UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Giovani Pieri

FORMAÇÃO DE REDES BLUETOOTH ENTRE PASSAGEIROS PARA APLICAÇÕES DE LEVANTAMENTO DE DADOS EM SISTEMAS DE TRANSPORTES COLETIVOS

Florianópolis

2016

Giovani Pieri

FORMAÇÃO DE REDES BLUETOOTH ENTRE PASSAGEIROS PARA APLICAÇÕES DE LEVANTAMENTO DE DADOS EM SISTEMAS DE TRANSPORTES COLETIVOS

Tese submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas para a obtenção do Grau de Doutor em Engenharia de Automação e Sistemas.
Orientador: Werner Kraus Jr.
Coorientador: Jean-Marie Farines

Florianópolis

2016

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Pieri, Giovani
Formação de Redes Bluetooth entre Passageiros para Aplicações de Levantamento de Dados em Sistemas de Transportes Coletivos / Giovani Pieri ; orientador, Werner Kraus Jr. ; coorientador, Jean-Marie Farines. - Florianópolis, SC, 2016.
195 p.

Tese (doutorado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Inclui referências

1. Engenharia de Automação e Sistemas. 2. Formação de redes Bluetooth. 3. Redes móveis pessoais. 4. Sensoriamento urbano com Smartphones. I. Kraus Jr., Werner. II. Farines, Jean-Marie. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. IV. Título.

FORMAÇÃO DE REDES BLUETOOTH ENTRE PASSAGEIROS PARA APLICAÇÕES DE LEVANTAMENTO DE DADOS EM SISTEMAS DE TRANSPORTES COLETIVOS

Giovani Pieri

Tese submetida à Universidade Federal de Santa Catarina como parte dos requisitos para obteçlão do grau de Doutor em Engenharia de Automação e Sistenas

---

Pro. Werner Kraus Jr., Dr.
Orientador

Prof. Jean-Marie Farines, Dr.
Co-orientador

Prof. Rômulo Silva de Oliveira, Dr.
Coordenador do Programa de Pós-Graduação
em Engenharia de Automação e Sistemas

Banca Examinadora:

Prof. Werner Kraus Jr., Dr.
Presidente

Prof. Célio Vinicius Neves de Albuquerque, Dr. − DCC/UFF

Prof. Joni da Silva Fraga, Dr. − DAS/UFSC

Prof. Carlos Barros Montez, Dr. − DAS/UFSC

Prof. Odilson Tadeu Valleu, Dr. − IFSC

Prof. Gustavo Medeiros de Araújo, Dr. − Campus Araranguá/UFSC

RESUMO

Engenheiros de tráfego e gestores dos sistemas de transportes coletivos necessitam de informações atualizadas sobre passageiros transportados para o gerenciamento de tais sistemas. Informações de ônibus são difíceis de obter; por exemplo, apenas o embarque de passageiros é registrado, portanto não há como assegurar o número de passageiros a bordo em um trecho qualquer do itinerário; também, não é possível determinar o ponto de destino dos passageiros, ainda que a origem destes possa ser estimada com base na bilhetagem eletrônica. Nos últimos anos, *smartphones* e dispositivos móveis vêm sendo usados para coleta de informações de trânsito e também sobre o transporte coletivo. Entretanto, estas aplicações não se comunicam com dispositivos próximos. Cada *smartphone* coleta dados individualmente. O estabelecimento de uma rede de *smartphones* permite a troca de informações com dispositivos próximos incentivando processamento local e troca de informações. Nesta tese é proposto um novo algoritmo chamado Bluemob para estabelecer redes usando a tecnologia *Bluetooth* entre passageiros embarcados no ônibus. Este algoritmo é projetado especificamente para as necessidades e particularidades de ambientes pequenos e dinâmicos. Dadas as dimensões de um ônibus, todos os passageiros estarão dentro do raio de comunicação um do outro. Além disso, passageiros entram e descem do ônibus frequentemente. Logo, o algoritmo de formação trata a saída e entrada de dispositivos na rede. Bluemob representa a informação da propensão da saída da rede de cada dispositivo e utiliza esta informação para organizar a topologia e estrutura da rede. A corretude de Bluemob é verificada, garantindo que o algoritmo atende aos requisitos e propriedades estipuladas. Validação do algoritmo é realizado através de simulação, que apontam melhora de até 75% nos tempos envolvidos para reconstrução da rede após a saída de nós da rede quando comparados a algoritmos clássicos. Avaliação qualitativa da estrutura interna da rede mostra que a estrutura imposta pelo algoritmo aumenta sua resiliência a saída de nós quando comparado a outros algoritmos.

Palavras-chave: Formação de redes Bluetooth. Redes móveis pessoais. Sensoriamento urbano com *Smartphones*.

ABSTRACT

Traffic engineers and public transit systems managers need updated information on passengers being transported, for managing and operating these systems. Bus related information are hard to gather; for instance, if only passenger boarding is registered, there is no way to ensure the number of passengers on board at some point in time along the itinerary; also, it is not possible to assert the destination of the passengers, despite that their origin can be inferred based on electronic fare systems. In the past few years, smartphones and mobile devices have been used to gather traffic and transit information. However, these application do not communicate with neighboring devices. Each smartphone gathers data individually. The establishment of a smartphone network allows the exchange of information among nearby devices, allowing local processing and information exchange. In this thesis it is proposed a new algorithm called Bluemob for establish networks among passengers on board of a bus using Bluetooth technology. This algorithm is designed specifically for the needs and particularities of small and dynamic environments. Given the relatively small size of a bus, all passengers are in communication range from one another. Besides that, passengers board and alight from the bus frequently. Therefore, the formation algorithm must deal with devices joining and leaving the network. Bluemob represents the likelihood of a node exiting the network and uses this information to better organize the network topology and structure. The correctness of Bluemob is verified, guaranteeing that the algorithm have the stipulated properties and requirements. Algorithm evaluation is done through simulation, that show an improvement of up until 75% on the times needed for rebuilding the network after the nodes leaving the network when compared to other algorithms. Qualitative evaluation of the internal structure of the network show that the structure achieved by the algorithm increases its resilience to nodes exiting when compared to other algorithm.

Keywords: *Bluetooth* network formation. Personal mobile networks. Mobile urban sensing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BRT	Bus Rapid Transit
Matriz OD	Matriz Origem-Destino
IPF	Iterative pro- portional fitting
MLE	Maximum likelihood estimation
APC	Automatic passenger counter
CMS	Comfort Measuring System
AVL	Automatic Vehicle Location
BLE	Bluetooth Low Energy
IEEE	Institute of Electrical and Electronics Engineers
PAN	personal area network
V2V	Vehicle to vehicle
EE	Escravo-Escravo
ME	Mestre-Escravo
ISM	Industrial Sientific and Medical
GFSK	Gaussian frequency-shift keying
FHSS	frequency-hopping spread spectrum
OSI	Open Systems Interconnection
LMP	link manager protocol
HCI	Host Controller Interface
L2CAP	Logical Link Control and Adaptation Protocol
SCO	Synchronous Connection-Oriented
ACL	Asynchronous Connection-Less
PPP	Point-to-Point Protocol
TCS	Telephone Control Protocol Specification
SDP	Service Discovery Protocol
FDM	Frequency Division Multiplexing
FHS	Frequency Hopping Synchronization
BSF	Bluetooth *Scatternet* Formation

SUMÁRIO

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Há oportunidades interessantes para uso de redes móveis *ad-hoc* formadas entre dispositivos em ambientes fechados. Por exemplo, robôs explorando uma construção podem se comunicar um com outro para melhorar procedimentos de mapeamento do ambiente; pessoas em um evento social podem conhecer umas as outras através da rede *ad-hoc*; e passageiros em ônibus podem contribuir na coleta independente de dados sobre passageiros em sistemas de transporte público, além de poderem interagir com os outro através da rede *ad-hoc*.

Este trabalho é orientado ao último caso. Parte da solução dos engarrafamentos no trânsito das cidades brasileiras é a difusão e incentivo ao uso de sistemas de transporte coletivos. Sistema de transporte baseados em ônibus, como o Bus Rapid Transit (BRT), é uma solução de transporte público com compromisso adequado entre preço de implantação e qualidade de serviço (ARIAS et al., 2008). O gerenciamento e a operação eficiente destes sistemas exigem informações atualizadas sobre o uso destes sistemas pelos seus usuários. Entretanto, informações como o número de pessoas aguardando em pontos de embarque, matrizes de origem-destino (ou seja, o número de viagens feitas pelos passageiros entre quaisquer dois pontos de paradas), número de passageiros a bordo em um momento qualquer do itinerário são de difícil obtenção usando os sistemas atualmente disponíveis, seja pelo seu alto custo de implantação de sistemas de coleta de dados, seja pela falta de sistemas automatizados adequados para sua obtenção.

Propomos um esquema para coleta de informações sobre passageiros em sistemas de transporte coletivo baseado em ônibus sem nenhuma infraestrutura. As vantagens da proposta são: primeiro, o baixo custo é importante para as cidades e operadores em situações onde operadores não estão dispostos a realizar investimentos adicionais nos serviços de transporte público; segundo, a operação do sistema de coleta independe da instalação e manutenção de equipamentos pelos operadores, cujos interesses escusos poderiam levá-los a não realizar a manutenção adequada de equipamentos para ocultar informações relevantes de órgãos de controle, especialmente em mercados mal regulados. Em casos onde estes fatores não são um problema, a proposta beneficia passageiros que desejam interagir entre si no ônibus.

A delegação a voluntários do processo de coleta de informações sobre o ambiente urbano que resulte em um retorno social é conhecido na

literatura por sensoriamento urbano (*urban sensing*) (BURKE et al., 2006; CAMPBELL et al., 2006). O uso muito difundido de aplicações como Waze (Waze Mobile, 2014) e Moovit (Moovit, 2014) atestam a viabilidade de sensoriamento urbano usando *smartphones* como sensores para coleta de dados. Trabalhos anteriores na literatura que fazem uso de *smartphones* para coleta de dados no paradigma de sensoriamento urbano são, por exemplo, Nericell (MOHAN; PADMANABHAN; RAMJEE, 2008) que usa *smartphones* para monitorar condições de trânsito e manutenção de rodovias através dos sensores do *smartphones* para automaticamente detectar frenagens, acelerações, lombadas, buracos e níveis de ruído; VTrack (THIAGARAJAN et al., 2009) que rastreia veículos para estimar o tempo de viagens entre dois pontos e, assim, identificar gargalos no tráfego; Thiagarajan et al. (2010) que descreve um sistema de rastreamento de ônibus e trem para estimar o tempo de chegada do próximo ônibus ou trem na parada; e Lin et al. (2010) que propõe um sistema para avaliar o nível de conforto do transporte coletivo através de dados dos sensores dos *smartphones*.

Trabalhos na literatura que lidam com a aquisição de dados sobre passageiros em sistemas de ônibus incluem Calabrese et al. (2011), Caceres, Widerberg e Benitez (2007) que propõem o uso de dados oriundos das torres celulares no rastreamento da posição das pessoas na cidade, e posterior construção da matriz Origem-Destino (matriz OD), ou seja, uma matriz informando o número de pessoas que realizaram cada um dos trajetos durante o período da pesquisa. As desvantagen de tais métodos se encontram na dependência de companhias de telecomunicação para obtenção dos dados; e da indisponibilidade sobre o método de transporte usados pelas pessoas. Ben Moshe, Hadas e Levi (2014) propõe a instalação de equipamentos dentro do ônibus para detectar e identificar dispositivos *Bluetooth* de passageiros para determinar a matriz OD. Portanto, equipamentos necessitam ser instalados no ônibus e os *smartphones* assumem um papel passivo na coleta de informações.

Nossa abordagem baseia-se na formação de rede pelas seguintes razões. Primeiro, é possível manter um registro da vida da rede do primeiro ao último ponto do itinerário do ônibus. Segundo, o envio de informações a servidores para processamento de dados pode ser feito por um passageiro, diminuindo o tráfego e ajudando com a consistência de dados. Terceiro, e apesar de não tratado nesta tese, é mais fácil confirmar que todos os *smartphones* estão no mesmo ônibus quando eles formam uma rede pois permite a troca de informações de sensores, como por exemplo traços de GPS. Na verdade, o problema de determinar que o *smartphone* está de fato embarcado em um ônibus já foi tratado anteriormente em outros trabalhos e revelou-se ser de difícil solução (REDDY et al., 2010; SHAFIQUE; HATO, 2015).

Bluetooth é adotado como interface de comunicação pois (1) é mais

eficiente energeticamente do que WiFi; (2) ônibus são particularmente pequenos (entre 12 m e 18 m) e o maior alcance fornecido pelo WiFi não é necessário para construir uma rede entre passageiros de um ônibus.

Considerando que a abordagem depende das redes *Bluetooth*, a questão da formação da rede se torna essencial. No cenário de interesse, a principal característica de qualquer algoritmo de formação de rede está na eficiência relacionada aos limites temporais. O tempo necessário para a rede se formar deve ser menor que o tempo de viagem entre duas paradas de ônibus consecutivas. Observa-se tempos de viagem típicos em torno de 1 minuto. Entretanto, há situações nas quais as paradas de ônibus se encontram pouco espaçadas (e.g., 250 m a 500 m). Considerando a aceleração, a velocidade de cruzeiro, a frenagem, e a velocidade média de 30 km/h (8.3 m/s), essas condições resultam em intervalos de tempo tão curtos quanto 30 s para pontos de paradas espaçadas 250 m.

Com o objetivo de tornar rápido o processo de formação de redes *Bluetooth* em cenários de alta volatidade, onde há entrada e saída de dispositivos da rede com frequência, esta tese apresenta o algoritmo Bluemob. Este algoritmo de formação de redes busca posicionar nós na rede de forma que passageiros com maior propensão a desembarcar não sejam alocados em posições da rede que causem o particionamento desta. Ao fazê-lo, ocorrem menos particionamentos na rede a cada ponto de parada, levando a um menor tempo de reconstrução da rede após a saída de um ponto de parada.

Bluemob gera uma topologia de rede refletindo a propensão de nós em permanecerem na rede. Entretanto, salientamos que o mesmo conceito pode ser usado quando há outras restrições que não sejam tempo. Por exemplo, a energia de cada um nós pode ser importante em cenários como comunicação multi-robô e reuniões sociais. Nestes casos, adaptações do Bluemob devem ser implementadas de tal forma que a topologia reflita critérios relacionados a energia.

1.2 OBJETIVO

Conforme indicado acima, o principal objetivo deste trabalho é desenvolver e analisar uma forma rápida e confiável para formação de redes espontâneas entre passageiros de um ônibus usando interface de comunicação *Bluetooth*. Na construção da solução, devem ser levados em consideração aspectos da aplicação na estruturação da rede, no contexto de um sistema de coleta de trajetos individuais de passageiros (para determinação de matrizes Origem-Destino, número de passageiros embarcados, em ônibus e informações correlatas) independente de equipamentos instalados nos ônibus, onde os

próprios passageiros embarcados coletam, tratam e disponibilizam as informações de interessam de forma voluntária por intermédio de seus *smartphones*. Esta abordagem de coleta voluntária de informações é conhecida como *crowdsensing*.

1.3 RESULTADOS OBTIDOS

Este trabalho apresenta os seguintes resultados:

1. Algoritmo de formação de redes espontâneas *Bluetooth* para uso ambientes pequenos (14 m x 14 m, onde quaisquer dois dispositivos possam estabelecer uma conexão *Bluetooth*), dinâmicos, e com alta volatilidade, denominado Bluemob, que leva em consideração aspectos da aplicação na estruturação de redes *Bluetooth*;
2. Proposta de determinação dos passageiros embarcados entre paradas para determinação dos trajetos individuais de passageiros e informações correlatas (e.g. Matriz Origem-Destino) sem necessidade de instalação de equipamentos de infraestrutura nos ônibus ou na cidade;
3. Proposta de uso de redes de comunicação *ad-hoc* no interior do ônibus em sistema *crowdsensing*;
4. Prova de propriedades do algoritmo de formação de redes espontâneas *Bluetooth* obtido.

Resultados do trabalho foram publicados em:

1. PIERI, G.; KRAUS, W.; FARINES, J.M. Bluemob:Algoritmo dinâmico para formação de redes *Bluetooth* em aplicações de sensoriamento urbano. 2015. XXXIII Brazilian Symposium on Computer Networks and Distributed Systems. Vitória, Maio 2015.

   Apresenta os princípio do algoritmo Bluemob e resultados preliminares obtidos com uso de simuladores.

2. MELO, A.S.; KRAUS, W.; FARINES, J.M.; PIERI, G. Abordagem de baixo custo para coleta de dados de transporte público usando *Smartphones*. 2015. XXIX ANPET. Ouro Preto, Novembro 2015.

   Aborda o uso de Smarphone na coleta de Matrizes Origem-Destino usando *Bluetooth*, e auxílio de infraestrutura no ônibus.

3. PIERI, G.; KRAUS JR., W.; FARINES, J.M. Bluemob: a network formation algorithm for bus riders. Wireless Networks, p. 1–15, 26 fev. 2016.

Apresenta o algoritmo Bluemob, a sua proposta de uso para estabelecimento de redes *ad-hoc* entre passageiros para levantamento de matrizes Origem-Destino, e os resultados experimentais obtidos por intermédio de simulação.

1.4 ESTRUTURA DO DOCUMENTO

O restante deste documento está organizado da seguinte forma: no Capítulo 2 apresentamos as principais informações necessárias no projeto, planejamento e operação de sistemas de transporte baseados em ônibus e os mecanismos atualmente disponíveis para sua obtenção. Apresentamos a proposta de um sistema para coleta destas informações pelos próprios passageiros de forma colaborativa baseada na formação de redes *ad-hoc* entre estes dentro dos ônibus. Em seguida, no Capítulo 3, apresentamos o funcionamento do *Bluetooth* e os principais algoritmos e métodos encontrados na literatura para formação de redes *ad-hoc* entre dispositivos. No Capítulo 4, descrevemos em detalhes o algoritmo Bluemob e seu funcionamento interno. No Capítulo 5, apresentamos resultados de estudos realizados usando simuladores onde descrevemos as características e comportamento do Bluemob. No Capítulo 6, realizamos um estudo simulado avaliando a aplicabilidade do Bluemob em uma aplicação de coleta de informações de trânsito urbano. Por fim, no Capítulo 7, retomamos os principais pontos e contribuições desta tese e apontamos direções para desenvolvimento futuro.

2 COLETA DE DADOS EM SISTEMAS DE TRANSPORTE PÚBLICO

A qualidade da informação disponível durante as fases de planejamento e operação são essenciais para garantir a qualidade do sistema de transporte coletivo (Ben Moshe; HADAS; LEVI, 2014). As informações normalmente usadas por engenheiros de transportes nestas tarefas são: (i) matriz origem-destino (matrizes OD) de passageiros; (ii) demanda de passageiros nos pontos de paradas; (iii) carregamento dos ônibus (i.e. o total de passageiros embarcados); (iv) localização dos ônibus; e, (v) condições de trânsito.

Estes dados também são usados para análise e melhoria de desempenho de sistemas de transporte coletivo. Por exemplo, matrizes OD são fundamentais para o ajuste dos tempos de intervalo dos ônibus, assim como para o controle operacional dinâmico de sistemas urbanos de transporte coletivo. Com as informações contidas na variação temporal destas matrizes OD, é possível estimar demandas e determinar estratégias operacionais de controle que se adaptem rapidamente às necessidades do passageiro (CHEN et al., 2011; MADANAT; HU; KROGMEIER, 1996; JIUH-BIING, 2005).

No que segue apresentamos as principais informações de interesse a operadores e gerentes de sistemas de transportes coletivos e os principais métodos usados por eles para obtenção destas informações. Por fim, apresentamos o sistema proposto por nós para obtenção de informações usando *smartphones* dos passageiros, sob paradigma de *crowdsourcing*.

2.1 TÉCNICAS EXISTENTES PARA LEVANTAMENTO DE INFORMAÇÕES

O levantamento de informações sobre os ônibus é feito predominantemente de três formas: (1) questionários e pesquisas feitas com os passageiros; (2) implantação de equipamentos especialmente desenvolvidos para aquisição de dados no interior dos ônibus; (3) fazendo com que o passageiro, através de seus dispositivos, voluntariamente colete e ceda informações sobre o sistema de transporte coletivo.

O desenvolvimento de questionários e pesquisas com os passageiros (primeiro método) para levantamento de informações nos ônibus costuma ter altos custos para realização. Logo, o intervalo entre as pesquisas costuma ser grande, levando à obsolência nos dados obtidos. Além disso, a abrangência da pesquisa é limitada, sendo que apenas uma pequena amostra dos passageiros é contabilizada nos resultados. Tratamentos estatísticos dos dados da

pesquisa devem ser usados para obtenção de dados confiáveis. Apesar destes problemas, ainda hoje algumas informações são obtidas apenas através desta técnica.

O uso de equipamentos instalados nos ônibus, o segundo método, é o mais difundido atualmente, havendo equipamentos especialmente desenvolvidos para coleta de informações. Uma vez instalados, capturam as informações de interesse e as enviam para centrais de processamento. A desvantagem deste método são os custos de instalação e manutenção dos equipamentos.

O terceiro método vem sendo adotado nos últimos anos, se aproveitando do aumento de poder computacional e popularização dos *smartphones* e tablets. Neste método os usuários cedem seus *smartphones* para captura de informações. Os primeiros trabalhos a proporem esta abordagem foram (BURKE et al., 2006; CAMPBELL et al., 2006). Este método é conhecido por vários nomes como *crowdsensing* (GANTI; YE; LEI, 2011), *participatory sensing*, *opportunistic sensing*, ou sensoriamento urbano (*urban sensing*). Neste paradigma os cidadãos deixam de ser meros consumidores de informação, passando a contribuir ativamente na geração da informação. A desvantagem desta abordagem é que os passageiros devem se sentir compelidos a usarem o dispositivo, havendo necessidade da adoção do sistema por um número crítico de usuários para que ele gere informações relevantes. Entretanto, aplicativos para *smartphones* recentes como Moovit (2014) mostram a viabilidade prática destas soluções, ainda que limitados à coleta de tempos de viagem. Diferentes rstratégias podem ser adotadas para estimular a instalação e uso do aplicativo por usuários, por exemplo, incentivo monetário aqueles que utilizam o aplicativo, fornecimento de informações relevantes sobre o sistema de transporte coletivo.

A seguir é feita uma revisão dos principais dados coletados e métodos usados na sua obtenção.

2.1.1 Matrizes Origem-Destino de Passageiros

Matrizes Origem-Destino (OD) são utilizadas para capturar informações sobre os fluxos de deslocamento dentro da cidade. As matrizes OD em sistemas de transporte coletivo indicam o fluxo de passageiros entre cada par de pontos de embarque/desembarque em uma rota.

Matrizes OD são representadas como uma matriz quadrada de tamanho N, onde N é o número de paradas na rota em questão. Cada elemento $a_{i,j}$ representa o número de passageiros que embarcaram no ponto i e desembarcaram no ponto j.

A construção da matriz OD exige o conhecimento dos pontos de em-

barque e desembarque de cada passageiros. De forma alternativa, pode-se determinar a matriz OD com o conhecimento dos passageiros que estão embarcados no veículo entre as paradas de ônibus. Estas duas informações são equivalentes, podendo-se derivar uma a partir da outra de forma trivial.

Atualmente, a forma mais usada para construção de matrizes OD em sistemas de transporte coletivo é através de contagem manual de passageiros por pesquisadores. Agências de transporte coletivo obtêm matrizes OD como um subproduto de pesquisas ocasionais realizadas com os passageiros a bordo do veículo. O resultado da pesquisada é extrapolado por meio de técnicas estatísticas. Periodicamente, contagem de embarques e desembarques é usada para atualizar as matrizes. A realização das pesquisas e contagem são caras e, consequentemente, infrequentes (BEN-AKIVA; MORIKAWA, 1989). Por exemplo, em uma pesquisa conduzida na cidade de Nova York em 2002 chegou-se a conclusão que matrizes OD são estimadas tipicamente entre 5 a 10 anos (BARRY et al., 2002).

A atualização e predição de matrizes OD em tempo real é um tema estudado na literatura de tráfego urbano (CHEN et al., 2011; SUN; FU; LI, ; SUGIYAMA et al., ; JOU et al., ; YAO; ZHAO; YU,). Entretanto, a maior parte destes estudos, parte do pressuposto que tanto o número de embarques quanto de desembarques ocorridos é conhecido. Conforme apontado em Li et al. (2011), nem sempre se tem à disposição sistemas para aferição dos desembarques.

Esta diferença é muito mais acentuada em sistemas que possuem controles de entrada e saída de passageiros. Entretanto, sistemas de transporte coletivo baseados em ônibus não costumam controlar o desembarque de passageiros. Sistemas de tarifação automática registram apenas o embarque de passageiros no ônibus, enquanto sistemas de contagem automatizada de passageiros, conhecidos como sistemas APC, possuem alto custo de instalação e manutenção sendo raro em sistemas de transporte público, especialmente no Brasil. Em sistemas com controle de entrada e saída de passageiros, dados de origens e destinos podem ser derivados a partir do sistema de bilhetagem eletrônico (GORDILLO, 2006).

Em Chen et al. (2011), é proposta uma metodologia para previsão a curto prazo da demanda de passageiros baseado nos padrões temporais exibidos pelos embarques e desembarque e número de passageiros em um dado itinerário.

2.1.1.1 Técnicas para estimativa de matrizes OD a partir de dados coletados manualmente

Uma vez que uma matriz OD foi construída, técnicas estatísticas são usadas para atualizá-la a partir de dados coletados automaticamente. Estes algoritmos necessitam de uma matriz OD inicial, denominada *seed matrix* (matriz-semente), e contagens de embarque e desembarque. Dois métodos são bastante usados: método de ajuste iterativo proporcional (Iterative proportional fitting - IPF) e o método de estimativa por máxima verossimilhança.

O método de ajuste iterativo proporcional, também conhecido como ajuste bi-proporcional ou algoritmo RAS, é um método estatístico usado para a reconstrução de matrizes de contingência proposto em (DEMING; STEPHAN, 1940). Ben-Akiva, Gartner e Wilson (1987) descreve várias técnicas para a junção de matrizes OD provenientes de diferentes fontes, entre elas o método de ajuste iterativo proporcional. Esta técnica continua sendo aplicada em trabalhos recentes, como por exemplo (McCord et al., 2010).

O método de ajuste iterativo proporcional pode ser usado para ajustar uma matriz bidimensional de dados de forma que o somatório de cada linha e de cada coluna entrem em acordo com valores pré-determinados obtidos de fontes alternativas. O mecanismo de ajuste modifica a matriz original gradualmente através de repetidas iterações até atingir as metas estipuladas.

