecast***: Descrito no capítulo 4 maximiza as idéias apresentadas em outros protocolos propostos anteriormente. Além disso, inclui um mecanismo de eleição particularmente desenvolvido para ambientes de tempo real que utilizem o protocolo de comunicação *Bluetooth*;

- **Protocolo de melhor esforço**: Caso ótimo para ambientes *Bluetooth*, entretanto não garante ordem tampouco é tolerante falhas.

Para cada um dos protocolos acima diferentes cenários foram executados, assim como algumas variações destes. As principais variáveis consideradas foram:

- Número de processos na rede: Os testes executados consideraram grupos de processos de tamanho 2, 3 e 4 em uma rede *Bluetooth* com características de tempo real;

- Grau de resiliência: Atualmente grande parte dos protocolos encontrados na literatura tem como característica o suporte a perda de mensagens. Para tanto, estes algoritmos quase em sua totalidade implementam o conceito de resiliência. Neste trabalho, para efeito de testes foram utilizados três diferentes níveis de resiliência para todos os protocolos executados: 1, 5 e 10 mensagens.

- Diferentes tipos de aplicação: Para se checar o comportamento dos protocolos quando executando com diferentes tipos de requisições, dois tipos de aplicações foram testados:

  - Jogo: Simula uma partida de tênis com até quatro jogadores. Objetivo principal é testar os algoritmos quando executando uma aplicação com requisitos típicos do ambiente de tempo real;
  - Gerador de números aleatórios: Através da mesma é possível se observar os protocolos quando estressados de diversas maneiras quanto ao número de mensagens difundidas na rede.

- Tipo de tráfego: Para os cenários foram considerados também dois tipos de distintos de tráfego na rede:
  - Com picos: Envio de determinada quantidade de dados após um tempo definido aleatoriamente;
  - Constante: Envio constante de dados com mínimo intervalo de tempo.

5.3 Arquitetura comum, métodos de aferição e resultados

Uma característica comum a implementação de cada um dos protocolos é a separação da arquitetura em diferentes camadas. Isso faz com que se possa obter dados mais concisos e precisos, uma vez que a arquitetura geral dos mesmos não varia.

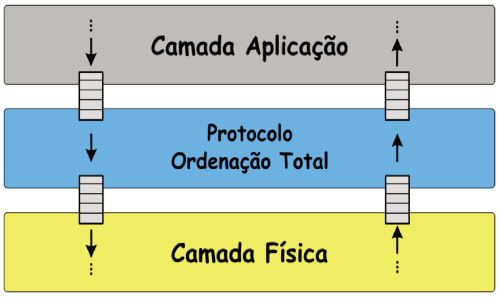


Figura 5.1: Arquitetura comum a todos os protocolos considerados

A Figura 5.1 descreve a arquitetura geral e comum dos protocolos de ordenação total considerados por este trabalho. Ambos são, basicamente, divididos em três diferentes camadas:

- **Camada física**: Interface de envio e recebimento de dados (mensagens). Para tanto, possui capacidade de acesso direto ao enlace físico do meio de comunicação;
- **Protocolo de ordenação**: Camada intermediária que age principalmente no sequenciamento das mensagens recebidas pela interface inferior (camada física). Além de garantir a ordem das mensagens, também é responsável pela atomicidade na entrega das mesmas a camada superior;
- **Camada de aplicação**: Não faz explicitamente parte do protocolo, entretanto, de acordo com o modelo de implementação utilizado esta é considerada, pois funciona como consumidora das mensagens ordenadas pela camada inferior.

Utilizando-se os dois tipos de aplicações descritas anteriormente, o comportamento de cada um dos algoritmos foi observado sob diferentes aspectos e pontos.

5.3.1 Ferramentas e métodos de obtenção dos resultados

O trabalho aqui apresentado tem como base principal os conceitos de mobilidade e ambientes de comunicação de tempo real. Tais fatores tornam impossível a não contextualização do mesmo com dispositivos de pouca capacidade de processamento e, consequentemente *softwares* embarcados. Esta última área da computação acaba por trazer ao trabalho uma nova classe de desafios por apresentar um ambiente de desenvolvimento mais complexo que aquele encontrado em ambientes computacionais mais robustos.

Para suplantar as limitações impostas pelo ambiente e obter os resultados necessários às conclusões, foram desenvolvidas (e reutilizadas) algumas ferramentas e métodos essenciais que, resumidamente podem ser divididas em três fases distintas:

- **Prototipação**:
  Baseada principalmente em uma ferramenta já existente [1], de uso livre para fins não comerciais, e escrita na linguagem Java, esta fase foi como uma pré-experimentação do trabalho desenvolvido. Uma das razões para a utilização desta aplicação na fase inicial foi a facilidade de se emular diversos dispositivos *Bluetooth* de forma virtual e bem aproximada do real, sem que para isso houvesse a necessidade de se operará-los fisicamente.

  Entretanto, a utilização da aplicação como forma de obtenção dos dados não é válida, uma vez que a simulação provida não pode ser tratada como um ambiente real, o que consequentemente impossibilita a obtenção de dados fidedignos. A Figura 5.2 apresenta o aspecto básico da interface de usuário da ferramenta utilizada. Além da emulação de dispositivos, é possível separá-los em classes distintas de acordo com seu poder computacional, por exemplo.

- **Testes**:
  A partir desta fase foram feitos testes com módulos básicos dos protocolos escolhidos. Tais testes foram necessários para se delimitar a classe de dispositivos móveis que melhor se adaptaria a execução dos protocolos de ordenação. Duas razões principais motivaram a utilização de uma fase de testes:

  – Depuração: Como citado previamente, quando se trabalha com aplicações embarcadas uma das grandes preocupações é saber o estado atual do dispositivo, de forma a encontrar e solucionar possíveis problemas. Como apresentado na Figura 5.3, toda a fase de remoção de erros/problemas contou com três diferentes aproximações:

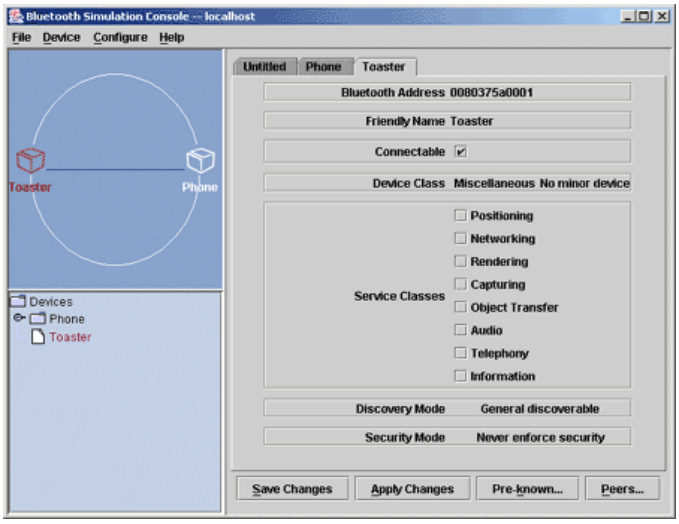


Figura 5.2: Interface apresentada pela ferramenta Impronto Simulator [1].

* Uma primeira rodada de remoção de *bugs* foi feita ainda no simulador Figura 5.3(a);
* Problemas não considerados na primeira fase ocorreram nesta. A solução passou pela implementação de uma nova arquitetura que agregava um novo membro (não participante do processo de difusão e utilizado como uma espécie de servidor de *debug*) àqueles já integrantes do grupo de processos, que era responsável por receber uma cópia de cada mensagem trocada pelos processos válidos e, ao final de cada execução, interpretar as mesmas e gerar os resultados ou relatório de erros.
  Apesar de válida sob alguns aspectos, tal aproximação se mostrou ineficiente por apresentar problemas de sincronização na recepção das mensagens vindas dos processos como apresentado na Figura 5.4.
* O modelo final utilizado para a recuperação dos dados e correção de erros foi básico. Cada um dos processos grava um histórico de todas as mensagens que trafegaram por suas interfaces de envio, recebimento e aplicação juntamente com uma marca de tempo local. Dessa forma, apesar de não ser possível se determinar um comportamento global para o grupo num dado momento, evitam-se os problemas de sincronização Figura 5.3(c).

– *Limitações dos dispositivos:*
Através destas duas primeiras fases foi possível se comprovar a limitação de alguns dispositivos e a necessidade de se considerar determinadas características

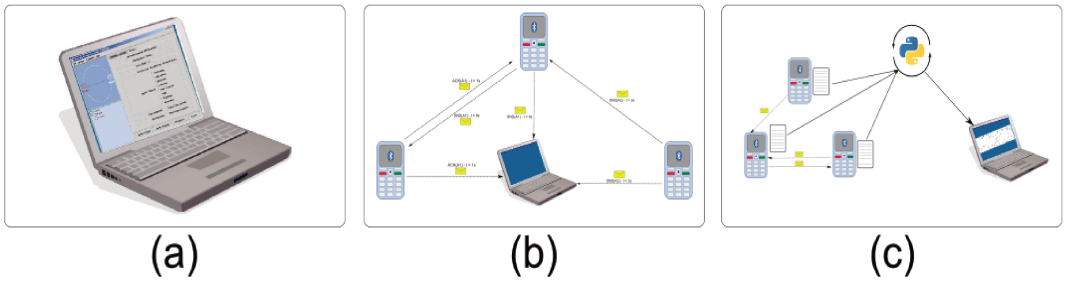


Figura 5.3: Diferentes arquiteturas para *log* de mensagens

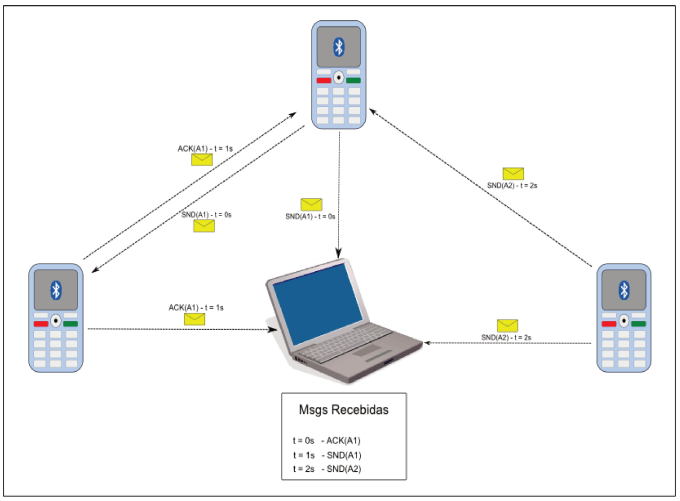


Figura 5.4: Exemplo de falha na sincronização de mensagens.

do *hardware* presente em outros. Algumas destas afetam principalmente o grau de conectividade de um nó ou ainda a capacidade de processamento do mesmo. Apesar de todos os dispositivos suportarem a especificação do padrão *Blueto-oth*, nem todos disponibilizam todo o conjunto de funcionalidades previstas pela mesma.

- Execução e obtenção dos resultados:
  O modelo descrito na Figura 5.3(c) foi o escolhido para ser utilizado na recuperação e análise dos dados vindos dos dispositivos. O processo conta basicamente com as seguintes fases:

  – Execução do protocolo (com os devidos parâmetros) nos dispositivos componentes da rede;

  - Os aparelhos, por sua vez, gravam todas as informações de suas interfaces de entrada, saída e aplicação em arquivos dentro do próprio dispositivo;
  - Após finalizada a execução, tais arquivos são recuperados, processados por um interpretador e os resultados analisados.

Exceto pelo simulador apresentado em [1], todas as outras ferramentas foram desenvolvidas para uso diretamente no trabalho aqui apresentado.

5.3.2 Tempo de entrega das mensagens

Cada um dos protocolos analisados (também aquele que visa o melhor esforço) implementa, mesmo que intrinsecamente, alguma estrutura que permite retratar a funcionalidade de resiliência desejada. Em todos eles, esta mesma estrutura de dados foi diretamente mapeada em uma fila de mensagens, onde, para que ocorra a entrega de uma dada mensagem, a mesma deve percorrer todas as posições desta fila.

Dessa forma, é correto se inferir que o valor da variável resiliência influencia diretamente no tempo de entrega dos dados, uma vez que os mesmos devem percorrer todo o caminho desde a camada física até a camada de aplicação. No trabalho aqui apresentado, o tempo considerado é dado por:

$T_{msg}(N) = T_{Ent}(N) - T_{Rec}(N)$, onde:

$T_{msg}(N)$: Tempo de entrega da mensagem (N);
$T_{Ent}(N)$: Tempo quando mensagem (N) é repassada do protocolo de ordenação para a camada de aplicação;
$T_{Rec}(N)$: Tempo quando mensagem (N) é recebida pela interface física de comunicação.

As seguintes considerações foram feitas quando da aferição dos resultados:

- Os cenários de teste foram separados segundo o número de nós e tipo de aplicações;
- Todos os valores para o tempo de entrega das mensagens são dados em milissegundos;
- 10 execuções para cada um dos cenários considerados: Apesar de aparentemente pequeno, este foi considerado o valor ótimo devido ao grande tempo (10 a 15 minutos) necessário para a execução de cada uma das rodadas de teste;
- Em cada uma das execuções, os dados foram obtidos após o envio de 1300 mensagens pela rede de difusão.

Além dos pontos acima, uma associação das aplicações testadas pode ser feita diretamente com dois tipos de tráfego de dados:

- **Tráfego com picos**: Os dados aqui mensurados relacionados a este tipo de tráfego não foram diretamente contextualizados a variáveis comumente levadas em conta quando analisando tráfego em rajadas, como:
  - Nível de distribuição das rajadas durante determinado período de tempo;
  - Duração do envido de dados durante um ciclo de rajada; e
  - Pico ou densidade dos dados enviados.

  Os dados aqui obtidos são relacionados a este tipo de tráfego por possuirem características facilmente associáveis a tal conceito. A utilização de uma aplicação baseada na interação dos usuários (Jogo) - que gera diferentes quantidade de dados num intervalo de tempo aleatório - acaba por indiretamente alcançar resultados e comportamentos próximos àqueles observados quando num ambiente de laboratório especificamente preparado para este tipo de avaliação.
- **Tráfego contínuo**: Pode ser claramente mapeado através da aplicação para geração de números aleatórios utilizada uma vez que a mesma envia informações continuamente com um intervalo de tempo bem definido.

Tráfego com picos

Os dados apresentados pelas tabelas subsequentes relacionam o valor da resiliência (1, 5 ou 10) com as diferentes execuções de um mesmo algoritmo. Dessa forma, a primeira coluna representa o valor da resiliência, enquanto que as outras colunas representam o tempo médio de entrega das mensagens em cada uma das dez execuções do protocolo.

- **Cenário 1**:
  - Aplicação: Jogo;
  - Protocolo: *Bluecast*;
  - Número de processos na rede: 2.

Analisando-se os dados apresentados nas Tabelas 5.1 e 5.2, é possível se observar que devido as características do tipo de aplicação utilizado para testes, uma grande variação no tempo de entrega das mensagens pode ser observado, dado que as mensagens só são enviadas sob demanda dos usuários. E estes, por sua vez, reagindo aos estados apresentados pela aplicação não possuem um conjunto de ações contínuas o que acaba por refletir

Res.	Tempo de entrega das mensagens *(ms)*										Média Geral (ms)
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
1	4.44	4.42	4.29	4.48	4.54	4.16	4.10	4.26	4.16	4.50	**4.33**
5	387.62	364.68	353.24	349.65	353.26	347.28	343.80	324.74	366.18	334.59	**352.50**
10	1034.28	951.80	870.07	950.27	704.58	709.98	687.30	777.93	770.73	728.53	**818.55**

Tabela 5.1: Tempo médio de entrega das mensagens para Cenário 1.

Res.	Tempo de entrega das mensagens *(ms)*										Média Geral (ms)
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
1	15.67	11.31	10.09	11.82	14.40	11.24	11.25	35.43	31.63	31.03	**18.39**
5	509.25	408.57	466.53	416.63	353.50	414.80	369.84	340.24	436.42	391.09	**410.69**
10	628.96	566.31	542.29	540.22	523.17	544.21	684.07	618.41	686.85	830.95	**616.54**

Tabela 5.2: Desvio padrão médio para entrega das mensagens no Cenário 1.

num comportamento heterogêneo para o tempo de entrega das mensagens durante uma dada execução.

