

# Um Algoritmo para Configuração de *Scatternets* Bluetooth

Wanderley Ravagnani Junior, Jacques Wainer, Edmundo R. M. Madeira

Instituto de Computação  
Universidade Estadual de Campinas  
CP 6176 13083-970 Campinas-SP-Brasil  
{wanderley.junior, wainer, edmundo}@ic.unicamp.br

**Resumo.** Bluetooth é uma tecnologia de rádio que permite dispositivos eletrônicos comunicarem a curta distância através de redes ad-hoc sem fio. Este artigo apresenta um algoritmo para configurar *scatternets* Bluetooth, na tentativa de satisfazer as necessidades específicas de comunicação entre cada par de dispositivos da rede. O artigo também mostra algumas simulações para testar a eficiência deste algoritmo. Os resultados mostraram que 99% das redes testadas foram satisfeitas.

**Abstract.** Bluetooth is a radio technology which allows electronic devices to communicate in short distances through ad-hoc wireless networks. This paper presents an algorithm developed to configure Bluetooth *scatternets*, in order to satisfy the specific communication requirements of each pair of network devices. The paper also shows some simulations to test the efficiency of this algorithm. The results showed that 99% of the tested networks were satisfied.

## 1. Introdução

Bluetooth é uma especificação de um padrão aberto para comunicação sem fio, de curto alcance (10 metros a 0 dBm [3]) e baixo custo entre dispositivos, através de conexões de rádio. Criada em 1998 por um consórcio das maiores empresas de telecomunicação e computação do mundo, a tecnologia Bluetooth permite que os usuários conectem uma ampla variedade de dispositivos fixos (PCs, impressoras, mouse, teclado, scanners, etc.) e móveis (laptops, PDAs, telefones celulares, etc.) de uma forma bastante simples, sem a necessidade de utilizar cabos de ligação, possibilitando a formação de redes ad-hoc.

Esse novo padrão opera na faixa de frequência ISM (*Industrial, Scientific, Medical*) centrada em 2,45 GHz e visa facilitar as transmissões de dados e voz em

tempo real, assegurar a proteção contra interferência e garantir a segurança dos dados transmitidos [1].

O Bluetooth visa a criação de um sistema de rádio, onde os dispositivos se comunicam formando uma rede denominada *piconet*, na qual podem existir até oito dispositivos interligados, sendo que esses dispositivos compartilham um mesmo canal (FH Bluetooth *Channel*). Para controlar o tráfego nesse canal, um desses dispositivos se torna o mestre da *piconet* e os demais se tornam escravos. Para evitar a colisão devido a múltiplas transmissões de dispositivos escravos, o mestre utiliza a técnica de *polling*, na qual somente um dispositivo pode transmitir por vez.

Tipicamente, em aplicações Bluetooth, várias *piconets* podem se sobrepor ou coexistir numa mesma área, formando um sistema ad-hoc denominado *scatternet*, onde as *piconets* que compõem este sistema não devem ser sincronizadas [1]. Nas *scatternets*, um dispositivo (nó) pode participar de diferentes *piconets* utilizando a técnica de *Time Division Multiplexing* (TDM). Isto é, um nó pode participar seqüencialmente de diferentes *piconets*, sendo que este pode somente estar ativo em uma *piconet* por vez [4]. É importante observar que se um nó é o mestre em uma *piconet*, ele não pode ser, de forma alguma, o mestre em outra, devido ao fato da sincronização dentro de uma *piconet* ser feita utilizando os parâmetros do mestre. Dessa forma, as *piconets* de uma *scatternet* podem se comunicar compartilhando um(ou mais) nó(s) com as outras *piconets*, como ilustra a Figura 1. O nó compartilhado é denominado **ponte**.

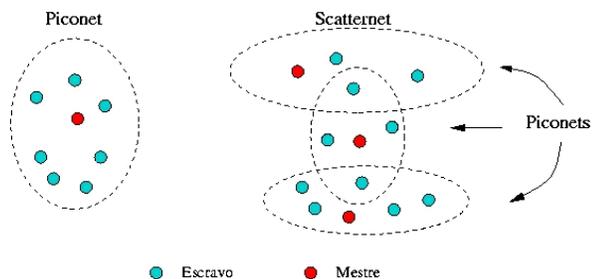


Figura 1 - Tipos de redes formadas entre dispositivos Bluetooth.