Nas matrizes OD, o somatório das linhas constituem os desembarques ocorridos naquele ponto. Enquanto o somatório das colunas constituem os embarques. A ideia do algoritmo é aproximar o somatório das linhas e das colunas da matriz OD dos valores observados em campo.

Este método exige que as medições de embarque e desembarques sejam bastante precisos para a geração de resultados satisfatórios. A matriz semente pode ser uma amostra pequena, entretanto deve ser representativa. O uso de uma matriz semente muito tendenciosa pode fazer com que o resultado seja pior do que o uso de uma matriz semente vazia.

Este método não funciona caso não ocorram desembarques ou embarques em algum ponto de parada. Uma abordagem usada para circunscrever esta limitação é atribuir a estas paradas um número pequeno de desembarques e embarques.

O método de estimativa por máxima verossimilhança (*Maximum likelihood estimation* - MLE) é um método estatístico utilizado para estimar parâmetros em um modelo. O objetivo do método é, dado um modelo estatístico (ex: uma distribuição normal) com parâmetros bem desconhecidos (média e desvio padrão), estimar o valor destes parâmetros a partir de uma amostra da população. A estimativa é feita buscando-se escolher valores para os parâmetros de modo a maximizar as probabilidades de se observar os da-

dos amostrados.

A aplicação do método MLE para realizar a estimativa de matrizes OD é abordado por Ben-Akiva, Gartner e Wilson (1987), sendo chamada neste trabalho de *Poisson Maximum Likelihood Estimator* (PMLE).

O método de estimação PMLE pressupõe que o sistema de transporte coletivo possui as seguintes características: (i) passageiros chegam individualmente, de acordo com um processo de Poisson; (ii) erro no sistema de contagem de passageiros é negligenciável; (iii) a opção de responder o questionário é um processo binomial; (iv) todos os passageiros podem embarcar no primeiro ônibus disponível; e, (v) As contagens de embarques e desembarques são variáveis independentes.

O sistema de transporte coletivo é modelado com as variáveis aleatórias:

- O fluxo entre dois pontos de parada i e j.
- O número de embarques em um ponto de parada i.
- O número de desembarques em um ponto de parada j.

PMLE varia a matriz OD semente maximizando a probabilidade dos fluxos, embarques e desembarques obtidos em campo serem observados. Ou seja, escolhe-se a matriz que tenha maior probabilidade de ter uma amostragem como a observada em campo. Este método é bastante flexível, mas parte de premissas fortes em relação ao sistema de transporte coletivo. E o sistema real pode acabar não se comportando como o modelo matemático espera.

2.1.1.2 Técnicas para estimativa de matrizes OD a partir de dados coletados automaticamente

Sistemas de transporte coletivo baseados em ônibus costumam possuir sistemas de tarifação automáticos. Através destes é possível determinar o momento do embarque de passageiros que pagaram. Ao cruzar esta informação com dados oriundos de sistemas de localização, como GPS, é possível determinar onde o passageiro embarcou. Entretanto, não há informações sobre os desembarques.

A reconstrução da matriz OD a partir de dados de embarque coletados pelo sistema bilhetagem eletrônica são apresentadas em alguns trabalhos na literatura. Estas técnicas fazem uso de extrapolação e inferência, sendo o resultado obtidos através da aplicação destas técnicas aproximações da realidade. Entre as estratégias usadas para aplicação em sistemas de transporte sobre trilhos pode-se citar(CUI, 2006):

- Passageiros iniciam sua próxima viagem na estação de destino da sua viagem anterior ou em uma estação próxima (método da próxima viagem)
- Passageiros terminam sua última viagem do dia em na estação onde estava no início da primeira viagem do dia (método da última viagem)
- Se um passageiro de metrô/VLT usa um ônibus subsequentemente, a intersecção entre as linhas do ônibus e do metrô/VLT indica o destino da viagem.

Dados provenientes de sistemas de tarifação possuem erros inerentes. Por exemplo, há embarques não contabilizados (p. ex. pagamento em dinheiro). Além disso, as estratégias usadas são de melhor esforço. Logo, espera-se erros na inferência dos destinos. Assim como deve-se definir a cada embarque de um passageiro, seu ponto de desembarque. Dado que estas informações não são fornecidas por sistemas de tarifação, métodos de ajuste são usados para inferir o destino dos passageiros embarcados em um ponto. Técnicas mais usuais usam os métodos anteriormente citados, extrapolando-os para incorporar dados provenientes de sistemas de contagem automático de passageiros. (BARRY et al., 2002; ZHAO; RAHBEE; WILSON, 2007; Wei Wang; John P. Attanucci; Nigel H.M. Wilson, 2011)

2.1.1.3 Determinação automatizada de matrizes OD

Até recentemente não haviam propostas para determinação de matrizes OD de forma automática. Equipamentos disponíveis em ônibus para coleta automatizada de informações não provêm dados suficientemente detalhados para determinar as trajetórias individuais dos passageiros. Dado este necessário para construção da matriz OD.

Entretanto em Ben Moshe, Hadas e Levi (2014) e Kostakos, Camacho e Mantero (2010) foram propostos os primeiros sistemas automáticos para determinação de matrizes OD. Os dois sistemas são semelhantes: dispositivos são instalados no interior do ônibus para detecção de dispositivos *Bluetooth* nas proximidades.

Em Kostakos (2008) os dispositivos são instalados próximos às portas de embarque e desembarque. Desta forma, os passageiros próximos às portas são identificados. De posse desta informação, e mediante processamento, é possível criar a matriz origem-destino.

O mecanismo proposto em Ben Moshe, Hadas e Levi (2014) é mais elaborado. Utiliza-se a intensidade do sinal dos dispositivos *Bluetooth* para determinar a posição do dispositivo no interior do ônibus. Desta forma,

consegue-se, além dos dados necessários para a construção da matriz OD, determinar a distribuição de passageiros no interior do veículo.

A desvantagem da abordagem adotada nestes trabalhos é que apenas dispositivos *Bluetooth* configurados como visíveis pelos passageiros podem ser detectados. Entretanto, a permanência de *smartphones* no estado visível é efêmera. *Smartphones* modernos impõem um limite de tempo em que o dispositivo pode ficar visível. Entretanto, mesmo com esta limitação, Kostakos, Camacho e Mantero (2010) mostram que é possível extrair uma matriz origem destino através desta técnica.

2.1.2 Demanda de passageiros nos pontos de paradas

A demanda de passageiro nos pontos de paradas é medida contando-se o número de passageiros aguardando para embarcar em pontos de parada. Usualmente engenheiros de tráfego urbano não têm acesso a esta informação. Exceções são alguns sistemas de transporte coletivos integrados, onde as catracas são colocadas no acesso aos pontos de parada. Ao usar o sistema de bilhetagem eletrônica para acesso ao ponto, associado a um sistema para contagem de embarques, é possível determinar quantos passageiros estão em um ponto. Entretanto, sistemas organizados desta forma são raros.

A primeira proposta para obtenção sobre demanda de passageiros nos pontos de parada de forma automática é Ben Moshe, Hadas e Levi (2014), onde, a exemplo do que é feito para detecção de passageiros embarcados, dispositivos são instalados nos pontos de parada para detecção de dispositivos *Bluetooth* nas proximidades. Através de um sistema de determinação de posicionamento baseado em intensidade do sinal dos rádios *Bluetooth*, Ben Moshe, Hadas e Levi (2014) consegue estimar o número de passageiros aguardando em um ponto de parada.

2.1.3 Número de passageiros embarcados

A determinação do número de passageiros que estão embarcados entre paradas é realizado por um sistema automático de contagem de passageiros (sistema APC). Estes sistemas são projetados para determinar o número de embarques e desembarques ocorridos em cada parada.

Sistemas APC adotam uma variedade de tecnologias para realizar a contagem de embarque e desembarque. Os métodos mais usados para realizar o sensoriamento são: sensores infravermelhos instalados nas entradas e saídas dos veículos, câmeras aliadas a tecnologias de processamento de imagens e

tapetes sensíveis a pressão colocados nas entradas e saídas dos veículos (CUI, 2006; FURTH et al., 2006).

Sistemas APC não são capazes de identificar os pontos individualizados de embarques e desembarques de cada um dos passageiros. É possível determinar apenas o número de passageiros que embarcaram em uma parada, ou o número de passageiros que desembarcaram. Entretanto, não é possível determinar quantos passageiros embarcaram em uma parada e desembarcaram em outra. Desta forma, estes sistemas não fornecem informações suficientes para a criação de matrizes OD. Além disso, segundo Furth et al. (2006), sistemas APC são raros mesmo em países desenvolvidos.

2.1.4 Nível de conforto em sistemas de transporte coletivos

Alguns trabalhos de *crowdsensing* buscam informações sobre transporte coletivo que são especialmente úteis para o planejamento de sistemas de transporte coletivo. Lin et al. (2010), é descrito um sistema para medir o nível de conforto dos passageiros do transporte coletivo, chamado *Comfort Measuring System* (CMS).

O CMS é composto por três partes: (i) coleta de dados através de sistema de sensoriamento voluntariamente cedidos pelos passageiros, responsável por analisar e avaliar as viagens no sistema de transporte coletivo; (ii) coleta de dados privados das operadoras de transporte coletivo com informações confiáveis, precisas e detalhadas sobre veículos usados; e, (iii) algoritmo responsável por relacionar os dados coletados em (1) e (2) para futuras análises e estatísticas.

CMS explora o GPS e o acelerômetro para medir o nível de conforto dos passageiros. O protótipo descrito em Lin et al. (2010) foi desenvolvido como um sistema offline. Dados brutos são coletados pelos *smartphones* e, posteriormente, processados para determinação do nível de conforto.

Um algoritmo de calibragem é usado para calcular as acelerações nos eixos X, Y e Z, fixados para o centro da terra. É usada técnica semelhante a Nericell (MOHAN; PADMANABHAN; RAMJEE, 2008) para determinação da direção de deslocamento. Mas ao contrário de Nericell, que usa apenas dados de acelerômetro, CMS também faz uso de sensores magnéticos e giroscópio.

O nível de conforto é calculado com base na aceleração máxima a que o passageiro foi submetido. Uma escala logarítmica de 6 níveis foi desenvolvida para determinação do conforto. Quanto menor o número nesta escala, mais confortável a viagem. Para associar a medição do conforto à uma linha de ônibus, CMS procura o itinerário de ônibus mais próximo àquele amos-

trado pelo GPS do *smartphone*.

2.1.5 Determinação do posicionamento do ônibus (rastreamento)

Durante a operação do sistema de transporte coletivo é de interesse saber o posicionamento dos ônibus em trânsito na malha viária. Esta informação é fornecida pelos sistemas de localização automático (*Automatic Vehicle Location* - AVL). O objetivo destes sistemas é rastrear a frota, fornecendo aos operadores a localização dos veículos durante a operação.

Os sistemas AVL usualmente determinam o posicionamento dos veículos usando: GPS (*Global Positioning System*); Odômetro e giroscópio combinado com informações fornecidas acerca da rota do ônibus; análise de sinais de rádio em locais chave; ou, alguma combinação dos métodos acima para prover alguma redundância (CUI, 2006).

Além do posicionamento do ônibus, sistemas AVL podem fornecer algumas informações adicionais como, por exemplo, a identificação de rota e horário, identificação do operador, identificação de paradas, identificação do fim da linha (FURTH et al., 2006). Sistemas AVL baseados em *crowdsensing* foram propostos na literatura. Por exemplo, Thiagarajan et al. (2010) descreve um sistema AVL onde os participantes (os usuários do sistema de transporte coletivo) instalam em seus *smartphones* o aplicativo. Com a ajuda de sensores como: GPS, acelerômetro e WiFi o aplicativo determina se o usuário está em um veículo de transporte coletivo ou não. Em caso afirmativo, mensagens de localização periódicas e anonimizadas são enviadas a uma central.

O cerne do sistema de Thiagarajan et al. (2010) é algoritmo de classificação de atividade. Este algoritmo determina, com base no acelerômetro, o modo de transporte do usuário. Isto permite ao algoritmo distinguir entre pedestres e passageiros de um veículo, e rastrear apenas os veículos.

A detecção de caminhada é baseado nos acelerômetros. É utilizado uma transformada discreta de Fourier para analisar a frequência predominante da variação do acelerômetro em um intervalo de tempo, sendo que a caminhada possui frequência característica de 1-3Hz. O modo de transporte determinado pelo classificador é usado para determinar quando o GPS deve ser ativado para realização do rastreamento.

2.1.6 Monitoramento das condições de tráfego

Informações em tempo-real sobre as condições de tráfego nas vias é de importante valor operacional para engenheiros de tráfego urbano. Estas informações são usualmente fornecidas por meio de observadores espalhados pela cidade. Recentemente, aplicações comerciais como Waze Mobile (2014) tem tido êxito no uso de *crowdsensing* para este fim, mostrando a viabilidade práticas destas aplicações. Na literatura as duas principais propostas baseadas em *crowdsensing* para avaliar tráfego urbano são Nericell e VTrack.

2.1.6.1 VTrack

VTrack (THIAGARAJAN et al., 2009) é um sistema de rastreamento de veículos baseado em *smartphones*. A partir dos dados de localização enviados pelos participantes, VTrack identifica gargalos no trânsito. Para estimular o uso do aplicativo, VTrack permite ao usuário consultar o tempo estimado de chegada em seu destino, levando em consideração o estado atual do trânsito.

Na arquitetura de VTrack, usuários com *smartphones* possuem uma aplicação responsável por enviar dados de posicionamento ao servidor periodicamente. O servidor executa um algoritmo de estimação de tempo de viagem para estimar o tempo de viagem em cada segmento da rodovia. As estimativas são então usadas para identificar gargalos no trânsito e alimentar sistema de planejamento de rotas em tempo real.

Segundo VTrack, o uso de *smartphones* para medir condições de tráfego traz dois desafios significativos:

1. Consumo de energia: determinar a posição do *smartphone* com frequência e precisão consome bastante energia. Diminuir a granularidade da amostragem ou empregar métodos menos precisos de localização contribui para menor consumo de energia. Um compromisso deve ser estabelecido entre acurácia e consumo de energia.

2. Amostras de posição imprecisas: o sistema de localização mais preciso encontrado em *smartphones*, o GPS, não está sempre disponível. O GPS costuma experimentar períodos de indisponibilidade dependendo da localização do usuário (os chamados cânions urbanos, costumeiramente localizados nos arredores de prédios altos e túneis).

VTrack envia fluxos de posições não precisas e estampilhas de tempo em intervalos irregulares. Um Modelo Oculto de Markov (Markov Hidden

Model - HMM) é usado para modelar a trajetória do veículo em um mapa da área. VTrack então realiza um processo de "map matching" responsável por associar cada posição amostrada com a posição mais provável no mapa rodoviário. Em seguido, produz estimativas do tempo de viagem para cada segmento de viagem percorrido.

2.1.6.2 Nericell

Nericell (MOHAN; PADMANABHAN; RAMJEE, 2008) utiliza os sensores dos *smartphones* para determinar as condições do trânsito, através de detecção de freadas e do nível de ruído do trânsito (por exemplo, buzinas). As condições de conservação das estradas são estimadas, detectando-se buracos e lombadas.

Quando em operação, Nericell mantém apenas o radio GSM e o acelerômetro ligados. A análise destes sensores, que possuem baixo consumo energético, é usada para avaliar a necessidade do uso de sensores mais precisos, porém que necessitam de mais energia para operarem, como o GPS. Com isso, busca-se diminuir o consumo de energia pela aplicação.

Os algoritmos usados em Nericell na detecção de lombadas, buracos e frenagem baseiam-se no acelerômetro do *smartphone*. Os acelerômetros de dispositivos móveis possuem 3 eixos, e mensuram a força exercida nos seus mecanismos de sensoriamento. Assim sendo, um acelerômetro bem calibrado estacionário reporta em seu eixo Z o valor de 1g, e 0g em seus dois outros eixos. Da mesma forma, quando um veículo acelera é reportado uma aceleração negativa em seu eixo X.

O uso do acelerômetro é dificultado por não haver conhecimento da posição relativa do *smartphone* em relação ao veículo. Os eixos do acelerômetro do *smartphone* podem não coincidir com os eixos reais do veículo. Para circunscrever este problema Nericell propõe a técnica de reorientação virtual do acelerômetro. A ideia de Nericell é identificar períodos quando o acelerômetro não está reportando movimentações para determinar a direção em que está o chão. Leituras de GPS são usadas para determinar a direção em que o dispositivo está se movendo.

2.2 PROPOSTA DE SISTEMA DE COLETA BASEADO EM *CROWDSENSING*

Conforme visto anteriormente, há escassez de alternativas para coleta de informações detalhadas sobre o trajeto de passageiros em sistemas

de transporte coletivo baseados em ônibus. Algumas informações, como passageiros embarcados entre pontos e carregamento dos ônibus, dependem de equipamentos especiais instalados nos mesmos, e outras, como matrizes OD, não são passíveis de serem determinadas automaticamente. Nesta seção descrevemos um sistema para coleta destas informações que usa os *smartphones* dos passageiros para determinar o conjunto de passageiros embarcados entre os pontos de parada. O sistema de coleta de informações proposto é um sistema distribuído composto por um número qualquer de *smartphones* e uma central de processamento. Parte-se do princípio que ao determinar quais *smartphones* estão embarcados, tem-se um subconjunto representativo dos passageiros do ônibus. Assim, o problema de determinação do conjunto de passageiros embarcados pode ser aproximado pela determinação de quais *smartphones* estão no interior do ônibus.

O ambiente operacional da proposta consiste de passageiros de ônibus contribuindo na coleta, pessoas no ponto de parada aguardando o embarque e a central de processamento que recebe as informações coletadas. Todo passageiro possui uma aplicação móvel instalada em seu *smartphone* que, em conjunto, identifica os demais passageiros embarcados que também possuem a aplicação. Uma rede é formada no interior do ônibus, e é usada para identificar os passageiros embarcados. Aqueles passageiros cujo *smartphone* integra a rede *ad-hoc* é considerado embarcado no ônibus. Consequentemente, a entrada e saída do *smartphone* da rede é uma aproximação do embarque e desembarque do passageiro da mesma, e esta informação pode ser usada para construção da matriz OD. Para determinar se um *smartphone* deixou a rede, espera-se a conexão *Bluetooth* ser desfeita. Em experimentos realizados com o protótipo descrito no anexo B, a detecção da desconexão pelo próprio *Bluetooth* é rápida (tempos observados foram menores que 1s), e para efeitos deste trabalho, consideramos este tempo desprezível. Sempre que se aproximam de um ponto de parada, o conjunto de passageiros identificado é enviado à central de processamento. Por sua vez, a central de processamento é um agregador das informações coletadas pelos *smartphones*. Seu papel é processar e agregar informações de viagens (origens/destino, carregamentos) a partir dos dados recebidos dos *smartphones*. A adoção de *smartphones* possibilita que o sistema de coleta não dependa de equipamentos instalados na infraestrutura ou nos ônibus, facilitando a implantação e manutenção do sistema.

Cada *smartphone* de passageiro embarcado possui um aplicativo instalado. Todo aplicativo possui um “Algoritmo de Formação de Rede” responsável pela formação e manutenção de uma rede *ad-hoc* com os outros aplicativos próximos. Um subsistema auxiliar “Processamentos auxiliares” é usado para processar os sensores do *smartphone* e determinar quando um

passageiro está chegando próximo a um ponto de parada. Ao se aproximar dos pontos de embarque e desembarques, os integrantes da rede *ad-hoc* são determinados, o histórico e contagem são contabilizados. Por fim os dados coletados são enviados pelo líder à central por intermédio do subsistema de "Envio a central". A central realiza o processamento e tratamento necessário às informações recebidas para disponibilizá-los aos interessados.

O aplicativo executado nos *smartphones* dos passageiros embarcados em um ônibus durante o processo de coleta de dados é composto pelos seguintes elementos: "Algoritmo de Formação de Rede", responsável pela formação e manutenção da rede *ad-hoc*; "Processamentos auxiliares", para processamento de sensores do *smartphone* e determinar quando um passageiro está chegando próximo a um ponto de parada; "Envio à central", executado apenas no líder da rede para envio dos dados coletados a cada ponto de parada. A central realiza o processamento e tratamento necessários às informações recebidas para disponibilizá-los aos interessados.

A partir da concepção geral apresentada acima, surgem questões associadas à forma de implementação em face das características do ambiente no qual o sistema opera. Inicialmente, é preciso destacar as razões pelas quais se opta pela formação de rede, conforme segue. Havendo uma rede no interior dos ônibus, torna-se possível determinar que todos os nós pertencem, de fato, ao mesmo veículo através dos traços de GPS de cada nó; em contraste, o problema de determinar se um nó isolado está em um ônibus não tem solução simples (REDDY et al., 2010; THIAGARAJAN et al., 2010; SHAFIQUE; HATO, 2015). Além disso, a consolidação dos dados antes do envio em apenas um nó ajuda na consistência da informação, ao contrário do caso em que nós individuais enviassem informações à central. Por último, sempre que algum nó da rede permanecer a bordo em um ponto de parada, é possível preservar o histórico do trajeto para envio à central, facilitando verificações adicionais de consistência de dados e confirmação de itinerário.

Uma segunda questão refere-se ao tipo de interface de comunicação a ser usada na formação da rede. Ônibus são relativamente pequenos, com comprimentos entre 12 m e 18 m. *Smartphones* típicos possuem duas interfaces de comunicação sem fio cujo alcance permite a conexão de quaisquer dois passageiros embarcados: WiFi e *Bluetooth*. O alcance do rádio WiFi depende da versão específica do protocolo em uso e de características da antena. Por exemplo, um *access point* 802.11b ou 802.11g com uma antena padrão pode ter alcances na faixa dos 100 m (PARIKH; KANABAR; SIDHU, 2010). *Bluetooth*, por outro lado, possui alcance típico de 20-30 m em rádios *Bluetooth* Classe 2 usados em *smartphones* (PEI et al., 2010). Sendo ambos igualmente apropriados para uso em relação ao alcance, *Bluetooth* foi escolhido ao invés de WiFi como interface de comunicação pois: (1) é mais eficiente energe-

ticamente do que WiFi (MOHAN; PADMANABHAN; RAMJEE, 2008); e (2) ônibus são pequenos (entre 12 m e 18 m) e o maior alcance fornecido pelo WiFi não é necessário para construir uma rede entre passageiros de um ônibus.

O método a ser usado na formação da rede, isto é, o algoritmo para formação baseado em *Bluetooth*, passa a ser um aspecto fundamental do sistema, consistindo no foco desta tese. Destaca-se que, na concepção do algoritmo, deve ser dada atenção à eficiência temporal no processo de formação, pois o conjunto de passageiros muda a cada embarque e desembarque nas paradas de ônibus. O tempo para finalizar a determinação do conjunto é igual ao tempo que o ônibus leva para se deslocar de um ponto de parada ao próximo. Portanto, há uma restrição temporal para que a coleta de informações dos passageiros no ônibus seja realizada. Para obter a eficiência desejada na formação da rede, propõe-se usar as características do ambiente a favor do processo de organização da topologia da rede. Parte-se da observação que em situações reais os nós têm propensões distintas a desembarcar; assim, o algoritmo de formação pode organizar a rede colocando nas extremidades desta, os nós cujos os desembarques são estimados para ocorrer mais cedo do que outros, que por sua vez, são colocados em posições estruturantes da rede. Tais ideias dão origem ao algoritmo Bluemob, apresentado a seguir.

2.3 ALGORITMO BLUEMOB: FORMANDO REDES COM TOPOLOGIA ADAPTADA AO AMBIENTE DA APLICAÇÃO

O sistema proposto de coleta de dados baseia-se na capacidade de *smartphones* de passageiros embarcados estabelecerem uma rede de forma autônoma e rápida. A tarefa de formação consiste em: (i) interconectar todos os passageiros embarcados no ônibus em uma única rede *Bluetooth*; e (ii) finalizar o processo dentro do tempo necessário para que o ônibus se desloque de um ponto de parada ao próximo.

Confirmou-se, usando um protótipo de sistema de formação de rede descrito no anexo B, que os tempos envolvidos para a formação da rede podem ser longos. No protótipo, usou-se a plataforma Beddernet (GOHS; GUNNARSSON; GLENSTRUP, 2011) para a formação de redes *Bluetooth* em dispositivos Android. O algoritmo adotado nesta plataforma não foi capaz de atender aos requisitos de tempo para completar a formação da rede, excedendo intervalos de 3 min para conectar cerca de doze dispositivos. O tempo excessivo observado com o protótipo motivou o estudo de técnicas de formação mais adequadas ao ambiente de uso da aplicação, ou seja, o espaço interior de ônibus em deslocamentos urbanos.

Para melhorar o desempenho do processo de formação da rede, observou-se que, uma vez inicializada, esta é particionada quando ocorrem desembarques de passageiros. Isto é, a dinâmica intrínseca ao processo de embarque e desembarque de passageiros causa o particionamento da rede nas paradas de ônibus. Entre o trajeto de uma parada a outra, não há mudança nos passageiros embarcados havendo um período de estabilidade, durante o qual a rede é reestabelecida, conectando as diferentes partições uma vez mais em uma única rede no interior do ônibus. Assim, a reconexão completa da rede seria mais rápida se a topologia fosse organizada para que os desembarques envolvessem apenas nós situados nas bordas da rede. Desta forma, o processo de formação envolveria apenas nós que embarcam no ônibus, diminuindo o tempo para restauração completa da rede. Caso contrário, se os nós fossem posicionados indistintamente, os desembarques afetariam com maior frequência as posições estruturantes da rede (aquelas com maior número de conexões com outros nós), aumentando o esforço computacional e o tempo de reconstrução desta.

Ao se observar os padrões de viagens de passageiros de ônibus, observa-se que há regularidades a serem aproveitadas como informação relevante ao processo de formação da rede. Por exemplo, um mesmo passageiro tende a ter deslocamentos similares em dias de trabalho ou mesmo em viagens de compras e lazer (SONG et al., 2010). Assim, o histórico dos deslocamentos do passageiro pode ser um forte indicativo do roteiro de uma viagem atual. Além disso, a posição do passegeiro no interior do ônibus pode indicar a propensão de desembarque, em caso de veículos muito carregados.

A partir dessas observações, surge a ideia de usar estimativas do ponto de desembarque de cada passageiro no processo de formação da rede. Assim, pode-se estruturar a rede para diminuir o particionamento devido ao desembarque de passageiros. Neste contexto foi desenvolvido o algoritmo Bluemob, que aborda a formação de redes *Bluetooth* através do posicionamento de nós com maior propensão a desembarcar em posições periféricas. Desta forma, a topologia da rede formada tem menos particionamentos a cada desembarque se comparada com uma rede formada com nós posicionados aleatoriamente, resultando em tempos menores de formação após cada parada de ônibus.