Além do comportamento descrito acima, o tempo de entrega das mensagens é também diretamente influenciado pelo valor selecionado para a resiliência do sistema. Situação esta, possível de ser observada quando se compara na Figura 5.5 as diferentes posições das curvas de tempo médio quando resiliência escolhida é: 1, 5 ou 10.

- **Cenário 2**:
  - Aplicação: Jogo;
  - Protocolo: *Bluecast*;
  - Número de processos na rede: 3.

Res.	Tempo de entrega das mensagens *(ms)*										Média Geral (ms)
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
1	3.86	4.22	3.88	4.70	4.10	3.99	4.19	8.82	9.50	3.85	**5.11**
5	321.81	308.13	351.96	316.74	294.62	304.04	305.26	481.48	321.89	287.77	**329.37**
10	919.47	976.30	956.66	812.46	752.68	774.83	849.26	902.79	716.54	779.55	**844.05**

Tabela 5.3: Tempo médio de entrega das mensagens para Cenário 2.

Apesar do aumento no número de nós/processos na rede, os tempos de entrega das mensagens acabam por se manter na mesma faixa de valores daqueles apresentados no Cenário 1. Isso se deve ao fato de que apesar de haver um número maior de mensagens

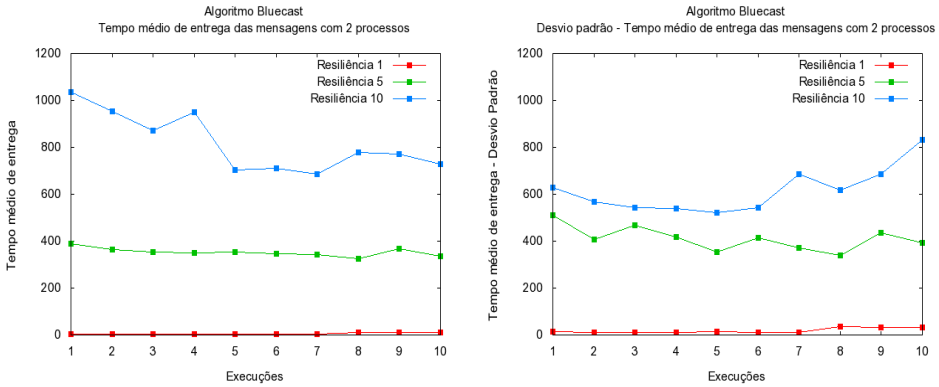


Figura 5.5: Tempo médio e desvio padrão para Cenário 1.

Res.	**Tempo de entrega das mensagens** *(ms)*										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
1	13.11	14.09	12.08	16.35	15.14	10.96	15.45	35.62	38.02	12.41	**18.32**
5	802.13	659.46	734.29	798.58	685.61	684.91	751.05	1586.16	775.05	629.92	**810.72**
10	920.44	1208.77	1080.88	2778.30	1080.63	1135.47	1208.18	1272.98	1185.71	1039.63	**1291.10**

Tabela 5.4: Desvio padrão médio para entrega das mensagens no Cenário 2.

trafegando pela rede, o tempo necessário para se processar uma mensagem (desde sua chegada na camada mais baixa até a entrega na camada de aplicação) continua com variações parecidas àquelas apresentadas no cenário anterior.

- **Cenário 3**:
  - Aplicação: Jogo;
  - Protocolo: *Bluecast*;
  - Número de processos na rede: 4.

O mesmo raciocínio aplicado ao Cenário 2, pode também ser aplicado aqui. Pelo mesmo motivo, o número de processos no grupo (4) acaba por não causar uma grande reação no tempo de entrega das mensagens. Entretanto, ainda assim é possível observar um pequeno aumento nestes valores quando resiliência do sistema é colocada em 5 ou 10. Dessa forma, é correto afirmar que se o número de processos for grande o suficiente, alguma sobrecarga mais significativa poderá ser causada pois o nó mestre necessitará enviar uma cópia das mensagens a todos os outros elementos.

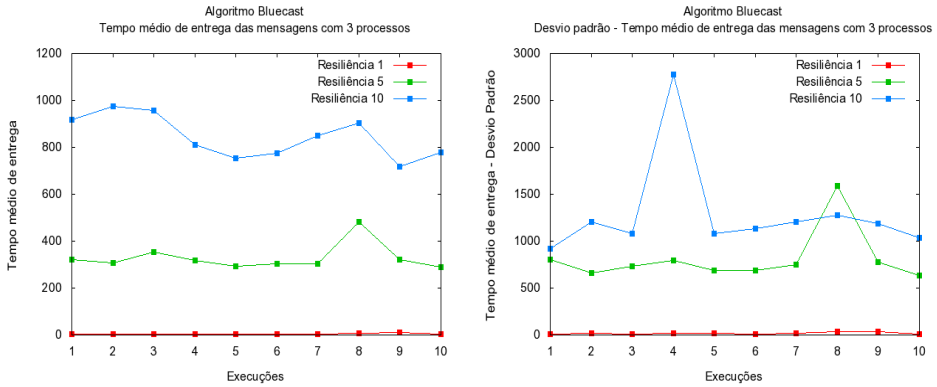


Figura 5.6: Tempo médio e desvio padrão para Cenário 2.

Res.	**Tempo de entrega das mensagens** ***(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
1	7.03	5.90	5.50	5.80	6.20	6.08	6.44	11.53	10.55	10.44	**7.55**
5	461.51	464.59	444.63	453.50	426.19	417.26	431.60	438.28	459.73	464.33	**446.16**
10	977.43	1047.55	963.39	983.19	994.44	1012.45	953.03	1029.16	1027.54	1004.90	**999.31**

Tabela 5.5: Tempo médio de entrega das mensagens para Cenário 3.

Ao se analisar os gráficos de tempo de entrega gerados por tais testes, é possível se notar que os resultados apresentados na Figura 5.5 se aproximam muito dos que podem ser observados em Figura 5.6. Contudo, para Figura 5.7, o resultado no aumento no número de nós pode ser observado através de um deslocamento das linhas que representam as execuções com a resiliência em 5 e 10.

- **Cenário 4**:
  - Aplicação: Jogo;
  - Protocolo: Chang & Maxemchuk;
  - Número de processos na rede: 2.

Atualmente, a grande maioria dos dispositivos mais simples (como telefones portáteis, fones de ouvido, MP3 *players*...) com suporte a redes *Bluetooth* possuem algumas limitações de *hardware* que os impedem de executar em sua totalidade todas as funcionalidades providas pelo padrão *IEEE 802.15*. Algumas destas são:

- Número de conexões: Algumas vezes limitado a um número menor que o previsto pela especificação do padrão;

Res.	Tempo de entrega das mensagens *(ms)*										Média Geral (ms)
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
1	12.10	11.01	10.17	10.13	12.28	11.31	11.33	21.08	20.17	21.03	**14.06**
5	853.77	921.31	1033.12	910.98	835.93	878.51	706.78	997.38	933.29	944.85	**901.59**
10	1009.38	1376.93	1061.52	1067.36	1187.84	1393.85	1126.37	1340.44	1429.70	1259.20	**1225.26**

Tabela 5.6: Desvio padrão médio para entrega das mensagens no Cenário 3.

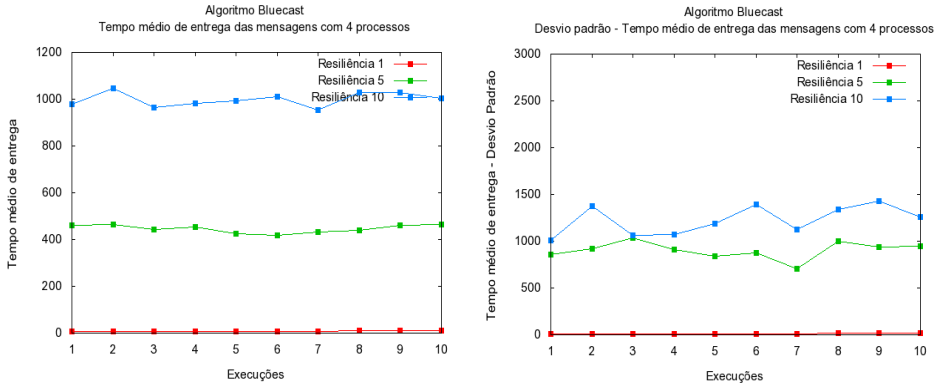


Figura 5.7: Tempo médio e desvio padrão para Cenário 3.

- *Piconets*:
  - Limitação quanto a entrada, formação e compartilhamento de dados em redes do tipo *Piconet*;
  - Limitação no número de *piconets* que podem ser formadas.

Dessa forma, devido as características do protocolo proposto por Chang&Maxemchuk [12], a execução dos testes e coleta dos dados do mesmo só foi possível utilizando-se 2 processos na rede. Isto porque, a principal limitação reside no fato de que nesta proposta, diferentemente das outras, todos os processos agem como mestres e escravos dentro do ambiente, necessitando assim que tal comportamento seja suportado pela arquitetura mais baixa.

Uma vez que o protocolo não foi desenvolvido para diretamente operar no ambiente *Bluetooth*, os valores obtidos para os tempos de entrega das mensagens estão bem acima daqueles apresentados nos cenários relativos ao protocolo *Bluecast* e melhor esforço.

Numa análise dos gráficos apresentados na Figura 5.8 é possível se confirmar o comportamento apresentado acima. Ainda, quando se compara estes resultados com os apresentados nas Figuras 5.5, 5.6 e 5.7 tem-se a exata noção do ganho apresentado pela primeira

Res.	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
1	672.34	484.47	948.83	291.93	408.48	266.61	813.09	650.91	514.94	504.41	**555.60**
5	1220.62	1853.81	1797.06	1077.31	2171.21	2099.77	1468.52	1716.38	1313.02	1550.13	**1626.78**
10	2168.48	2804.56	2869.66	2764.72	2262.02	2603.84	2324.59	3384.47	2223.26	3084.63	**2649.02**

Tabela 5.7: Tempo médio de entrega das mensagens para Cenário 4.

Res.	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
1	504.65	483.50	551.38	340.80	381.92	310.75	636.05	511.61	348.06	349.64	**441.84**
5	705.76	1050.98	831.85	576.91	1103.90	1077.91	805.45	959.62	715.24	1034.35	**886.2**
10	657.61	925.81	745.09	1552.75	694.24	621.54	589.05	1534.65	658.05	1152.29	**913.11**

Tabela 5.8: Desvio padrão médio para entrega das mensagens no Cenário 4.

proposta analisada (*Bluecast*).

- **Cenário 5**:
  - Aplicação: Jogo;
  - Protocolo: Melhor esforço;
  - Número de processos na rede: 2.

Res.	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
1	6.62	8.24	7.91	7.94	7.94	7.84	7.15	9.50	9.95	10.10	**8.32**
5	508.18	460.96	401.57	505.37	462.03	447.60	458.73	479.51	501.08	482.37	**470.74**
10	792.71	772.08	918.87	829.76	875.78	740.49	726.84	868.53	882.21	842.73	**825.00**

Tabela 5.9: Tempo médio de entrega das mensagens para Cenário 5.

Os resultados apresentados no Cenário 5 se mantêm dentro da margem esperada. É possível se comparar o mesmo com outros dois cenários:

- Cen. 1: Apesar de apresentar valores de tempo algumas vezes menores, ambos os cenários (1 e 5) tem seus valores consistentes uma vez que uma pequena variação no tempo é causada pela natureza da aplicação de tempo real que está sendo executada;
- Cen. 4: Como esperado, quando a comparação ocorre com este cenário, os valores são bem menores, uma vez que no último não se tem a implicação de sequenciamento e envio de mensagens de ACK.

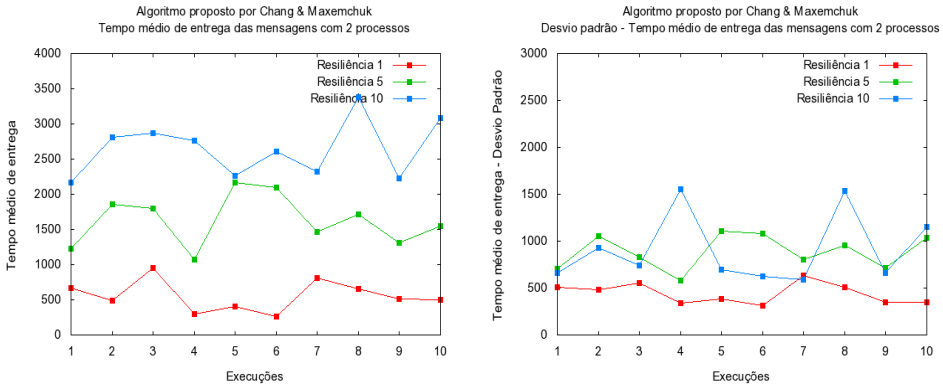


Figura 5.8: Tempo médio e desvio padrão para Cenário 4.

Res.	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
1	9.07	12.03	12.17	10.53	10.03	10.82	9.62	19.18	20.16	20.00	**13.36**
5	865.18	1290.85	277.33	512.56	442.65	1183.35	735.06	403.27	541.00	362.14	**661.34**
10	690.02	424.10	960.11	417.75	465.15	271.21	310.44	342.06	413.82	385.15	**467.98**

Tabela 5.10: Desvio padrão médio para entrega das mensagens no Cenário 5.

- **Cenário 6**:
  - Aplicação: Jogo;
  - Protocolo: Melhor esforço;
  - Número de processos na rede: 3.

Res.	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
1	8.58	9.14	7.04	8.08	8.16	7.25	7.93	10.28	10.94	10.27	**8.77**
5	556.19	549.62	543.71	538.26	523.94	487.52	432.17	474.28	424.12	510.58	**504.04**
10	1127.75	1326.12	1089.61	1016.93	1029.69	1000.33	1044.30	836.42	1166.19	1112.10	**1074.94**

Tabela 5.11: Tempo médio de entrega das mensagens para Cenário 6.

Assim como no cenário anterior (5), aqui ainda é impossível analisar o impacto causado pela adição de um novo processo ao grupo. Isto porque, os tempos se aproximam daqueles obtidos em Cenário 2.

- **Cenário 7**:

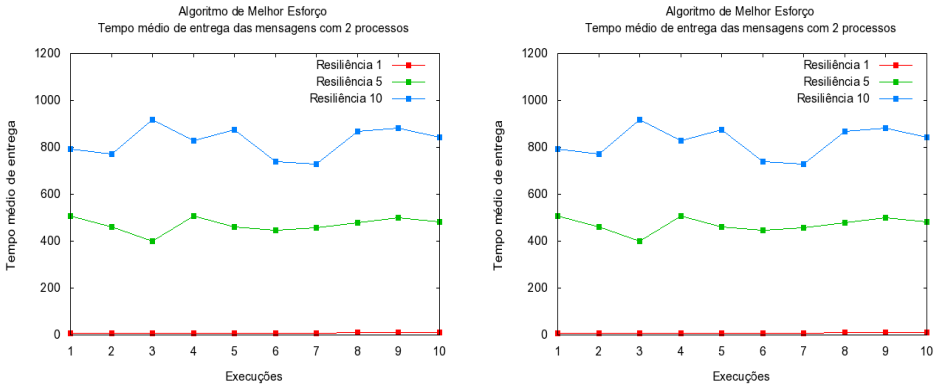


Figura 5.9: Tempo médio e desvio padrão para Cenário 5.

Res.	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
1	12.22	13.39	9.47	11.56	11.97	12.87	13.23	19.36	18.47	17.97	**14.05**
5	714.09	702.59	1355.65	2380.43	982.41	640.54	679.10	624.13	796.93	614.86	**949.07**
10	1439.50	3961.63	960.70	1137.84	1517.07	896.18	939.47	791.79	1113.24	839.65	**1359.71**

Tabela 5.12: Desvio padrão médio para entrega das mensagens no Cenário 6.

- Aplicação: Jogo;
- Protocolo: Melhor esforço;
- Número de processos na rede: 4.