Um canal Bluetooth possui um throughput de 1 Mbit/s entre os dispositivos pertencentes à mesma *piconet* [2]. Porém ainda não sabemos o valor do throughput entre *piconets*, já que é necessário saber o tempo que um nó leva para mudar de uma *piconet* para outra e quanto tempo um nó fica ativo em cada *piconet*. Poucos trabalhos foram encontrados referenciando este assunto, como é o caso de [7] e [8], entretanto, o tráfego entre *piconets* não foi diretamente abordado.

A formação de uma *scatternet* Bluetooth é um assunto importante a ser ressaltado, pois o tempo e o modo como essa formação é feita podem afetar o desempenho da rede. Esse problema foi tratado por [5] e [6], onde estes partem de um número de nós  $N$  criando uma topologia de rede de forma distribuída. Porém, ambos não consideram a necessidade de comunicação entre os dispositivos pertencentes à rede, o que pode levar a uma perda de desempenho. Por exemplo, numa rede gerada pelos trabalhos acima citados, onde um laptop e uma impressora, que precisam se comunicar a uma taxa muito alta, podem estar dispostos em *piconets* diferentes, distantes uma da outra, fazendo com que o tráfego dessa *scatternet* fique comprometido.

Devido às restrições de largura de banda e tendo em vista a necessidade de um dispositivo  $i$  qualquer, pertencente a uma *scatternet*  $S$ , comunicar com qualquer outro dispositivo  $j$  pertencente a mesma *scatternet*  $S$  a uma taxa  $T_{ij}$ , a proposta desse artigo é expor e mostrar a eficiência de um algoritmo que projetamos para configurar essa rede Bluetooth (*scatternet*  $S$ ) na tentativa de satisfazer a necessidade de comunicação ( $T_{ij}$ ) de todo dispositivo  $i, j$ , onde  $i \neq j$ . A idéia desse algoritmo é dispor os dispositivos na rede de tal forma que os dispositivos, que mais necessitam comunicar entre si, pertençam a uma mesma *piconet*.

Este artigo está estruturado da seguinte maneira: A Seção 2 refere-se à modelagem da nossa proposta, a Seção 3 descreve o gerador de redes aleatórias Bluetooth utilizado para testar a eficiência do nosso algoritmo, a Seção 4 descreve o algoritmo guloso, a Seção 5 ilustra os

resultados obtidos e finalmente a Seção 6 traz a conclusão e trabalhos futuros.

## 2. Modelagem

Para modelar nossa proposta, algumas suposições devem ser feitas para a configuração das redes Bluetooth de forma a satisfazer as necessidades de comunicação acima descritas:

- A *scatternet* possui  $N$  dispositivos, e os  $N$  dispositivos estão próximos entre si (10 metros).
- A *scatternet* é totalmente conexa, ou seja, todas as *piconets* pertencentes a *scatternet* podem comunicar entre si. Isso normalmente faz com que não haja um grande número de *piconets* numa mesma área, reduzindo assim a interferência o que, segundo [9] e [10], melhora o desempenho da rede;
- Como o throughput entre *piconets* ainda é uma questão aberta, nós assumimos que o throughput entre duas *piconets* quaisquer pertencentes a uma *scatternet* é no máximo de 100 kbits/s, pois suportaria, além de tráfego em rajadas, a transmissão de um canal de voz que requer uma taxa de 64 kbits/s [3].

Com os itens acima descritos, poderemos agora mostrar como geramos as redes para testar a eficiência do algoritmo guloso, bem como expor em detalhes como esse algoritmo funciona.