O entendimento das especifidades de Bluemob e das escolhas tomadas em seu projeto requer explanação do funcionamento do protocolo *Bluetooth*, em particular dos aspectos únicos a esta tecnologia que afetam o processo de formação de redes, como por exemplo, o processo de descoberta de dispositivos. Também é necessário o conhecimento sobre os principais algoritmos propostos na literatura para formação de redes *Bluetooth*, e sua classificação. Estes dois assuntos serão abordados no próximo capítulo, em preparação à

apresentação de Bluemob no Capítulo 4.

2.4 CONSIDERAÇÕES FINAIS

Apresentou-se neste capítulo uma revisão dos principais métodos de obtenção de dados operacionais de transporte coletivo. Foram revisados também os métodos que se baseiam em "*crowdsensing*", usando dispositivos móveis (tais como os *"smartphones"*) para obtenção dos dados de interesse. Observou-se que os métodos revisados não buscam cooperação entre dispositivos para coleta dos dados, tornando a tarefa mais difícil por conta da incerteza de localização do dispositivo. Isto motivou a criação da proposta de coletar dados em sistemas de transporte coletivo com base em redes *ad-hoc* formada entre *"smartphones"* de passageiros usando o suporte de comunicação *Bluetooth*.

Para viabilizar a proposta, é necessário dispor de um algoritmo de formação de redes que seja capaz de finalizar a conexão de todos os nós a bordo antes da chegada do ônibus ao próximo ponto de parada. Experimentos com um protótipo baseado em algoritmo de formação de propósito geral denominado "Beddernet" indicaram que os tempos envolvidos seriam inviáveis se este algoritmo fosse utilizado.

O ponto de partida para formulação de algoritmo mais eficiente para o ambiente da aplicação é a constatação de que o desembarque de passageiros é o evento crítico no histórico da rede em termos do requisito de tempo para formação. Também, constata-se que os passageiros têm propensões diferentes de desembarque em cada ponto, podendo ser formuladas técnicas de estimativas da tendência de permanência a bordo com base em diferentes critérios. Portanto, um algoritmo eficiente de formação de rede pode aproveitar o conhecimento das estimativas de desembarque para construir a rede com topologia tal que os nós propensos a desembarcar situem-se nas extremidades da rede. Assim, os desembarques tenderão a produzir menos partições da rede, agilizando a formação após a partida do ponto de parada.

O algoritmo Bluemob foi formulado para implementar as ideias apresentadas acima visando a eficiência no processo de formação. Em linhas gerais, o algoritmo emprega regras heurísticas baseadas na estimativa do ponto de desembarque de cada nó para construir a rede com topologia adequada ao ambiente da aplicação. O restante deste trabalho apresenta elementos básicos para a formulação do algoritmo (relativos à interface *Bluetooth* e algoritmos de formação) bem como o detalhamento das propriedades e do desempenho da formação de rede com Bluemob.

Há questões em aberto não tratadas neste trabalho. A primeira refere-

se à viabilidade do sistema, tanto comercial como técnica. Do ponto de vista comercial, seria preciso avaliar a aceitação por parte dos usuários (passageiros, operadores e gestores) e formas de políticas de incentivo do uso, eventualmente financeiro ou de recompensas promocionais. Pelo lado técnico, há desafios interessantes quanto ao desenvolvimento de produto baseado no algoritmo desenvolvido, ao tratamento de possíveis faltas, e à atualização com base em novas tecnologias ora em estágio de introdução, como o *Bluetooth* de baixa energia (Bluetooth Low Energy - BLE) introduzida na versão 4.0 da especificação *Bluetooth* (Bluetooth SIG, 2010). O aspecto de tratamento de faltas refere-se à interferência de pessoas nas proximidades que não estão dentro do ônibus mas que instalaram o aplicativo para auxiliar na coleta de dados quando estão embarcados. Este problema é percebido pontos de paradas, já que há potencialmente passageiros nos pontos de parada com o aplicativo instalado que não embarcam no ônibus, por estarem esperando outra linha. Logo, deve-se diferenciar estas pessoas de passageiros e evitar a sua incorporação à rede, e consequente contabilização como um passageiro do ônibus. Algumas estratégias podem ser adotadas para minimizar este problema, por exemplo, evitar a entrada e saída de passageiros na rede *ad-hoc* nas cercanias dos pontos de parada, ou esperar que o nó esteja presente por um tempo pré-determinado antes de aceitá-lo na rede. De forma análoga, há pessoas que se recusam a participar independente de possíveis incentivos que sejam oferecidos. Métodos devem ser utilizados para estimar qual a proporção destes usuários para que se possa avaliar a qualidade e representatividade da matriz OD obtida.

Por último, destaca-se que a heurística de formação que diferencia os nós de acordo com sua propensão a permanecer a bordo pode ser substituída por outros critérios em aplicações diferentes. Por exemplo, conforme apresentado no Capítulo 1, pode haver interesse em formar redes *ad-hoc* com robôs explorando um ambiente fechado. Neste caso, um critério que evita particionamentos é a distância de um nó ao centro da rede; quanto mais afastado, maior a tendência a perder contato devido ao alcance limitado do *Bluetooth*. Outro exemplo é dado pelos ambientes de interação social, nos quais o uso de *Bluetooth* pode substituir o WiFi para economia de bateria. Neste caso, o critério mais adequado seria o tempo restante para atingir-se o nível crítico de carga da bateria.

3 REDE *AD-HOC* DE DISPOSITIVOS USANDO TECNOLOGIA DE COMUNICAÇÃO SEM FIO *BLUETOOTH*

Neste capítulo apresentamos a tecnologia de comunicação sem fio *Bluetooth*. Especial atenção é dada aos mecanismos de estabelecimento de conexão, os procedimentos de INQUIRY e PAGE, devido ao seu papel no projeto de algoritmos de formação de redes *Bluetooth*. Por fim, descrevemos os principais algoritmos encontrados na literatura para criação de redes *Bluetooth* chamadas *Scatternets*. Apresentamos como estes algoritmos são classificados, suas principais características e seu funcionamento.

3.1 DESCRIÇÃO GERAL DO *BLUETOOTH*

Bluetooth é uma tecnologia de comunicação sem fio projetada para fornecer conectividade a curta distância para dispositivos eletrônicos. Seu objetivo é fornecer uma alternativa sem fio e com baixo consumo energético para interconectar dispositivos pessoais. É usado atualmente, por exemplo, para redirecionar o áudio das chamadas de *smartphones* para o sistema de áudio de automóveis, conectar fones de ouvido a *smartphones* sem uso de fios, entre outras aplicações.

3.1.1 Visão geral

Em 1994, Ericsson Mobile Communications iniciou um estudo de viabilidade de um interface de rádio de baixo consumo para ser usada entre os celulares Ericsson e seus acessórios. Em fevereiro de 1998, IBM, INTEL, Nokia e Toshiba se juntaram à Ericson para fundar a SIG *Bluetooth*. Posteriormente, diversas companhias ingressaram no SIG. Como por exemplo, 3Com, Agere (Lucent Technologies), Microsoft e Motorola. Atualmente, SIG possui mais de 2500 companhias abrangendo os mais diversos campos do conhecimento como telefonia, computadores pessoais, carros, e processamento digital. Sendo uma especificação industrial aberta, todos os membros do SIG *Bluetooth* podem usá-lo gratuitamente em seus serviços e produtos. Além disso, a tecnologia *Bluetooth* é padronizada pela IEEE sob a referência 802.15.1.

Bluetooth viabiliza a criação de redes de área pessoais (*personal area network* - PAN). Opera na frequência de 2.4GHz, possui baixo consumo, e permite a comunicação a algumas dezenas de metros. Devido a suas pro-

priedades, *Bluetooth* é perfeito para estabelecer redes no interior de veículos. Inclusive alguns pesquisadores propõem o uso de *Bluetooth* em redes veiculares (V2V). Entretanto, o uso é limitado devido ao alcance e tempo necessário para estabelecimento das redes.

Na especificação *Bluetooth*, atualmente em sua versão 4.0 (Bluetooth SIG, 2010), são descritas o funcionamento do hardware e de software dos dispositivos. Nas camadas inferiores da pilha de protocolo, detalha-se o funcionamento do hardware de rádio e modulação de sinal. Nas demais camadas, usualmente implementadas em software e gerenciadas por microprocessadores, os protocolos de comunicação são descritos detalhadamente.

3.1.2 Estrutura das redes *Bluetooth*: *Piconets* e *Scatternets*

Redes *Bluetooth* se organizam em *Piconets*: uma rede com topologia estrela na qual há um mestre conectado a até sete escravos. De acordo com a especificação *Bluetooth*, a comunicação entre dois dispositivos *Bluetooth* é possível apenas se ambos os dispositivos pertencem a uma mesma *piconet*. Um mestre pode possuir até 255 escravos, entretanto apenas 7 destes estão ativos na *Piconet* simultaneamente. Entretanto, evita-se atribuir a um mestre mais de 7 escravos devido aos custos envolvidos. A *Piconet* é apenas um canal de comunicação bidirecional entre o mestre e seus escravos.

A comunicação dentro de uma *Piconet* segue um padrão de comunicação mestre/escravo. Logo, não há comunicação direta entre escravos de uma *Piconet*; portanto, os pacotes devem ser roteados através do mestre. *Bluetooth* segmenta o tempo em *slots* com 625 μs de duração cada. Durante este *slot* de tempo um pacote pode ser enviado.

A sincronização da comunicação entre dispositivos *Bluetooth* no tempo é feito por intermédio de uma variável interna denominada *clock*, disponível em todo dispositivo *Bluetooth*. O *clock* é um número inteiro incrementado a cada $312,5$ μs, equivalente a metade do tempo de um *slot* de comunicação. Para comunicar-se com um escravo, o mestre envia uma mensagem informando que ele deve realizar a transmissão no próximo *slot*. Caso tenha uma informação a transmitir, o escravo responde a solicitação no próximo *slot*.

A especificação *Bluetooth* impede que um dispositivo atue como mestre de mais de uma *Piconet*, sendo a identificação da *Piconet* o seu mestre. Entretanto, não há limite no número de *Piconets* a que um dispositivo pode pertencer atuando como um escravo. Ao permitir que dispositivos pertençam a mais de uma *Piconet* simultaneamente, a especificação permite que *Piconets* sejam interconectadas. Esta rede formada por duas ou mais *Piconets* interconectadas é chamada de *Scatternet*. Duas *Piconets* podem se inter-

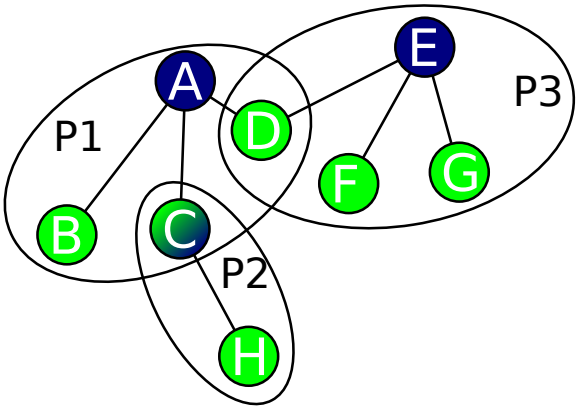


Figura 1: Exemplo de uma rede *Bluetooth* com três *piconets* formando uma *scatternet*. Os nós A, C e E atuam como mestres das *Piconets* P1, P2 e P3, respectivamente. O nó D é uma ponte Escravo-Escravo, enquanto o nó C é uma ponte Mestre-Estravo. Os demais nós exercem papel de escravo.

conectar através de pontes Escravo-Escravo (EE) ou pontes Mestre-Escravo (ME). Nas pontes ME o nó compartilhado entre as duas *Piconets* atua como escravo em uma delas e mestre na outra. De forma análoga, nas pontes EE a ponte atua como escravo em ambas as *Piconets*. A especificação permite que duas *Piconets* sejam interconectadas por mais de duas pontes. A Figura 1 ilustra uma rede *Bluetooth Scatternet* formada por 3 *Piconets*.

Não é possível que um dispositivo atue simultaneamente em duas *Piconets*. Um algoritmo de escalonamento entre-*Piconets* é usado para chavear entre as *Piconets*, desta forma, em um dado instante as pontes estão participando de apenas uma *Piconet*. Logo, devido ao fato de que toda a comunicação de uma *Piconet* deve ser roteada através do mestre, o uso de pontes ME incorre em maiores custos do que pontes EE. Pois, durante o tempo no qual a ponte ME está atuando como escravo de outra *Piconet*, a comunicação da *Piconet* na qual ele atua como mestre é interrompida, já que ele não pode atuar na coordenação e roteamento de pacotes de sua própria *Piconet*. Isto faz com que haja uma diminuição do *troughput* da rede e aumento na latência da comunicação na rede *Bluetooth*, especialmente quando pacotes precisam ser roteados pela *Piconet* na qual a ponte ME é mestre.

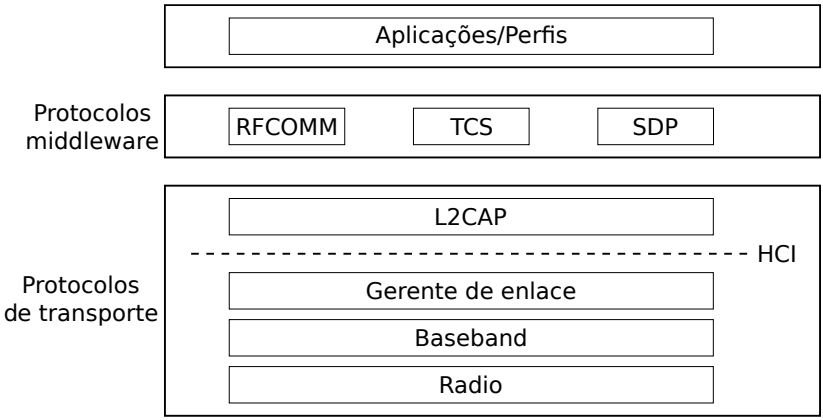


Figura 2: Pilha de protocolos *Bluetooth*

3.1.3 Camadas na pilha *Bluetooth*

Os protocolos das camadas *Bluetooth*, ilustrados na Figura 2, podem ser separados em duas categorias: protocolos de transporte e protocolos *middleware*. Os protocolos de transporte são responsáveis pela troca de informações entre dois dispositivos, sendo exclusivos à tecnologia *Bluetooth*. Os protocolos *middleware* fornecem abstrações para aplicações, isolando detalhes específicos dos protocolos de transporte.

As camadas *Bluetooth* são:

1. **Radio -** camada que define as características técnicas dos rádios *Bluetooth*. *Bluetooth* opera na banda não licenciada de 2.4GHz do espectro ISM (*Industrial Sientific and Medical*), conjunto de faixas de frequências reservadas internacionalmente para uso industrial, científico e médico. A modulação usada é a *Gaussian frequency-shift keying* (GFSK), aliada a técnica de *frequency-hopping spread spectrum* (FHSS) com até 79 diferentes frequências.

2. ***Baseband* -** equivalente à camada física do modelo OSI (*Open Systems Interconnection*), especifica o controlador de enlace (*Link Controller*). Esta camada define o funcionamento das estruturas básicas da rede *Bluetooth*: as *Piconets* e canais de comunicação *Bluetooth*. Especifica os mecanismos *Bluetooth* para descoberta de dispositivos e formação de conexão.

3. **Protocolo de gerenciamento de enlace (*link manager protocol - LMP)***

- protocolo responsável pelo processo de autenticação de dispositivos, estabelecimento e configuração dos canais de comunicação.

4. **Interface Controladora de Sistema (*Host Controller Interface* - HCI) -** interface padronizada para acesso uniforme às função da camada de *baseband* e LMP.

5. **Protocolo de Adaptação e Controle do Enlace Lógico (Logical Link Control and Adaptation Protocol - L2CAP) -** equivalente à camada de enlace do modelo OSI, é responsável por: segmentar e remontar pacotes; fornecer qualidade de serviço; abstrair grupos de dispositivos *Bluetooth* e mapeá-los sobre a *Piconet*; multiplexar; e, demultiplexar fluxos nos enlaces físicos. Define e fornece as camadas superiores mecanismos de comunicação orientadas a conexão (*Synchronous Connection-Oriented* - SCO), como enlaces não orientados a conexão (*Asynchronous Connection-Less* - ACL).

Os protocolos *middleware* usam os serviços fornecidos pela camada L2CAP para implementar uma interface familiar para uso pelas aplicações. Os principais protocolos *middleware* são:

1. **RFCOMM -** emula portas seriais (RS-232) sobre a camada L2CAP. Este protocolo permite o uso de protocolos genéricos, como *Point-to-Point Protocol* (PPP), sobre *Bluetooth*. Note que sobre PPP, pode-se inclusive usar protocolos como TCP/IP.

2. **Protocolo de controle telefônico (*Telephone Control Protocol Specification - TCS* -** define protocolo de sinalização de chamadas usados por dispositivos *Bluetooth* para estabelecer chamadas de dados ou voz.

3. **Protocolo de descoberta de serviços (*Service Discovery Protocol - SDP* -** provê meios de uma aplicação descobrir o conjunto de serviços disponíveis em outro dispositivo *Bluetooth* e determinar as características destes dispositivos. Este protocolo trata apenas do procedimento de descoberta, não define aspectos relacionados ao acesso e uso do serviço.

3.2 ESTABELECIMENTO DE CONEXÕES *BLUETOOTH*

Bluetooth adota, em sua camada física, uma estratégia de multiplexação por divisão de frequência (*Frequency Division Multiplexing* - FDM). Ou seja, o meio de transmissão é dividido em faixas de frequência não-sobrepostas, permitindo que múltiplos comunicadores compartilhem o mesmo

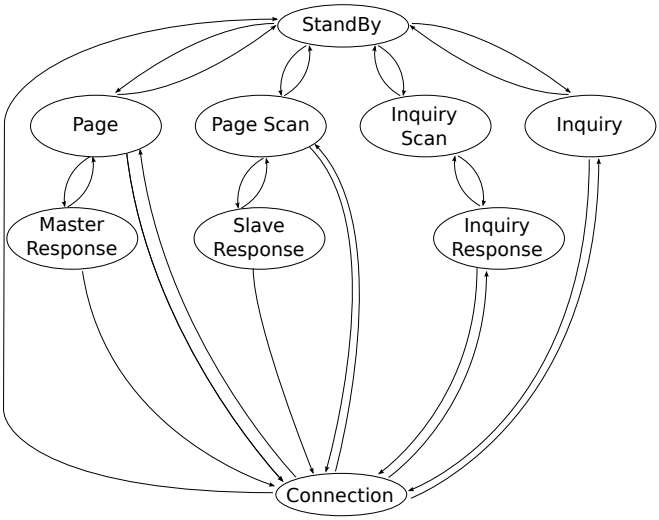


Figura 3: Máquina de estados *Bluetooth* para estabelecimento de uma conexão.

meio. A técnica adotada por *Bluetooth* é FHSS, que consiste em transmissor e receptor usarem uma sequência pseudoaleatória de canais de comunicação para a troca de informações. A cada troca de mensagens, um canal de comunicação é usado. Isto permite que várias *Piconets* coexistam e troquem informações simultaneamente.

A adoção de FHSS no *Bluetooth*, faz com que dispositivos precisem acordar a sequência pseudoaleatório usada na comunicação. Apenas desta forma um canal de comunicação pode ser estabelecido entre eles. Para isto, é necessário que dois dispositivos encontrem um ao outro tanto na frequência quanto no tempo. Este problema é conhecido como problema da Descoberta de Dispositivos.

O estabelecimento de conexão *Bluetooth* envolve duas etapas: Descoberta de Dispositivos, também conhecido como *inquiry*; e, estabelecimento de conexão, conhecido como *page*. O processo de estabelecimento de conexão segue uma máquina de estados, ilustrada na Figura 3. Nela são mostradas os estados de descoberta de dispositivos (*inquiry*, *inquiry scan* e *inquiry response*) assim como os estados de estabelecimento de conexão (*page*, *page scan*, *slave scan* e *master response*) que serão detalhados a seguir.

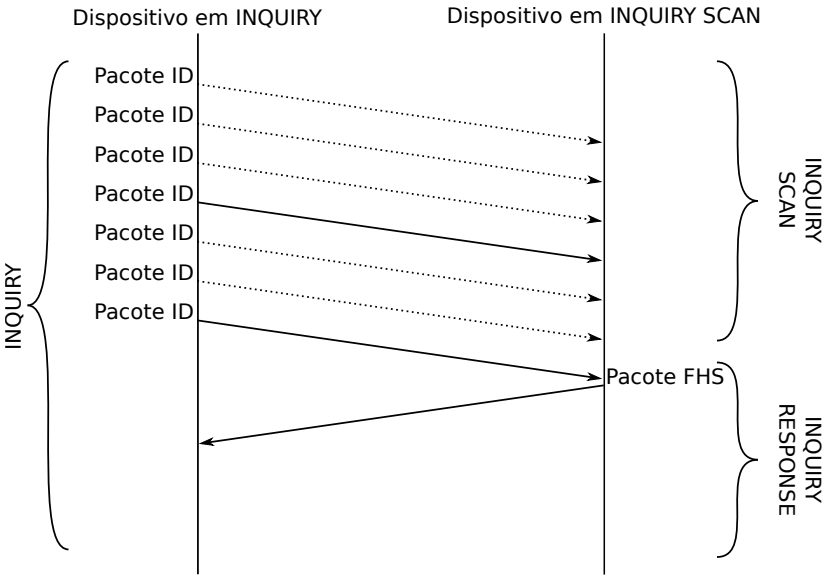


Figura 4: Diagrama ilustrando os pacotes trocados durante o processo de INQUIRY. Mensagens pontilhadas são perdidas.

3.2.1 Processo de Descoberta de Dispositivos

Dispositivos *Bluetooth* que desejam participar do processo de descoberta de dispositivos utilizam o protocolo cuja troca de mensagens é ilustrada na Figura 4. Este protocolo possui os estados INQUIRY e INQUIRY SCAN. Os dispositivos devem escolher um destes para ingressar. O procedimento de descoberta é assimétrico. Dispositivos que estão no estado de INQUIRY descobrem dispositivos que estão no estado INQUIRY SCAN. Os dispositivos em INQUIRY SCAN não obtêm informações dos dispositivos que estão em INQUIRY, apenas enviam informações sobre si para que dispositivos em INQUIRY sejam capazes de estabelecer uma conexão com eles. Assim como um dispositivo em INQUIRY não é capaz de descobrir outro dispositivo no mesmo estado.

De acordo com a especificação *Bluetooth*, quando um dispositivo *Bluetooth* deseja descobrir outros dispositivos ao seu redor ele deve entrar no estado INQUIRY. Quando no estado INQUIRY, o dispositivo *Bluetooth* anuncia sua existência fazendo o *broadcast* de um pacote especial de 68 bits conhecido como pacote ID. Apesar de conhecido como pacote ID, não possui

LSB											MSB
34	24	1	1	2	2	8	16	24	3	26	3
Bits de paridade	LAP	EIR	não definido	SR	Reservado	UAP	NAP	Classe do Dispositivo	LT_ADDR	CLOCK	Modo Page Scan

Figura 5: Pacote de dados FHS transmitido usado no procedimento de INQUIRY.

nenhuma informação sobre a identidade do emissor. O pacote ID é enviado em diferentes frequências na esperança de encontrar algum outro dispositivo ouvindo na mesma frequência, ao mesmo tempo. O dispositivo em INQUIRY alterna em uma sequência pré-definida de 32 frequências dentre as 79 usadas pelo *Bluetooth*.

Em contrapartida, dispositivos que desejam ser descobertos entram no estado INQUIRY SCAN. Uma vez nesse estado o dispositivo passa a ouvir o meio em busca de pacotes ID gerados pelos dispositivos no estado INQUIRY. Os dispositivos no estado INQUIRY SCAN alternam entre as frequências que dispositivos no estado INQUIRY transmitem pacotes ID. Entretanto, a velocidade em que o dispositivo em INQUIRY SCAN realiza os saltos é bem menor que dispositivos em INQUIRY. Quando um dispositivo no estado INQUIRY SCAN recebe um pacote ID, ele responde fazendo o *broadcast* de um pacote chamado FHS (*Frequency Hopping Synchronization*). O pacote FHS, cujo formato é mostrado na Figura 5, possui duas informações essenciais ao processo de INQUIRY sobre o dispositivo no estado INQUIRY SCAN: o seu endereço *Bluetooth* de 48 bits (formado pelos campos UAP, LAP e NAP), e o valor do seu *clock*. O *clock* é usado para sincronização durante o estabelecimento de conexão.

Nós no estado INQUIRY usam seus relógios para gerar a sequência de saltos dentre as 32 frequências usadas pelo dispositivo que está no estado INQUIRY. Este conjunto de 32 frequências é segmentado em dois trens, chamados de A e B. Cada um possui 16 frequências.

Ao iniciar o processo de INQUIRY, o dispositivo escolhe qual dos dois trens usar. A escolha é arbitrária. Cada trem de frequências é repetido 256 vezes. Dado que existem 16 frequências dentro de um trem, um dispositivo *Bluetooth* leva $10ms$ ($625\ \mu s \times 16$) para transmitir em todas as frequências. Após isso, troca-se o trem de frequência. Como cada trem é repetido 256 vezes, a cada $2,56\ s$ ($256 \times 10\ ms \times 16 = 2,56\ s$) é trocado o trem em uso.

A frequência usada durante o processo de INQUIRY é calculada segundo a fórmula:

$$F = [CLK_{16-12} + k + (CLK_{4-2,0} - CLK_{16-12})\ mod\ 16]\ mod\ 32$$

em que F e a frequência usada para INQUIRY, k indica o trem que está em uso ($k = 24$ para o trem A, e $k = 8$ para trem B), CLK_{i-j} se referem ao número representado pelos bits entre o $i^{\text{ésimos}}$ e $j^{\text{ésimos}}$ bit do *clock*, inclusive. O *clock* é referenciado como CLK_0. *CLK* ou CLK_1 é igual ao CLK_0 dividido por dois (ou seja, incrementado a cada 625 μs). A cada 1.28 segundos, CLK_{16-12} é incrementado, o que faz com que uma frequência seja trocada entre os trens A e B.