Res.	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
1	4.91	6.52	6.53	6.03	6.00	6.20	6.02	8.52	8.67	8.38	**6.78**
5	357.69	359.85	342.29	355.01	347.91	359.58	321.34	332.60	390.11	350.48	**351.69**
10	1041.70	885.92	931.76	757.09	790.45	736.10	793.89	723.48	831.58	767.25	**825.92**

Tabela 5.13: Tempo médio de entrega das mensagens para Cenário 7.

No Cenário 7, já é possível se notar um outro comportamento. Com o acréscimo de mais um processo ao grupo (agora com 4 nós) observa-se uma pequena variação no tempo de entrega das mensagens diferentemente dos tempos bem próximos, apresentados em Cenários 1/5 e 2/6.

Através da análise destes resultados é possível se concluir que um ganho claro só começa a ocorrer à partir de um grupo de processos de tamanho mínimo igual a 4. Em

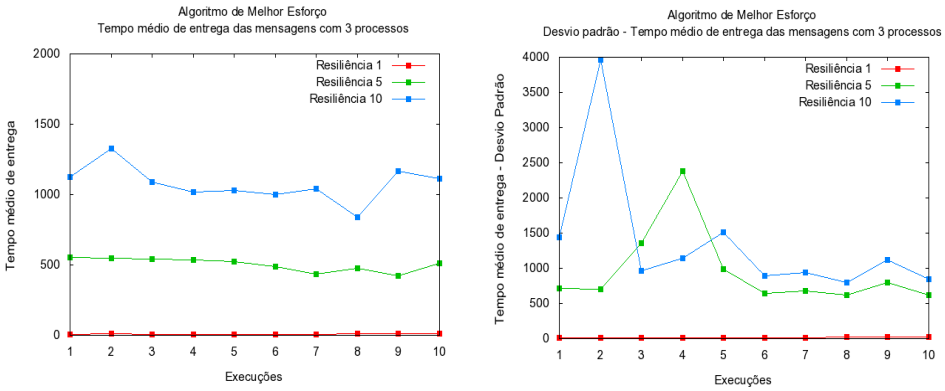


Figura 5.10: Tempo médio e desvio padrão para Cenário 6.

Res.	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
1	9.00	11.17	10.67	9.16	8.92	11.34	9.26	17.24	14.65	18.51	**11.99**
5	518.94	453.55	433.39	489.86	477.30	489.65	303.43	355.54	543.82	396.97	**446.25**
10	1328.43	1223.06	1613.68	1611.11	1444.45	1451.23	1413.75	2165.94	1268.36	1209.72	**1472.97**

Tabela 5.14: Desvio padrão médio para entrega das mensagens no Cenário 7.

ambientes com número reduzido de processos é impossível se notar o pequeno ganho obtido com o protocolo de Melhor Esforço.

Apesar de apresentar valores de tempo melhores para ambos os casos (grandes e pequenos grupos de processos) o protocolo de Melhor Esforço não garante premissas básicas de ambientes de ordenação atômica, como:

- Ordem;
- Garantia de entrega;
- Suporte a falhas.

Dessa forma, a conclusão obtida quanto ao tempo de entrega de mensagens é favorável à utilização do algoritmo *Bluecast* uma vez que o mesmo apresenta o melhor comportamento dentro do ambiente de comunicação considerado.

Tráfego contínuo

Um dos principais pontos levados em conta pelos resultados apresentados na subseção 5.3.2 foram os valores possíveis para a variável resiliência, que diretamente interfere no

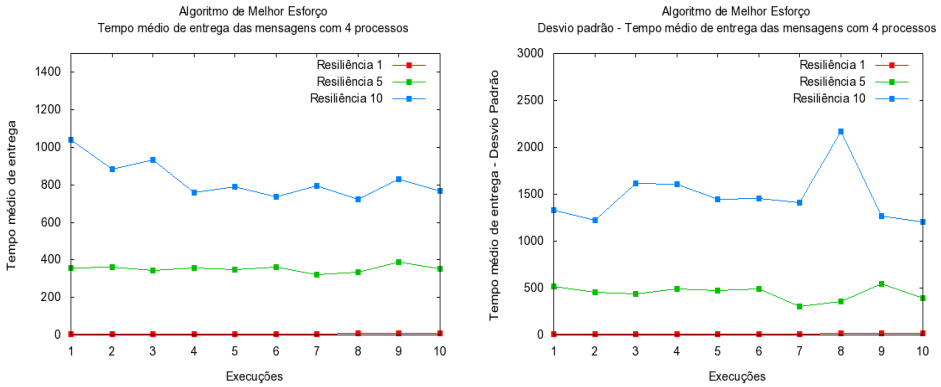


Figura 5.11: Tempo médio e desvio padrão para Cenário 7.

nível de confiabilidade e interatividade que pode ser provida ao usuário. Apesar disso, este outro conjunto de testes aqui apresentado privilegia a análise do desempenho apresentada por cada um dos protocolos de ordenação total em detrimento do grau de confiabilidade/interatividade. Fator este que é claramente demonstrado através da ênfase na questão do intervalo de envio de cada mensagem através do enlace de transporte.

Para uma melhor definição do escopo dos testes, determinadas variáveis do sistema tiveram seus valores devidamente delimitados:

- Resiliência: 1;
- Rodadas de execução: 10;
- Intervalo de envio de mensagens pela rede: $30ms$, $40ms$ e $50ms$;

As tabelas apresentadas a seguir relacionadas a este conjunto de cenários descrevem a relação entre o tempo de envio de cada mensagem e os tempos médios de entrega das mesmas nos dispositivos. A primeira coluna representa o intervalo no tempo de envio de duas mensagens sequenciais. As demais colunas representam o tempo médio de entrega das mensagens em cada uma das dez execuções do protocolo.

- **Cenário 8**:
  - Aplicação: Gerador de números aleatórios;
  - Protocolo: *Bluecast*;
  - Número de processos na rede: 2.

Intervalo de envio (*ms*)	**Tempo de entrega das mensagens*(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
30	2.52	2.40	2.33	2.28	2.22	2.27	2.35	2.24	2.33	2.39	**2.33**
40	0.83	2.19	1.91	2.09	2.27	2.25	2.25	2.25	2.26	2.07	**2.04**
50	2.33	2.39	2.39	2.26	2.27	2.19	1.89	2.66	2.12	2.34	**2.28**

Tabela 5.15: Tempo médio de entrega das mensagens para Cenário 8.

Intervalo de envio (*ms*)	**Tempo de entrega das mensagens*(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
30	4.57	3.60	3.76	4.64	9.92	3.68	3.73	3.31	3.46	3.70	**4.44**
40	3.38	3.51	1.78	2.48	3.70	2.57	3.73	3.98	7.52	3.04	**3.57**
50	3.82	4.34	4.34	3.44	3.40	3.28	2.74	3.26	3.47	3.71	**3.58**

Tabela 5.16: Desvio padrão médio para entrega das mensagens no Cenário 8.

Os resultados apresentados no Cenário 8 demonstram claramente que o conjunto, número de processos mais intervalo de envio não influencia diretamente no tempo de entrega das mensagens quando existem apenas 2 processos na rede. Por tal motivo, é possível se observar na Figura 5.12 que o gráfico à esquerda apresenta uma curva com valores praticamente contínuos durante todas as 10 execuções do protocolo.

- **Cenário 9**:
  - Aplicação: Gerador de números aleatórios;
  - Protocolo: *Bluecast*;
  - Número de processos na rede: 3.

Intervalo de envio (*ms*)	**Tempo de entrega das mensagens*(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
30	2.39	2.12	2.35	2.92	2.32	2.24	2.33	2.05	2.32	2.82	**2.39**
40	2.24	1.97	2.07	2.36	2.41	2.04	2.24	2.05	2.15	2.33	**2.19**
50	1.93	2.26	2.38	1.97	1.98	1.95	1.98	2.07	2.24	2.05	**2.08**

Tabela 5.17: Tempo médio de entrega das mensagens para Cenário 9.

Novamente neste cenário, assim como no anterior, o aumento no número de processos participantes do *broadcast* acaba por não gerar nenhuma sobrecarga que venha a acarretar uma perda no desempenho do algoritmo. Numa comparação entre as Figuras 5.12 e 5.13 é possível se observar que o comportamento para o tempo médio de entrega estão ambos nas mesmas faixas de valores.

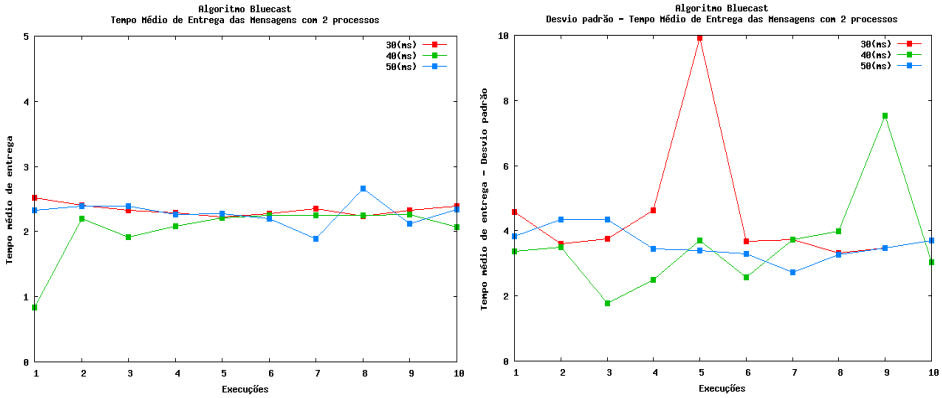


Figura 5.12: Tempo médio e desvio padrão para Cenário 8.

Intervalo de envio (*ms*)	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
30	3.51	2.81	3.77	2.36	3.50	3.94	3.51	2.27	3.94	2.45	**3.21**
40	3.82	2.55	2.65	4.68	4.38	2.30	2.95	3.15	3.95	4.28	**3.47**
50	1.81	4.39	4.69	2.43	2.31	1.95	4.06	2.60	3.36	3.45	**3.11**

Tabela 5.18: Desvio padrão médio para entrega das mensagens no Cenário 9.

- **Cenário 10**:
  - Aplicação: Gerador de números aleatórios;
  - Protocolo: *Bluecast*;
  - Número de processos na rede: 4.

Intervalo de envio (*ms*)	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
30	1.90	2.35	1.99	1.91	2.20	2.23	1.92	2.28	2.21	2.33	**2.13**
40	1.93	2.56	2.23	2.26	2.17	1.98	1.94	1.88	1.90	1.93	**2.08**
50	2.09	2.12	2.26	2.17	2.23	2.15	2.30	2.01	2.23	1.91	**2.15**

Tabela 5.19: Tempo médio de entrega das mensagens para Cenário 10.

Contextualizando-se o Cenário 10 com os cenários 8 e 9 é possível se observar que não existe praticamente nenhuma variação no tempo de entrega das mensagens quando se acrescenta novos componentes ao grupo de comunicação. Entretanto, um fator interessante a ser ressaltado é o comportamento observado quando deste aumento progressivo.

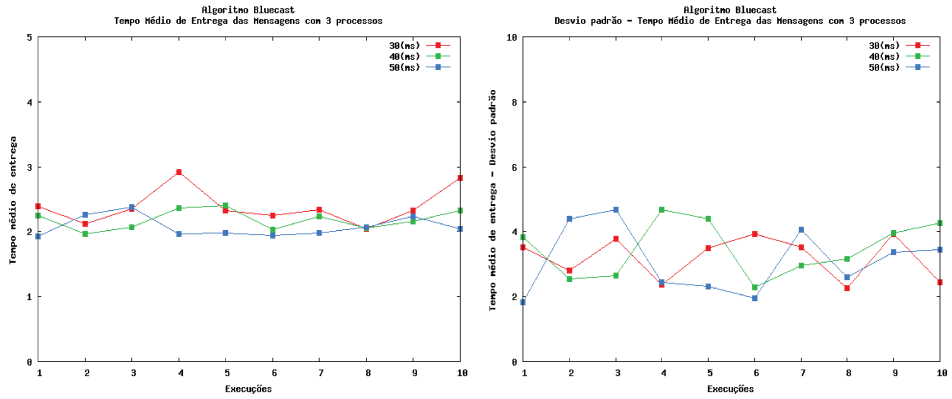


Figura 5.13: Tempo médio e desvio padrão para Cenário 9.

Intervalo de envio (*ms*)	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
30	1.77	4.10	3.26	2.36	3.68	4.73	3.28	4.41	3.54	4.55	**3.57**
40	2.91	5.07	4.54	3.77	4.21	6.77	3.31	2.60	19.82	2.09	**5.51**
50	3.72	3.08	3.83	3.35	3.61	3.90	3.08	2.46	3.02	2.46	**3.25**

Tabela 5.20: Desvio padrão médio para entrega das mensagens no Cenário 10.

Conforme são adicionados mais processos, a variação no tempo médio de entrega fica cada vez menor tendendo a ser uniformizada.

Tal comportamento é de certa forma explicado através das próprias características do protocolo. Uma vez que o mestre tem mais processos para enviar dados, os processos escravos têm mais tempo entre o recebimento de duas mensagens consecutivas. Por conseguinte, facilita-se que uma mensagem não fique aguardando para ser entregue e se evita causar um *overhead* no ambiente.

O algoritmo de melhor esforço foi também utilizado para este tipo de aplicação, assim como apresentado para o algoritmo *Bluecast*. Os cenários 11 a 13 descrevem os resultados obtidos quando da execução deste.

- **Cenário 11**:
  - Aplicação: Gerador de números aleatórios;
  - Protocolo: Melhor esforço;
  - Número de processos na rede: 2;

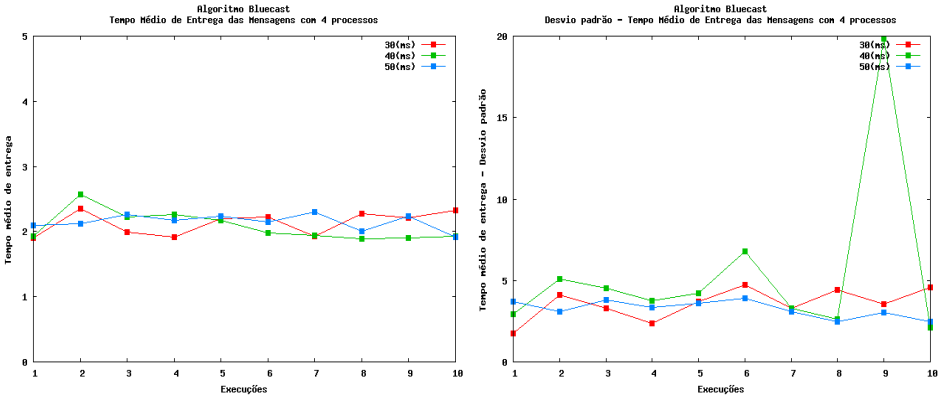


Figura 5.14: Tempo médio e desvio padrão para Cenário 10.

Intervalo de envio (*ms*)	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
30	1.51	1.43	1.70	1.70	1.69	1.77	1.78	1.40	1.56	1.80	**1.63**
40	1.74	1.50	1.49	2.72	0.88	1.59	1.55	1.30	1.64	1.64	**1.61**
50	1.27	1.68	0.88	1.61	1.51	1.52	1.51	1.53	1.70	1.61	**1.48**

Tabela 5.21: Tempo médio de entrega das mensagens para Cenário 11.

Mesmo com apenas 2 processos na rede, o algoritmo de melhor esforço consegue fazer a entrega das mensagens sem que isso cause sobrecarga alguma ao sistema como um todo. Fato este que pode ser notado se avaliando o tempo de entrega das mensagens para os três cenários de teste que foram considerados.

- **Cenário 12**:
  - Aplicação: Gerador de números aleatórios;
  - Protocolo: Melhor esforço;
  - Número de processos na rede: 3;

Intervalo de envio (*ms*)	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
30	4.37	2.27	4.24	3.94	3.31	3.79	3.73	2.66	3.18	3.08	**3.46**
40	4.96	2.72	2.98	3.88	1.03	3.48	2.93	1.80	3.89	5.08	**3.28**
50	2.63	3.58	1.27	2.92	3.17	2.75	2.67	3.59	4.83	3.05	**3.05**

Tabela 5.22: Desvio padrão médio para entrega das mensagens no Cenário 11.