## 3. Gerador de redes aleatórias Bluetooth

Em nosso projeto desenvolvemos um algoritmo que gera redes Bluetooth aleatoriamente e que são de alguma forma factíveis (ou seja, cada rede tem pelo menos uma solução) em relação às necessidades de comunicação entre os dispositivos da rede. As redes geradas servirão para testar a eficiência do algoritmo guloso (que será apresentado posteriormente). A rede aleatória com  $N$  dispositivos é gerada da seguinte forma:

1. Os  $N$  dispositivos são dispostos aleatoriamente dentro de um vetor VET;
2. O número de dispositivos dentro de cada *piconet* é escolhido aleatoriamente (no mínimo três e no máximo oito), de tal forma que a soma do número de dispositivos dentro de todas as *piconets* da rede seja  $N$ .
3. As *piconets* são preenchidas sequencialmente com os elementos do vetor VET.

Em seguida criamos uma estrutura para armazenar as necessidades de comunicação dos dispositivos na rede. Para tanto, utilizamos uma matriz triangular inferior de throughputs (MT), onde para cada par de dispositivos  $i, j$  (onde  $i > j$  e  $i \neq j$ ),  $MT[i][j]$  significa o quanto  $i$  e  $j$  necessitam se comunicar na rede, o que chamamos de **throughput desejado** entre  $i$  e  $j$ . Finalmente preenchamos a matriz MT da seguinte forma:

- 1.1 Para cada comunicação (dentro de uma *piconet* ou entre duas *piconets*) da *scatternet*, o valor do throughput utilizado (TU) será um valor aleatório que varia de 40% a 100% de seu throughput máximo (1Mbit/s *intra-piconet* ou 100 Kbits/s *inter-piconet*).
- 1.2 Calcula-se o número de enlaces que a comunicação possuirá, onde um enlace, aqui, representa a conexão entre dois quaisquer dispositivos  $i$  e  $j$  (onde  $i \neq j$ ):

- Se a comunicação for *inter-piconet*, o número de enlaces será igual ao número de dispositivos da *piconet*  $p_i$  vezes o número de dispositivos da *piconet*  $p_j$ .
- Se a comunicação for *intra-piconet* (*piconet*  $p_i$ ), o número de enlaces será igual a somatória de  $t$ , onde  $t$  varia de ( $n^\circ$  de dispositivos de  $p_i - 1$ ) até 1, ou seja:

$$\text{Número de enlaces} = \sum_{t=(n^\circ \text{ de dispositivos de } p_i - 1)}^1 (t)$$

- 1.3 Para cada enlace, que conecta quaisquer dois dispositivos  $i$  e  $j$ , a matriz de throughputs desejados,  $MT[i][j]$  (onde  $i > j$  e  $i \neq j$ ) é igual a um valor aleatório entre zero e TU, sendo que a soma de todos os throughputs pertencentes a comunicação que engloba este enlace será TU.

Com isso, criamos aleatoriamente uma *scatternet* Bluetooth que possui pelo menos uma solução. A matriz de throughputs desejados MT e o número de dispositivos na rede N, servirão de entrada para o algoritmo guloso que será apresentado a seguir.

É importante observar que a simetria de tráfego entre  $i$  e  $j$  (onde  $i \neq j$ ), em situações reais, não existe, mas isso não faz com que os algoritmos descritos neste artigo percam a generalidade, já que a matriz de throughputs desejados MT poderia ser totalmente preenchida e utilizada pelos algoritmos.

## 4. Algoritmo Guloso

A idéia do nosso algoritmo guloso é colocar os dispositivos que mais precisam se comunicar, dentro de uma mesma *piconet*, já que o canal *intra-piconet* é o que possui o maior throughput (1 Mbit/s). O algoritmo recebe como entrada o número de dispositivos (N) e a matriz de throughputs desejados (MT). O pseudocódigo do algoritmo é apresentado na Figura 2 e os passos são citados a seguir:

1. Para cada dispositivo  $i$ , cria-se uma fila  $F_i$  contendo os ( $N - i$ ) dispositivos, em ordem decrescente de acordo com os throughputs desejados entre  $i$  e todos os outros dispositivos da rede, para sabermos com quem  $i$  deseja comunicar mais. Também cria-se um ponteiro  $P_i$ , inicialmente nulo, que apontará para a fila do dispositivo  $i$ . Quando o dispositivo  $i$  for o escolhido para entrar na rede, o dispositivo apontado por  $P_i$  será o seu candidato a ser o próximo elemento a entrar na rede. Esse passo é ilustrado pelas linhas 2 a 22 da Figura 2. Dessa forma, colocamos os dispositivos que mais desejam se comunicar próximos uns dos outros.
2. O primeiro dispositivo  $i$  a ser escolhido para entrar na rede é aquele que deseja comunicar mais com um outro dispositivo, ou seja, aquele que possui o maior throughput desejado em toda rede. Então criamos a primeira *piconet* e o inserimos lá. Em seguida removemos  $i$  de todas as outras filas e fazemos seu ponteiro  $P_i$  apontar para sua fila  $F_i$ , pois  $P_i$  apontará para o seu candidato a ser o próximo a entrar na rede. Como mostra a Figura 2 nas linhas 23 a 34.
3. A partir daí, a escolha dos outros dispositivos a entrar na rede é feita da seguinte forma (linhas 35 a 73 da Figura 2):
  - a. Para cada dispositivo  $i$ , analisa-se o seu candidato que é apontado por  $P_i$  a entrar na rede. O candidato a ser escolhido deve respeitar os seguintes critérios (linhas 42 e 43 da Figura 2):
    - i. A soma do throughput interno da *piconet* em questão, incluindo o candidato do dispositivo  $i$ , deve ser menor que 1 Mbit/s.
    - ii. A média do throughput desejado entre  $i$  e os dispositivos já escolhidos da *piconet* em questão ( $p_i$ ) deve ser maior que um valor de corte  $C_{p_i}$  correspondente a essa *piconet*. Esse valor de corte  $C_{p_i}$ , bem como a necessidade de seu uso, será explicada mais adiante. Esse critério não é válido quando  $p_i$  possuir menos que três dispositivos inseridos.

```

Procedimento Guloso (int corte[]);
1. Início;
2. // Inicia estruturas.
3. para (i=0; i<N; i++)
4.   Inicia_Fila(F[i]);
5.   P[i]=NULO;
6. fim para;

7. // Preenche as filas (decrecentemente) com os
8. // throughputs desejados para cada dispositivo,
9. // e também guarda o dispositivo que quer falar
10. // mais.
11. maior=0;
12. para (i=0; i<N; i++)
13.   para (j=0; j<N; j++)
14.     se (i≠j) então
15.       Inere_Fila(F[i], Throughput(i, j), j);
16.       se (Throughput(i, j) > maior) então
17.         maior=Throughput(i, j);
18.         maior_i=i;
19.       fim se;
20.     fim se;
21.   fim para;
22. fim para;

23. // Escolhe o primeiro candidato:
24. // o que quer falar mais (maior_i).
25. np=0; j=0; n_aux=1;
26. rede.pico[np].elementos[j++]=maior_i;

27. // Os elementos da fila de maior_i agora são
28. // candidatos a entrar na rede.
29. P[maior_i]=F[maior_i].prim;

30. // Ajusta as filas removendo o maior_i de todas elas.
31. para (i=0; i<N; i++)
32.   se (i≠maior_i) então
33.     Remove_Fila(F[i], Throughput(i, maior_i),
34.     maior_i);
34. fim para;

35. enquanto (n_aux<N) faça
36.   // Escolhe o candidato que quer falar mais
37.   // com os dispositivos escolhidos anteriormente,
38.   // verificando se esse cabe na piconet.
39.   escolheu=falso;
40.   maior=-1;

41.   para (i=0; i<N e j<8; i++)
42.     se (P[i]≠NULO) então
43.       se (Throughput_Medio>maior e
44.         Soma_Intra(pico[np], P[i].id)<=1Mbit/s
45.         e (Throughput_Medio>corte[np]*j ou
46.         j<3)) então
47.         maior=soma;
48.         maior_i=P[i].id;
49.         escolheu=verdadeiro;
50.       fim se;
51.     fim se;
52.   fim para;

53. // Se alguém foi escolhido.
54. se (escolheu) então
55.   // Insere o candidato na piconet.
56.   rede.pico[np].elementos[j++]=maior_i;
57.   n_aux++;
58.   // Os elementos da fila de maior_i agora
59.   // são candidatos a entrar na rede.
60.   P[maior_i]=F[maior_i].prim;

61. // Ajusta as filas removendo o maior_i de
62. // todas elas.
63. para (i=0; i<N; i++)
64.   se (i≠maior_i e Vazia(F[i])=falso) então
65.     Remove_Fila(F[i], Throughput(i, maior_i),
66.     maior_i);
67.     se (P[i] ≠NULO) então P[i]=F[i].prim;
68.   fim se;
69.   fim para;
70. senão // Se ninguém foi escolhido.
71.   // Preenche o restante da piconet com -1,
72.   // e inicia a próxima.
73.   rede.pico[np].n_elementos=j;
74.   para (; j<8; j++) rede.pico[np].elementos[j]=-1;
75.   np++; j=0;
76.   fim se;
77. fim enquanto;

78. // Preenche o restante da última piconet com -1.
79. rede.pico[np].n_elementos=j;
80. for (; j<8; j++) rede.pico[np].elementos[j]=-1;
81. np++;
82. Fim.