Na versão 1.2 do *Bluetooth* foi introduzido um modo de Busca por Dispositivos chamado modo entrelaçado. Seu objetivo é otimizar o processo de busca através do uso de dois trens de frequência simultaneamente pelo processo em INQUIRY SCAN. O processo em INQUIRY SCAN seleciona duas frequências, uma de cada trem. Então, ouve durante um tempo $T_{w\_inquiry\_scan}$ mensagens enviadas usando a frequência do primeiro trem. Em seguida, o processo é repetido usando a frequência do segundo trem, pelo mesmo período de tempo. Após dois períodos $T_{w\_inquiry\_scan}$ o processo em INQUIRY SCAN seleciona outras 2 frequências e repete o processo. Espera-se que o tempo em que cada dispositivo ouve em uma dada frequência em busca de pacotes ID ($T_{w\_inquiry\_scan}$) seja menor que a metade do tempo usado no modo não-entrelaçado.

3.2.2 Processo de Page e Page Scan

Uma vez que o endereço *Bluetooth* de outro dispositivo é conhecido, através do processo de INQUIRY, uma conexão *Bluetooth* pode ser estabelecida com ele. O processo de estabelecimento da conexão *Bluetooth* é conhecido como *paging*.

O *paging* possui uma dinâmica semelhante ao processo de descoberta de dispositivos. Assim como o processo de INQUIRY, o processo de estabelecimento de conexão é assimétrico. O dispositivo mestre entra no estado PAGE, enquanto o escravo entra no estado PAGE SCAN. O dispositivo mestre deve ter conhecimento do endereço *Bluetooth* do escravo para que seja possível o estabelecimento da conexão. Entretanto, o escravo não sabe quem irá conectar-se a ele. A Figura 6 ilustra como o procedimento de *paging* ocorre.

Ao iniciar o protocolo, o dispositivo escravo escuta por pacotes enviados pelo mestre. Assim como no processo de INQUIRY, o escravo troca periodicamente a frequência na qual está escutando o meio. A sequência de frequências usadas pelo escravo é determinada usando o endereço *Bluetooth* do escravo. O dispositivo mestre, em PAGE, por ter conhecimento do endereço *Bluetooth* do escravo, é capaz de calcular a sequencia de 32 frequências usada pelo escravo. Entretanto, o mestre não se sabe em qual frequência es-

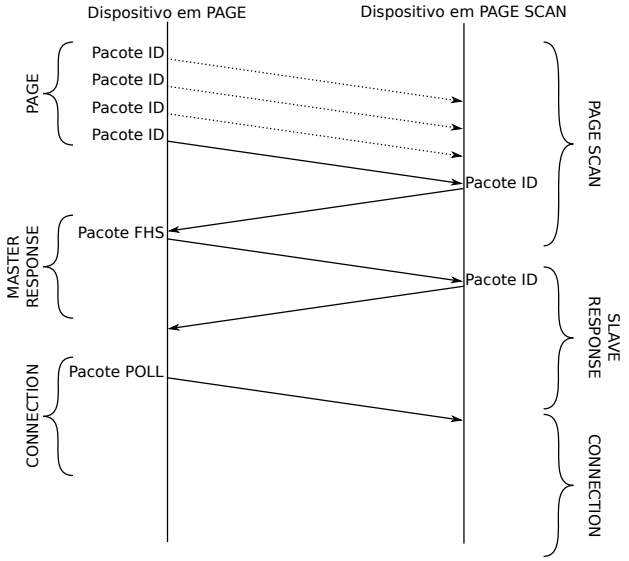


Figura 6: Diagrama ilustrando os pacotes trocados durante o processo de PAGE. Mensagens pontilhadas são perdidas.

pecífica o escravo está ouvindo, pois seus relógios não estão sincronizados.

Para descobrir a frequência que o escravo está ouvindo, o mestre faz uma estimativa do *clock* do escravo. A estimativa costuma ser feita usando-se informações de comunicações passadas. Por exemplo, usando informações de CLK recebidas no pacote FHS durante o processo de INQURIY. Usando esta informação, o mestre pode estimar o canal em que o dispositivo está ouvindo. Quanto mais precisa a estimativa, menos tempo é necessário para que o mestre encontre a frequência do escravo.

O procedimento de *paging* independe da estimativa do *clock* do escravo. Durante o *paging*, é usado um mecanismo capaz de sincronizar os relógios do mestre e escravo, viabilizando a comunicação. O mestre transmite uma série de pacotes ID usando não só a frequência calculada a partir do *clock* estimado, mas usando frequências próximas. Caso, após a transmissão do pacote ID, o mestre não receba uma resposta do escravo (por meio de um pacote ID) o mestre passa a usar frequências cada vez mais longe do ponto onde ele previamente havia estimado que o escravo estaria transmitindo.

Quando o escravo recebe um pacote ID do mestre, indicando que o mestre o encontrou, ele envia a resposta ao mestre (pacote ID) e entra no estado SLAVE RESPONSE. De forma análoga, ao receber a mensagem do

escravo, o mestre ingressa no estado MASTER RESPONSE. No momento em que esta troca de mensagens ocorre, há uma sincronização entre os clocks mestre e escravo. Ou seja, tanto o mestre quanto o escravo sabem em que ponto da sequência de frequências cada um estará recebendo e transmitindo mensagens ao longo do tempo. Isto viabiliza a troca de informações para estabelecimento do canal de comunicação.

Ao entrar no estado MASTER RESPONSE, um pacote FHS é transmitido pelo mestre ao escravo. O escravo, no estado SLAVE RESPONSE, ao recebê-lo é informado do *clock* e endereço *Bluetooth* do mestre. Uma mensagem ID é enviada ao mestre confirmando a recepção das informações e o escravo entra no estado CONNECTION. Até este instante, a transmissão de pacotes e as frequências de escuta são feitas segundo a sequência de frequências calculadas a partir do endereço *Bluetooth* e *clock* do escravo. Neste momento, o escravo passa a usar a sequência de frequências da *Piconet*, calculadas com o *clock* e endereço *Bluetooth* do mestre.

Ao receber o pacote ID de confirmação do cliente, o mestre também passa a usar a sequência de frequências da *Piconet*, calculadas a partir de seu endereço e *clock*. Então o mestre transita ao estado CONNECTION, e envia um pacote POLL ao escravo. Com isto, é caracterizado o início da atividade do escravo na *Piconet* e a conclusão do processo de estabelecimento da conexão.

3.3 ALGORITMOS PARA CONSTRUÇÃO DE *SCATTERNETS*

A formação de *Scatternets* (ou *Bluetooth Scatternet Formation* - BSF) consiste em definir como as *Piconets* são formadas a partir de um conjunto de dispositivos *Bluetooth* que se conhecem entre si, assim como a topologicamente da rede *Bluetooth* resultante (JEDDA et al., 2013). Os algoritmos BSF devem atribuir o papel de mestre, escravo ou ponte a cada um dos participantes da rede e assegurar a conectividade desta, respeitando ainda as restrições impostas pela tecnologia *Bluetooth*, como o limite de sete escravos por mestre.

Os algoritmos BSF tentam resolver o problema de formação da *scatternet* otimizando diferentes métricas. As principais métricas segundo (JEDDA et al., 2013) são:

- Tempo: o tempo necessário para formar uma *scatternet* conectada, envolvendo todos os dispositivos.

- Número de *piconets*: minimizando-o, mantendo a maioria dos mestres com sete escravos em suas *Piconets*.

- Número de pontes mestre-escravo e escravo-escravo: priorizando pontes EE sobre pontes ME. A vazão oferecida por pontes ME é menor que pontes EE devido ao período em que a ponte deixa de atuar como mestre para atuar como escravo em outra *Piconet*. Durante este período, os escravos da *Piconet* são incapazes de trocar dados devido a falta de um mestre ativo.

- Número de papéis por nó: igual ao número de *Piconets* as quais um nó pertence. Quando minimizado, o número médio de papéis propicia ganhos de eficiência na troca de mensagens.

Os algoritmos BSF podem ser classificados baseado em características em (STOJMENOVIC; ZAGUIA, 2006):

- Algoritmos *Single-hop* vs. *multi-hop*: algoritmos *single-hop* pressupõem que todos os dispositivos encontram-se dentro do alcance de comunicação *Bluetooth*, permitindo que quaisquer dois destes se conectem. Os algoritmos *Multi-hop* correspondem ao caso da existência de dispositivos fora do alcance de comunicação um do outro, necessitando o roteamento de mensagens. Ressalta-se que a *Scatternet* resultante de algoritmos Single-hop podem necessitar roteamento pela existência de nós a 2 ou mais saltos de distância. Esta nomenclatura no contexto de algoritmos BSF diz respeito apenas a possibilidade de conexão entre dispositivos *Bluetooth*, e não a topologia da *Scatternet* resultante.

- Algoritmos centralizados vs distribuídos: no caso centralizado, o algoritmo BSF é executado numa unidade central de processamento, como um servidor externo ou um líder. BTCP (SALONIDIS et al., 2005) é um exemplo de algoritmo desta classe. No caso distribuído, o processamento é local, dados são trocados entre vizinhos e a decisão da topologia final da rede é realizada de forma distribuída.

- Descoberta dinâmica vs descoberta estática: em algoritmos com descoberta estática, existe uma fase preliminar de descoberta responsável por identificar todos os dispositivos *Bluetooth* próximos. BlueStar (PETRIOLI; BASAGNI; CHLAMTAC, 2003), BlueMesh (PETRIOLI; BASAGNI; CHLAMTAC, 2004), BlueNet (WANG; THOMAS; HAAS, 2002), BlueMIS (ZAGUIA; DAADAA; STOJMENOVIC, 2008) são algoritmos com descoberta estática. Os algoritmos com descoberta dinâmica não possuem uma etapa preliminar de descoberta, sendo este procedimento realizado junto ao algoritmo de formação. Portanto, algoritmo com descoberta dinâmica são melhores adaptados a cenários onde o conjunto de dispositivos não é conhecido a priori, ou quando

Algoritmo	Single-hop	Multi-hop	Descoberta dinâmica	Descoberta estática
BTCP	X			X
BlueStar		X		X
BlueMesh		X		X
BlueNet		X		X
BlueMis		X		X
Law	X		X	
BTSpin		X	X	
TSF	X		X	
Shaper		X	X	
Beddernet		X	X	

Tabela 1: Algoritmos BSF e suas características

novos dispositivos surgem durante o processo de formação de *Scatternet*.

Há na literatura diversas propostas de algoritmos BSF. A Tabela 1 apresenta os principais algoritmos e como estes são classificados.

A aplicação de transporte coletivo possui particularidades que favorecem algoritmos Single-hop. Primeiramente, os dispositivos estão dentro do alcance um do outro, dado que o alcance de 20 metros do *Bluetooth* de *smartphones* é suficiente para cobrir todo o ônibus. Em segundo lugar, devido aos embarques e desembarques de passageiros, há uma constante entrada e saída de nós da rede. Em vista destas observações, o algoritmo BSF ideal para esta aplicação deve ser *Single-hop* com descoberta dinâmica. Dentre os algoritmos constantes na Tabela 1, dois possuem as duas características: Law (LAW; MEHTA; SIU, 2003) e TSF (TAN et al., 2002).

No que segue descreveremos o funcionamento dos dois algoritmos de formação de *Scatternet* mais relevantes ao presente trabalho: Law e TSF. Estes dois algoritmos realizam descoberta dinâmica de nós, ou seja, de forma integrada ao algoritmo de formação. Portanto, não possuem uma etapa preliminar de descoberta de nós. Devido à integração do algoritmo com o sistema de descoberta de nós, algoritmos pertencentes a esta classe costumam adaptar-se bem a situações onde há entrada e saídas de nós da *scatternet*. É indiferente do ponto de vista do algoritmo se um dispositivo acabou de chegar ou se já estava presente desde o início em seus arredores mas demorou a ser descoberto. A desvantagem em relação a algoritmos com descoberta estática é que a visão mais abrangente provida pela fase de coleta permite uma melhor otimização que algoritmos dinâmicos. Os algoritmos dinâmicos criam a *scat-*

ternet através da união de componentes menores em componentes maiores. Eventualmente um único componente é formado incorporando todos os nós.

Para fins de completude, os demais algoritmos enumerados na tabela 1 serão brevemente descritos posteriormente. Além de algoritmos multi-hop com descoberta dinâmica, também abordaremos algoritmos com descoberta estática.

3.3.1 Algoritmo Law

Law (LAW; MEHTA; SIU, 2003) é um dos primeiros algoritmos BSF dinâmicos a serem propostos na literatura. Seu objetivo é alcançar um balanço adequado entre: o número de *Piconets*, o número de papéis por nó e o diâmetro da *Scatternet*. Ele é caracterizado como um algoritmo distribuído, *single-hop*, e com descoberta dinâmica.

Law define um *componente* como sendo um nó desconectado, uma *piconet*, ou uma *scatternet*. Um dos mestres do componente atua como o seu *líder*. Caso o componente seja formado por um nó desconectado, então ele próprio é o líder do componente. Um componente deve satisfazer as seguintes propriedades:

(i) cada líder ou possui pelo menos um escravo não-compartilhado ou não possui nenhum escravo;

(ii) cada mestre possui menos de sete escravos.

O algoritmo Law forma a *scatternet* unindo dois componentes. Durante cada rodada de duração δ, o líder do componente tenta unir-se com outro componente. Ele realiza o processo de INQUIRY ou requisita que um de seus escravos não compartilhados entre no estado de INQUIRY SCAN. Caso o componente seja um nó desconectado, ele próprio entra no estado INQUIRY SCAN. A decisão entre realizar o processo de INQUIRY ou entrar no estado de INQUIRY SCAN é feita aleatoriamente pelo líder do componente com probabilidade P de realizar o processo de INQUIRY. Law espera que todas as uniões sejam completadas dentro do tempo de duração δ da rodada.

Quando o líder realizando o processo de INQUIRY descobre um nó no estado INQUIRY SCAN, Law estabelece uma conexão entre eles, com o nó no estado INQUIRY tornando-se mestre do nó em estado INQUIRY SCAN. Dados sobre a estrutura dos componentes são trocados e um processo de reorganização da topologia ocorre. Law utiliza a seguinte nomenclatura durante o processo de reorganização: u é o líder realizando INQURIY; v é o nó realizando INQUIRY SCAN; w é o líder do componente de v. O algoritmo de Law define 6 reorganizações possíveis dependendo da estrutura dos componentes:

1. Quando *v* também é o líder *w* (ou seja, o componente é composto de apenas um nó desconectado) e *u* possui menos de sete escravos.
2. Quando *v* também é o líder *w* e *u* possui exatamente sete escravos.
3. Quando as *Piconets* de *u* e *w* são unidas em uma única *Piconet* com menos de sete escravos.
4. Quando a *Piconet* de *y* e *w* são unidas em uma *Piconet* com sete escravos e *u* estava sozinho em seu componente.
5. Quando a *Piconet* de *u* e *w* são unidas em uma *Piconet* com sete escravos e *u* possui escravos.
6. Quando a *Piconet* de *u* e *w* não podem ser unidas em uma única *Piconet* por ultrapassarem o limite de 7 escravos.

Para cada uma destas reorganizações, o algoritmo seleciona o líder do novo componente e modifica as conexões de tal forma que as propriedades sejam mantidas. Uma vez finalizado o processo é repetido em uma outra rodada pelo líder do novo componente.

3.3.2 TSF

TSF (TAN et al., 2002) é um algoritmo BSF distribuído, *single-hop*, e com descoberta dinâmica. Os componentes no algoritmo TSF são árvores. Portanto, grafo da rede sendo formada pelo TSF, a qualquer instante, é uma floresta. TSF classifica os nós de acordo com o papel desempenhado por eles dentro da estrutura de árvore da *Scatternet* a qual pertence. Nó raiz é um nó que não possui um pai. Cada árvore possui um nó raiz. Nó livre é uma árvore de tamanho 1. Nó da árvore são os nós que não são nem nó raiz e nem nó livre.

Nós alteram entre estados de INQUIRY e INQUIRY SCAN, tentando integrar-se a uma outra árvore. Ao encontrar uma outra árvore, as duas árvores são unidas, respeitando as seguintes três regras:

1. nós livre podem se conectar apenas a outros nós livres ou a nós da árvore. No primeiro caso, um dos nós se torna mestre enquanto o outro se torna escravo na *Piconet* recém formada. No segundo caso, o nó livre se torna escravo do nó da árvore.
2. Nós raízes de árvores com mais de um nó podem se conectar apenas a outros nós raízes. Uma das raízes se torna mestre enquanto a outra se torna escrava na *Piconet* recém formada.

3. Nós de árvore não tentam formar árvores maiores com nós que não são livres.

Seguindo estas regras na formação da rede, ao fim do processo uma *Scatternet* é formada com topologia de árvore. TSF não impõe limites ao número de escravos dos nós, e a árvore resultante não é balanceada. Logo, casos podem ocorrer onde a árvore se degenera em uma lista.

3.3.3 Demais algoritmos com descoberta dinâmica

3.3.3.1 BTSpin

BTSpin (GHOSH et al., 2004) é um algoritmo BSF distribuído, com descoberta dinâmicas e *multi-hop*. O principal objetivo de BTSpin é prover uma solução realista em um cenário *ad-hoc*. Por este motivo BTSpin adota topologia *mesh* ao invés de uma baseada em árvore. Segundo Petrioli e Basagni (2002), a topologia *mesh* é superior nestes cenários.

BTSpin propõe uma técnica chamada *spin*, na qual cada mestre de uma *Piconet* agenda a execução de um "spin" pelos seus escravos. A ordem de "spin" dos escravos segue um esquema *round-robin*. Um nó realizando *Spin* realiza um processo de INQUIRY e em seguida entra no estado de INQUIRY SCAN. Enquanto um escravo está realizando o *spin*, o mestre pode continuar se comunicando com os demais escravos. Desta forma é possível comunicar-se durante a execução de BTSpin. Ao término do *spin* de um escravo, o mestre realiza um *spin*. Note que pontes escravo-escravo não realizam *spin*, pois o número máximo de *Piconets* desempenhados por um nó em BTSpin é dois. De forma análoga, ao conectar-se a sete escravos o mestre deixa de realizar *spin*.

Assim que dois nós em *spin* descobrem um ao outro, uma *Piconet* temporária é formada entre eles. Nesta *Piconet*, o nó que estava no estado INQUIRY assume papel de mestre e o outro nó o papel de escravo. Informações sobre os nós e suas *Piconets* são trocadas e os componentes são mesclados, tornando-se um.

Os autores mostram, através de simulação, que o algoritmo forma *Scatternets* com características comparáveis a outros algoritmos na literatura. Entretanto, não é limitado a ambientes *single-hop* como no caso de Law e TSF.

3.3.3.2 SHAPER

SHAPER (CUOMO; BACCO; MELODIA, 2003) é um algoritmo BSF distribuído, com descoberta dinâmica e multi-hop. SHAPER organiza as *Scatternet* em topologias baseadas em árvore. Entretanto, complexidade adicional é introduzida no algoritmo para torná-lo multi-hop.

Os nós são classificados em:

1. **raiz** - um nó que não possui um pai. Cada árvore possui um nó raiz.

2. **livre** - é uma árvore de tamanho 1.

3. **não-raiz** - Nó da árvore são os nós que não são nem nó raiz e nem nó livre.

O funcionamento de SHAPER é semelhante ao funcionamento de STF. Nós alternam entre estados INQUIRY e INQUIRY SCAN para descobrir um ao outro.

O procedimento de união ocorre sempre que dois nós de árvores distintas descobrem um ao outro. Os nós formam uma *Piconet* temporária, usada para coordenar a reorganização. No que segue, S_B é o escravo desta *Piconet* temporária, M_A é o mestre da *Piconet* temporária, T_A e T_B são as árvores as quais M_A e S_B pertencem, respectivamente. SHAPER define quatro procedimentos para união de T_A e T_B, dependendo da função exercida por M_A e S_B em seus componentes. A Tabela 2 mostra para cada situação possível qual o procedimento de união é usado pelo algoritmo.

$S_B \setminus M_A$	Raiz	Não-raiz	Livre
Raiz	A_1	A_2	A_3
Não-raiz	A_3	A_4	A_3
Livre	A_2	A_2	A_2

Tabela 2: Procedimentos de união realizados por SHAPER

O procedimento A_1 ocorre quando os dois nós M_A e S_B são raízes de suas árvores. Neste caso, a raiz de uma das árvores se torna pai da outra. A raiz da árvore resultante será a raiz da maior árvore.

O procedimento A_2 é executado em quatro casos: (1) quando um nó raiz se torna mestre de um nó livre; (2) quando um nó não-raiz se torna mestre de um nó raiz; (3) quando um nó não-raiz se torna mestre de um nó livre; (4) quando um nó livre se torna mestre de outro nó livre. Nestes casos, nenhuma ação adicional é necessária para mesclar as árvores.

O procedimento A_3 é executado: (1) quando um nó raiz se torna mestre de um nó não-raiz; (2) quando um nó livre se torna mestre de um nó raiz; (3) quando um nó livre se torna mestre de um nó não raiz. Nestes casos, os papéis dos nós são invertidos. Isto é, o mestre se torna escravo e o escravo se torna mestre.

O procedimento A_4 ocorre quando um nó não-raiz se torna mestre de outro nó não-raiz. Neste caso, uma das árvores é modificada permitir a união. Para isto, o algoritmo exige que T_B seja menor que T_A. Caso isso não seja verdade no início do procedimento de união, o algoritmo faz $T_B = T_A$, $T_B = T_A$, $M_A = S_B$, $S_B = M_A$. Para manter a coerência, os papéis da conexão entre M_A e S_B são invertidos. Em seguida, os papéis de todas as conexões encontradas entre S_B a raiz da árvore T_B são invertidas para que a nova raiz da árvore T_B possa ser mantida como filha de M_A. A reconfiguração de T_B é feita recursivamente. O procedimento recursivo consiste em, dado um nó N_K: (1) se N_K é um nó raiz, a reorganização está finalizada. (2) senão, inverte-se os papéis de N_K e seu pai. Em seguida, aplica-se este procedimento ao pai de N_K. Aplicando este procedimento em T_B a partir do nó S_B faz com que S_B se torne um nó raiz em T_B.

Seguindo estas reconfigurações, SHAPER gera uma *Scatternet* com topologia em árvore. Ao contrário de TSF, SHAPER não se limita a ambiente *single-hop*. Ao definir um meio de nós não-raiz se unirem a qualquer outro nó que descobrir, a restrição imposta por TSF é eliminada.

3.3.3.3 Beddernet

Beddernet (GOHS; GUNNARSSON; GLENSTRUP, 2011) se propõe a ser mais que um algoritmo BSF, ao contrário dos outros trabalhos descritos neste capítulo. Beddernet se denomina como um *middleware* voltado ao desenvolvimento de aplicações distribuídas sobre redes espontâneas criadas usando *Bluetooth*. Além da função de algoritmo BSF, Beddernet possui um algoritmo de roteamento baseado em DSDV, um sistema de descoberta de serviços, métodos de descoberta de aplicações entre outros aspectos relacionados à sua função como *middleware*. Beddernet possui uma implementação de referência *open-source* disponível. Esta implementação foi feita em Android, e devido a isto, foi usada como ponto de partida para o protótipo descrito no anexo B.

O algoritmo BSF usado por Beddernet pode ser classificado como um algoritmo com descoberta dinâmica e multi-hop. O algoritmo funciona da seguinte forma: cada dispositivo ao ser iniciado decide se irá realizar o procedimento de INQUIRY ou o procedimento de INQUIRY SCAN. A decisão

é tomada aleatoriamente, sendo que há uma probabilidade de 33% do dispositivo realizar o procedimento de INQUIRY. Caso o dispositivo ingresse no estado de INQUIRY SCAN, ele permanece neste estado por um tempo aleatório entre 0 e 2 s. Durante este intervalo de tempo o dispositivo pode ser descoberto por outros que estejam no estado de INQUIRY. Caso o dispositivo ingresso no estado INQUIRY, o dispositivo realiza a busca por vizinhos que estejam no estado INQUIRY SCAN. O dispositivo permanece neste estado por 10.24 segundos, tempo indicado para realização de busca por dispositivos *Bluetooth* e encontrar todos os vizinhos que estejam no estado INQUIRY SCAN (HUANG; RUDOLPH, 2007). Finalizado o processo de INQUIRY, o algoritmo pode evoluir de duas maneiras: (1) caso nenhum dispositivo tenha sido encontrado; ou, (2) dispositivos foram encontrados. No caso (1) o dispositivo ingressa automaticamente no estado INQUIRY SCAN, onde permanecerá por até 2 s. No caso (2) tenta-se estabelecer conexões *Bluetooth* com todos os dispositivos encontrados até que o limite de 7 escravos sejam atingidos. Caso não possua nenhum escravo após tentar estabelecer conexões com os dispositivos encontrados, o dispositivo ingressa no estado INQUIRY SCAN onde permanece por até 2 s. Por outro lado, se ele possui escravos, o *smartphone* irá realizar o processo de Inquiry scan mas o tempo máximo que permanecerá neste estado é proporcional ao número de escravos que possui. A ideia por trás deste comportamento é que dispositivos com poucos escravos devem fazer INQUIRY com maior frequência que aqueles com muitos escravos. Desta forma o número de escravos por mestre é balanceado na rede. O cálculo do tempo máximo de espera de um dispositivo com escravos é dado por $(n+1) \times 60\ s$, sendo n o número de escravos do dispositivo. Por exemplo, caso um dispositivo tenha 4 escravos após estabelecer as conexões $(4+1) \times 60\ s = 300\ s$. Então, um número aleatório entre 0 e 300 s será sorteado e o dispositivo permanece em INQUIRY SCAN durante este período. Findo este período, o dispositivo ingressa no estado de INQUIRY.

Um protótipo foi desenvolvido que utiliza Beddernet como algoritmo para formação da *Scatternet* entre *smartphones*, descrito em detalhes no anexo B. A implementação se mostrou insatisfatória e instável. Redes com poucos dispositivos levavam minutos para serem formadas. Eventualmente o algoritmo parava de evoluir, e novas conexões não eram formadas. Creditou-se estes problemas a erros de sincronização de threads na implementação do sistema.

3.3.4 Algoritmos com descoberta estática

Algoritmos BSF com descoberta estática assumem que cada nó conhece a identidade de um número suficiente de vizinhos para garantir a conectividade da rede antes do início da execução do algoritmo BSF. Logo, há uma fase preliminar externa ao algoritmo de descoberta de dispositivos.

A fase de descoberta preliminar utilizada em todos os algoritmos com descoberta estática são derivados da técnica proposta por Salonidis et al. (2005). Nesta proposta, todos os nós alternam entre os estados INQUIRY e INQUIRY SCAN, permanecendo em um destes estados por um período de tempo aleatório. Esta técnica torna o processo assimétrico de INQUIRY, em um processo simétrico. O pseudo-código descrevendo esta técnica é mostrado no Algoritmo 1. No que segue, esta técnica será referida como VARREDURA_ALTERNANTE.