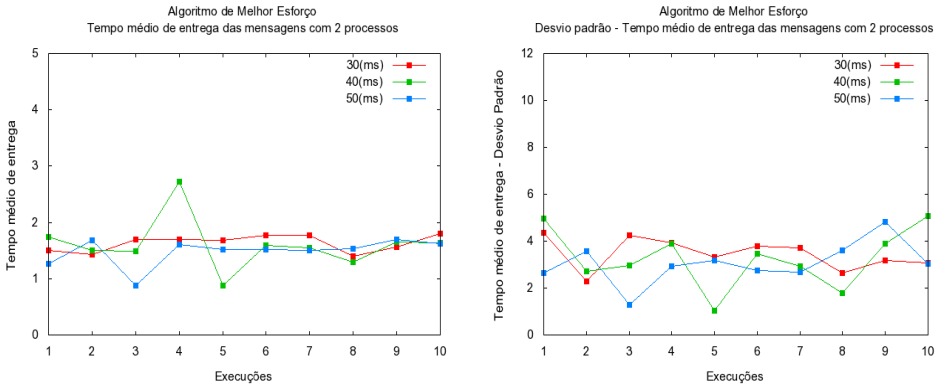


Figura 5.15: Tempo médio e desvio padrão para Cenário 11.

Intervalo de envio (*ms*)	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
30	1.71	1.73	1.76	1.47	1.54	1.84	2.10	1.48	1.52	1.79	**1.69**
40	1.40	1.38	1.45	1.53	1.58	1.55	1.69	1.62	1.45	1.40	**1.51**
50	1.42	1.53	1.55	1.48	1.52	1.52	1.61	1.52	1.31	1.46	**1.49**

Tabela 5.23: Tempo médio de entrega das mensagens para Cenário 12.

Apesar da diminuição no número de processos componentes do ambiente, os tempos de entrega das mensagens não são influenciados. Isto porque a participação de 4 componentes no grupo acaba por não causar nenhuma sobrecarga separadamente a cada um dos nós participantes, como pode ser comprovado pela análise dos resultados apresentados neste cenário.

Um dado que também pode ser observado ao se comparar este cenário como o anterior, diz respeito a distribuição dos valores obtidos para o desvio padrão. Nota-se que quanto maior o número de processos no grupo, maior será também a variação no tempo de entrega das mensagens. Isto é diretamente causado pela atuação do processo mestre (responsável pelo envio das mensagens a todos os outros processos), que com mais escravos, necessita

Intervalo de envio (*ms*)	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
30	4.66	3.77	3.40	2.82	2.30	5.95	9.81	2.47	3.54	3.70	**4.24**
40	2.18	2.41	3.40	3.16	3.21	2.81	3.38	2.72	2.78	2.92	**2.90**
50	4.27	2.99	3.47	2.89	3.71	3.03	4.21	3.27	4.08	2.93	**3.49**

Tabela 5.24: Desvio padrão médio para entrega das mensagens no Cenário 12.

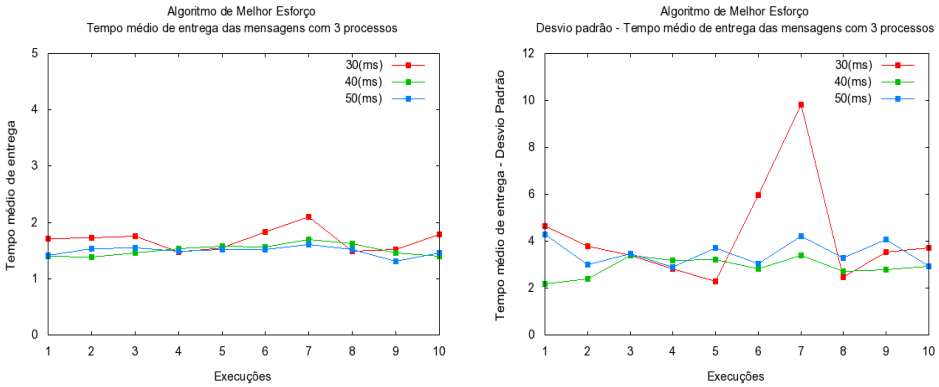


Figura 5.16: Tempo médio e desvio padrão para Cenário 12.

de um maior intervalo de tempo para que todos recebam os dados.

- **Cenário 13**:
  - Aplicação: Gerador de números aleatórios;
  - Protocolo: Melhor esforço;
  - Número de processos na rede: 4;

Intervalo de envio (*ms*)	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
30	1.34	1.35	1.47	1.25	1.28	1.28	1.34	1.15	1.29	1.33	**1.31**
40	1.39	1.48	1.59	1.17	1.30	1.18	1.09	1.18	1.48	1.51	**1.34**
50	1.51	1.40	1.49	1.64	1.50	1.51	1.51	1.35	1.52	1.68	**1.51**

Tabela 5.25: Tempo médio de entrega das mensagens para Cenário 13.

Intervalo de envio (*ms*)	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
30	4.07	2.52	4.92	3.35	4.12	3.06	4.43	2.61	3.85	3.17	**3.61**
40	5.13	4.51	5.59	2.38	2.83	5.58	2.64	2.45	4.79	5.13	**4.10**
50	4.32	3.82	10.22	5.67	4.01	4.84	4.57	4.08	5.00	4.97	**5.15**

Tabela 5.26: Desvio padrão médio para entrega das mensagens no Cenário 13.

O cenário 13 apresenta resultados compatíveis com execuções prévias do protocolo de melhor esforço para ambientes com 4 processos. Se comparados com os dados apresentados

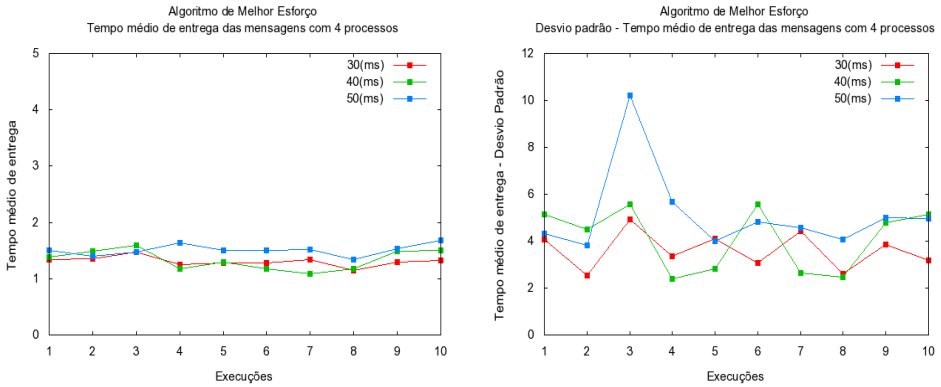


Figura 5.17: Tempo médio e desvio padrão para Cenário 13.

pelo algoritmo *Bluecast*, nota-se que ambos se aproximam bastante. Entretanto, por não implementar os conceitos de ordenação e suporte a falhas, o protocolo de melhor esforço acaba por atingir melhores tempos nas entregas das mensagens.

Para o protocolo proposto por Chang&Maxemchuk também foram executados a mesma classe de testes - variando-se o intervalo de envio das mensagens. Os resultados são apresentados no cenário 14.

- **Cenário 14**:
  - Aplicação: Gerador de números aleatórios;
  - Protocolo: Chang&Maxemchuk;
  - Número de processos na rede: 2;

Intervalo de envio (*ms*)	**Tempo de entrega das mensagens*(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
30	179.25	178.97	179.33	180.86	179.09	177.49	178.78	180.19	179.04	184.12	**179.71**
40	190.44	189.24	191.55	184.98	184.70	183.99	185.72	183.65	184.78	183.99	**186.30**
50	178.90	208.92	197.49	197.67	198.23	202.50	198.37	201.89	201.73	201.71	**198.74**

Tabela 5.27: Tempo médio de entrega das mensagens para Cenário 14.

Os dados obtidos dos testes com o protocolo proposto por Chang&Maxemchuk são relativamente altos se comparados aos dois protocolos anteriormente descritos. Tal comportamento é causado basicamente pela natureza do protocolo, que utiliza uma mensagem de resposta a cada mensagem de difusão enviada. Tal fator, além de aumentar o tráfego

Intervalo de envio (*ms*)	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
30	34.45	35.02	35.36	50.17	35.40	33.65	34.23	42.67	34.33	38.10	**37.34**
40	37.99	35.09	39.02	33.28	32.78	32.99	40.01	34.34	33.24	32.88	**35.16**
50	37.81	63.05	49.91	51.43	49.82	49.65	49.17	58.54	51.01	53.15	**51.35**

Tabela 5.28: Desvio padrão médio para entrega das mensagens no Cenário 14.

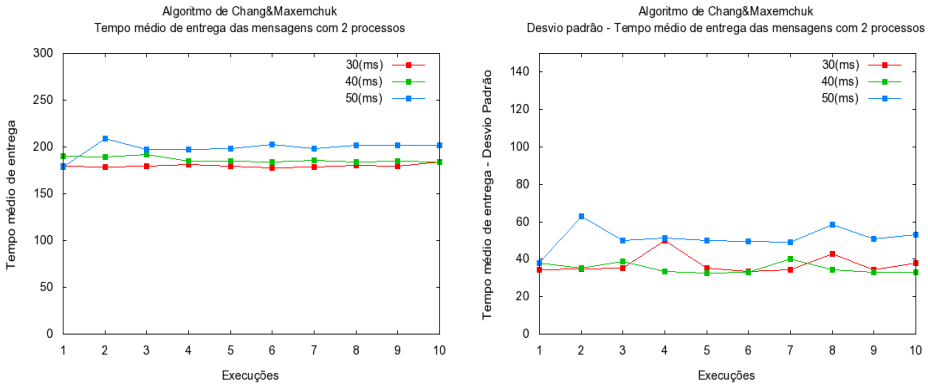


Figura 5.18: Tempo médio e desvio padrão para Cenário 14.

interno da rede causa ainda uma sobrecarga no tempo de entrega das mensagens, pois uma dada mensagem só pode ser entregue após receber a confirmação de recebimento pelo processo responsável pelo sequenciamento no dado momento.

Mestres vs. Escravos

Outro ponto avaliado neste protocolo de ordenação foi o comportamento dos processos mestres em relação aos processos escravos. Os próximos cenários apresentam os resultados quando se compara o tempo de entrega das mensagens entres os mestres e escravos numa mesma execução.

Os resultados apresentados pelas tabelas deste grupo descrevem os tempos médios de entrega das mensagens para cada um dos grupos de processos presentes na rede (mestre e escravos). A primeira coluna descreve o papel considerado enquanto as demais listam os tempos médios de entrega das mensagens em cada uma das dez execuções do algoritmo.

- **Cenário 15**:
  - Aplicação: Gerador de números aleatórios;
  - Protocolo: *Bluecast*;

– Número de processos na rede: 2;

– Frequencia de envio das mensagens: 30(ms).

Processo	**Tempo de entrega das mensagens** ***(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
Mestre	2.52	2.40	2.33	2.28	2.22	2.27	2.35	2.24	2.33	2.39	**2.33**
Escravo-1	2.92	2.92	3.25	5.19	4.34	2.81	2.99	4.47	5.05	3.03	**3.70**

Tabela 5.29: Tempo médio de entrega das mensagens para Cenário 15.

Processo	**Tempo de entrega das mensagens** ***(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
Mestre	4.57	3.60	3.76	4.64	9.92	3.68	3.73	3.31	3.46	3.70	**4.44**
Escravo-1	4.84	8.56	6.45	14.49	11.99	5.26	5.85	12.98	13.72	7.12	**9.13**

Tabela 5.30: Desvio padrão médio para entrega das mensagens no Cenário 15.

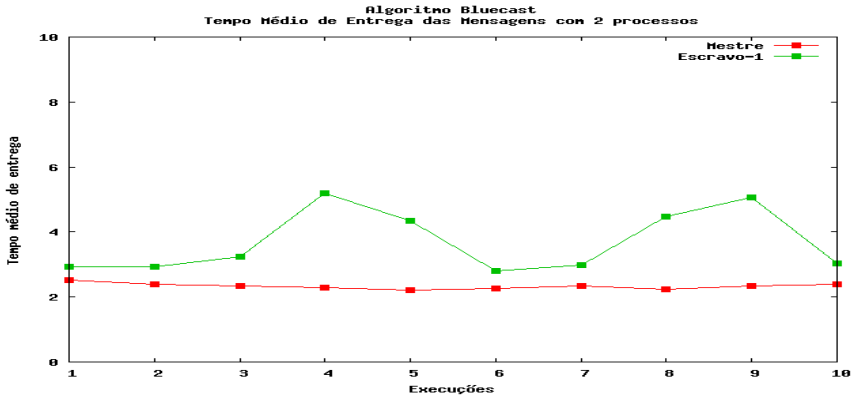


Figura 5.19: Tempo médio e desvio padrão para Cenário 15.

Analisando-se os resultados apresentados no cenário 15 se observa a disparidade entre o comportamento de um processo mestre para com o de um processo escravo. Previsivelmente, o elemento mestre tem um melhor desempenho, uma vez que para o transporte das mensagens este não utiliza o enlace físico (transporte via ar).

Como observado na Figura 5.19, o nó escravo apresenta uma variação no tempo de entrega maior e mais frequente que a observada no processo mestre. Tal comportamento acaba sendo um reflexo da degradação do tempo médio de entrega dos dados.

- **Cenário 16**:
  - Aplicação: Gerador de números aleatórios;
  - Protocolo: *Bluecast*;
  - Número de processos na rede: 3;
  - Frequência de envio das mensagens: 30(ms).

Processo	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
Mestre	2.39	2.12	2.35	2.92	2.32	2.24	2.33	2.05	2.32	2.82	**2.39**
Escravo-1	3.16	2.73	3.25	4.25	5.55	3.01	2.98	2.89	3.09	2.98	**3.39**
Escravo-2	3.46	2.65	3.40	3.27	3.04	3.07	3.93	2.82	2.68	3.12	**3.14**

Tabela 5.31: Tempo médio de entrega das mensagens para Cenário 16.

Processo	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
Mestre	3.51	2.81	3.77	2.36	3.50	3.94	3.51	2.27	3.94	2.45	**3.21**
Escravo-1	6.21	3.49	7.08	7.20	6.82	5.54	5.37	4.40	5.28	7.54	**5.89**
Escravo-2	6.46	7.97	7.40	9.46	8.53	8.37	6.39	8.93	7.95	8.98	**8.04**

Tabela 5.32: Desvio padrão médio para entrega das mensagens no Cenário 16.

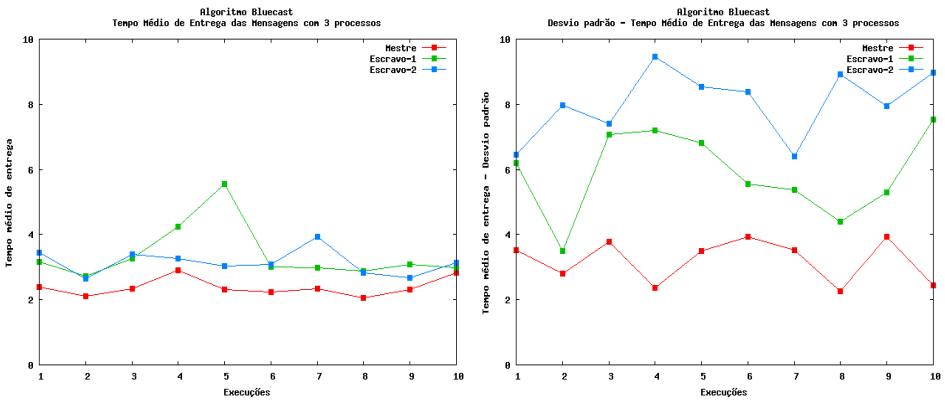


Figura 5.20: Tempo médio e desvio padrão para Cenário 16.