```

Figura 2. Pseudocódigo do Algoritmo Guloso.

- b. O candidato escolhido  $i$  é aquele que obedeceu aos critérios acima citados e que mais deseja comunicar com os dispositivos já escolhidos pertencentes à *piconet* em questão (linhas 44 a 46 da Figura 2). Em seguida inserimos  $i$  nessa *piconet*, iniciamos a fila  $F_i$ , fazendo  $P_i$  apontar para  $F_i$  e também atualizamos todas as outras filas excluindo das tais o dispositivo escolhido (linhas 50 a 65 da Figura 2). Se nenhum candidato foi escolhido, ou se o número de dispositivos escolhidos para a *piconet*  $p_i$  for igual a oito, encerra-se essa *piconet* e criamos uma nova (linhas 66 a 72 da Figura 2), desde que o número de candidatos escolhidos seja menor que  $N$ , quando então encerra-se o algoritmo.

A complexidade desse algoritmo guloso é  $O(N^3)$ , pois para cada dispositivo escolhido para integrar a rede (onde são escolhidos  $N$  dispositivos), percorre-se todas as  $N$  filas de tamanho  $(N-1)$ , removendo esse dispositivo escolhido. É importante observar que esse não é um algoritmo de otimização e sim uma heurística que tenta satisfazer a necessidade de comunicação da rede.

O critério de corte  $C_{p_i}$ , acima citado, é muito importante, pois ele limita o número de dispositivos dentro de cada *piconet*  $p_i$ , já que sem ele podemos inserir nessa *piconet* alguns dispositivos que querem comunicar muito pouco com os dispositivos já escolhidos. Isso pode fazer com que uma nova *piconet*  $p_j$  seja criada e o próximo dispositivo inserido nela talvez queira comunicar muito com o último dispositivo inserido em  $p_i$ , podendo assim extrapolar o throughput *inter-piconet* que é de 100 kbits/s. O corte possibilita colocar o último dispositivo de  $p_i$  e o dispositivo de  $p_j$  na mesma *piconet*, no caso  $p_j$ .

Os valores de corte foram extraídos depois de inúmeros testes, tendo em vista que ele deve ser menor que 100 Kbits/s (throughput *inter-piconet*). Os melhores resultados obtidos foram com o valor de corte entre 7 e 25 Kbits/s.

Para encontrar um valor de corte adequado para cada *piconet* da rede, geramos a rede através do algoritmo guloso com o valor de corte igual a sete para todas as *piconets*. Em seguida, para as  $(NP-1)$  *piconets*, vamos aumentando o valor de corte para cada duas *piconets* consecutivas e gerando a rede gulosa, até que o limite superior de corte (25 Kbits/s) seja atingido ou quando o throughput (*inter-piconet*) entre essas *piconets* seja menor que 100 Kbits/s, já que não é preciso se preocupar com o throughput *intra-piconet*, pois o algoritmo guloso garante que esse throughput não ultrapassa o limite de 1Mbit/s. Esse procedimento