Algoritmo 1 Pseudo-código ilustrando a técnica de descoberta de dispositivos *Bluetooth* proposta em (SALONIDIS et al., 2005)

```
1: procedure VARREDURA_ALTERNANTE
2:     p ← random number between 0 and 1
3:     if p < 0.5 then
4:         INQUIRYFOR(random(T_min, T_max))
5:         INQUIRYSCANFOR(random(T_min, T_max))
6:     else
7:         INQUIRYSCANFOR(random(T_min, T_max))
8:         INQUIRYFOR(random(T_min, T_max))
9:     end if
10: end procedure
```

O método de VARREDURA_ALTERNANTE foi aperfeiçoado em trabalhos posteriores (BASAGNI et al., 2004; SATO; MASE, 2002). Estes trabalhos propõem que tão logo um dispositivo descubra outro, uma *Piconet* temporária seja estabelecida para garantir o conhecimento mútuo entre eles. Ou seja, caso um nó A descubra um nó B, adicionando-a a sua lista de vizinhos, garante-se que o nó A está lista de vizinhos de B.

Vários algoritmos BSF com descoberta estática pressupõem que durante a fase de descoberta os nós conhecem todos os vizinhos dentro do seu raio de comunicação. Desta forma, pode-se modelar a rede como um grafo de disco unitário onde os nós são os vértices dos grafos e as arestas indicam que um nó é vizinho de outro. Entretanto, o método de VARREDURA_ALTERNANTE não é capaz de determinar todos os vizinhos dos nós em tempo hábil. Simulações apresentadas por Basagni et al. (2004) mostram que, após 20 segundos de simulação, os nós não haviam descoberto todos os seus vizinhos.

Duas solução foram propostas para este problema. A primeira solu-

ção, apresentada por Basagni et al. (2004), adiciona uma fase posterior a fase de descoberta na qual os nós se conectam a vizinhos para trocar a lista de todos os nós conhecidos. Em seguida, cada nó tenta estabelecer uma conexão com todos os nós conhecidos que não sejam vizinhos. Caso a conexão seja estabelecida com sucesso, o nó é adicionado a lista de vizinhos. Mostrou-se que esta técnica oferece uma garantia estatística de um grafo de disco unitário ser criado.

A segunda solução, apresentada por Dubhashi et al. (2007), é uma modificação da técnica de VARREDURA_ALTERNANTE. Cada nó descobre ou é descoberto por no máximo c vizinhos; sendo c uma constante igual a 5, 6 ou 7. Os autores provam analiticamente que esta modificação gera com alta probabilidade uma rede conectada. O uso desta técnica de descoberta em conjunção com o algoritmo BlueStars (PETRIOLI; BASAGNI; CHLAMTAC, 2003) é conhecido como Blue Pleiades (DUBHASHI et al., 2007). A limitação desta técnica é que a argumentação dos autores acerca da probabilidade da rede ser conectada é baseada na premissa de que os nós estão distribuídos uniformemente em um plano.

3.3.4.1 BTCP

BTCP (SALONIDIS et al., 2005) é um algoritmo BSF centralizado, *single-hop* com descoberta estática. BTCP opera em 3 fases: (1) eleição de coordenador; (2) determinação de papéis; (3) estabelecimento de conexões.

Durante a fase 1, um processo assíncrono e distribuído de eleição de coordenador é iniciado. O coordenador, uma vez eleito, saberá o número, identidades e os *clocks* de todos os nós participantes do processo de construção da *scatternet*.

O processo de eleição é similar ao processo de busca por vizinhos VARREDURA ALTERNANTE. Nós alternam entre os estados INQUIRY e INQUIRY SCAN aleatoriamente. Cada nó possui uma variável associada chamada $VOTES$, inicialmente com valor 1. Uma vez que um dispositivo encontra outro através do processo de INQUIRY uma conexão *Bluetooth* é estabelecida entre eles. O nó com maior $VOTES$ é dito o vencedor. Caso tenham o mesmo valor para $VOTES$, o nó com maior endereço *Bluetooth* é declarado o vencedor. A variável $VOTES$ do vencedor é incrementada com o valor da $VOTES$ do perdedor. O perdedor repassa ao vencedor as informações necessárias para estabelecimento de uma conexão *Bluetooth* de todos os nós que ele derrotou e sobre si mesmo. Em seguida, o vencedor encerra a conexão com o perdedor. Este procedimento é realizado $N-1$ vezes, sendo N o número de nós que irão formar a *scatternet*. O vencedor do último confronto

será o líder.

A fase 2 consiste em atribuir papéis a cada um dos nós, determinando quem será mestre e/ou escravo de quem. Primeiramente, é calculado o número P de *Piconets* da *Scatternet*. O coordenador se torna mestre, e escolhe outros $P-1$ nós para atuarem como mestres. $\frac{P(P-1)}{2}$ nós são escolhidos para atuarem como pontes. Os demais nós são distribuídos entre os mestres para atuarem como escravos. Por fim o coordenador estabelece uma conexão temporária com os mestres para informar quem são os seus escravos e pontes.

Durante a fase 3, os mestres, escravos e pontes estabelecem as conexões definidas pelo coordenador na etapa 3.

3.3.4.2 BlueStar

O algoritmo *BlueStar* (PETRIOLI; BASAGNI; CHLAMTAC, 2003) é um algoritmo BSF distribuído, *multi-hop* com descoberta estática. BlueStar possui três fases: (1) descoberta da topologia; (2) formação de *piconets*; (3) conexão das *piconets* em *scatternets*.

A primeira etapa consiste na descoberta de dispositivos vizinhos. BlueStar exige que o conhecimento de vizinhança seja mútuo. Isto é, um nó A possui o nó B em sua lista de vizinhança se, e somente se, o nó B também possuir o nó A em sua lista de vizinhança.

A segunda etapa consiste na formação de *Piconets*. Cada nó possui um número identificador único associado. Nós com maior ID entre seus vizinhos iniciam processo de captura dos seus vizinhos com menor ID como escravos em uma *Piconet*. Durante esta fase, é imposta a condição de que um escravo pertence a apenas uma *Piconet*. Caso um nó descubra que todos os seus escravos com ID maior se tornaram escravos, ele assume papel de mestre e passa a formar uma nova *Piconet* com os nós ainda não capturados. Ao fim deste processo, serão criadas várias *Piconets* isoladas.

A terceira etapa consiste na escolha de nós ponte, para interconectar as *Piconets*. *Piconets* vizinhas são definidas como aquelas cujos mestres podem ser interconectados através de um escravo (2 saltos) ou dois escravos (3 saltos). A escolha das pontes é feita de acordo com a seguinte regra: para quaisquer dois mestres, os escravos que formam um caminho de 2 ou 3 saltos entre eles serão pontes. Caso exista mais de um caminho de 2 ou 3 saltos entre dois mestres, pode-se escolher os nós de apenas um dos caminhos para serem ponte. Esta estratégia garante a conectividade em um cenário *multi-hop*, caso exista.

3.3.4.3 BlueMesh

O algoritmo *BlueMesh* (PETRIOLI; BASAGNI; CHLAMTAC, 2004) é um algoritmo BSF distribuído, *multi-hop* com descoberta estática. BlueMesh possui duas fases: (1) descoberta da topologia; (2) formação de *scatternet*.

Assim como outros algoritmos estáticos, o objetivo da fase de descoberta da topologia é identificar seus vizinhos. Entretanto, no BlueMesh, além dos vizinhos imediatos, cada nó identifica os nós a 2 saltos. Este algoritmo parte do pressuposto que os nós estão espalhados em um espaço bidimensional; os nós formam um grafo de disco unitário, ou seja, a aresta entre dois nós indica que a distância entre os nós é menor que o alcance de comunicação; e, o grafo não se modifica durante a execução do algoritmo.

A fase 2 de BlueMesh procede em sucessivas iterações. Cada nó realiza duas atividades durante uma iteração: seleção de papel e seleção de ponte.

Durante o processo de seleção de papel, os nós com maior ID dentre seus vizinhos, conhecidos como "nós iniciais", tornam-se mestres. Os mestres escolhem os escravos em duas etapas: primeiro escolhe-se os escravos que cobrem todos os demais. Ou seja, os nós que não foram inclusos são vizinhos destes. Em seguida completa-se a *Piconet* com os escravos remanescentes, até no máximo 7 escravos. O mestre então notifica a todos os seus vizinhos se eles serão seus escravos ou não.

Os demais nós aguardam que seus vizinhos com maior ID os informem se são ou não seus escravos. Caso não tenham se tornado escravos, ele se tornará um mestre. Eles então passam a operar da mesma forma como se fossem um "nó inicial", escolhendo seus escravos dentre seus vizinhos e notificando-os. Caso tenham se tornado escravos, então ele informa este fato a todos os vizinhos com ID menor. Esta informação é necessária para que um vizinho de ID maior possa decidir se tornar um mestre e dar seguimento ao algoritmo.

Concluída esta etapa, inicia-se o processo de seleção de ponte. Todos os escravos enviam a seus mestres informações sobre: os papéis de seus vizinhos; a lista de mestres de seus vizinhos; e, se é escravo de algum mestre. Com estas informações cada mestre decide os escravos que se tornarão pontes. Caso dois mestres vizinhos possuam escravos em comum, um deles é escolhido como ponte. Se não existir nenhum escravo em comum, são escolhidos dois escravos vizinhos para atuar como pontes-intermediárias.

As pontes, mestres e escravos terminam a execução do algoritmo. As pontes intermediárias prosseguem para a iteração $i+1$ para formar novas *Piconets* e interconectá-las.

3.3.4.4 BlueNet

O algoritmo *BlueNet* (WANG; THOMAS; HAAS, 2002) é um algoritmo BSF distribuído, *multi-hop* com descoberta estática.

O algoritmo BlueNet forma *Scatternets* seguindo três regras:

- Evitar formar *Piconets* dentro de uma *Piconet*;
- Evitar conectar uma ponte mais de uma vez a uma mesma *Piconet*;
- Manter o número de escravos equilibrado. Nem poucos, nem muitos.

BlueNet possui três fases. Durante a primeira fase, o algoritmo constrói a lista de vizinhos, de forma análoga a outros algoritmos com descoberta estática. Após finalizada a descoberta de vizinhos, inicia-se a formação de *Piconets*. Nós entram no estado PAGE aleatoriamente, tentando convidar *n* de seus vizinhos a se juntar a futura *Piconet*. Uma vez que um nó se torna um escravo, ele para de tentar estabelecer conexões e ignora pedidos. Quando a fase 1 termina, varias *Piconets* desconectadas são formadas, com alguns nós *Bluetooth* deixados desconectados de outras *Piconets*. Escravos são informados sobre os participantes da *Piconet*, desta forma evita-se formação de *Piconets* dentro da mesma *Piconet* nas próximas fases.

Durante a fase 2, nós desconectados são incorporados às *Piconets* formadas na fase 1. Durante esta fase, eventualmente um nó desconectado pode se incorporar a mais de uma *Piconet*. Neste caso, uma ponte é formada. Os nós desconectados se incorporam a no máximo *n Piconets*.

Na fase 3, os mestres das *Piconets* instruem seus escravos a formarem conexões com vizinhos que não pertencem a *Piconet*. Desta forma, há uma alta probabilidade de formação de uma *Scatternet* conectada.

3.3.4.5 BlueMIS

O algoritmo *BlueMIS* (ZAGUIA; DAADAA; STOJMENOVIC, 2008) é um algoritmo BSF distribuído, *multi-hop* com descoberta estática.

Seu objetivo é simplificar o método de formação de *Scatternets* usada no BlueMesh. Isto é feito interpretando o problema de seleção de escravos como um problema de conjunto independente máximo. Com isso, o número de iterações no BlueMIS é no máximo dois, ao passo que BlueMesh não impõe um limite superior ao número de reorganizações possíveis.

Na teoria dos grafos, um conjunto independente é um conjunto de vértices não adjecentes entre si. Ou seja, para todo par de vértices do conjunto

independente, não há uma aresta os conectando. Um conjunto independente máximo é um conjunto independente que não permite a adição de nenhum outro vértico, pois tal adição faz com que o conjunto deixe de ser um conjunto independente.

No BlueMis, ao fim do procedimento de descoberta de vizinhos os dispositivos têm acesso aos seus vizinhos de 1-salto. Durante o processo de construção do conjunto independente máximo informações são trocadas entre vizinhos a medida que for necessário o conhecimento dos vizinhos a 2-saltos. Uma vez conhecidos os vizinhos a 1-salto, inicia-se a primeira iteração. Durante esta fase, todos os nós criam sua própria *Piconet*. Os escravos de um dado nó u é igual ao conjunto independente máximo dos seus vizinhos $MIS(u)$.

Cada nó tem associado a si duas chaves: $chave1$ e $chave2$. Estas chaves são conhecidas pelos vizinhos.

Para determinar $MIS(u)$, o nó u itera sobre os seus vizinhos. A cada iteração é escolhido aquele com menor $chave1$, conhecido como v. u obtém o conjunto de vizinhos de v e o conjunto de escravos de v ($MIS(v)$). Caso $u \notin MIS(v)$, o nó descoberto v é adicionado ao conjunto $MIS(u)$ e u é declarado mestre de v. Se $u \in MIS(v)$, u compara a $chave2$ de v e u, caso a $chave2(u) < chave2(v)$, u se torna mestre de v, v é adicionado a $MIS(u)$ e u e removido de $MIS(v)$. Uma vez feito isso, v e todos os seus vizinhos são marcados como tendo sido processados.

A segunda iteração visa simplificar a estrutura da *Piconet*, eliminando *Piconets* supérfluas. Para isto são definidos dois procedimentos: um para ser executado pelos mestres e outro para os escravos. É importante que os escravos executem seu procedimento antes dos mestres iniciarem a execução dos seus.

O procedimento usado pelos escravos consiste em procurar os seus mestres que não possuem nenhum outro mestre e possuem um único escravo. Neste caso, a relação entre os dois é invertida. Ou seja, o mestre torna-se escravo. A menos que o número máximo de escravos (sete) tenha sido atingido.

Uma vez concluído este processo pelos escravos, os mestres verificam a estrutura de sua *Piconet* determinando se esta pode ser eliminada. Esta simplificação é considerada apenas pelos mestres que não são escravos de nenhuma outra *Piconet*. Estes mestres verificam se é possível tornar-se escravo. Em caso afirmativo, a *Piconet* é destruída e o antigo mestre se torna escravo.

Os autores mostram que caso BlueMIS não apresenta bom desempenho quando aplicado em grafo de disco unitário denso. Isto é, quando os nós estão todos no alcance um do outro. Para redes moderadamente densas, BlueMIS apresenta desempenho similar a BlueMesh.

3.4 CONSIDERAÇÕES FINAIS

Nesta seção abordamos as principais características da tecnologia *Bluetooth*. Foram descritas as estruturas de comunicação *Bluetooth*, baseada em *Piconets* e *Scatternets*; os procedimentos de estabelecimento de canais de comunicação entre dispositivos; e, técnicas e algoritmos existentes na literatura para formação de *Scatternets*.

Os protocolos de acesso ao meio e os mecanismos usados no *Bluetooth* para estabelecimento de canais de comunicação dificultam a formação de *Scatternets*. Além do problema da descoberta de dispositivos próximos, uma *Piconet* deve ser estabelecida entre dois dispositivos para que os mesmos possam trocar informações. O estabelecimento de uma conexão *Bluetooth* ocorre em duas etapas. Na primeira, o dispositivo entra no estado INQUIRY/INQUIRY SCAN para, em um segundo momento, estabelecer a conexão através do processo de *paging*.

É possível, em algumas situações, omitir o procedimento de INQUIRY no estabelecimento de uma *Piconet*, desde que o endereço *Bluetooth* e *clock* do dispositivo com o qual se quer conectar sejam conhecidos, informações estas enviadas no pacotes FHS pelo escravo ao mestre durante o processo de INQUIRY. Por exemplo, caso algum dispositivo na rede possua estas informações, ela pode ser compartilhada com os demais integrantes da rede. Desta forma, uma conexão pode ser estabelecida através do processo de *paging*. Esta técnica é usada por vários algoritmos de formação de *Scatternets*, como por exemplo, BTCP (SALONIDIS et al., 2005).

Em *Scatternets* esta técnica pode trazer vantagens, pois a estimativa de *clock* recebida no pacote FHS durante o processo de inquiry está bem atualizada. Isto implica em um processo de *paging* bastante rápido. Além disso, evita-se que um procedimento de busca seja realizado, o que é bastante lento.

As solução ao problema formação de *Scatternets* são separados em dois grupos: algoritmos com busca estática e algoritmos com busca dinâmica. A diferença entre eles está na forma como a busca por dispositivos *Bluetooth* próximos é realizada. Algoritmos baseados em busca estática operam em fases, determinando o conjunto de vizinhos a priori. Algoritmos baseados em busca dinâmica realizam os dois processos concomitantemente.

Algoritmos estáticos, quando comparados aos algoritmos dinâmicos, conseguem, em geral, otimizar melhor os parâmetros de desempenho e as características desejáveis da *Scatternet*. Algoritmos com descoberta estática, uma vez findo o estágio de descoberta, pressupõem que novos dispositivos não serão incorporados na *Scatternet*. Ou seja, há conhecimento global de vizinhança durante a montagem da *Scatternet*. Mesmo que um nó tenha ape-

nas conhecimento parcial da vizinhança (vizinhos a 1 ou 2-saltos), como é o caso dos algoritmos BSF distribuídos, o conhecimento global existe. Assim, é possível organizar a *Scatternet* otimizando as métricas definidas levando em consideração este conhecimento global.

Algoritmos com descoberta dinâmica não possuem distinção entre os estágios de descoberta e formação da *Scatternet*. Desta forma, nunca há um conhecimento global sobre vizinhança durante a montagem da *Scatternet*. Um dispositivo não é capaz de discernir se existem ou não outros dispositivos próximos a serem incorporados a *Scatternet*, pois não se sabe se há dispositivos que ainda não foram descobertos. Desta forma, algoritmos dinâmicos possuem maiores dificuldades em realizar otimizações globais. Mas, são bem adaptados a situações onde há mobilidade de nós, pois dispositivos podem ingressar na *Scatternet* a qualquer momento, ao contrário de algoritmos com descoberta estática, que devem ter todos os integrantes da *Scatternet* identificados durante a etapa de descoberta.

Esta característica faz dos algoritmos dinâmicos interessantes para aplicações em situações de mobilidade, como a estudada nessa tese. Nestes cenários, dispositivos entram e saem das *Scatternet* durante a viagem de ônibus.

Visto que Beddernet não se mostrou uma alternativa viável para uso no sistema de coleta de informações de transporte coletivo, passamos a considerar outros algoritmos BSF. Conforme visto anteriormente, os algoritmos BSF podem ser classificados entre *single-hop* e *multi-hop* e entre *descoberta dinâmica* ou *descoberta estática*.

Considerando que na aplicação de transporte coletivo todos os dispositivos estão dentro do alcance um do outro (considerando o alcance de 20 metros do *Bluetooth* de *smartphones*, suficiente para cobrir todo o ônibus); e há entrada e saída constante de passageiros do ônibus, e por consequência da *Scatternet*. O algoritmo BSF ideal para esta aplicação deve ser *Single-hop* com descoberta dinâmica.

Dentre os algoritmos analisados, dois possuem ambas as características desejadas: Law (LAW; MEHTA; SIU, 2003) e TSF (TAN et al., 2002). Entretanto, estes algoritmos não levam em consideração que alguns nós da *Scatternet* têm maior propensão a deixá-la, tampouco a qualidade de comunicação fornecida pela camada de enlace *Bluetooth*. Esta situação é intrínseca do ambiente no qual opera. Desta forma, os algoritmos supracitados não aproveitam este conhecimento da aplicação para otimizar a reorganização após desembarques. Dada estas observações, no próximo capítulo propomos um algoritmo de BSF inédito que busca considerar aspectos ligados à aplicação para melhor posicionar os nós dentro da *Scatternet* para diminuir o número de partições causadas na *Scatternet* pelos desembarques. Este algoritmo é

capaz tanto de considerar que alguns nós da *Scatternet* têm maior propensão a deixá-la, quanto aspectos relacionados a camada de enlace, ao permitir que estas informações sejam codificadas em um peso que indica o quão apto é um nó para assumir uma função estruturante da *Scatternet*. Entretanto para simplificar a discussão, no que segue consideraremos que o peso codifica apenas a questão de propensão de desembarque.

4 FORMAÇÃO DE *SCATTERNETS* ENTRE PASSAGEIROS DE ÔNIBUS: ALGORITMO BLUEMOB

No ambiente de operação de Bluemob observa-se que (1) os dispositivos modernos são capazes de estabelecer canais de comunicação a distâncias de até 20 metros, o que é suficiente para cobrir todo o ônibus (PEI et al., 2010); e, (2) há entrada e saída constante de nós da rede, correspondente aos embarques e desembarques do ônibus.

Desta forma, um algoritmo BSF para nossa aplicação deve possuir as seguintes características:

- *Single-hop*: nos cenários considerados, o alcance máximo dos dispositivos *Bluetooth*, é suficiente para cobrir toda a área onde a rede opera. Logo, dispensa-se o uso de algoritmos *multi-hop*, evitando a complexidade inerente a estes.

- Descoberta Dinâmica: o algoritmo deve ser capaz de incorporar novos dispositivos a *scatternet* e suportar a saída de dispositivos mantendo a conectividade da rede. Algoritmos BSF com descoberta dinâmica são mais adequados a estes cenários, já que possuem mecanismos de descoberta e incorporação de dispositivos a *scatternet*. Algoritmos com descoberta estática apenas definem a topologia da rede, sem levar em consideração a incorporação e saída de nós após a rede ter sido formada.

Dentre os algoritmos revisado no Capítulo 3, os algoritmos Law (LAW; MEHTA; SIU, 2003) e TSF (TAN et al., 2002) são os dois algoritmos BSF que se enquadram nas características acima: *single-hop* com descoberta dinâmica. Entretanto, estes algoritmos não levam em consideração que alguns nós da rede têm maior propensão a deixar a *scatternet*. Esta situação é intrínseca do ambiente no qual Bluemob opera. A atribuição de pesos diferentes aos passageiros do ônibus de acordo com a propensão destes em desembarcar traduz um modelo de comportamento a ser considerado na formação, possibilitando alocar nós com menor propensão de sair da rede em posições de mestre e ponte. Desta forma, durante a saída dos nós da rede haverá um menor número de particionamentos em relação a algoritmos sem modelo de comportamento, pois os nós que saem da rede não ocuparão posições estruturantes. Menos particionamentos acarretam tempo menor para restaurar a conectividade da rede.

Possíveis indicadores da propensão de um passageiro para desembarcar são: (i) o tempo que o passageiro está a bordo, considerando que quanto

maior este tempo, mais próximo está o seu ponto de desembarque; (ii) a posição do passageiro no interior do ônibus, pois a proximidade dos passageiros às portas de saída indica uma propensão a desembarcar, principalmente em ônibus lotados; (iii) o conhecimento prévio sobre padrões de embarques e desembarques em uma dada rota de ônibus, obtido a partir de dados históricos de origem-destino que fornecem a probabilidade de desembarque em uma parada dado o local onde o embarque ocorreu; (iv) nível de bateria do *smartphone*. Neste trabalho, o modelo de comportamento é traduzido em um número que indica o ponto de ônibus de saída do nó. Embora o processo de obtenção da estimativa não seja discutido, variam-se os níveis de precisão da estimação para avaliar o quão crítico é este processo.

Bluemob é um algoritmo BSF *single-hop* com descoberta dinâmica, a exemplo de TSF e Law. Assim como Law, Bluemob forma *Scatternets* com topologia *mesh*. Ressalta-se que no contexto de algoritmos BSF, *single-hop* se refere apenas a característica na qual quaisquer dois dispositivos podem estabelecer uma conexão *Bluetooth* entre eles. Entretanto, isto não implica que a topologia da rede resultante do algoritmo BSF não necessite de roteamento (seja uma rede de um salto). Topologia *mesh* propicia maior liberdade durante a reorganização da *scatternet* que ocorre durante a entrada e saída de nós da rede que alternativas, como topologias em árvore adotada em TSF. Ao contrário dos outros algoritmos, Bluemob considera a propensão de cada nó deixar a rede durante a construção da *scatternet*. Para quantificar a propensão em deixar a rede, cada nó possui um peso associado.

A melhoria no desempenho do algoritmo pode ser medida pelo tempo que necessita para reestabelecer a rede após os nós a deixarem. No cenário em estudo, a rede deve ser sempre reconstruída antes da chegada do ônibus na próxima parada. O tempo de viagem entre paradas, então, impõe um *soft deadline* que precisa ser respeitado para que as aplicações possam identificar os passageiros embarcados que desejam participar da rede.

4.1 PRINCÍPIOS DO BLUEMOB

Bluemob introduz a noção de peso aos nós da rede durante o processo de formação para representar a propensão de um nó permanecer embarcado. Faça h representar esta propensão; ela pode ser arbitrariamente definida, desde que atenda ao requisito de Bluemob que necessita de uma relação de ordem total no domínio de h.

Assim como outros algoritmos BSF (LAW; MEHTA; SIU, 2003; TAN et al., 2002), um *componente* é definido como sendo um nó desconectado, uma *Piconet*, ou uma *Scatternet*. Cada componente possui um *líder* esco-

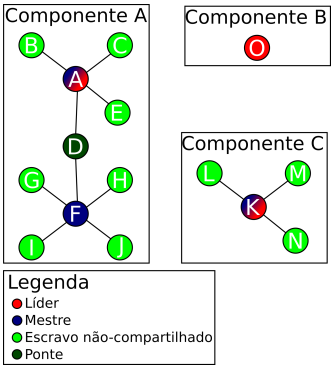


Figura 7: Estrutura interna de componentes: uma *Scatternet* (componente A), um nó solitário (componente C) e uma *Piconet* (componente B).

lhido entre seus mestres. Os líderes de diferentes componentes coordenam o processo de união. Além disso, um componente válido deve satisfazer as seguintes propriedades:

(1) **Liderança única** - o componente possui um único líder;

(2) **Limite superior de escravos do líder** - o líder deve possuir no máximo seis escravos;

(3) **Limite superior de escravos** - os mestres devem possuir menos que sete escravos;

(4) **Disponibilidade para busca** - o líder deve possuir um escravo não-compartilhado, ou não possuir nenhum escravo;

(5) **Propriedade organizacional** - todo escravo não-compartilhado possui peso h menor ou igual ao peso h de seu mestre.