Nos resultados apresentados em cenário 16 é possível se observar a disparidade dos dados obtidos para um processo mestre em comparação com um elemento que atua como

escravo. Os gráficos apresentado na Figura 5.20 ilustram exatamente tal comportamento ao explicitar que em comparação com o cenário anterior, a diferença entre os tempos obtidos para o mestre e escravos se mantém praticamente os mesmos.

É possível se observar ainda um aumento na variação do tempo de entrega destas mensagens. Mais uma vez, ao se analisar o gráfico relacionado ao desvio padrão é possível notar uma maior constância no tempo de entrega dos dados para o processo mestre enquanto que os outros elementos apresentam um comportamento não uniforme.

- **Cenário 17**:
  - Aplicação: Gerador de números aleatórios;
  - Protocolo: *Bluecast*;
  - Número de processos na rede: 4;
  - Frequência de envio das mensagens: 30(ms).

Processo	**Tempo de entrega das mensagens*(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
Mestre	1.90	2.35	1.99	1.91	2.20	2.23	1.92	2.28	2.21	2.33	**2.13**
Escravo-1	3.05	2.90	2.65	2.74	2.83	2.88	4.80	2.81	2.81	2.85	**3.03**
Escravo-2	3.13	2.98	3.53	3.22	3.40	3.17	3.19	2.93	2.65	2.92	**3.11**
Escravo-3	2.92	2.79	3.02	3.03	2.70	2.75	2.99	3.00	2.79	2.78	**2.88**

Tabela 5.33: Tempo médio de entrega das mensagens para Cenário 17.

Processo	**Tempo de entrega das mensagens*(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
Mestre	1.77	4.10	3.26	2.36	3.68	4.73	3.28	4.41	3.54	4.55	**3.57**
Escravo-1	6.25	8.16	5.25	5.59	5.95	6.39	17.29	7.62	5.74	6.71	**7.49**
Escravo-2	7.84	12.22	12.30	9.95	11.79	10.85	12.89	9.45	7.52	10.06	**10.49**
Escravo-3	4.33	5.56	5.91	6.42	4.04	6.21	8.19	7.99	5.07	11.22	**6.49**

Tabela 5.34: Desvio padrão médio para entrega das mensagens no Cenário 17.

O mesmo comportamento, assim como nos dois casos de teste anteriores, é aqui também observado. O processo mestre tem um melhor desempenho que os demais escravos principalmente por não utilizar, como citado anteriormente, a camada física para transporte dos dados. Por outro lado, é também compreensível que o comportamento dos escravos se assemelhe bastante já que executam as mesmas ações na mesma ordem. A pequena variação apresentada entre os mesmos se deve principalmente às características de *hardware* inerentes a cada um dos dispositivos onde os testes foram executados.

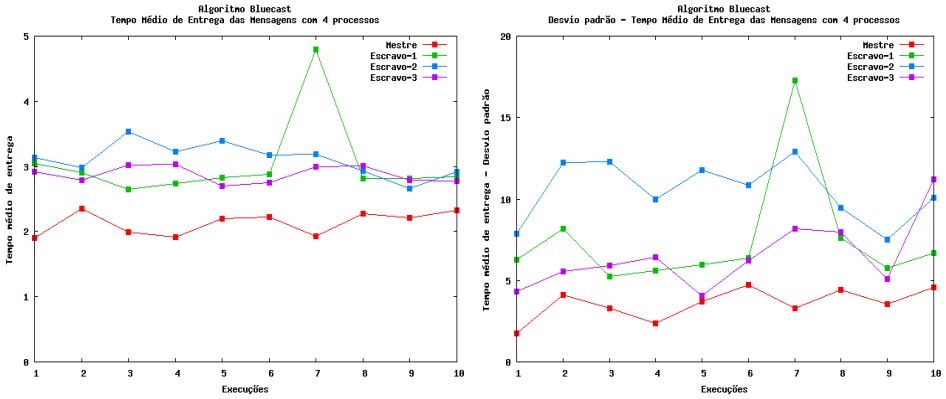


Figura 5.21: Tempo médio e desvio padrão para Cenário 17.

A arquitetura proposta para o desenvolvimento do algoritmo de melhor esforço se baseia principalmente no conceito de mestre e escravo que por sua vez é uma característica própria da especificação do ambiente *Bluetooth*. Dessa forma, é prudente que aqui também se analise o comportamento dos dispositivos mestres em relação aos escravos.

- **Cenário 18**:
  - Aplicação: Gerador de números aleatórios;
  - Protocolo: Melhor esforço;
  - Número de processos na rede: 2;
  - Frequência de envio das mensagens: 30(ms).

Processo	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
Mestre	1.91	1.43	1.99	1.90	1.69	1.77	1.78	1.40	1.56	1.80	**1.72**
Escravo-1	2.22	2.01	3.80	3.86	2.36	2.34	2.42	2.15	2.09	3.84	**2.71**

Tabela 5.35: Tempo médio de entrega das mensagens para Cenário 18.

Assim como nos testes desta natureza para o protocolo *Bluecast*, os resultados aqui diferem quando se compara o nó mestre e os escravos. Pelo mesmo motivo anteriormente apresentado, o nó mestre não utiliza nenhuma capacidade de processamento para receber a mensagem, pois a coloca diretamente na fila de processamento de sua camada mais baixa.

Processo	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
Mestre	4.37	2.27	4.24	3.94	3.31	3.79	3.73	2.66	3.18	3.08	**3.46**
Escravo-1	4.09	3.49	11.96	11.97	5.78	5.79	6.66	5.04	4.13	12.57	**7.15**

Tabela 5.36: Desvio padrão médio para entrega das mensagens no Cenário 18.

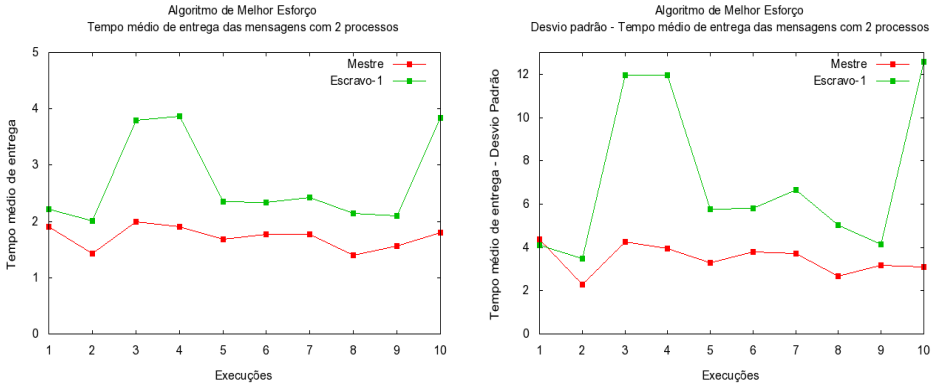


Figura 5.22: Tempo médio e desvio padrão para Cenário 18.

Outro ponto que é necessário se notar diz respeito à variação no tempo de entrega das mensagens. Através da Figura 5.22 é possível notar que o nó escravo possui uma grande variação no tempo de entrega das mensagens. Isto é causado basicamente pela falta de uniformidade na recepção das mesmas, uma vez que diferentemente dos processos mestres, os escravos dispendem certo nível de processamento para receber e interpretar as mensagens que chegam através do enlace físico.

- **Cenário 19**:
  - Aplicação: Gerador de números aleatórios;
  - Protocolo: Melhor esforço;
  - Número de processos na rede: 3;
  - Frequência de envio das mensagens: 30(ms).

Em comparação com os resultados apresentados em cenário 18, este apresenta uma equalização nos tempos médios de entrega das mensagens. Entretanto, os processos escravos apresentam também aqui o mesmo comportamento quanto à variação no tempo de entrega. Exceto pelo mestre, todos os outros nós não possuem uma uniformidade neste quesito devido ao mesmo motivo apresentado no cenário 18.

Processo	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
Mestre	1.71	1.73	1.76	1.47	1.54	1.84	2.10	1.48	1.52	1.79	**1.69**
Escravo-1	2.50	3.08	4.40	3.73	2.58	2.72	2.88	3.37	4.09	2.84	**3.22**
Escravo-2	2.33	2.13	1.89	2.23	2.33	2.10	2.30	2.62	2.29	2.58	**2.28**

Tabela 5.37: Tempo médio de entrega das mensagens para Cenário 19.

Processo	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
Mestre	4.66	3.77	3.40	2.82	2.30	5.95	9.81	2.47	3.54	3.70	**4.24**
Escravo-1	4.45	7.30	13.97	12.15	6.31	5.37	5.69	10.95	12.72	5.38	**8.43**
Escravo-2	10.62	8.38	6.49	9.66	10.11	8.49	10.08	13.13	9.89	11.99	**9.88**

Tabela 5.38: Desvio padrão médio para entrega das mensagens no Cenário 19.

- **Cenário 20**:
  - Aplicação: Gerador de números aleatórios;
  - Protocolo: Melhor esforço;
  - Número de processos na rede: 4;
  - Frequência de envio das mensagens: 30(ms).

Processo	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
Mestre	1.34	1.35	1.47	1.25	1.28	1.28	1.34	1.15	1.29	1.33	**1.31**
Escravo-1	2.23	2.07	4.07	4.07	1.89	1.99	2.17	5.02	3.91	2.52	**2.99**
Escravo-2	2.48	1.92	2.91	2.83	2.25	2.30	2.21	2.65	2.78	2.00	**2.43**
Escravo-3	1.99	1.91	2.55	2.16	1.65	2.10	1.70	2.16	2.24	1.84	**2.03**

Tabela 5.39: Tempo médio de entrega das mensagens para Cenário 20.

Novamente, no cenário 20 os dados apresentados são compatíveis com as afirmações anteriormente descritas nos cenários 18 e 19. Entretanto, é possível agora se notar um comportamento parecido se comparados somente os processos escravos, uma vez que invariavelmente o processo mestre sempre apresentará um tempo menor e mais uniforme na entrega das mensagens.

Apesar de não haver uma definição clara dos conceitos de mestre e escravo para o algoritmo proposto por Chang&Maxemchuk, nos cenários de teste aqui propostos houve uma separação nos papéis de cada processo: Um deles é encarregado de somente enviar os dados, enquanto o outro atua somente como um receptor. Dessa forma, se tornou

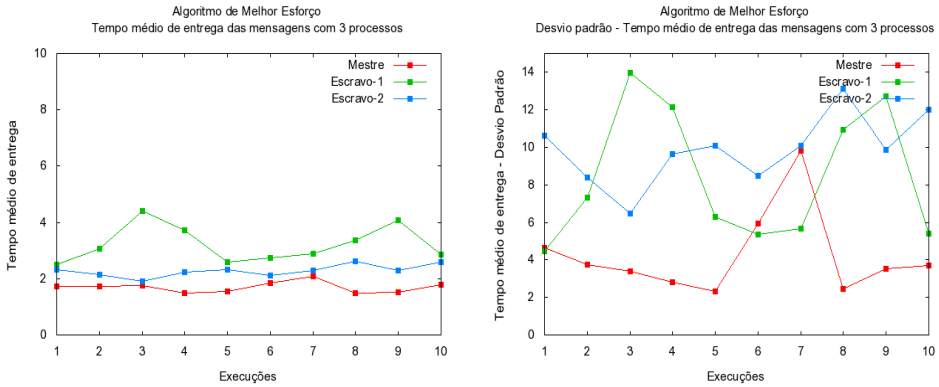


Figura 5.23: Tempo médio e desvio padrão para Cenário 19.

Processo	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
Mestre	4.07	2.52	4.92	3.35	4.12	3.06	4.43	2.61	3.85	3.17	**3.61**
Escravo-1	5.29	5.95	15.86	16.81	6.03	5.30	6.48	19.45	16.02	7.86	**10.51**
Escravo-2	13.07	10.25	15.89	15.79	10.71	12.84	9.67	13.31	13.44	8.80	**12.38**
Escravo-3	7.46	5.75	9.83	8.77	4.15	8.34	3.03	7.55	7.46	4.95	**6.73**

Tabela 5.40: Desvio padrão médio para entrega das mensagens no Cenário 20.

plausível a comparação dos tempos médios de entrega das mensagens em cada um destes participantes do grupo.

- **Cenário 21**:
  - Aplicação: Gerador de números aleatórios;
  - Protocolo: Chang&Maxemchuk;
  - Número de processos na rede: 2;
  - Frequência de envio das mensagens: 30(ms).

Processo	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
Mestre	179.25	178.97	179.33	180.86	179.09	177.49	178.78	180.19	179.04	184.12	**179.71**
Escravo-1	527.49	1059.61	702.34	1455.57	527.85	872.32	1053.51	529.53	526.06	544.66	**779.89**

Tabela 5.41: Tempo médio de entrega das mensagens para Cenário 21.

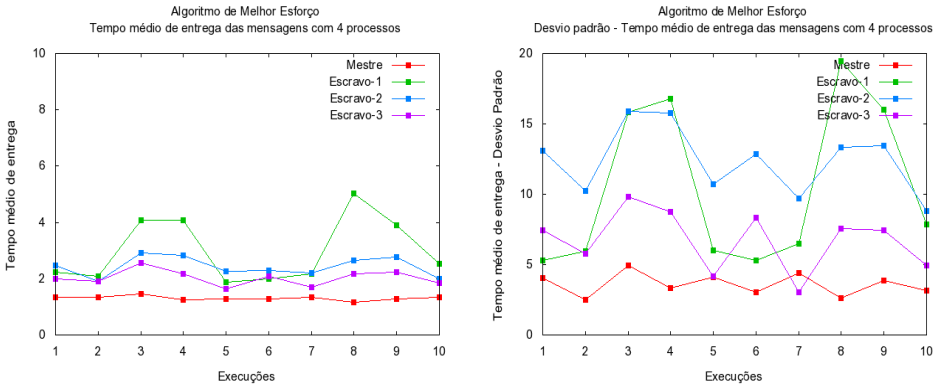


Figura 5.24: Tempo médio e desvio padrão para Cenário 20.

Processo	**Tempo de entrega das mensagens *(ms)***										**Média Geral (ms)**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
Mestre	34.45	35.02	35.36	50.17	35.40	33.65	34.23	42.67	34.33	38.10	**37.34**
Escravo-1	112.06	376.23	184.13	813.54	144.83	256.36	370.48	128.04	99.95	110.04	**259.57**

Tabela 5.42: Desvio padrão médio para entrega das mensagens no Cenário 21.

Pelos dados apresentados nos resultados do cenário 21 nota-se que aqui, a disparidade entre o comportamento dos dois tipos de processos é ainda maior. Em alguns casos, os valores para o nó receptor chega a ser aproximadamente 8x maior que no processo responsável pelo envio das mensagens. Como já citado, tal fato é causado principalmente pelas características do protocolo, onde o processo receptor acaba por ter que receber e interpretar dados adicionais para cada mensagem de difusão recebida.

A disparidade dos resultados fica ainda maior quando comparados junto aos resultados obtidos previamente em outras execuções de outros protocolos (principalmente quando comparados com o melhor caso). Tal diferença no desempenho dos protocolos pode, num ambiente de tempo real, causar certo desconforto ao usuário ao prover uma resposta demasiadamente lenta as requisições feitas pelo mesmo.

5.3.3 Número de mensagens trocadas

Cada um dos protocolos aqui aferidos possui um modo diferente de receber, enviar e sequenciar (ou não) as informações que trafegam pela rede. Dependendo do modo como tais operações são executadas, um *overhead* pode ser acrescentado ao conjunto de dados compartilhados. Dessa forma, para o mesmo tipo de aplicação considerada na subseção 5.3.2, foi aferido o número de mensagens total compartilhadas pelos processos que faziam

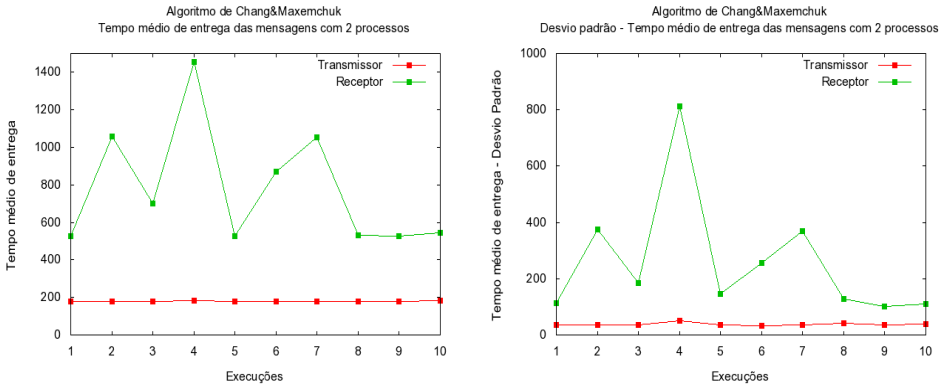


Figura 5.25: Tempo médio e desvio padrão para Cenário 21.

parte do ambiente de comunicação. Para tanto, as seguintes condições foram consideradas:

- Condição de parada: Entrega da mensagem de número 1300 a aplicação;
- Resiliência = 1;
- 10 execuções para cada um dos cenários considerados.