termina quando a necessidade de comunicação da rede for totalmente satisfeita ou quando se esgotarem todas as tentativas. O pseudocódigo desse procedimento encontra-se na Figura 3.

```

Procedimento Gera_Nete;
1. Início;
2. para (i=0; i<MAX_NP; i++) corte [i]=7;
3. Guloso(corte);
4. para (i=0; i<NP-1 e rede.pior<0; i++)
5.   enquanto (Throughput Inter(i, i+1)>100 Kbits/s
              e corte[i]<25) faça
6.     se (corte[i]>corte [i+1]) então
7.       corte [i]=corte [i+1]+1;
8.       corte [i+1]=corte [i];
9.     senão
10.      corte [i]=corte [i]+1;
11.      corte [i+1]=corte [i];
12.     fim se;
13.     Guloso(corte);
14.     se (rede.pior>melhor_rede.pior) então
15.       Copia_Nete(rede, melhor_rede);
16.     fim enquanto;
17. fim para;
18. se (melhor_rede.pior<0) então
19.   Busca_Local(melhor_rede);
20. Fim.

```

Figura 3. Pseudocódigo do Procedimento.

Com isso a complexidade do nosso algoritmo aumenta para  $O(N^4)$ , já que esse procedimento gasta  $O(N)$  para percorrer todas as *piconets*.

Se tal necessidade ainda não foi satisfeita, fazemos uma busca local na melhor rede gerada pelo procedimento acima e em seguida devolvemos o melhor resultado encontrado. A busca local funciona da seguinte maneira:

- Primeiramente, criamos uma matriz triangular superior de slacks (MS), onde definimos: **slack interno** de uma *piconet*  $i$  ( $MS[i][i]$ ) como sendo 1 Mbit/s menos a somatória do throughput interno da *piconet*  $i$ , e **slack inter-piconet** ( $MS[i][j]$ ) como sendo 100 Kbits/s menos a somatória do throughput entre as *piconets*  $i$  e  $j$ . Isso significa que, quando  $MS[i][i]<0$ , falta banda interna para a *piconet*  $i$ , o que prejudica o desempenho da rede. Caso contrário ( $MS[i][i]\geq 0$ ), a *piconet* possui banda suficiente para suportar o tráfego entre seus dispositivos. E quando  $MS[i][j]<0$ , falta banda entre as *piconets*  $i$  e  $j$ , o que também prejudica o desempenho da rede. Caso contrário ( $MS[i][j]\geq 0$ ), a comunicação *inter-piconet* possui banda suficiente para suportar o tráfego entre as *piconets*  $i$  e  $j$ . É importante citar que ao

construirmos a matriz MS, guardamos o valor do menor elemento pertencente a MS, o que chamamos de **pior(MS)**.

- Em seguida percorremos toda a rede verificando se a troca entre quaisquer dois dispositivos pertencentes a *piconets* diferentes melhora o desempenho da rede. Essa verificação é feita analisando a matriz MS. Se para cada troca, melhoramos o valor de **pior(MS)**, essa troca é aceita, entretanto somente a melhor troca de todas é feita, ou seja, a troca que consegue o melhor valor para **pior(MS)** é realizada. Ao fazer a troca, atualizamos a matriz MS e percorremos toda a rede novamente. Isso é feito até que mais nenhuma troca seja realizada ou quando temos **pior(MS)>0**, o que significa que não está faltando banda na rede.

A melhor rede encontrada (ou seja, a rede que possui o melhor **pior(MS)**) é devolvida como resultado da busca local.

A complexidade dessa busca é  $O(N^3)$ , pois para percorrermos a rede verificando as trocas gastamos  $O(N^2)$  e para cada verificação, analisamos duas linhas e duas colunas da matriz triangular MS, gastando para isso  $O(N)$ .

Portanto a complexidade total do nosso algoritmo é  $O(N^4)$  e os resultados obtidos por ele serão ilustrados no próximo tópico.

Também construímos um algoritmo de força bruta para resolver este tipo de problema, onde permutamos todos os dispositivos entre as *piconets*. Porém para satisfazer as necessidades de uma rede com vinte dispositivos, ele levou mais de 20 segundos.