A figura 7 ilustra a estrutura interna dos componentes formados pelo Bluemob. Cada um dos componentes possui apenas um líder: na *Scatternet* do Componente A o líder é o nó A; no Componente B o líder é o nó O; e, na *Piconet* do Componente C é o nó K. Em componentes constituídos por mais de um nó, ou seja, *Scatternets* e *Piconets*, um dos mestres das *Piconets* atua também como líder. Os escravos não-compartilhados (nós verde-claro) são aqueles escravos que pertencem apenas a uma *Piconet*, ou seja, não atuam como pontes.

O princípio básico do algoritmo BSF é o processo de união de dois componentes mantendo as propriedades acima. Cada propriedade tem um

objetivo. A propriedade 1 limita a ocorrência de uma união em cada rodada. Caso mais de um líder em um componente fosse permitido, haveria mais de um nó realizando INQUIRY e/ou INQUIRY_SCAN em um mesmo componente. A propriedade 2 garante que o líder seja capaz de conectar-se a um outro componente sem que o limite de sete escravos por mestre seja violado. A propriedade 3 é imposta pela especificação do *Bluetooth* e impede que o limite de sete escravos ativos seja violado pelos mestres dos componentes. A propriedade 4 impõe a existência de um escravo não-compartilhado capaz de entrar no estado INQUIRY_SCAN, tornando o componente passível de ser descoberto e unido a outro componente cujo líder esteja realizando processo de INQUIRY. Por fim, a propriedade 5 restringe a topologia da rede, garantindo que escravos não-compartilhados possuem peso h menor que seus mestres. Em outras palavras, os escravos não-compartilhados possuem maior propensão a deixar a rede que seus mestres.

Para cada reorganização de componentes, o Bluemob escolhe um novo líder e modifica as conexões existente para que as propriedades sejam mantidas. O algoritmo segue em rodadas até que todos os nós pertençam a um único componente. Durante cada rodada, um componente pode descobrir novos componente ou se tornar visível. Líderes escolhem aleatoriamente entre realizar o processo de INQUIRY do *Bluetooth* ou requisitar a um de seus escravos não compartilhados que entrem no estado de INQUIRY SCAN. Componentes formados por apenas um nó não possuem um escravo não compartilhado, portanto o líder realiza o procedimento INQUIRY SCAN. Estas operações devem ser finalizadas dentro de um intervalo de tempo δ.

Quando um líder realiza um processo de INQUIRY e descobre um nó no estado de INQUIRY SCAN, uma conexão é formada. O líder que realiza o processo de INQUIRY é denominado u; o escravo não compartilhado de u com maior h é denominado y; o nó no estado INQURIY SCAN é v; e, o líder do componente de v é w. No início do processo de reorganização, v se torna escravo de u e dados sobre os dois componentes são trocados. Esta conexão pode ser temporária, dependendo da estrutura do componente.

No Bluemob o líder sempre escolhe o escravo não compartilhado com maior h para se tornar visível (realizar o processo de INQUIRY SCAN). Esta é uma diferença entre Bluemob e outros algoritmos (LAW; MEHTA; SIU, 2003; TAN et al., 2002) para facilitar o procedimento de reorganização. Isto é usado por Bluemob para evitar desconexões e reconexões durante a reestruturação do novo componente, pois no inicio do processo de reorganização o algoritmo sabe que o líder se conectou ao escravo com maior h do outro componente.

4.2 DETALHAMENTO DO ALGORITMO

Para descrição do algoritmo é usada a seguinte notação:

- $h(x)$: é o peso associado ao nó x codificando a propensão deste em permanecer na rede. Quanto maior o $h(x)$ mais tempo ele permanecerá na rede;
- $S(x)$: o conjunto de escravos de x;
- $M(x)$: o conjunto de mestres de x;
- $S_u(x)$: o conjunto de escravos não compartilhados de x;
- $Y_p(u)$: o conjunto de escravos não compartilhados de u que têm h maior que ou igual a todos os outros escravos livres de u, i. é: $Y_p(u) \leftarrow \{y \in S_u(u) | \forall k \in S_u(u) : h(y) \geq h(k)\}$

Bluemob opera em rodadas, sendo que cada rodada possui uma duração δ durante a qual o líder tenta realizar a união com outro componente. No início de uma rodada (linha 3 do algoritmo 2) o líder realiza um processo INQUIRY (linha 6) ou requisita a um de seus escravos não-compartilhados que realize INQUIRY SCAN (linha 12). A escolha entre qual dos dois processos (INQUIRY ou INQUIRY SCAN) será realizado é aleatória com probabilidade P de realizar INQUIRY (linha 5). Caso o componente consista de um único nó, o líder realiza o processo INQUIRY SCAN ele mesmo (linha 9). É importante notar que este algoritmo pressupõe que todas as operações da rodada, incluindo o processo de união, sejam concluídas antes da rodada terminar.

Quando a conexão *Bluetooth* é criada, o procedimento WHEN CONNECT é executado por quem estabeleceu a conexão (linha 16), ou seja, pelo líder em INQUIRY. Este procedimento coleta dados sobre os dois componentes que estão sendo unidos. Em seguida, o processo de reorganização da topologia é iniciado (linha 28).

As informação coletadas no estabelecimento da conexão são: o mestre em INQUIRY (u); o nó em INQUIRY SCAN (v); o líder do componente de v (w); o escravo não-compartilhado de u com maior h (y). É importante salientar que, se v é um escravo de u ($v \in S(u)$) e o componente de v possui mais do que um nó ($v \neq w$), então v também será escravo do líder w ($v \in S(w)$). Em outras palavras, quando o líder u encontra um outro componente, uma conexão é estabelecida entre v e u sendo que u assume o papel de mestre de v. Também, por construção, se v não está sozinho em seu componente, então w será mestre de v. Isto é importante, pois v pertence tanto a $S(w)$ quanto

Algoritmo 2 Bluemob executando no nó i

```
1: P ← (0,1); δ ← duração da rodada
2: isLeader_i ← true; h_i ← h(i)
3: procedure START ROUND
4:     p ← random number between 0 and 1
5:     if p < P then
6:         INQUIRY
7:     else
8:         if S(i) = ∅ then
9:             INQUIRYSCAN(i)
10:        else
11:            y ← any element ∈ Y_p(i)
12:            INQUIRYSCAN(y)
13:        end if
14:    end if
15: end procedure
16: procedure WHEN CONNECT(to)
17:    u ← i; v ← to
18:    if M(v) = {u} then
19:        w ← v
20:    else
21:        w ← M(v) − {u}
22:    end if
23:    if Y_p(i) = ∅ then
24:        y ←⊥
25:    else
26:        y ← qualquer elemento ∈ Y_p(i)
27:    end if
28:    REORGANIZE(u, v, w, y)
29: end procedure
30: procedure BECOME LEADER(u)
31:    leader_u ← true
32:    Agendar a execução do método Start Round a cada δ s
33: end procedure
34: procedure RETIRE(u)
35:    leader_u ← false
36:    Parar a execução do método Start Round a cada δ s
37: end procedure
```

a $S(u)$. Este fato deve ser levado em consideração durante o processo de reorganização.

4.3 REORGANIZAÇÕES

A reorganização ocorre sempre que um componente que está em INQUIRY se conecta a outro componente no estado INQUIRY SCAN (linha 28 no Algoritmo 2), sendo descrita no algoritmo 3. O objetivo do algoritmo é evitar alocar os papéis de mestre e pontes aos nós que possuem maior propensão a sair da *Scatternet*. Sendo Bluemob um algoritmo single-hop, pode-se estabelecer uma conexão *Bluetooth* entre quaisquer dois nós n_i e n_j. Isto garante que, no algoritmo de reorganização, quaisquer nós desconectados du-

rante a reorganização podem ser conectados.

O peso h influencia a estrutura do componente de duas formas: através da propriedade organizacional imposta aos componentes, que faz com que escravos não-compartilhados tenham peso h menor ou igual ao peso h de seu mestre; e, através da forma como escravos não compartilhados são escolhidos dentro de uma *Piconet* para realizar uma dada função. A escolha é feita sempre a partir do conjunto Y_p, que contém aqueles escravos não compartilhados com maior h. Este conjunto caracteriza os escravos mais adequados a deixar de ser escravos e tornar-se mestre ou ponte, dentro de uma *Piconet*.

Quando o líder em INQUIRY descobre outro componente, uma reorganização é feita de forma que o componente resultante respeite as propriedades estabelecidas. Bluemob escolhe uma dentre treze reorganizações, dependendo da estrutura dos componentes.

1. Se $v = w$, o componente descoberto é composto por um nó, pois um líder w só entra em INQUIRY SCAN neste caso. Para este cenário, temos cinco casos:

   (a) Caso 1: $|S(u)| < 7$ e $h(u) > h(v)$

   (b) Caso 2: $|S(u)| < 7$ e $h(u) \leq h(v)$

   (c) Caso 3: $|S(u)| = 7$ e $h(u) \geq h(v)$ e $h(y) \leq h(v)$

   (d) Caso 4: $|S(u)| = 7$ e $h(u) \geq h(v)$ e $h(y) > h(v)$

   (e) Caso 5: $|S(u)| = 7$ e $h(u) < h(v)$

2. Se $v \neq w$, o componente descoberto é uma *Piconet* ou *Scatternet*. Neste cenário, temos os seguintes casos:

   (a) Caso 6: $|S(u)| + |S(w)| < 7$ e $h(u) \geq h(w)$

   (b) Caso 7: $|S(u)| + |S(w)| < 7$ e $h(u) < h(w)$

   (c) Caso 8: $|S(u)| + |S(w)| = 7$ e $|S(u)| = 1$ e $h(u) < h(v)$

   (d) Caso 9: $|S(u)| + |S(w)| = 7$ e $|S(u)| = 1$ e $h(u) \geq h(v)$

   (e) Caso 10: $|S(u)| + |S(w)| = 7$ e $|S(u)| \neq 1$ e $h(v) \geq h(w)$

   (f) Caso 11: $|S(u)| + |S(w)| = 7$ e $|S(u)| \neq 1$ e $h(v) < h(w)$

   (g) Caso 12: $|S(u)| + |S(w)| > 7$ e $|S(u)| \neq 1$ e $h(u) < h(w)$

   (h) Caso 13: $|S(u)| + |S(w)| > 7$ e $|S(u)| \neq 1$ e $h(u) \geq h(w)$

Para cada caso, Bluemob define os papéis dos nós das *Piconets* dos líderes (u e w), refazendo conexões e determinando quem será o líder do novo componente.

Algoritmo 3 Procedimento para reorganização dos componentes (continua)

```
1: procedure CASE 1(u, w, v, y)
2:     RETIRE(w)
3: end procedure
4: procedure CASE 2(u, w, v, y)
5:     RETIRE(u)
6:     m ← S(u)
7:     DISCONNECT(m∪{w},u)
8:     CONNECT(m∪{u}−{w},w)
9: end procedure
10: procedure CASE 3(u, w, v, y)
11:     RETIRE(u)
12:     DISCONNECT({y},u)
13:     CONNECT({y},w)
14: end procedure
15: procedure CASE 4(u, w, v, y)
16:     RETIRE(u)
17:     RETIRE(v)
18:     BECOME LEADER(y)
19:     DISCONNECT({v},u)
20:     CONNECT({v},y)
21: end procedure
22: procedure CASE 5(u, w, v, y)
23:     RETIRE(u)
24:     DISCONNECT({y},u)
25:     CONNECT({v},y)
26: end procedure
27: procedure CASE 6(u, w, v, y)
28:     RETIRE(w)
29:     m ← slavesOf(w)
30:     DISCONNECT(m,w)
31:     CONNECT(m∪w−v,u)
32: end procedure
33: procedure CASE 7(u, w, v, y)
34:     RETIRE(u)
35:     m ← slavesOf(u)
36:     DISCONNECT(m,u)
37:     CONNECT(m∪u−w,y)
38: end procedure
39: procedure CASE 8(u, w, v, y)
40:     RETIRE(u)
41:     RETIRE(w)
42:     BECOME LEADER(v)
43:     DISCONNECT({v},u)
44:     CONNECT({u},v)
45: end procedure
46: procedure CASE 9(u, w, v, y)
47:     RETIRE(w)
48:     DISCONNECT({v},w)
49:     CONNECT({w},u)
50: end procedure
```

As reorganizações foram projetadas para que o componente resultante da união dos componentes atendam às propriedades de corretude impostas pelo algoritmo. Para atender este objetivo, cada uma das reorganizações pode: (1) destruir conexão *Bluetooth* entre dois dispositivos dentro do componente;

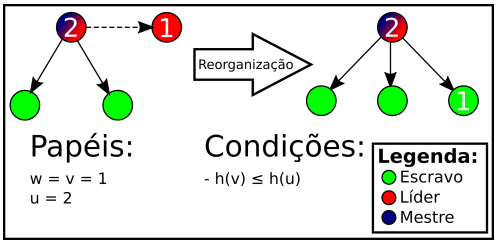


Figura 8: Estrutura dos componentes antes e depois da reorganização caso 1

(2) tornar um dispositivo mestre de outro dispositivo pertencente ao mesmo componentes; (3) aposentar um dispositivo que era líder; (4) tornar um dispositivo líder. O algoritmo 3 descreve todas as reorganizações possíveis, e as transformações topológicas necessárias para que o componente resultante da união atenda as especificações. As transformações topológicas são definidas usando-se duas funções: $Disconnect(A,b)$ e $Connect(B,b)$. A função *Disconnect* possui dois parâmetros: um conjunto de dispositivos A e um nó b. Ao ser invocada, todos os dispositivos em A são desconectados do nó b. A função *Connect*, por sua vez, também recebe dois parâmetros: um conjunto de dispositivos A e um nó b. Ao ser invocada, todos os dispositivos em A tornam-se escravos do nó b.

Além destes dois procedimentos, são usadas as funções *Retire* e *Become Leader*. Estas funções são usadas para fazer com que um líder deixe de sê-lo e para tornar um nó o líder, respectivamente. Ambas recebem um parâmetro identificando o nó que se tornará o líder (ou deixará de sê-lo).

4.3.1 Caso 1: $w = v \wedge |S(u)| < K \wedge h(v) \leq h(u)$

A Figura 8 exemplifica uma reorganização para o caso 1. Nesta situação, o nó u, que realiza o processo de INQUIRY, descobre o nó v e estabelece a conexão entre u e v representada pela linha tracejada. Nas reorganizações do caso 1, v e w são o mesmo nó. Isto significa que o componente que estava realizando o procedimento de INQUIRY SCAN está sozinho em seu componente.

Dado que o número de escravos de u é menor do que 7, é possível agrupar os nós em uma única *Piconet* sem violar o limite superior de escravos do líder. A questão pendente é a determinação de qual será o mestre da *Piconet*. Mas como $h(v) \leq h(u)$, não há empecilhos em deixar que v seja escravo de u. Desta forma, u será o novo líder do componente, enquanto v é

Algoritmo 3 Procedimento para reorganização dos componentes (conclusão)

```
51: procedure CASE 10(u, w, v, y)
52:     RETIRE(u)
53:     RETIRE(w)
54:     BECOME LEADER(v)
55:     m ← slavesOf(u)
56:     DISCONNECT(m,u)
57:     CONNECT(m − {y,v},w)
58:     CONNECT({u,y},v)
59: end procedure
60: procedure CASE 11(u, w, v, y)
61:     RETIRE(u)
62:     DISCONNECT({v},u)
63:     CONNECT({w},u)
64:     m ← slavesOf(w)
65:     DISCONNECT(m − {v},w)
66:     CONNECT(m − {v},u)
67: end procedure
68: procedure CASE 12(u, w, v, y)
69:     RETIRE(w)
70:     availableSlots ← K − |slavesOf(w)|
71:     availableSlaves ← |slavesOf(u)| − 2
72:     howManyToMove ← min(availableSlots, availableSlaves
73:     slavesToMove ← a subset of slavesOf(u) − y,v with size howManyToMove
74:     DISCONNECT(slavesToMove,u)
75:     CONNECT(slavesToMove,w)
76: end procedure
77: procedure CASE 13(u, w, v, y)
78:     RETIRE(u)
79:     availableSlots ← K − |slavesOf(u)|
80:     availableSlaves ← |slavesOf(w)| − 2
81:     howManyToMove ← min(availableSlots, availableSlaves
82:     slavesToMove ← a subset of slavesOf(w) − v with size howManyToMove
83:     DISCONNECT(slavesToMove,w)
84:     CONNECT(slavesToMove,u)
85:     DISCONNECT({v},u)
86:     CONNECT({w},u)
87: end procedure
```

aposentado e permanece escravo de u.

Nesta reorganização não é necessária o estabelecimento ou destruição de nenhuma conexão. O nó v já é escravo de u, pois u estava em INQUIRY. Portanto, a única ação necessária é aposentar o nó v de seu status de líder.

4.3.2 Caso 2: $w = v \wedge |S(u)| < K \wedge h(v) > h(u)$

A Figura 9 exemplifica uma reorganização para o caso 2. Nesta situação, o nó u, que realiza o processo de INQUIRY, descobre o nó v e estabelece a conexão entre u e v representada pela linha tracejada. Nas reorganização do caso 2, v e w são os mesmos nós. Isto significa que o componente que estava realizando o procedimento de INQUIRY SCAN está sozinho em seu

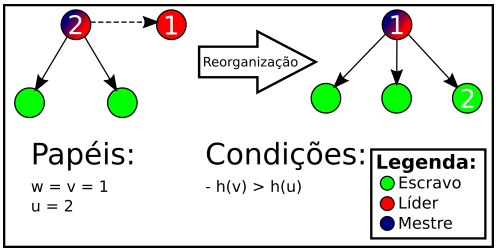


Figura 9: Estrutura dos componentes antes e depois da reorganização caso 2

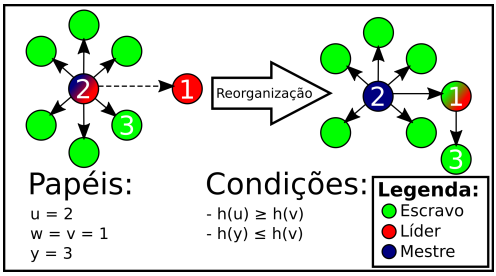


Figura 10: Estrutura dos componentes antes e depois da reorganização caso 3

componente.

Dado que o número de escravos de u é menor do que 7, é possível agrupar os nós em uma única *Piconet* sem violar o limite superior de escravos do líder.

Dado que, ao contrário do caso 1, $h(v) > h(u)$, v não pode se tornar escravo de u pois violaria a propriedade 5. Logo, na reorganização caso 2, deve-se reestruturar a *Piconet*. O que é feito é a inversão dos papéis entre v e u: v se torna mestre de u. Além disso, todos os escravos de u são transferidos para v. Isto é possível pois caso o escravo de u seja um escravo não compartilhado, então seu h será menor ou igual ao $h(u)$ que por sua vez é menor do que $h(v)$.

4.3.3 Caso 3: $w = v \wedge |S(u)| = K \wedge h(u) \geq h(v) \wedge h(y) \leq h(v)$

A Figura 10 exemplifica uma reorganização para o caso 3. Nesta situação, o nó u, que realiza o processo de INQUIRY, descobre o nó v e estabelece a conexão entre u e v representada pela linha tracejada. Nas reorganização do caso 3, v e w são os mesmos nós. Isto significa que o componente que

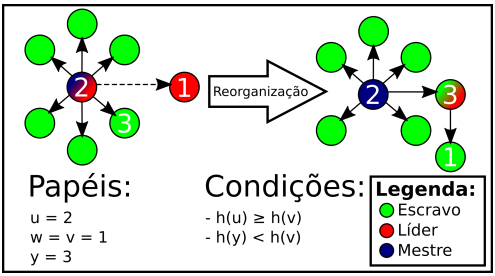


Figura 11: Estrutura dos componentes antes e depois da reorganização caso 4

estava realizando o procedimento de INQUIRY SCAN está sozinho em seu componente.

A diferença entre os casos anteriores (1 e 2) e o caso 3 é que o número de escravos de u é exatamente 7. Nesta situação, u não pode simplesmente se tornar o líder, pois violaria a propriedade 2.

Como não é possível unificar as *Piconets* dos líderes em uma única *Piconet*, cria-se uma nova *Piconet* para o líder. Dado que $h(u) \geq h(v)$ e $y(h) \leq h(v)$, é possível simplesmente transferir um escravo não compartilhado y de u para w.

4.3.4 Caso 4: $w = v \wedge |S(u)| = K \wedge h(u) \geq h(v) \wedge h(y) > h(v)$

A Figura 11 exemplifica uma reorganização para o caso 4. Nesta situação, o nó u, que realiza o processo de INQUIRY, descobre o nó v e estabelece a conexão entre u e v representada pela linha tracejada. Nas reorganização do caso 4, v e w são os mesmos nós. Isto significa que o componente que estava realizando o procedimento de INQUIRY SCAN está sozinho em seu componente.

Ao contrário do caso 3, no caso 4 tem-se que $y(h) > h(v)$. Nesta situação, y não pode se tornar escravo não compartilhado de v, pois isto violaria a propriedade 5.

Na reorganização caso 4 ao invés de transformar y no escravo não compartilhado de v, faz-se de y o novo líder e v seu escravo não compartilhado.

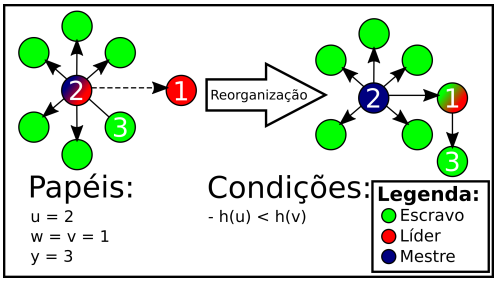


Figura 12: Estrutura dos componentes antes e depois da reorganização caso 5

4.3.5 Caso 5: $w = v \wedge |S(u)| = K \wedge h(u) < h(v)$

A Figura 12 exemplifica uma reorganização para o caso 5. Nesta situação, o nó u, que realiza o processo de INQUIRY, descobre o nó v e estabelece a conexão entre u e v representada pela linha tracejada. Nas reorganização do caso 5, v e w são os mesmos nós. Isto significa que o componente que estava realizando o procedimento de INQUIRY SCAN está sozinho em seu componente.

Esta reorganização é igual a reorganização do caso 3. Não é possível unificar as *Piconets* dos líderes em uma única *Piconet*, cria-se uma nova *Piconet* para o líder. Dado que $h(u) \geq h(v)$ e $y(h) \leq h(v)$, é possível simplesmente transferir um escravo não compartilhado y de u para w.

Mas ao contrário do caso 3 onde $h(w) \geq h(v)$, neste caso $h(u) < h(v)$. Logo, apesar das modificações feitas na topologia da rede serem as mesmas, as motivação e justificativas para cada uma das propriedades são diferentes.

4.3.6 Caso 6: $w \neq v \wedge |S(w)| + |S(u)| < K \wedge h(u) \geq h(w)$

A Figura 13 exemplifica uma reorganização para o caso 6. Nesta situação, o nó u, que realiza o processo de INQUIRY, descobre o nó v e estabelece a conexão entre u e v representada pela linha tracejada. Nas reorganização do caso 5, v e w são diferentes, logo a *Piconet* descoberta é composta de só um dispositivo.

Nesta reorganização é possível juntar todos os nós da *Piconet* dos líderes u e w em uma só *Piconet* sem violar a propriedade 2. Dado que $h(u) \geq h(w)$, u será o novo líder e w será um escravo de u. Os escravos de w são desconectados de w e reconectados a u. Logo a *Piconet* de w é desfeita.

Note que é possível que u não tenha nenhum escravo, sendo um com-

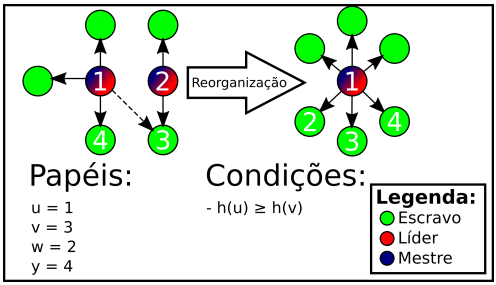


Figura 13: Estrutura dos componentes antes e depois da reorganização caso 6

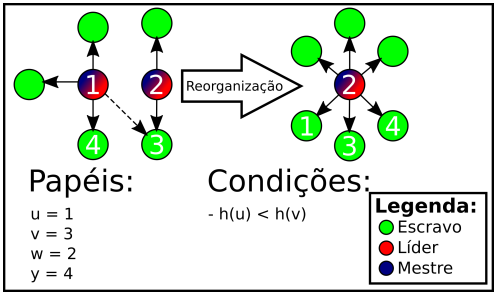


Figura 14: Estrutura dos componentes antes e depois da reorganização caso 7

ponente vazio.

4.3.7 Caso 7: $w \neq v \wedge |S(w)| + |S(u)| < K \wedge h(u) < h(w)$

A Figura 18 exemplifica uma reorganização para o caso 7. Nesta situação, u realiza o processo de INQUIRY, descobre o nó v e estabelece a conexão entre u e v representada pela linha tracejada. O mestre de v é w, líder do componente. O peso de u é menor que o de w ($h(u) < h(w)$). Dado o número de escravos na *Piconet* de cada um dos líderes, $|S(u)| + |S(w)| < 7$, os nós de ambas as *Piconets* podem ser unificadas em uma única *Piconet* sem violar o limite superior de escravos do líder (neste exemplo, seis).

A questão pendente em relação à reorganização é a seguinte: deve u se tornar escravo de w e w se tornar o novo líder, ou deve w se tornar escravo de u e u assumir o papel de novo líder?

Para que o novo componente seja válido, as propriedades devem ser respeitadas após a reorganização. Como $h(u) < h(w)$, u não pode ser mestre

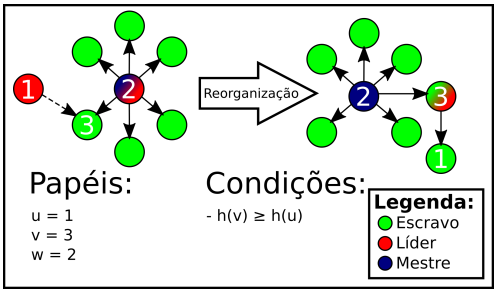


Figura 15: Estrutura dos componentes antes e depois da reorganização caso 8

de w, então u se torna escravo de w. Os escravos de u tornam-se escravos de w pois a terceira propriedade diz que todo escravo não compartilhado deve possuir h menor ou igual ao h de seu mestre. Logo, todos os escravos não-compartilhados de u possuem um h menor que u, e, por transitividade, menor que w. Consequentemente, o algoritmo pode concluir que w é o novo líder, e todos os escravos de u se tornam escravos de w sem que nenhuma propriedade seja violada, conforme ilustrado na Figura 14.