Os dados representados através das tabelas deste grupo de testes descrevem a relação número de mensagens enviadas / algoritmo. A primeira coluna representa o protocolo considerado por cada uma das dez execuções seguintes (representadas através das demais colunas da tabela).

- **Cenário 22**:
  - Aplicação: Jogo;
  - Número de processos na rede: 2.

Protocolo	**Número de mensagens compartilhadas**										**Média Geral**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
Melhor Esforço	1880	1902	1885	1877	1890	1898	1904	1901	1938	1910	**1898.5**
Chang&Max.	2613	2613	2615	2613	2612	2611	2615	2613	2612	2612	**2612.9**
Bluecast	1850	1852	1876	1890	1815	1837	1844	1893	1875	1845	**1857.7**

Tabela 5.43: Número de mensagens compartilhadas na rede para Cenário 22.

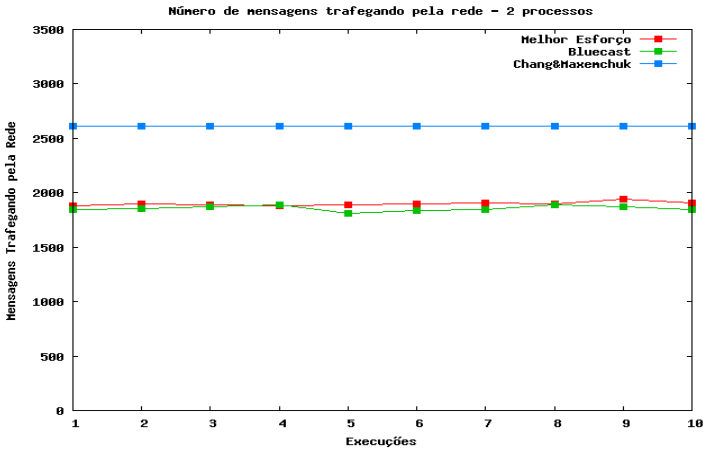


Figura 5.26: Número de mensagens compartilhadas para Cenário 22.

Os resultados apresentados neste cenário representam bem o comportamento esperado para cada um dos protocolos postos à prova. Inicialmente, por não se preocupar em prover garantias básicas de ordem e suporte a falhas, o protocolo de melhor esforço pode ser considerado como sendo a referência em termos de "caso ótimo" para o número de mensagens trocadas na rede de *broadcast*. Dessa forma, é possível se observar que os valores alcançados por tal algoritmo são relativamente baixos devido às características do mesmo. Consequentemente tem-se um baixo número de mensagens trafegando durante toda a execução, o que acaba por refletir num também pequeno *overhead* no compartilhamento dos dados.

O protocolo de *Bluecast* apresenta também um bom resultado. Apesar incluir algumas mensagens de controle, mesmo assim, alcança um resultado muitas vezes melhor que aquele apresentado pelo algoritmo de melhor esforço e ainda garante ordem e entrega das informações.

Por outro lado, devido a características inerentes a especificação do protocolo, a idéia apresentada por Chang&Maxemchuk, apesar de possuir um comportamento linear no número de mensagens enviadas, gera um grande ruído no ambiente ao enviar confirmações para cada uma das mensagens de "dados úteis". Além disso, apesar dos resultados apresentados considerarem apenas um ambiente composto de dois processos, tais valores tendem a formar uma curva de crescimento cada vez mais íngreme. Isto porque para cada novo processo integrante, duas novas mensagens são enviadas pela rede (uma com os dados úteis e outra - ACK - que serve como confirmação da primeira).

- **Cenário 23**:

– Aplicação: Jogo;

– Número de processos na rede: 3.

Protocolo	**Número de mensagens compartilhadas**										**Média Geral**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
Melhor Esforço	3538	3711	3326	3778	3723	3759	3746	3774	3859	3883	**3709.7**
Bluecast	3559	3448	3483	3434	3478	3407	3483	3450	3568	3470	**3478.0**

Tabela 5.44: Número de mensagens compartilhadas na rede para Cenário 23.

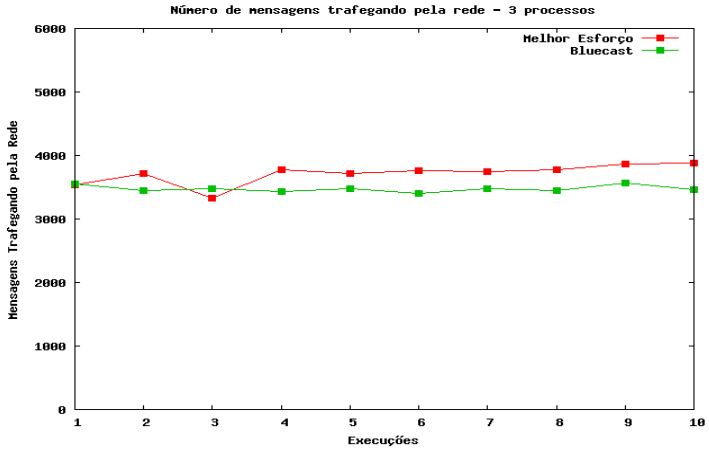


Figura 5.27: Número de mensagens compartilhadas para Cenário 23.

Os resultados apresentados neste cenário refletem o mesmo comportamento descrito no cenário anterior. Entretanto, é válido fazer uma ressalva a algumas das execuções apresentadas, pois, algumas vezes, o protocolo *Bluecast* apresenta resultados melhores que o algoritmo de melhor caso. Isto se deve basicamente a uma característica do tipo de aplicação que faz uso dos protocolos de ordenação. Uma vez que ambos os algoritmos empregam os conceitos de mestre/escravo, os seguintes pontos devem ser considerados:

- Todos os processos podem enviar e receber informações dentro do ambiente de comunicação;

- Todas as mensagens compartilhadas devem obrigatoriamente passar pelo processo mestre, que é quem conhece todos os componentes do grupo;

- Dados enviados por escravos acabam por ter *overhead* de uma mensagem (pois esta necessita primeiramente chegar ao mestre, para posteriormente ser compartilhada com os outros integrantes);

Dados enviados pelo processo mestre não geram sobrecarga no sistema pois não trafegam mais que uma vez pelo ambiente.

Assim sendo, as execuções apresentadas na Figura 5.27 que podem causar certo nível de estranheza, se explicam através dos pontos acima. Estas nada mais são que rodadas onde os nós escravos acabaram por ter uma participação mais ativa principalmente no "envio" de dados pela rede.

- **Cenário 24**:
  - Aplicação: Jogo;
  - Número de processos na rede: 4.

Protocolo	**Número de mensagens compartilhadas**										**Média Geral**
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
Melhor Esforço	4762	4878	4776	4756	4758	4758	4845	4609	4590	4874	**4760.6**
Bluecast	4870	4713	4735	4652	4613	4928	4634	4761	4814	4788	**4750.8**

Tabela 5.45: Número de mensagens compartilhadas na rede para Cenário 24.

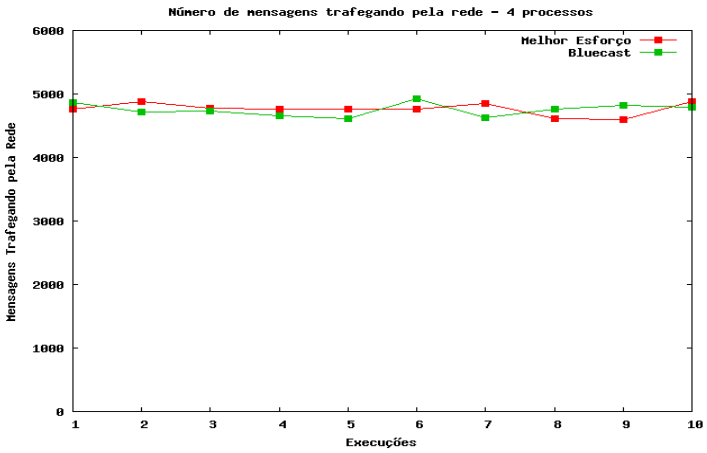


Figura 5.28: Número de mensagens compartilhadas para Cenário 24.

Com o aumento no número de integrantes da rede, um comportamento conhecido se repete. Um aumento linear no número de mensagens trocadas pode ser observado para

ambos os protocolos considerados. A mesma consideração feita para o cenário anterior pode aqui ser utilizada onde os resultados apresentados pelo algoritmo *Bluecast* superam os descritos para o de "melhor esforço".

Numa conclusão geral, nota-se que, para o protocolo *Bluecast* a inserção de ordem, suporte a falhas e garantia de entrega acaba por causar um impacto mínimo no número de mensagens trafegando pela rede, e muitas vezes - devido a características da aplicação utilizada - apresenta até resultados melhores que a solução apresentada para o "melhor caso".

5.3.4 Comportamento em condições de erro

Os cenários descritos anteriormente foram basicamente baseados em situações e execuções corretas. Entretanto, assim como em ambientes reais, aqui também é prudente que se considere a existência de falhas. Para tanto, as seguintes características foram consideradas em relação ao ambiente:

- Modelo de medição: Os tempos de cada uma das mensagens foram recuperados diretamente na camada mais interna do protocolo quando do momento de recebimento dos dados. Os valores foram considerados tendo-se como base a diferença no tempo de recebimento de duas mensagens consecutivas;
- Condição de parada: Entrega da mensagem de número 1300 a aplicação;
- Frequência de envio: 1 mensagem a cada 50 (ms);
- Resiliência = 10;
- 10 execuções para cada um dos cenários considerados;
- Falhas: Simulação de erros no envio das mensagens de número 200, 400, 600, 800, 1000 e 1200.
- **Cenário 25**:
  - Aplicação: Gerador de números aleatórios;
  - Protocolo: *Bluecast*;
  - Número de processos na rede: 2.

Os resultados apresentados pela Figura 5.29 descrevem o tempo de recepção (intervalo de tempo entre a recepção de duas mensagens consecutivas pela camada mais inferior do protocolo) entre cada uma das 1300 mensagens de difusão trocadas pelo sistema durante

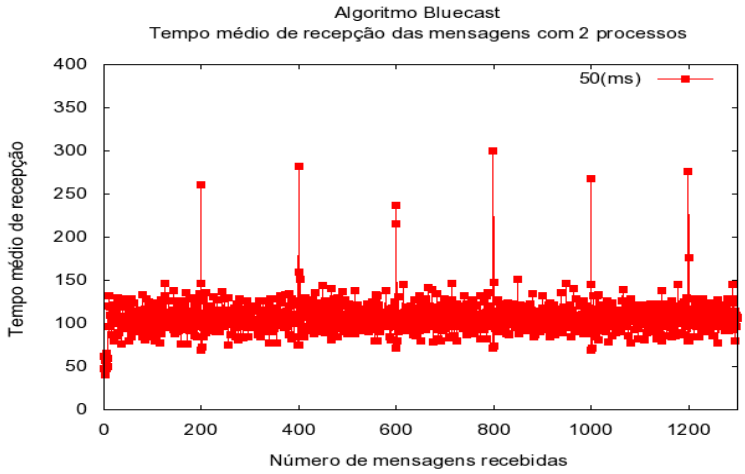


Figura 5.29: Tempo de recepção das mensagens para Cenário 25.

a execução do teste. Através deste nota-se claramente a influência no sistema quando da ocorrência de algum erro no envio ou recepção de determinadas mensagems. Com excessão das mensagens de número 200, 400, 600, 800, 1000 e 1200 (e algumas diretamente afetadas pelo comportamento destas), todas as outras possuem uma variação contínua no tempo de recepção. Tal comportamento é justificado através de características do próprio protocolo, que, quando nota a falta de determinada mensagem, envia uma requisição ao mestre para recuperar a mesma.

- **Cenário 26**:
  - Aplicação: Gerador de números aleatórios;
  - Protocolo: Chang&Maxemchuk;
  - Número de processos na rede: 2.

Devido as características de tempo para a retransmissão das mensagens, os valores apresentados pela Figura 5.30 são maiores que aqueles descritos para o protocolo *Bluecast* no cenário anterior. Entretanto, é necessário salientar que apesar da demora na recuperação de uma mensagem, a variável tempo de espera para retransmissão pode ser reconfigurada para um valor de tempo menor, de modo que esta ocorra com maior frequência. Por outro lado, um valor muito baixo pode causar uma sobrecarga na rede, já que os processos componentes podem não ter tempo suficiente para receber, validar e enviar a confirmação associada a uma determinada mensagem.

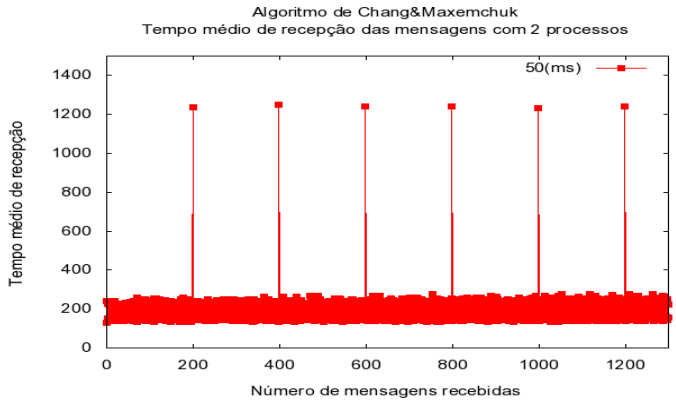


Figura 5.30: Tempo de recepção das mensagens para Cenário 26.

5.3.5 Comportamento sob influência do ambiente

O objetivo principal deste cenário é demonstrar o comportamento do ambiente *Bluetooth* quando seus componentes comunicamentes são separados por diferentes distâncias. No mundo real, tais resultados são de grande importância uma vez que, frequentemente diferentes dispositivos não estão fisicamente próximos no momento da formação de uma rede de compartilhamento de dados.

Assim como os resultados apresentados pelos cenários 25 e 26, os valores mensurados pelo cenário 27 levam em consideração o intervalo de tempo entre a recepção de duas mensagens sequenciais.

As tabelas seguintes contextualizam os tempos médios de recepção das mensagens quando se varia a distância entre os dispositivos. A primeira coluna representa a distância considerada entre os dois nós, enquanto as demais colunas descrevem o tempo médio de recepção em cada uma das dez execuções.

- **Cenário 27**:
  - Aplicação: Gerador de números aleatórios;
  - Protocolo: *Bluecast*;
  - Número de processos na rede: 2;
  - Distância considerada: 0(m) e 10(m).

A diferença entre os valores apresentados pelas tabelas 5.46 e 5.47 está no papel exercido pelos dispositivos utilizados em cada uma das rodadas de teste. Os resultados

Distância	Tempo médio de recepção das mensagens (Transmissores)										Média Geral (ms)
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
d = 0(m)	95.00	95.00	112.00	111.00	104.00	104.00	103.00	104.00	111.00	113.00	**105.2**
d = 10(m)	105.00	104.00	107.00	107.00	105.00	105.00	105.00	112.00	112.00	113.00	**107.5**

Tabela 5.46: Tempo médio de recepção das mensagens para Cenário 27 (Transmissores).