## 5. Resultados Obtidos

Os testes foram feitos num computador com processador Intel Pentium II 300, com 192 Mbytes de RAM, usando sistema operacional Windows 98, e os algoritmos foram implementados na linguagem C. Foram geradas pelo nosso gerador de redes aleatórias Bluetooth três mil redes, sendo que quinhentas foram geradas com dez dispositivos, quinhentas com vinte, quinhentas com trinta, quinhentas com quarenta, quinhentas com cinquenta e finalmente quinhentas com sessenta dispositivos na rede. Usamos o limite de sessenta dispositivos, pois dificilmente teremos, numa casa ou num escritório, mais que sessenta dispositivos na rede dispostos a uma distância de 10 metros entre si. Os resultados obtidos são apresentados na Tabela 1.

Nº Dispositivos	10	20	30	40	50	60	Total
Nº Redes Geradas	500	500	500	500	500	500	3000
Nº Buscas Locais	0	5	15	32	37	46	135
Não Satisfeitas	0	1	1	8	10	13	33
% Satisfeitas	100,0	99,8	99,8	98,4	98,0	97,4	98,9

Tabela 1. Resultados obtidos com throughput *inter-piconet* de no máximo 100 Kbits/s.

Nas redes compostas por 10 dispositivos, nenhuma delas necessitou fazer a busca local e todas foram satisfeitas pelo algoritmo. Com 20 dispositivos, apenas de 1% fizeram a busca local e 99,8% de todas as redes foram satisfeitas. Com 30 dispositivos, 3% fizeram a busca local e mais de 99% das redes foram satisfeitas. Com 40 dispositivos, 6,4% necessitaram fazer a busca local e 98,4% das redes foram satisfeitas. Com 50 dispositivos, 7,4% fizeram a busca local e 98% das redes foram satisfeitas. Finalmente, nas redes com 60 dispositivos, 9,2% necessitaram fazer a busca local e 97,4% das redes foram satisfeitas. Na média, 4,5% de todas as redes geradas fizeram a busca local e 98,9% das redes foram satisfeitas pelo algoritmo. O tempo médio gasto pelo algoritmo para tentar satisfazer a rede foi de 0,06 segundos, sendo que a rede que levou mais tempo foi uma rede de 60 dispositivos que gastou 2,8 segundos para ser satisfeita.

Para analisarmos a eficiência do algoritmo se o throughput *inter-piconet* for diferente de 100 Kbits/s, ou seja, para sabermos a sua dependência em relação ao throughput *inter-piconet*, fizemos novos testes com o mesmo número de redes e de dispositivos, só que agora com um throughput *inter-piconet* de 50 Kbits/s. Os resultados são apresentados na Tabela 2 abaixo:

Nº Dispositivos	10	20	30	40	50	60	Total
Nº Redes Geradas	500	500	500	500	500	500	3000
Nº Buscas Locais	0	7	10	25	32	46	120
Não Satisfeitas	0	0	1	3	3	5	12
% Satisfeitas	100,0	100,0	99,8	99,4	99,4	99,0	99,6

Tabela 2. Resultados obtidos com throughput *inter-piconet* de no máximo 50 Kbits/s.

Analisando a Tabela 2, verificamos que os resultados diferem pouco. Nas redes compostas por 10 dispositivos, nenhuma delas necessitou fazer a busca local e todas foram satisfeitas pelo algoritmo. Com 20 dispositivos, apenas de 1,4% fizeram a busca local e todas as redes foram satisfeitas. Com 30 dispositivos, 2% fizeram a busca local e 99,8% das redes foram

satisfeitas. Com 40 dispositivos, 5% necessitaram fazer a busca local e mais de 99% das redes foram satisfeitas. Com 50 dispositivos, 6,4% fizeram a busca local e mais de 99% das redes foram satisfeitas. Com 60 dispositivos, 9,2% necessitaram fazer a busca local e 99% das redes foram satisfeitas. Na média, 4% de todas as redes geradas fizeram a busca local e 99,6% das redes foram satisfeitas pelo algoritmo. O tempo médio gasto para tentar satisfazer as necessidades de comunicação dessas redes foi de 0,02 segundos e a rede que gastou mais tempo foi uma de 60 dispositivos e também necessitou de 2,8 segundos para ser satisfeita.