4.3.8 Caso 8: $w \neq v \wedge |S(w)| + |S(u)| = K \wedge |S(u)| = 1 \wedge h(v) \geq h(u)$

A Figura 15 exemplifica uma reorganização para o caso 8. Nesta situação, u realiza o processo de INQUIRY, descobre o nó v e estabelece a conexão entre u e v representada pela linha tracejada. O mestre de v é w, líder do componente. O peso de v é maior ou igual ao peso de u ($h(v) \geq h(u)$).

Nesta reorganização, não é possível unificar as *Piconets* de u e w em uma, pois a *Piconet* do líder possuiria 7 nós. Com isso, a propriedade 2 seria violada.

Uma particularidade, desta reorganização é que o componente de u é composto apenas por ele. Logo y, o escravo não compartilhado de u, não existe durante a reorganização.

Nesta situação, como $h(v) \geq h(u)$, é possível fazer v o novo líder e tornar u o escravo não compartilhado de v.

4.3.9 Caso 9: $w \neq v \wedge |S(w)| + |S(u)| = K \wedge |S(u)| = 1 \wedge h(v) < h(u)$

A Figura 16 exemplifica uma reorganização para o caso 9. Nesta situação, u realiza o processo de INQUIRY, descobre o nó v e estabelece a

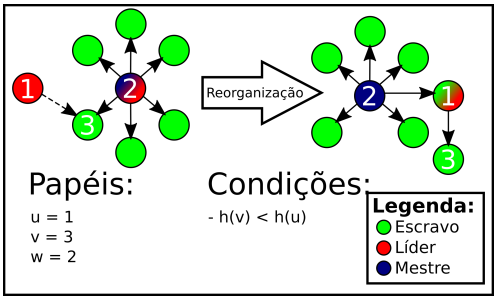


Figura 16: Estrutura dos componentes antes e depois da reorganização caso 9

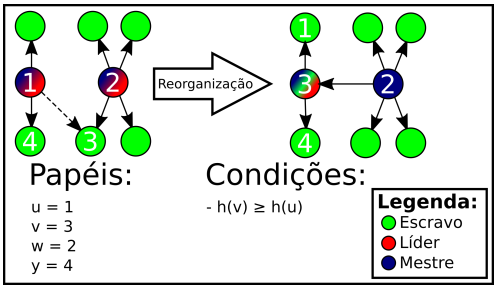


Figura 17: Estrutura dos componentes antes e depois da reorganização caso 10

conexão entre u e v representada pela linha tracejada. O mestre de v é w, líder do componente.

Nesta reorganização, assim como na reorganização 8, o componente de u é composto apenas por ele. Logo y, o escravo não compartilhado de u, não existe durante a reorganização.

A diferença entre a reorganização 8 e a 9 está na relação entre os pesos de v e u. Na reorganização 9, $h(u) > h(v)$. Isto impossibilita que u se torne um escravo não compartilhado de v.

Portanto, nesta reorganização, u permanece como líder e v assume papel de escravo não compartilhado de u.

4.3.10 Caso 10: $w \neq v \wedge |S(w)| + |S(u)| = K \wedge |S(u)| > 1 \wedge h(v) \geq h(u)$

A Figura 17 exemplifica uma reorganização para o caso 10. Nesta situação, u realiza o processo de INQUIRY, descobre o nó v e estabelece a

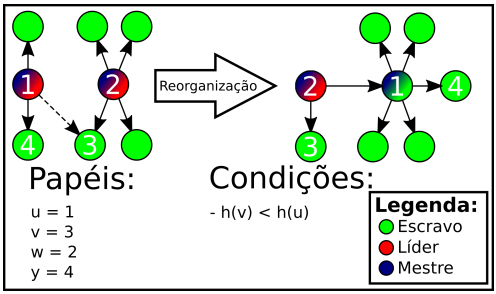


Figura 18: Estrutura dos componentes antes e depois da reorganização caso 11

conexão entre u e v representada pela linha tracejada. O mestre de v é w, líder do componente. O peso de v é maior ou igual ao peso de u ($h(v) \geq h(u)$).

Nesta reorganização, não é possível unificar as *Piconets* de u e w em uma, pois a *Piconet* do líder possuiria 7 nós. Com isso, a propriedade 2 seria violada. Mas ao contrário das reorganização 8 e 9, na reorganização 10 u possui escravos. Logo, existe um nó y, ou seja, u possui um escravo não compartilhado.

Nesta situação, como $h(v) \geq h(u)$, é possível fazer v o novo líder e tornar u o escravo não compartilhado de v. O nó y é deixado como escravo não compartilhado de v e todos os nós de u com exceção de y são movidos para w.

4.3.11 Caso 11: $w \neq v \wedge |S(w)| + |S(u)| = K \wedge |S(u)| > 1 \wedge h(v) < h(u)$

A Figura 18 exemplifica uma reorganização para o caso 11. Nesta situação, u realiza o processo de INQUIRY, descobre o nó v e estabelece a conexão entre u e v representada pela linha tracejada. O mestre de v é w, líder do componente. O peso de v é maior ou igual ao peso de u ($h(v) \geq h(u)$).

Nesta reorganização, não é possível unificar as *Piconets* de u e w em uma, pois a *Piconet* do líder possuiria 7 nós. Com isso, a propriedade 2 seria violada. Mas ao contrário das reorganização 8 e 9, na reorganização 11 u possui escravos. Logo, existe um nó y, ou seja, u possui um escravo não compartilhado.

Nesta situação, como $h(v) < h(u)$. Isso permite mover todos os escravos de w para u, deixando apenas v para que o novo líder w tenha um escravo não compartilhado.

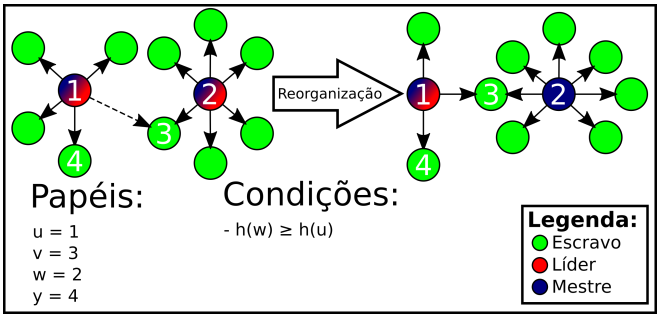


Figura 19: Estrutura dos componentes antes e depois da reorganização caso 12

4.3.12 Caso 12: $w \neq v \wedge |S(w)| + |S(u)| > K \wedge h(w) \geq h(u)$

A Figura 19 exemplifica uma reorganização para o caso 12. Nesta situação, u realiza o processo de INQUIRY, descobre o nó v e estabelece a conexão entre u e v representada pela linha tracejada. O mestre de v é w, líder do componente.

Nesta reorganização, não é possível unificar as *Piconets* de u e w em uma, pois há mais de 8 nós juntando-se as duas *Piconets*.

Na reorganização 12 $h(w) \geq h(u)$. Isso permite realocar os escravos de u para w, esvaziando a *Piconet* do novo líder u. v é mantido como ponte entre u e w.

4.3.13 Caso 13: $w \neq v \wedge |S(w)| + |S(u)| > K \wedge h(w) < h(u)$

A Figura 20 exemplifica uma reorganização para o caso 13. Nesta situação, u realiza o processo de INQUIRY, descobre o nó v e estabelece a conexão entre u e v representada pela linha tracejada. O mestre de v é w, líder do componente.

Não é possível unificar as *Piconets* de u e w em uma, pois as duas *Piconets* unidas possuem mais de 8 nós.

Ao contrário da reorganização 12: nesse caso, $h(w) < h(u)$. Portanto, não é possível realocar os escravos de u para w, mas o contrário é. Neste caso, v não é mantido como ponte entre u e w. O nó v é deixado como escravo não compartilhado de w. w torna-se escravo de u.

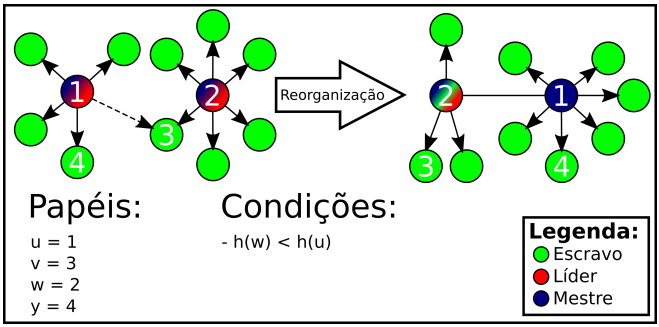


Figura 20: Estrutura dos componentes antes e depois da reorganização caso 13

4.3.14 Propriedades do algoritmo

As organizações são projetadas de tal forma que as propriedades sejam mantidas após a união. Isto permite que as uniões de componentes ocorram indefinidamente.

Todas as reorganização atendem o requisito que o componente resultante da união de dois componentes A e B atende a todas as 5 propriedades caso A e B atendam a estas propriedades. A prova para o primeiro caso das reorganizações (onde $w = v \wedge |S(u)| < K \wedge h(v) \leq h(u)$) é mostrado abaixo, as demais reorganizações são apresentadas no anexo A deste documento.

Lema 1. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 1, após a união (MERGE) o componente resultante da união possui um líder.*

Demonstração. A prova é por contradição. Assume-se que a premissa é falsa: após a união existe mais de um líder ou não existe nenhum líder. Na linha 2 do Algoritmo 3, w, líder do componente B (linha 18 do Algoritmo 2), é aposentado. Antes da união, o componente B respeita todas as propriedades, e segundo a propriedade 1 pode-se concluir que após a finalização da reorganização, nenhum nó que pertencia a B é líder. Dado que: (1) não há nenhum líder pertencente a B; (2) a reorganização 1 não torna nenhum nó líder; e, (3) nenhum nó pertencente a A é aposentado; todos líderes de A são líderes do componente após união. Como nenhum nó pertencente ao componente B é líder, todos os líderes do componente após a união são líderes do componente A. Logo, antes da reorganização o componente A não possui nenhum líder ou possui pelo menos 2 líderes. Isto é uma contradição, pois sabe-se devido

a propriedade 1 que o componente A possui exatamente 1 líder. Conclui-se que a premissa é verdadeira e existe um líder após o término da reorganização. □

Lema 2. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 1, após a união (MERGE) o líder possuirá no máximo* $K-1=6$ *escravos.*

Demonstração. A prova é por contradição. Assume-se que a premissa é falsa: após a união o líder possui mais de $K-1=6$ escravos. A ocorrência da reorganização CASO 1 exige que as condições da linha 2 do Algoritmo 3 sejam satisfeitas, portanto $|S(u)| < K$ (equivalente a $|S(u)| \leq K-1$). Como v foi aposentado na linha 2 do Algoritmo 3, e u, o líder de A (linha 17 do Algoritmo 2) não foi aposentado, o líder do componente após a união é u. Como nenhum novo escravo é atribuído a u durante a reorganização, após a conclusão da união o número de escravos de u permanece o mesmo. Logo, $|S(u)| \leq K-1$. Isto é uma contradição, portanto conclui-se que $|S(u)| \leq K-1$ e $|S(u)| > K-1$ □

Lema 3. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 1, não existe mestre com mais de* $K=7$ *escravos.*

Demonstração. A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre com mais de $K=7$ escravos. A ocorrência da reorganização CASO 1 exige que as condições da linha 2 do Algoritmo 4 sejam satisfeitas, portanto $|S(u)| < K$ e $v=w$. $v=w$ implica que w, líder do componente B, estava no estado de INQUIRY SCAN. Isto ocorre apenas se o componente B for composto por um único nó (linha 9 do Algoritmo 2). Logo, não há nenhum mestre no componente B. Como $|S(u)| < K$ e nenhum escravo foi adicionado a nenhum nó durante a reorganização, deve existir um mestre diferente de u com mais de K escravos no componente A antes da união. Entretanto, o componente A respeita a propriedade 3. Portanto todo mestre em A antes da união possui $|S(u)| < K$. Isto é uma contradição. Portanto, a premissa é verdadeira. □

Lema 4. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 1, após a união (MERGE) o líder possui um escravo não compartilhado ou não possui nenhum escravo.*

Demonstração. A ocorrência da reorganização CASO 1 exige que as condições da linha 2 do Algoritmo 4 sejam satisfeitas, portanto $v=w$. $v=w$ implica que w, líder do componente B, estava no estado de INQUIRY SCAN. Portanto, não há nenhum mestre em B. Isto ocorre apenas se o componente

B for composto por um único nó (linha 9 do Algoritmo 2). Como $w \in S(u)$, $S(w) = \emptyset, M(w) = \{u\}$, por definição w é um escravo não compartilhado de u. Portanto, o líder u possui pelo menos um escravo não compartilhado: w. □

Lema 5. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 1, após a união (MERGE) todos os escravos não-compartilhados possuem h menor ou igual ao h de seu mestre.*

Demonstração. A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um escravo l cujo h é maior que o h de seu mestre.

A ocorrência da reorganização CASO 1 exige que as condições da linha 2 do Algoritmo 4 sejam satisfeitas, portanto $h(v) \leq h(u)$ e $v = w$. $v = w$ implica que w, líder do componente B, estava no estado de INQUIRY SCAN. Isto ocorre apenas se o componente B for composto por um único nó (linha 9 do Algoritmo 2). Portanto, não há nenhum escravo em B antes da união. Mas, ao iniciar o processo de reorganização, v torna-se escravo de u. Como $h(v) \leq h(u) \implies v \neq l$. Logo l não pertence ao componente B antes da reorganização. Isto implica que l pertence ao componente A. Sabe-se que o componente A respeita a propriedade 5. Portanto todos os escravos não compartilhados em A antes da reorganização possuem h igual ou maior que seus mestres. A única mudança nas conexões feita durante a reorganização é o estabelecimento de uma conexão entre v e u. Portanto, para quaisquer dois nós que estavam em A, existe uma conexão após a reorganização se, e somente se, existe um conexão entre eles antes da reorganização. Logo, todo nó não compartilhado após a reorganização que pertence a A, era um nó não compartilhado de A antes da união. Portanto l não pertence a A. Isto é uma contradição, e portanto a premissa é verdadeira. □

A proposição abaixo é consequência dos teoremas acima relativos ao caso 1 das reorganização e aqueles expressos no anexo A deste documento. Esta proposição diz que o componente resultante da união de dois componentes que atendem as 5 propriedades requeridas também as atende, independente da reorganização usada durante o processo de união.

Lema 6. *Dado que dois componentes A e B respeitam as propriedades antes da união, segue que: (1) o componente resultante possui um único líder; (2) o líder do componente resultante possui no máximo seis escravos; (3) não existe nenhum mestre com mais de sete escravos no componente resultante; (4) o líder do componente resultante possui pelo menos um escravo não-compartilhado; (5) todos os escravos do componente resultante que não são pontes possuem h menor ou igual ao h de seu mestre.*

Com este resultado prova-se que os componentes formados pelo algoritmo atendem as propriedades necessárias para o correto funcionamento do algoritmo:

Teorema 1. *Todo componente respeita as seguintes propriedades: (1) o componente possui um único líder; (2) o líder deve possuir no máximo seis escravos; (3) os mestres devem possuir menos que sete escravos; (4) o líder deve possuir um escravo não-compartilhado ou nenhum escravo; (5) todo escravo que não é uma ponte possui h menor ou igual ao h de seu mestre.*

Demonstração. A prova é feita por indução para o número de nós do componente.

Caso base: Quando o componente possui apenas um elemento l.

1. Quando o componente possui um único membro, como visto na linha 2 do algoritmo 2, o nó l é inicializado como líder.

2. Quando não existe nenhum outro nó no componente, l possui 0 escravo e como $0 \leq K-1$ a proposição é válida.

3. Quando não existe nenhum outro nó no componente, então l não é o mestre (ao menos um nó atuando como escravo de l é necessário para se tornar mestre). Logo, pode-se afirmar que não existe mestre no componente e então não existe mestre com mais de sete escravos.

4. Quando não existe nenhum outro nó no componente, seu líder não pode possuir nenhum escravo. Logo, pode-se afirmar que a propriedade de que o líder possui pelo menos um escravo não-compartilhado ou não possui nenhum escravo é verdadeira.

5. Quando não existe nenhum outro nó no componente além de l, não há neste componente nenhum escravo não-compartilhado. Logo pode-se afirmar que todos os nós que a propriedade afirmando que todos as pontes e escravos não-compartilhados possuem h menor ou igual ao h de seu mestre é verdadeira.

Conclui-se então que todas as cinco propriedades se verificam para componentes com um elemento.

Passo de indução: O componente A possui tamanho k

Se o componente A de tamanho k existe, então ele foi formado através do processo de união de um componente B e tamanho b, e um componente C cujo tamanho $c = k - b$.

Dado que $b < k$ e $c < k$ pode-se afirmar, pela hipótese de indução, que C e B são componentes válidos pois respeitam as cinco propriedades.

Logo, pode-se afirmar que as propriedades são verdadeiras devido ao lema 67 do anexo A. □

4.3.15 Comparação com algoritmos BSF dinâmicos e single-hop

Na literatura foram propostos diferentes algoritmos BSF, conforme apresentado no Capítulo 3. Os dois principais algoritmos BSF com descoberta dinâmica e single-hop são TSF e Law. O princípio de funcionamento destes algoritmos é o mesmo do Bluemob: componentes são unidos sucessivamente até que um único componente reste contendo todos os nós. Entretanto, a forma como este processo de união é realizado difere entre os três algoritmos.

A primeira diferença entre TSF e os outros dois algoritmos (Bluemob e Law) está na topologia do componente. Os componentes em Bluemob e Law são mantidos sob uma topologia Mesh, pois apesar da topologia resultante não conter ciclos, um nó pode possuir mais de um pai. TSF, ao contrário, impõe ao componente uma estrutura em árvore. Devido a isto, o processo de união de TSF é completamente diferente de Law e Bluemob pois deve preservar as restrições impostas por uma topologia em árvore na estruturação da rede.

A segunda diferença encontra-se na forma como o processo de busca por componentes é coordenado. No TSF os nós integrantes de um componente são independentes e todos realizam o processo de busca *Bluetooth* (INQUIRY e INQUIRY SCAN) de forma autônoma e não coordenada. Portanto, não há em TSF a figura de líder como ocorre em Bluemob. Em Bluemob o processo de busca *Bluetooth* dentro de um componente é coordenado e gerenciado por intermédio de um líder.

Bluemob e Law são mais semelhantes entre si, especialmente ao compará-los à TSF. Isto se deve ao fato da estrutura interna dos componentes mantidos por eles ser semelhante. São três as principais semelhanças entre a Law e Bluemob:

- Ambos fazem uso de um líder responsável por coordenar as atividades de busca por novos componentes para realizar a união.
- Ambos mantém um nó não compartilhado junto ao líder para que ele realize procedimentos *Bluetooth* custosos sem prejudicar a rede.
- Ambos baseiam o processo de união de componente em uma tarefa de união que se repete a cada δ segundos. Note que o processo de formação é continuo e que δ é menor que o tempo de deslocamento entre paradas de ônibus, portanto, entre paradas, ocorre mais de uma rodada de formação de rede.

Ao comparar Bluemob e Law o principal aspecto que faz com que os algoritmos possuam comportamentos diferentes está na forma como as reor-

ganizações são realizadas e como Bluemob incorpora h nestas reorganizações para melhor estruturar a topologia da rede. O impacto nas reorganizações pode ser visto diretamente no número de casos de reorganização existentes em Bluemob e o número de casos existentes em Law. Enquanto Bluemob necessita de treze diferentes casos, o algoritmo Law necessita seis reorganizações:

1. Se $v = w$ (o componente possui apenas um nó v), tem-se dois casos:
   (a) Caso 1: $|S(u)| < 7$
   (b) Caso 2: $|S(u)| = 7$
2. Se $v \neq w$ (as *Piconets* de u e w são mescladas em uma *Piconet*), então:
   (a) Caso 3: $|S(u)| + |S(w)| < 7$
   (b) Caso 4: $|S(u)| + |S(w)| = 7$ e $|S(u)| = 1$
   (c) Caso 5: $|S(u)| + |S(w)| = 7$ e $|S(u)| > 1$
3. Caso 6: $|S(u)| + |S(w)| > 7$

Note que para cada condição em Law, existe 2 ou mais em Bluemob. Estes casos adicionais são consequência da introdução do peso h em Bluemob, havendo um maior número de combinações de componentes a serem consideradas na reorganização. Note que:

1. Caso 1 de Law é desmembrado nos Casos 1 e 2 de Bluemob
2. Caso 2 de Law é desmembrado nos Casos 3, 4 e 5 de Bluemob
3. Caso 3 de Law é desmembrado nos Casos 6 e 7 de Bluemob
4. Caso 4 de Law é desmembrado nos Casos 8 e 9 de Bluemob
5. Caso 5 de Law é desmembrado nos Casos 10 e 11 de Bluemob
6. Caso 6 de Law é desmembrado nos Casos 12 e 13 de Bluemob

O impacto da introdução do h afeta também as provas de corretude do algoritmo. Em Bluemob foi introduzida uma nova propriedade que limita as topologias possíveis de serem formadas. Essa propriedade diz que h de mestres deve ser maior que o h de escravos não compartilhados. Isto faz com que nem toda topologia de mesh seja válida para Bluemob. Law não possui esta restrição, portanto aceita a existência destas topologias que são inválidas para Bluemob.

Ao necessitar manter esta propriedade, as provas de corretude das reorganizações são afetadas devendo ser considerada em Bluemob.

4.4 CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentado o novo algoritmo de formação de *Scatternets* Bluemob. Este é um algoritmo *Single-hop*, com descoberta dinâmica projetado para operar em ambientes pequenos onde os nós não apresentam a mesma propensão de deixar a rede.

O algoritmo foi projetado para operar no interior de ônibus, com o objetivo de formar uma rede *ad-hoc* entre os passageiros de um ônibus. Com isso é viabilizada aplicações sensíveis a contexto para coleta de informações sobre os ônibus.

Entretanto, este algoritmo não é limitado ao contexto de sistema de transporte coletivo baseados em ônibus. Pode-se aplicar este algoritmo e técnica a outros contextos como, por exemplo, economia de energia e posicionamento. Neste caso, o objetivo deixa de ser a diminuição no número de partições ocasionadas pelo desembarque de passageiros e seus *smartphones*.

Por exemplo, em *Scatternets* mestres e pontes consomem mais energia que escravos, pois necessitam rotear mensagens. Isto implica em maior número de mensagens transmitidas e recebidas por estes nós, sendo portanto, interessante alocar nestas posições chave da rede nós com maior carga energética. Este cenário poderia ser útil em uma pequena festa ou conferência, onde uma *Scatternet* espontânea fosse montada.

No exemplo do consumo energético, pode-se usar Bluemob fazendo com que o peso h reflita o nível energético dos nós. Desta forma, Bluemob posiciona nós com maior capacidade energética nestas posições. O objetivo é portanto aumentar a média da capacidade energética de mestres e pontes. A viabilidade de aplicação neste contexto deve ser estudada em mais detalhes para definir o consumo energético dos dispositivos gastos no processo de formação. Este estudo não será realizado no contexto deste trabalho.

5 VALIDAÇÃO DO ALGORITMO BLUEMOB

Avaliamos o algoritmo Bluemob levando em consideração métricas qualitativas e quantitativas selecionadas entre aqueles descritas no capítulo 2. O objetivo é validar o algoritmo Bluemob, mostrando que as características exigidas pelo ambiente de operação, e aplicações que usam a rede são atendidas pelo Bluemob. Portanto, busca-se mostrar que o desempenho apresentado por Bluemob para formar a rede é suficiente para reconstruir a rede de nós a bordo de um ônibus após o embarque e desembarque de passageiros em um tempo menor que o tempo típico de viagem entre pontos de paradas. O objetivo secundário é mostrar as características estruturais das *Scatternets* formadas por Bluemob. Por exemplo, aspectos como diâmetro da rede e número de *Piconets*. Estas características intrínsecas a topologia e estrutura da *Scatternet* afetam a performance de aplicação que as usam. Por exemplo, diâmetro da rede influi no número de saltos necessários para rotear mensagens na rede, e consequentemente, no tempo de transmissão de mensagens.

5.1 AMBIENTES DE SIMULAÇÃO

A avaliação Bluemob foi feita por meio de simulação. Foram usados dois simuladores. O primeiro tem a meta de averiguar os tempos necessários para conclusão das reorganização. Para isto, é levado em consideração a pilha de comunicação *Bluetooth*. O segundo simulador busca verificar a evolução da rede mediante entrada e saída de dispositivos *Bluetooth*, em um contexto de transporte coletivo em ônibus.

Optou-se pelo uso de dois simuladores pois o simulador usado em outros trabalhos na literatura para simular redes *Bluetooth* não é capaz de capturar a dinâmica de embarque e desembarque de passageiros em um ônibus ao longo de um itinerário. A inclusão desta capacidade ao simulador mostrou-se tecnicamente difícil, pois (1) o desenvolvimento do simulador está estagnado a alguns anos, exigindo um ambiente especial de desenvolvimento de difícil operação; (2) há pouca documentação sobre a arquitetura interna do simulador, dificultando a análise e projeto de modificações e extensões ao modelo. Entretanto, em nosso caso, este simulador é adequado para determinar os tempos envolvidos no processo de reorganizações. Um segundo simulador foi desenvolvimento para avaliar o comportamento das reorganizações no ambiente de ônibus, capturando a lógica de formação das redes e a dinâmica de embarques e desembarques dos passageiros ao longo do itinerário, abstraindo as camadas inferiores avaliadas anteriormente.

5.1.1 Ambiente de simulação com pilha *Bluetooth*

O primeiro ambiente de simulação visa analisar os aspectos temporais relacionados ao algoritmo. Logo, é necessário levar em consideração a pilha de protocolos *Bluetooth* e suas particularidades. Adotou-se nesse trabalho o simulador NS2 (The VINT Project,) estendido com o modelo de simulação UCBT (WANG; AGRAWAL,). Este modelo de simulação implementa a pilha de protocolos *Bluetooth* de acordo com a especificação *Bluetooth* 1.1. UCBT é o principal simuladores *Bluetooth* usado na literatura para avaliar algoritmos BSF. Foi desenvolvido na Universidade de Cincinatti.