Distância	Tempo médio de recepção das mensagens (ms) - (Receptores)										Média Geral (ms)
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	
d = 0(m)	99.00	103.00	130.00	124.00	117.00	116.00	113.00	114.00	116.00	118.00	**115.0**
d = 10(m)	120.00	136.00	122.00	124.00	120.00	118.00	134.00	132.00	132.00	127.00	**126.5**

Tabela 5.47: Tempo médio de recepção das mensagens para Cenário 27 (Receptores).

apresentados na tabela 5.46 consideram somente os dados vindos dos dispositivos transmissores, isto é, responsáveis pelo envio das mensagens. Analogamente, 5.47 apresenta os dados resultantes quando os dispositivos considerados na análise são aqueles que desempenham o papel de receptores, e portanto não participam do envio de informações.

Os resultados apresentados nas tabelas 5.46 e 5.47 demonstram que de forma geral, os valores para os tempos médios de recepção das mensagens não variam muito apesar dos processos estarem ou não distantes fisicamente. Entretanto, na grande maioria dos casos, quando não separados por 10(m) de distância, os tempos tendem a ser ligeiramente melhores se comparados ao outro caso.

Valores notoriamente diferentes podem ser encontrados quando se associa aos processos os papéis de transmissor ou receptor. Neste contexto, uma diferença visível e persistente pode ser notada na comparação entre os dados apresentados na tabela 5.46 em relação aqueles listados na 5.47. Isto porque, como citado anteriormente, quando agindo como transmissor, um processo tem sua própria mensagem trafegando pela interface física de comunicação. Fato este que evita causar um *overhead* para o dispositivo.

Figura 5.31 apresenta dois gráficos considerando os tempos de recepção das mensagens pelos processos transmissores (lado direito) e receptores (lado esquerdo). Os dados apresentados consideram somente uma das 10 execuções, entretanto, o mesmo comportamento acaba por ser repetido a todas as outras rodadas de teste. Em uma análise superficial é possível notar algumas peculiaridades a cada uma das representações associadas aos papéis exercidos por cada um dos processos.

Pelo lado dos dispositivos transmissores é possível notar que existe um limiar mínimo para o tempo de recebimento das mensagens. Como justificado anteriormente, tal comportamento se deve a natureza do tráfego gerado, uma vez que os dados não trafegam pelo meio físico (neste caso o ar).

Resultados contrastantes ao primeiro podem ser observados no gráfico que representa o tempo de recepção das mensagens nos dispositivos que não enviam dados pela rede. Nestes, a distribuição do tempo de recepção de cada mensagem varia muito em razão do tempo, justamente por sofrer influência de um fator externo, uma vez que os dados trafegam através da camada física. Para determinadas mensagens, podem ser observados valores de tempo até menores que a média do gráfico anterior.

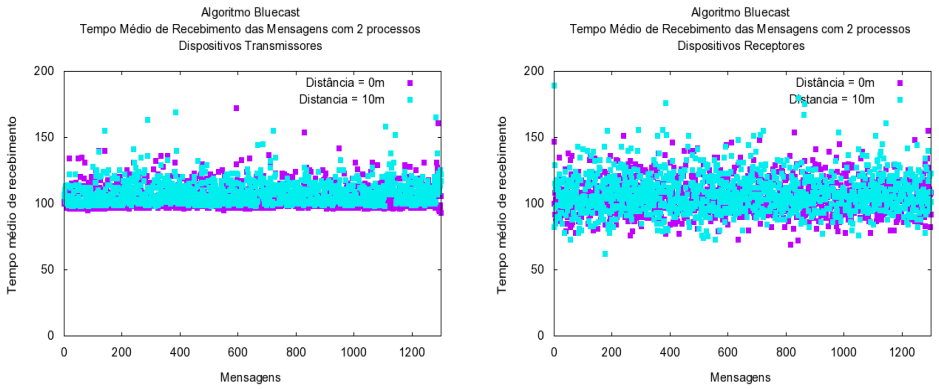


Figura 5.31: Tempo de recepção das mensagens para Cenário 27.

5.3.6 Análise Global

Através da análise inicial e dos 27 cenários de teste apresentados no decorrer deste capítulo é possível se observar algumas características e comportamentos peculiares a cada um dos protocolos que foram considerados. A viabiliade de implementação dos algoritmos foi separada basicamente pela capacidade de prover interatividade ao usuário de um ambiente com limitados recursos, diferencial este que ficou visível também durante todos os cenários de teste. Tal heterogeneidade é parcialmente justificada através da característica de cada um, que em seu determinado tempo foi baseado em diferentes características de cada ambiente.

Nos cenários de teste inicias, a principal variável a ser considerada foi o tempo que o protocolo ou algoritmo levava para processar uma determinada mensagem e entregá-la a camada de aplicação. Isto se reflete diretamente no nível de interatividade possível de ser provido - quanto menores os valores, melhor será a experiência final do usuário. Nos cenários de 1 a 7 o comportamento de cada trabalho foi analisado quando da variação de diversos valores que influenciam diretamente no desempenho do sistema, e o que se observou foi um sensível ganho ao se utilizar o algoritmo *Bluecast*. Na comparação direta

entre cenários equivalentes este sempre alcançou valores menores que o outro protocolo e bem próximos ao caso ótimo.

Além de se considerar ambientes reais, os protocolos foram também analisados sob influência de aplicações que requeriam a utilização de um tráfego contínuo de troca de dados entre os componentes da rede. Os resultados inerentes aos mesmos podem ser observados através dos cenários 8 a 14. Estes demonstram novamente um melhor aproveitamento do protocolo *Bluecast* em detrimento do outro algoritmo considerado. Entretanto, devido a características da aplicação de testes se observa um comportamento mais uniforme nos valores dos dados apresentados.

Tanto o protocolo *Bluecast* quanto o de melhor caso utilizam uma arquitetura baseada nos conceitos de mestre e escravo. Baseando-se nisso, é possível imaginar que o comportamento em cada um destes pode variar quando exposto a condições reais de teste. Os dados apresentados nos cenários 15 a 21 explicitam através de dados reais a variação no desempenho de cada um dos dispositivos. Nestes é possível notar sempre e claramente a superioridade dos processos mestres em relação aos escravos devido ao fator hierárquico que faz com que as mensagens enviadas pelo mestre não trafeguem através do enlace físico da rede, o que consequentemente evita a estes um *overhead*.

Outro importante fator considerado foi o tráfego gerado na rede quando ambos os algoritmos executam a mesma tarefa - execução até a entrega da mensagem de número 1300 à camada de aplicação. Os cenários 22 a 24 apresentam resultados que demonstram uma grande disparidade causada principalmente devido a uma grande variação nos dados apresentados pelos protocolos *Bluecast* e o proposto por Chang&Maxemchuk em [12]. O número de mensagens enviadas pelo segundo algoritmo chega a ser aproximadamente 40% maior que no primeiro, fator este que pode vir a causar uma sobrecarga na rede quando da utilização com aplicações que demandem tráfego intenso.

Os resultados apresentados nos cenários 25 e 26 focam na análise do comportamento dos protocolos quando expostos a condições adversas que acabam por culminar com a perda de mensagens. O objetivo principal de tais cenários é medir o tempo necessário para a recuperação e/ou retransmissão de uma mensagem em relação ao primeiro envio da mesma. Devido a características inerentes a implementação dos protocolos, os valores explicitados através das Figuras 5.29 e 5.30 apresentam uma grande diferença no desempenho de cada um. Isto porque, o algoritmo *Bluecast* trabalha através de requisições para as mensagens perdidas, enquanto a idéia apresentada por Chang&Maxemchuk se baseia em *timeouts* e retransmissões de mensagens até que se receba uma confirmação. Contudo, somente tal fator não pode ser utilizado como justificativa para tal comportamento, uma vez que a diminuição no tempo de retransmissão por causar uma sobrecarga de dados inúteis trafegando na rede, para o caso deste segundo protocolo.

Finalmente, de modo a simular um possível cenário real de utilização dos protocolos,

separou-se os dispositivos a 10 metros de distância e novos dados foram coletados. Os resultados são descritos através do cenário 27, onde é possível se observar que a tal distância o impacto é mínimo na execução do protocolo, e uma vez aplicado a um ambiente real, o comportamento da aplicação será aquele observado quando os elementos estão fisicamente próximos.

Capítulo 6

Conclusões

A associação das áreas *Bluetooth*, ambientes de tempo real e ordenação de mensagens traz novos desafios a problemas que antes eram estudados separadamente. Os protocolos e algoritmos apresentados no decorrer do capítulo 5 exemplificam tais estudos assim como a falta de relacionamento e contextualização dos mesmos a cenários e ambientes reais. Dessa forma, concentra-se na junção destes fatores o principal foco do trabalho aqui apresentado, quando se associa problemas teóricos a aplicações reais - algumas atualmente possíveis de serem desenvolvidas e outras já encontradas no mercado.

Para tornar possível a utilização de conceitos tão heterogêneos, uma comparação inicial entre as diversas soluções atualmente existentes foi executada. Os resultados demonstraram que os protocolos, em quase sua totalidade pecavam em algum detalhe que impossibilitariam sua aplicação ao ambiente desejado. Dessa forma, o melhor dos protocolos analisados foi selecionado para a execução de testes reais junto a outros dois algoritmos - melhor caso e um protocolo específico para o ambiente considerado por este trabalho.

Analisando de forma geral, a principal contribuição apresentada por este trabalho é o desenvolvimento de um novo protocolo de ordenação de mensagens para comunicações por difusão - com peculiaridades já citadas anteriormente. Mais especificamente, este se destaca por ter sido desenvolvido com o foco num ambiente de tempo real com recursos limitados.

Apesar de uma idéia inédita e inovadora, o desenvolvimento deste trabalho foi permeado por desafios imprevisíveis nas fases de análises teóricas. A dificuldade no gerenciamento e manipulação de dispositivos com poucas funcionalidades e baixa capacidade de processamento se acentua no momento da contextualização com a classe de problemas e o ambiente considerado. Como forma de superar determinadas limitações apresentadas no decorrer da fase de desenvolvimento, ferramentas de depuração e análise de dados foram desenvolvidas e outras reutilizadas de forma a maximizar o processo de obtenção dos resultados. Contudo, algumas outras diversas limitações necessitaram ser absorvidas e superadas para que o trabalho chegasse a sua conclusão:

- **Simulações**: O desenvolvimento de sistemas para dispositivos embarcados muitas vezes se baseia inicialmente em simuladores e/ou emuladores que reproduzem com certo grau de fidelidade o dispositivo físico. No caso da tecnologia *Bluetooth*, os simuladores existentes atualmente não levam em conta características e limitações físicas (*hardware*) dos aparelhos, entretanto, mesmo sem representar fielmente o dispositivo real, uma prototipação foi inicialmente desenvolvida junto a um emulador [1], e posteriormente descartada na fase de depuração e execução dos testes justamente por não ser sensível aos detalhes citados anteriormente.;

- **Especificação *Bluetooth***: O protocolo descrito no capítulo 4 nasceu a partir de uma observação feita nos dispositivos móveis que serviram aos testes iniciais. A grande maioria dos aparelhos analisados, apesar de implementar a pilha de protocolos *Bluetooth*, colocava restrições de *hardware* a algumas variáveis do sistema, como:

  - Número máximo de conexões: De acordo com a especificação do protocolo *Bluetooth*, cada dispositivo pode fazer um máximo de 7 conexões. Entretanto, a grande maioria dos aparelhos analisados reduz este número a 3 ou 4 (devido as características e aplicações nas quais o dispositivo é utilizado, como por exemplo telefonia, transferência de arquivos, etc). Tal restrição reduz as possibilidades de utilização de algoritmos mais elaborados, uma vez que estes podem requerer recursos não disponíveis num dado *hardware*;
  - Criação de redes *scatternets*: Além do número de conexões, outro fator importante é a impossibilidade de criação de *scatternets* (capítulo 2) nos dispositivos mais simples. Somente aqueles com recursos mais avançados estão aptos a criação deste tipo de rede de comunicação mais elaborado que o modelo comum - *piconet*.

- **Analisadores**: A execução dos testes e posterior coleta dos dados em cada um dos aparelhos gera um grande conjunto de informações que necessitam ser analisadas. Para cada um dos cenários considerados no capítulo 5 deste trabalho, um diferente analisador foi desenvolvido como forma de se processar as informações provenientes da execução.

Com o objetivo de suplantar as limitações apresentadas na codificação e execução dos protocolos já existentes na literatura, o algoritmo *Bluecast* foi desenvolvimento de forma a suportar a solução do problema descrito, sem no entanto, descartar as características dos ambientes e dispositivos fisicamente presentes.

6.1 Trabalhos Futuros

O ponto de maior lacuna durante todo o processo de desenvolvimento do trabalho foi a falta de ferramentas específicas para o ambiente. Emuladores do protocolo *Bluetooth* disponíveis hoje não possuem opções de configuração ou considerações em relação a fatores externos como: distanciamento, ruídos, capacidade de processamento, etc. Baseando-se nisso, como trabalho futuro sugere-se a criação de um ambiente de testes que possibilite tal tipo de desenvolvimento de modo a encurtar o processo de implantação de novas aplicações junto aos dispositivos reais.

O conjunto de testes aqui apresentado é suficiente para se obter conclusões claras a respeito do comportamento de determinados protocolos quando submetidos a cenários específicos. Entretanto, muitas outras opções de testes e aferições não foram executadas por fugirem ao escopo. Desse modo, é prudente como um trabalho futuro haja um aumento no conjunto de cenários de teste avaliados. Isto pode permitir se enxergar maneiras de otimizar o protocolo aqui apresentado ou ainda qualquer um dos outros algoritmos existentes na literatura.

Apêndice A

O Protocolo de *Chang & Maxemchuk*

A.1 Introdução

Dentre os protocolos de ordenação total e *broadcast* confiável mais conhecidos atualmente, um deles se destaca pelo uso consistente dos conceitos de ordenação de mensagem e pela fácil adaptação a diferentes ambientes computacionais.

Proposto em 1984 por Chang e Maxemchuk, o protocolo apresentado em [12] utiliza as idéias de grupos de processos [33], *broadcast/multicast* confiável e suporte a falhas de modo a prover um ambiente para o envio e recebimento de mensagens com garantia de ordem e entrega. Além desta, outra característica marcante da proposta é a existência de um processo sequenciador temporário e dinâmico, responsável pelo envio das confirmações de recebimento das mensagens aos outros nós do grupo.

A.1.1 Arquitetura

A idéia macro do protocolo é básica, composta principalmente por uma camada, chamada pelo autor de *broadcast protocol* e localizada entre os enlaces de aplicação e transmissão. O funcionamento da camada de *broadcast* é totalmente ancorado no conceito de filas de mensagens, tanto para o recebimento quanto para o envio. Dessa forma, garante-se que as mensagens recebidas e enviadas pelo processo são tratadas sequencialmente, facilitando de certa maneira a ordenação e detecção de erros pelo próprio protocolo. Outro fato relevante ao protocolo fica por conta de uma característica no envio das mensagens, onde:

- Para uma dada mensagem *m0* enviada por P, *m1* somente será remetida por P após o recebimento da confirmação de entrega de *m0* por todos os outros processos do grupo.

Figura A.1 apresenta uma descrição de alto-nível da hierarquia proposta por Chang e Maxemchuk.

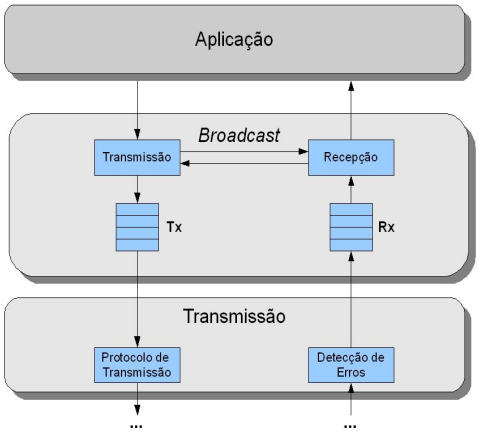


Figura A.1: Arquitetura do protocolo proposto em [12]

Através de determinadas premissas é possível se obter as principais características que delineiam o protocolo:

- Todas as mensagens enviadas e recebidas pelo protocolo são consideradas e tratadas como pacotes únicos, sem relação direta e/ou dependência com outros pacotes;
- Mensagens com erros de transmissão são descartadas;
- Retransmissões de mensagens perdidas (por qualquer que seja o motivo) é responsabilidade do protocolo;
- Quando do recebimento de uma mensagem, a mesma não é diretamente entregue à camada de aplicação. Esta permanece inválida até que o protocolo possa garantir que a mesma foi recebida por todos os processos do grupo;
- O protocolo implementa a noção de um sequenciador de mensagens. Isto é, um processo com uma função especial: definir a ordem de entrega das mensagens.