Esses dados mostram que o algoritmo possui uma dependência muito pequena em relação a comunicação *inter-piconet*.

## 6. Conclusão e Trabalhos Futuros

Este artigo apresenta um algoritmo que configura uma rede Bluetooth de acordo com as necessidades de comunicação entre os dispositivos, na tentativa de satisfazê-las. Os resultados mostraram a eficiência do algoritmo já que ele conseguiu satisfazer em média 99% das redes, com uma média de tempo muito baixa (0,06 segundos), e também mostraram que este algoritmo possui uma dependência muito pequena em relação ao throughput *inter-piconet*.

A comunicação entre *piconets* é um assunto muito importante e deve ser estudada em detalhes, já que necessitamos saber o throughput *inter-piconets*, pois ele influenciará na criação das pontes da rede. Essas pontes influenciarão na topologia da rede, tendo em vista que, dependendo do número de pontes por *piconet*, conseguimos criar uma topologia de rede diferente, sendo que essa topologia pode ser decisiva para o desempenho da rede.

Outros tópicos importantes a serem estudados como trabalhos futuros são:

- Como será realizada a formação da *scatternet*. Os trabalhos [5] e [6] apresentam a idéia da *scatternet* ser criada formando uma árvore de respostas (de baixo para cima) pelos dispositivos (nós) da rede para eleger um líder. Ao ser eleito, esse líder, possui informações referentes a todos os nós da rede, podendo então, definir quem são os mestres, os escravos e as pontes da *scatternet*.
- Como reconfigurar a *scatternet* para satisfazer ou voltar a satisfazer as necessidades de comunicação da rede, pois essa necessidade varia com o tempo e com a entrada e/ou saída de dispositivos da rede. Tendo em vista que já

possuímos um algoritmo de configuração, mas não sabemos se ele também é eficiente para reconfigurar a rede, a idéia aqui é testá-lo na reconfiguração em contrapartida a um algoritmo que utilize a rede atual como ponto de partida para a reconfiguração, já que o nosso algoritmo cria uma rede totalmente nova sem saber a topologia da rede antiga.

## Referências

- [1] Bluetooth Core Specification, version 1.1.  
[on line]: <http://www.bluetooth.com/>.
- [2] HAARTSEN, J.; **The Bluetooth Radio System**, IEEE Personal Communication, pp. 28-36, Fevereiro, 2000.
- [3] HAARTSEN, J.; NAGHSHINEH, M.; INOUE, J.; **Bluetooth: Vision, Goals, and Architecture**, Mobile Computing and Communications Review, Volume 1, Número 2, pp. 1-8.
- [4] HAARTSEN, J.; **BLUETOOTH—The universal radio interface for ad hoc, wireless connectivity**, Ericsson Review N° 3, pp. 110-117, 1998.
- [5] RAMACHANDRAN, L.; KAPOOR, M.; SARKAR, A.; AGGARWAL, A.; **Clustering Algorithms for Wireless Ad Hoc Networks**, SIG Mobile 2001, 2001.
- [6] SALONIDIS, T.; BHAGWAT, P.; LAMAIRE, R.; **Proximity awareness and ad hoc establishment in Bluetooth**, 2001.
- [7] KALIA, M.; GARG, S.; SHOREY R.; **Scatternet Structure and Inter-Piconet Communication in the Bluetooth System**, IEEE National Conference on Communications, New Delhi, 2000.
- [8] OLIVEIRA, R. A. R.; LOUREIRO, A. A. F.; **Caracterização de Topologias Dinâmicas no Bluetooth**, III Workshop de Comunicação sem Fio e Computação Móvel, Recife.  
[on line]: <http://www.dcc.ufmg.br/~rabelo>
- [9] KARNIK, A.; KUMAR, A.; **Performance Analysis of the Bluetooth Physical Layer**, IEEE International Conference on Personal Wireless Communication, Dezembro, 2000.
- [10] ZÜRBE, S.; **Considerations on Link and System Throughput of Bluetooth Networks**. IEEE International Symposium on Personal, Indoor e Mobile Radio Communications, Volume 2, pp. 1315-1319, 2000.