A.1.2 Execução

O funcionamento do protocolo é dividido em diferentes fases de modo a construir um ambiente estável para a troca de mensagens.

- **Fase de reforma**: Seguindo a especificação do protocolo, anteriormente à fase de troca de mensagens, é necessário se construir o ambiente no qual as mesmas

serão enviadas/recebidas. Para tanto, é apresentada uma série de passos e tarefas justamente com esta finalidade: Construção de um ambiente propício para o compartilhamento de informações utilizando ordenação total.

A fase de reforma é executada quando do início do protocolo (para a construção da rede) e também quando da detecção de uma falha. Ao final desta, tem-se como objetivo a criação de uma lista contendo os processos corretos presentes no ambiente. Ao mesmo tempo, esta lista é utilizada como uma referência (compartilhada por todos os processos) para a ordem de passagem do *token*. Ao final da fase de reforma, a lista gerada deve possuir as duas seguintes propriedades:

- Somente uma lista válida pode existir; e
- Nenhuma das mensagens entregues na lista anterior deve ser perdida.

O algoritmo gerador da lista de *tokens*, consequentemente da fase de reforma, prossegue da seguinte maneira:

- Quando da detecção de uma falha por determinado processo, este inicia a fase de reforma de modo a gerar uma nova lista válida de *tokens*. Cada um dos processos presentes no ambiente é convidado a participar desta nova lista, que por sua vez só se tornará válida se passar nos seguintes testes:
  - **Teste de maioria**: Uma nova lista, para se tornar válida deve ser aceita pela maioria dos processos presentes no ambiente. Dessa forma, garante-se que somente uma nova lista será validada pelos processos;
  - **Teste de sequência**: Garante que um processo só aceitará uma lista se esta possuir um número de sequência maior que o da última lista conhecida por ele. Se algum processo falha no teste de sequência, este diz ao nó originador (que iniciou a fase de reforma) seu número de sequência, que será levado em conta para a geração da nova lista;
  - **Teste de resiliência**: Garante que não será perdida nenhuma das mensagens já "entregues" pela lista antiga. Num ambiente com resiliência L, pelo menos $L+1$ processos devem receber uma determinada mensagem de forma que em uma nova lista, se um destes $L+1$ processos está presente, nenhuma das mensagens antigas serão perdidas. Para que o sistema passe pelo teste de resiliência é necessário que se conheça o número de sequência da última lista válida assim como a última mensagem entregue quando da utilização desta. Esta operação, normalmente é feita da seguinte maneira:

· Cada um dos processos que aceitam a nova lista dizem ao nó originador o valor de sequência da última mensagem válida entregue por eles;

· A última mensagem válida da lista com o maior número de sequência, é utilizada no teste de resiliência;

· O teste de resiliência é considerado bem sucedido se a nova lista contém o processo que enviou a mensagem com o maior número de sequência ou um dos L outros sites que conhecem tal mensagem, de forma que esta poderia ser recuperada aos processos que não a receberam.

Após o término da fase de reforma, um dos processos é escolhido como *token site*, responsável por sequenciar as mensagens de *broadcast* até que o *token* seja passado para o próximo processo da lista.

- **Fase normal**: É a fase de troca de mensagens em si. O protocolo garante que cada mensagem enviada é recebida por todos os processos válidos que integram o ambiente. A fase de reforma é invocada todas as vezes que os processos de um grupo estão utilizando listas de *tokens* diferentes ou ainda, quando alguma das seguintes mensagens não tem sua confirmação de recebimento enviada:

  – Mensagens de *broadcast* necessitam do envio explícito de uma mensagem de confirmação (ACK);

  – Sempre que requisitada, a retransmissão de uma mensagem perdida deve ser executada;

  – Quando o *token* é passado para o próximo site da lista, este deve confirmar explicitamente o recebimento do mesmo.

  Se após R tentativas de retransmissão a mensagem não é recebida, então considera-se que o processo falhou.

  Considerando as premissas acima, a execução normal do protocolo prossegue através de subfases, onde os processos armazenam as seguintes informações:

  – A versão corrente da lista de *tokens* que está lendo utilizada;

  – Uma lista contendo o identificador da próxima mensagem a ser enviada por cada um dos outros processos componentes da rede - a utilização desta lista evita que o recebimento de uma mesma mensagem seja confirmada mais de uma vez;

  – O valor do identificador da próxima mensagem de *broadcast* a ser recebida, de modo que cada processo consiga ordenar as mensagens recebidas utilizando este identificador.

A manutenção da fase de envio/recebimento de mensagens é baseada principalmente em três funcionalidades diferentes:

- **Transmissão**: Um processo retransmite uma determinada mensagem até que possua uma confirmação de recebimento enviada pelo processo sequenciador (processo com o *token*). Se após R retransmissões, a confirmação não é recebida, considera-se que houve uma falha;

- **Sequenciamento**: A principal função do processo que possui o *token* é o sequenciamento das mensagens. Consequentemente, este mesmo processo é o responsável pelo envio das mensagens de confirmação (ACK). Além disso, como o protocolo propõe um esquema de sequenciamento dinâmico, a cada mensagem de confirmação enviada o *token* é passado para o próximo processo da lista, que por sua vez, passa a ter essa responsabilidade. Dessa forma, tem-se um sequenciador para cada mensagem de *broadcast* e se evita que o sistema caia no caso de falha do sequenciador - para sistemas/ambientes onde o envio de mensagens possui um intervalo muito grande, o *token* pode também ser passado explicitamente através de uma mensagem específica após um tempo R;

- **Entrega**: Após uma mensagem ter sido sequenciada, consequentemente sua confirmação ter sido enviada, a entrega à camada de aplicação só ocorre após L passagens do *token*. Isto garante que $L+1$ processos receberam a mensagem. E, no caso de uma falha, a fase de reforma pode ser concluída - e recuperadas todas as antigas mensagens que foram entregues - desde que não mais que L processos estejam faltosos.

A.1.3 Suporte a falhas

A recuperação às falhas providas pelo protocolo proposto por Chang & Maxemchuk baseia-se principalmente em duas características distintas do protocolo:

- **Resiliência**: A replicação das mensagens de *broadcast* para os processos componentes da rede é garantida principalmente através do conceito de resiliência (L).

  Num sistema L-Resiliente, uma mensagem só é entregue a camada de aplicação após ser recebida por $L+1$ processos. Partindo-se deste conceito, o sistema passa a suportar que até L processos falhem ao mesmo tempo. Neste pior caso, pelo menos um processo que participa da fase de reforma possui todas as mensagens entregues pelos nós componentes da lista anterior, de modo que as mensagens faltantes nos outros processos possam ser facilmente recuperadas pelo protocolo.

  Entretanto, claramente é possível se notar uma espécie de troca entre o suporte a falhas e o tempo despendido para a entrega das mensagens na camada de aplicação.

Isto é, quanto maior a resiliência - consequentemente suporte a um número maior de falhas - maior também é o tempo que as informações levam para terem sua ordem total garantida e serem entregues a aplicação.

- **Sequenciador móvel**: A implementação de um sequenciador móvel vem a suprir as necessidades de dois problemas que ocorrem em protocolos baseados em sequenciadores fixos:
  - Impossibilidade de saber se todos os outros processos receberam as mensagens enviadas; e
  - No caso de falha do nó sequenciador, este pode ter sequenciado um determinado número de mensagens que não foram recebidas pelo outros processos componentes do grupo.

  No protocolo, para que um nó aceite o *token* e se torne o sequenciador das mensagens, este primeiro necessita ter recebido todas as mensagens sequenciadas até aquele momento. Desse modo, após L passagens de *token*, garante-se que pelo menos $L+1$ processos receberam uma mesma mensagem. Logo, não é mais necessário retê-la indefinidamente visando uma futura retransmissão.

Referências Bibliográficas

[1] Bluetooth impronto simulator. http://www.rococosoft.com/, 2009.

[2] E-donkey network. http://www.emule-project.net/, 2009.

[3] Ieee 802.11 - wireless local area networks - the working group for wlan standards. http://www.ieee802.org/11/, 2009.

[4] Kazaa network. http://www.kazaa.com/, 2009.

[5] T. Abdelzaher, A. Shaikh, F. Jahanian, and K. Shin. Rtcast: Lightweight multicast for real-time process groups. *Real-Time and Embedded Technology and Applications Symposium, IEEE*, 0:250, 1996.

[6] T. Alrabiah and T. Znati. Delay-constrained, low-cost multicast routing in multimedia networks. *J. Parallel Distrib. Comput.*, 61(9):1307–1336, 2001.

[7] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella. The totem single-ring ordering and membership protocol. *ACM Trans. Comput. Syst.*, 13(4):311–342, 1995.

[8] Infrared Data Association. Specification of object exchange protocol. version 1.3, January 2003.

[9] V. C. Barbosa. *An Introduction to Distributed Algorithms*. MIT Press, Cambridge, MA, USA, 1996.

[10] A. Basu, B. Charron-Bost, and S. Toueg. Simulating reliable links with unreliable links in the presence of process crashes. In *WDAG '96: Proceedings of the 10th International Workshop on Distributed Algorithms*, pages 105–122, London, UK, 1996. Springer-Verlag.

[11] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.

[12] J. Chang and N. F. Maxemchuk. Reliable broadcast protocols. *ACM Trans. Comput. Syst.*, 2(3):251–273, September 1984.

[13] H. Chen, T. Finin, and A. Joshi. An ontology for context-aware pervasive computing environments. *Workshop on Ontologies and Distributed Systems*, August 2003.

[14] M. Chou and R. S. Chang. Blueline: A distributed bluetooth scatternet formation and routing algorithm. In *Parallel and Distributed Computing and Networks*, pages 153–158, 2004.

[15] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri. *Scheduling in Real-Time Systems*. John Wiley & Sons, LTD, London, UK, 2002.

[16] F. Cristian. Synchronous atomic broadcast for redundant broadcast channels. *Real-Time Systems*, 2(3):195–212, 1990.

[17] F. Cristian. Asynchronous atomic broadcast. *IBM Technical Disclosure Bulletin*, 33(9):115–116, 1991.

[18] F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to byzantine agreement. *Inf. Comput.*, 118(1):158–179, 1995.

[19] F. Cristian, S. Mishra, and G. Alvarez. High-performance asynchronous atomic broadcast. *Distributed System Engineering Journal*, 33(2):109–128, 1997.

[20] X. Défago and P. Schiper, A.; Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 36(4):372–421, 2004.

[21] R. Giaffreda, A. Karmouch, A. Jonsson, A. M. Karlsson, M. I. Smirnov, R. Glitho, Ericsson Sweden, and Teliasonera Sweden. Context-aware communication in ambient networks. 2004.

[22] R. Guerraoui and A. Schiper. Total order multicast to multiple groups. In *ICDCS '97: Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS '97)*, pages 578–585, Washington, DC, USA, 1997. IEEE Computer Society.

[23] V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical report, Ithaca, NY, USA, 1994.

[24] A.S. Huang and L. Rudolph. *Bluetooth Essentials for Programmers*. Cambridge University Press, New York, NY, USA, 2007.

[25] P. Jalote. *Fault Tolerance in Distributed Systems*. Prentice Hall Press, Englewood Cliffs, NJ, USA, 1994.

[26] D. Johnson. Hardware and software implications of creating bluetooth scatternet devices. In *IEEE Conference in Africa (AFRICON)*, volume 1, pages 211–215, September 2004.

[27] M. F. Kaashoek, A. S. Tanenbaum, and S. F. Hummel. An efficient reliable broadcast protocol. *SIGOPS Oper. Syst. Rev.*, 23(4):5–19, 1989.

[28] M.F. Kaashoek and A. S. Tanenbaum. An evaluation of the amoeba group communication system. In *ICDCS '96: Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS '96)*, pages 436–447, Washington, DC, USA, 1996. IEEE Computer Society.

[29] D. Kammer, G. McNutt, B. Senese, and J. Bray. *Bluetooth Application Developer's Guide: The Short Range Interconnect Solution*. Syngress Publishing, Inc., Rockland, MA, USA, 2002.

[30] S. Khanvilkar, F. Bashir, D. Schonfeld, and A. Khokhar. Multimedia networks and communication index. 2008.

[31] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[32] A. D. Kshemkalyani and M. Singhal. *Distributed Computing - Principles, Algorithms and Systems*. Cambridge University Press, New York, NY, USA, 2008.

[33] L. Lamport. The implementation of reliable distributed multiprocess systems. *Computer Networks*, 2:95–114, 1978.

[34] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[35] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[36] P. A. Laplante. *Real-Time Systems: Design and Analysis*. IEEE Press. John Wiley & Sons, LTD., Danvers, MA, USA, 3rd edition, 2004.

[37] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., San Francisco, USA, 1996.

[38] D. F. Macedo, L. B. Oliveira, and A. A. F. Loureiro. Integrando redes overlay em redes de sensores sem fio. In *5o. Workshop de Comunicação Sem Fio (WCSF)*, pages 190–198, 2003.

[39] P. M. Melliar-Smith, L. E. Moser, and V. Agrawala. Broadcast protocols for distributed systems. *IEEE Trans. Parallel Distrib. Syst.*, 1(1):17–25, 1990.

[40] B. A. Miller and C. Bisdikian. *Bluetooth Revealed: The Insider's Guide to an Open Specification for Global Wireless Communications*. Upper Saddle River NJ: Prentice Hall, New Jersey, USA, 2001.

[41] M. Miller. *Discovering Bluetooth*. SYBEX Inc., Alameda, CA, USA, 2001.

[42] A. Mostefaoui, E. Mourgaya, and M. Raynal. An introduction to oracles for asynchronous distributed systems. *Future Gener. Comput. Syst.*, 18(6):757–767, 2002.

[43] Q. Ni and M. Sloman. An ontology-enabled service oriented architecture for pervasive computing. In *ITCC '05: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05)*, volume 2, pages 797–798, Washington, DC, USA, 2005. IEEE Computer Society.

[44] C. Pamuk and E. Karasan. Sf-devil: Distributed bluetooth scatternet formation algorithm based on device and link characteristics. In *ISCC '03: Proceedings of the Eighth IEEE International Symposium on Computers and Communications*, pages 646–651, Washington, DC, USA, 2003. IEEE Computer Society.

[45] D. Reading-Picopoulos and A. A. Abouzeid. A bluetooth scatternet formation algorithm for networks with heterogeneous device capabilities. In *ICOIN*, pages 295–305, 2003.

[46] J. Rey, B. Mathieu, D. Lozano, S. Herborn, K. Ahmed, S. Schmid, S. Göbbels, F. Hartung, and M. Kampmann. Media aware overlay routing in ambient networks. In *Proceedings of PIMRC05*, page 5, Berlin, Germany, Sep 2005.

[47] Bluetooth Special Interest Group (SIG). Bluetooth protocol - specification of the bluetooth system. core package. version 1.2. http://www.bluetooth.com/, November 2003.

[48] A. S. Tanenbaum and M. v. Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.

[49] P. Urban, I. Shnayderman, and A. Schiper. Comparison of failure detectors and group membership: Performance study of two atomic broadcast algorithms. In *In Proc. International Conf. on Dependable Systems and Networks*, pages 645–654, 2003.

[50] P. Veríssimo. Real-time data management with clock-less reliable broadcast protocols. In *Workshop on the Management of Replicated Data*, pages 20–24, 1990.

[51] V.P. Verma and A.A. Chandak. Distributed bluetooth scatternet formation algorithm. *IEEE Global Telecommunications Conference*, 3:1274–1278, August 2003.

[52] P. Veríssimo and M. Raynal. Time in distributed system models and algorithms. In *Advances in Distributed Systems, Advanced Distributed Computing: From Algorithms to Systems*, pages 1–32, London, UK, 1999. Springer-Verlag.

[53] R. Williams. *Real-Time Systems Development*. Butterworth-Heinemann - Elsevier, Oxford University, UK, 2006.