UCBT simula a pilha de protocolos *Bluetooth* em detalhes. Simula grande parte da especificação da camada *Baseband* e protocolos LMP, L2CAP, BNEP. Inclusive aspectos como salto de frequência, descoberta de dispositivo, protocolo de estabelecimento de conexões que possuem efeito significativo sobre os processos de reorganização e união de componentes de Bluemob. A Figura 21 mostra a pilha *Bluetooth* conforme especificação e a estrutura implementada pelo UCBT.

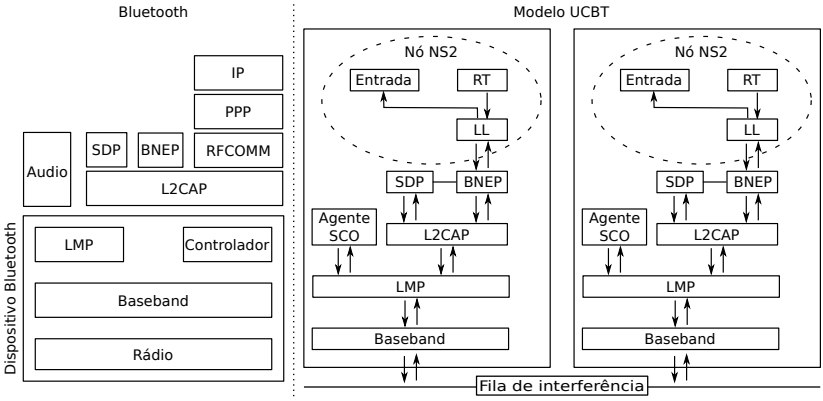


Figura 21: Estrutura da pilha de protocolos *Bluetooth* e a estrutura do simulador UCBT.

O canal de rádio não é modelado explicitamente. São usados dois modelos para simulação do canal físico: um modelo de perda de pacotes; e um modelo de interferência para comunicação entre *Piconets*. O modelo de perda de pacotes é baseado no modelo usado por BlueHoc (KUMAR,).

A simulação UCBT implementa os protocolos *Bluetooth* como um único protocolo no nível de enlace no NS2. UCBT fornece infraestrutura para o estudo e simulação de algoritmos para formação de *Scatternets*. Isto é

feito através de um ponto de extensão no nó NS2. Um mecanismo baseado em eventos é oferecido, onde funções com assinatura pré-definidas são chamadas quando ocorrem eventos na pilha *Bluetooth*. Eventos como estabelecimento de uma conexão *Bluetooth*, destruição de uma conexão, recebimento de mensagens possuem funções associadas para que o algoritmo de formação atue. O algoritmo de formação tem completo acesso à pilha *Bluetooth*, podendo executar procedimentos exportados pelas diferentes camadas.

5.1.2 Ambiente de simulação sem pilha *Bluetooth*

Os algoritmos Bluemob e Law baseiam-se em uma tarefa a ser executada a cada δ s. O valor de δ deve ser grande o suficiente para que todas as operações de busca de dispositivos nas proximidades e união de componentes sejam finalizadas. Portanto, δ encapsula todos os tempos envolvidos nas operações realizadas pela pilha *Bluetooth*. Isto inclui o processo de descoberta de dispositivos próximos, estabelecimento de conexões, e a reorganização dependentes da pilha de protocolos *Bluetooth*.

O tempo necessário para formar uma *Scatternet* com todos os dispositivos presentes no interior do veículo é igual a δ vezes o número de rodadas executadas pelo líder do componente. Quanto menor o valor de δ, menor o tempo disponível para estabelecimento completo da *Scatternet*.

O simulador *ad-hoc* usado na simulação dos experimentos apresentados nesta seção fazem uso deste fato para abstrair os detalhes da pilha *Bluetooth*. Este simulador, ao invés de determinar o tempo necessário para estabelecer a *Scatternet*, mede o número de rodadas para formação da *Scatternet*. A topologia da *Scatternet* resultante também é determinada. Este simulador *ad-hoc* consiste em sua essência na implementação dos algoritmos 3 e 4 em Python, portanto, para reprodutibilidade dos resultados obtidos por intermédio do simulador *ad-hoc*, o leitor é convidado a implementar estes algoritmos.

Para desenvolvimento deste simulador, optou-se inicialmente por usar a infraestrutura provida por OmNET++. OmNET++ é um simulador a eventos discretos bastante usado para simulação de redes de comunicação. Entretanto, optou-se por descartá-lo pois a maior parte de suas funcionalidades não são necessárias para atingir os objetivos da simulação. Decidiu-se projetar um simulador simplificado especificamente desenvolvido para atender as necessidades específicas dos experimentos e cenários de transporte coletivo de interesse para testar e analisar Bluemob. O simulador foi implementado em Python. A estrutura do simulador é mostrada na Figura 22.

O simulador mantém um dígrafo representando o estado atual das conexões *Bluetooth* em memória, modelado através da classe *Cenario*. Os dis-

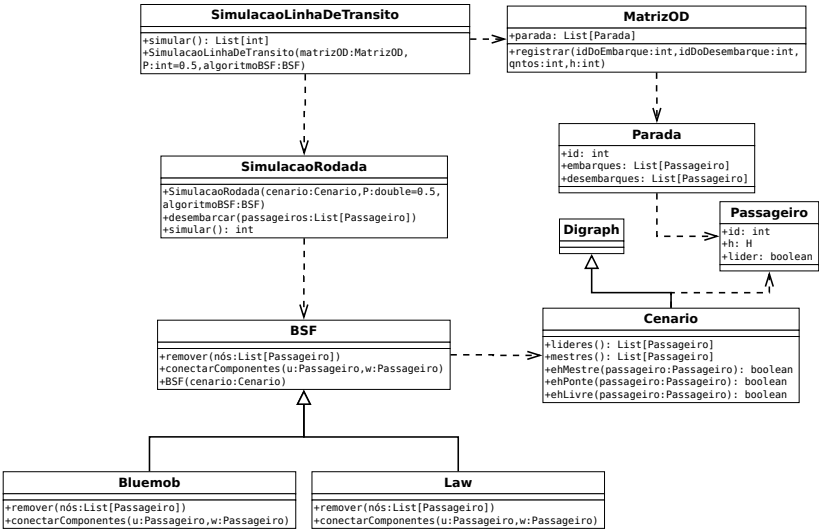


Figura 22: Estrutura do simulador BSF especializado implementado em Python

positivos *Bluetooth* são instâncias da classe *Passageiro*, e são os vértices no dígrafo. As conexões *Bluetooth* entre dispositivos são representadas pelas arestas. O sentido da aresta no dígrafo indica os papéis mestre e escravo exercidos pelos dispositivos na conexão. O dispositivo que exerce o papel de mestre da conexão é o vértice-inicial, enquanto o vértice-final representa o dispositivo que está atuando como escravo.

Não projetou-se um simulador genérico de algoritmos BSF, e sim um simulador especializado em algoritmos BSF com descoberta dinâmicas cuja estrutura é baseada na união de componentes. Possuindo uma estrutura que reflete as características da família de algoritmos como Law e Bluemob. Foram implementados neste simulador o algoritmo Law e Bluemob. Os algoritmos BSF simulados são subclasses da interface *BSF*.

A simulação de uma rodada do algoritmo BSF é definida pela classe *SimulacaoRodada*. Esta classe possui um método *simular*, na qual uma rodada do algoritmo é simulada. O algoritmo de simulação de uma rodada BSF consiste é ilustrada no fluxograma da Figura 23. A simulação de uma rodada consiste em determinar para cada líder o estado que ele estará nesta rodada. Há 50% de chance de um líder assumir estado INQUIRY, e 50% de chance de ingressar no estado INQUIRY SCAN. Uma vez que os estados dos líderes foram determinados, o algoritmo organiza de forma aleatória pares de líderes

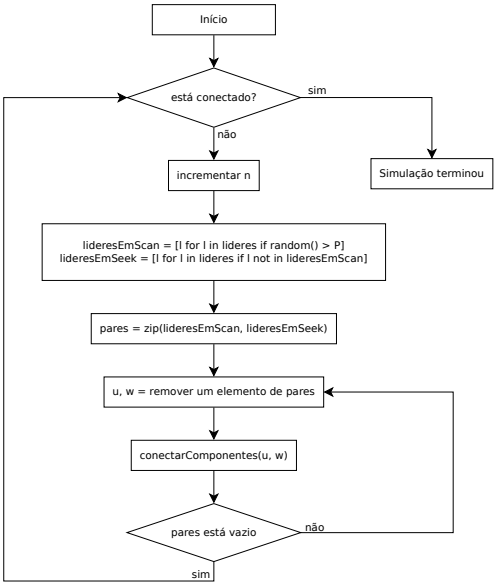


Figura 23: Diagrama de fluxo da simulação *ad-hoc*

com estados diferentes. Ou seja, serão formados pares u e w, sendo que u está no estado INQUIRY e w no estado INQUIRY SCAN. Este sorteio simula o processo *Bluetooth* INQUIRY.

Para cada par de líderes u e w, uma aresta (u, v) é adicionada ao grafo da rede. v é o dispositivo que está realizando o procedimento de INQUIRY SCAN na componente de w, segundo o algoritmo BSF sendo simulado. Por fim, o processo de união de componentes do algoritmo sendo simulado é executado. Após todos os componentes terem sido unidos, o simulador considera a rodada encerrada.

Os algoritmos BSF implementados no simulador possuem mecanismo para remoção de nós. A saída de nós ocorre sempre entre rodadas. Não há suporte no simulador para a saída de nós da rede durante uma rodada. Quando um nó deixa a rede, o vértice associado é removido do grafo, assim como todas as arestas a ele associados. Então o procedimento de desconexão do algoritmo BSF é executado. Isto é usado pelos algoritmos Bluemob e Law para que todos os componentes mantenham as invariantes após a saída do nó.

Além da simulação de cada rodada do algoritmo BSF, foi projetado um mecanismo para descrição de experimentos através da modelagem de cenários de transporte coletivo. Isto é realizado através da definição de uma matriz Origem-Destino. Primeiramente, o número de pontos de ônibus é de-

terminado. A cada ponto é atribuído um número. Em seguida determina-se para cada par de pontos (x,y) o número de passageiros que embarcaram em x e desembarcaram no ponto y. O simulador então realiza o seguinte processo de simulação: (1) remove-se do grafo todos os vértices programados para desembarcar na parada de ônibus atual; (2) adiciona-se ao grafo todos os vértices representando os pontos de embarque atual; (3) são executadas rodadas até que o grafo esteja conectado; (4) armazena-se o número de rodadas necessárias para forma a rede; (5) caso haja mais uma parada a frente, retorna-se ao passo 1, caso contrário, a simulação é considerada encerrada.

5.2 AVALIAÇÃO QUANTITATIVA DOS TEMPOS PARA CONCLUSÃO DAS REORGANIZAÇÕES

O objetivo deste experimento é avaliar os tempos envolvidos para completar as reorganizações por Bluemob. Isto permite determinar o tempo mínimo da rodada de Bluemob δ. Como as reorganizações devem ser finalizadas antes da próxima rodada ser iniciada, δ deve ser maior ou igual ao tempo necessário para concluir as reorganizações.

Realizou-se simulações usando a implementação do Bluemob no UCBT para determinar os tempos envolvidos em cada uma das 13 reorganizações. Para isso, projetou-se um experimento onde 2 *Piconets* com estrutura pré-determinada são formadas. Neste instante há dois componentes, e consequentemente, dois líderes. Um dos componentes, cujo líder é u, é colocado para realizar o processo de INQUIRY, enquanto o outro componente, cujo líder é w, é colocado no estado INQUIRY SCAN. Feito isto, inicia-se um cronômetro e os dois líderes iniciam a rodada de Bluemob. Ao finalizar a reorganização o cronômetro é parado e o tempo armazenado.

Todo líder no Bluemob possui menos que sete escravos. Há portanto 49 configurações possíveis para estruturação das duas *Piconets*. Além do tamanho das *Piconets* dos líderes, Bluemob leva em consideração os pesos h dos nós da *Piconet* do líder para determinar a reorganização a ser feita.

De forma a contemplar todas as combinações possíveis, projetamos um cenário de simulação para cada combinação de tamanho de *Piconet* e para cada variação de pesos. Há quatro nós cujos pesos são avaliados por Bluemob na escolha da reorganização a ser realizada: u, w, y, v. Varia-se os pesos destes de tal forma que todas as ordens relativas entre os pesos destes nós possuam um cenário associado.

Cada um destes cenários é simulado 40 vezes. Obtém-se os tempos máximo, mínimo e médio para cada reorganização, que estão sumarizadas na Figura 24.

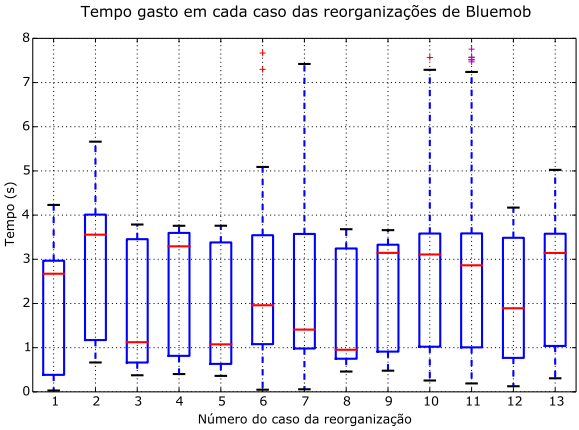


Figura 24: *Box plot* dos tempos envolvidos em cada uma das reorganizações de Bluemob

Observou-se que o maior tempo para reorganizar dois componentes é 7.75 s. Portanto, o tempo δ de duração de uma rodada deve ser maior que 7.75 s.

Na implementação de Bluemob feita no UCBT é simulada utiliza apenas o mecanismo de INQUIRY e INQUIRY SCAN para o estabelecimento de conexões. Conforme comentado no capítulo 3, o estabelecimento de uma conexão pode ser otimizado quando os *clocks* e endereços *Bluetooth* são conhecidos. Nestes casos, pode-se omitir o processo de INQUIRY e prosseguir diretamente ao procedimento de estabelecimento de conexão PAGE. Com isso, será observada uma diminuição nos tempos de estabelecimento de conexão. O procedimento INQUIRY costuma ser algumas ordens de magnitude mais lento que o PAGE. Um segundo ponto de otimização não realizada na implementação simulada diz respeito ao mecanismo de inversão de papéis. Nestes casos, o algoritmo faz com que os nós que terão seus papéis invertidos sejam desconectados e uma nova conexão entre eles realizada, através do processo de INQUIRY. É possível otimizar este processo em uma implementação real, utilizando o mecanismo de inversão de papéis disponibilizados pelo *Blueto-oth* (na camada L2CAP). Ao inverter os papéis mestre e escravo desta forma não há necessidade de desconectar para, em seguida, reconectar os nós.

Dado que estas otimizações não foram usadas durante a simulação, conclui-se que o tempo de 7.75s observado é um limite superior para o valor de δ. Entretanto, em uma implementação otimizada do protocolo, espera-se que os tempos envolvidos nas reorganizações sejam menores que os observa-

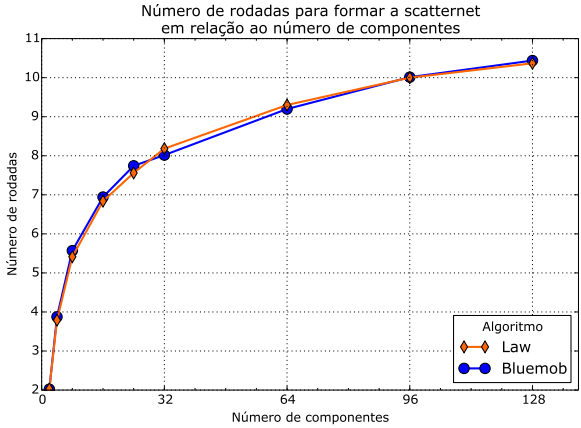


Figura 25: Número de rodadas para formação de rede a partir de uma rede desconectada, quantidade de nós variando de 0 a 128

dos na simulação.

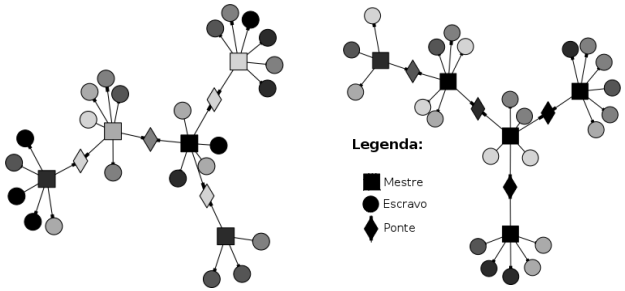
O tempo da rodada do algoritmo de Law é pelo menos da mesma magnitude daqueles verificados para Bluemob. Isto porque a reorganização mais demorada (número 6) de Bluemob cujo tempo máximo é 7.75 s é idêntica a uma das reorganizações usadas no algoritmo Law. Logo, δ de Law tem limite inferior igual a 7.75 s. Deste ponto em diante, escolhemos que $\delta = 8$ s.

5.3 AVALIAÇÃO QUANTITATIVA PARA FORMAÇÃO E REFORMAÇÃO DE REDES *AD-HOC*

5.3.1 Formação a partir de uma rede desconectada

Neste experimento é avaliado o comportamento de Bluemob para formar uma rede a partir de nós desconectados. Este cenário equivale à formação da rede no ônibus entre seu ponto de origem até o primeiro ponto de parada. A simulação é executada até ser formada uma *Scatternet* conectada com todos os nós. O número de rodadas necessárias para formar a *scatternet* é mostrado na Figura 25) para quantidades de nós variando entre 2 e 128.

Observa-se que o tempo para formação da rede está entre $\delta \times 10 = 80$ s e $\delta \times 11 = 88$ s com 128 passageiros. Dado que o tempo típico entre paradas é 60 s, com 128 passageiros não se consegue formar a rede até a próxima parada. Entretanto, esta situação não é usual durante a trajetória do



(a) Law et. al (b) Bluemob

Figura 26: Uma rede *Bluetooth* formada pelo (a) algoritmo Law, e (b) algoritmo Bluemob; quanto mais clara a tonalidade do nó, maior sua propensão a deixar a rede

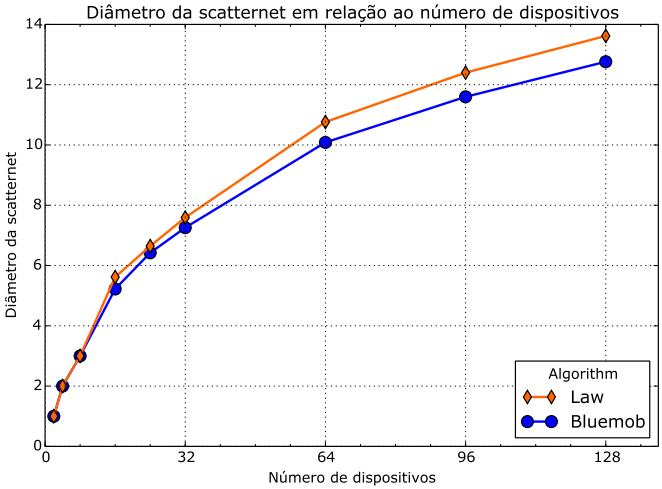
ônibus, sendo observada normalmente apenas na origem do itinerário. Neste ponto, os desembarques não costumam ocorrer durante as primeiras paradas, permitindo ultrapassagem da restrição.

O número de rodadas aumenta logaritmicamente em relação ao número de componentes, ou seja o aumentos do tempo de formação em relação ao aumento de passageiros é sub-linear.

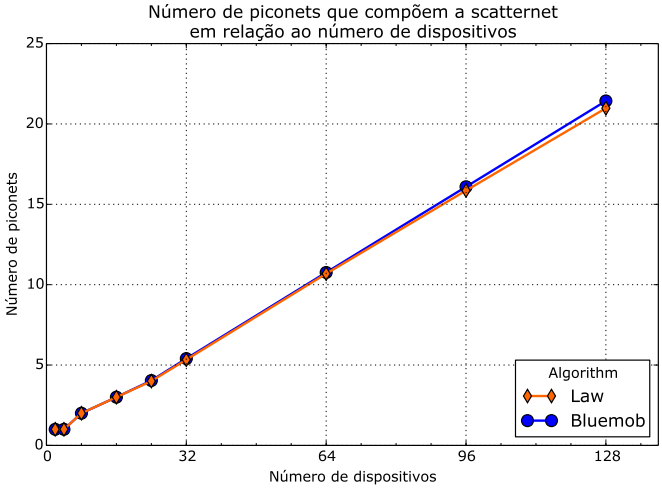
Em comparação ao algoritmo Law, pode-se ver na Figura 25 que não há diferenças em relação ao número de rodadas necessárias para a formação da rede. Isto é, não há penalidade devido à maior complexidade da união de componentes de Bluemob. Este comportamento pode ser explicado pela forma como a busca por componentes é realizada em rodadas (LAW; MEHTA; SIU, 2003).

Do ponto de vista da topologia da rede formada, há uma melhora qualitativa no posicionamento dos nós, conforme mostrado na Figura 26, na qual os nós com tonalidade mais clara apresentam maior propensão a deixar a rede. Na rede formada com Bluemob (na Figura 26b) mestres e pontes tem tonalidade mais escura que os escravos. Logo, quando um nó deixa a rede nesta topologia, é geralmente um escravo e não ocorrem particionamentos. Já na rede formada pelo algoritmo Law (na Figura 26a) vários mestres e pontes possuem tonalidade clara, indicando maior chance de particionamento da rede por conta de desembarques.

Estruturalmente, as *Scatternets* formadas pelos algoritmos Bluemob e Law são similares. O diâmetro médio das *Scatternets* formadas por Bluemob, mostrado na Figura 27a é um pouco menor que a média alcançada por Law. Isto implica em menor número de saltos para roteamento de mensagens na



(a) Diâmetro médio das *Scatternets*



(b) Número médio de *Piconets*

Figura 27: Características das *Scatternets* resultantes do processo de formação de *Scatternet* de Bluemob

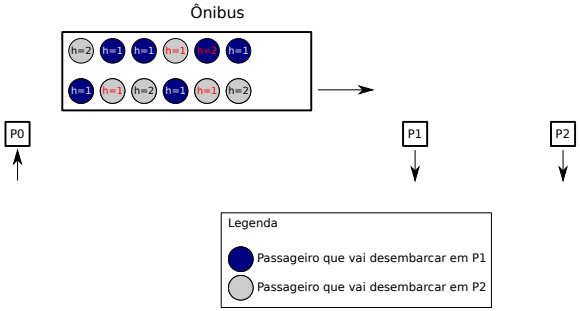


Figura 28: Cenário usado para o terceiro experimento

Scatternet formada por Bluemob.

O número de *Piconets* que constituem a *Scatternet* aumenta com o número de nós tanto em Bluemob quanto em Law. Este comportamento é esperado pois as *Piconets* são limitadas a 7 escravos e 1 mestre. Conforme visto na Figura 27b a diferença no número de *Piconets* entre os algoritmos é mínima.

Ao analisar estes gráficos, conclui-se que os algoritmos são equivalente em aspectos estruturais das *Scatternets* formadas por eles. Mesmo a diferença entre o diâmetro das redes formadas pelos dois algoritmos é pequena. Bluemob cria diâmetros menores que Law pois há um maior número de pontes mestre-escravo.

5.3.2 Reformação da rede após desembarque de passageiros

O objetivo do segundo experimento é avaliar o efeito do arranjo de nós quando passageiros deixam a rede, mostrando as melhoras obtidas com a introdução do peso h no procedimento de reorganização durante o desembarque.

O cenário no qual este experimento é realizado consiste em 3 pontos de paradas, denominadas pontos 0, 1 e 2, conforme ilustrado na Figura 28. n nós embarcam no ponto 0 do itinerário e iniciam o procedimento de construção da *Scatternet*. Após a *Scatternet* ser formada, o ônibus chega ao ponto 1. Neste ponto, uma fração dos nós desembarca do ônibus, deixando a rede. Isto implica no particionamento da *Scatternet*. Após o desembarque ser concluído, mede-se o número de rodadas necessárias para a rede se reconectar entre os pontos 1 e 2. Após a *Scatternet* ser reconectada, o ônibus atinge o ponto de parada 2 e todos os passageiros desembarcam, finalizando o experi-

mento.

Aos nós que desembarcam no ponto 1 é atribuído o peso 1 a h. Aos nós que desembarcam no ponto 2 é atribuído valor 2 a h.

Parte dos passageiros têm seus pesos atribuídos erroneamente. Desta forma é possível verificar o efeito destes erros nos tempos de reconexão da rede após o desembarque. A inclusão deste erro é feita invertendo o peso de uma porcentagem dos passageiros que desembarcam em 1 e em 2. Aos passageiros que desembarcam em 1 atribui-se o peso 2, ao invés do peso correto 1. De forma análoga, aos passageiros que desembarcam em 2 atribui-se o peso 1.

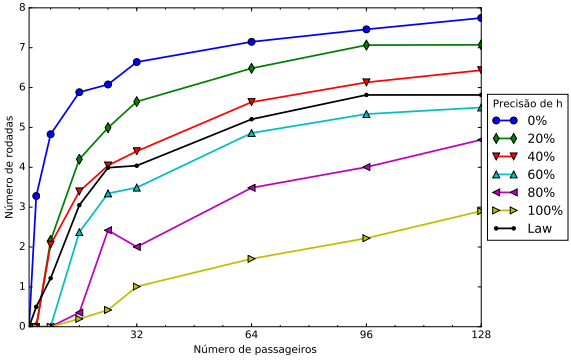
A precisão x indica a porcentagem de passageiros que possuem peso h correto. Ou seja, uma precisão de $x\%$ significa que $x\%$ dos passageiros que desembarcam em 1 possuem $h = 2$ e $x\%$ dos passageiros que desembarcam em 2 possuem $h = 1$.

Os experimentos foram simulados 400 vezes para cada uma das situações. O número médio de rodadas para reconectar a rede é mostrado na Figura 29. Três experimentos foram feitos: com 25%, Fig. 29a, com 50%, Fig. 29b, e 75%, Fig. 29c, dos nós deixando a rede. Pelas figuras, observa-se novamente o aumento logarítmico do número de rodadas com o número de nós. Em relação à precisão das estimativas, fica claro que quanto melhor for estimada a saída, melhor Bluemob se comporta. Nota-se que no pior caso, com precisão 0% e 128 passageiros, o tempo para reconstrução da rede ultrapassa 60s. Com o aumento da precisão de h obtém-se tempos na faixa 30s, atingindo o objetivo de reconstrução da rede entre paradas. Observa-se que a partir do momento em que precisão de h ultrapassa 50%, Bluemob supera o desempenho de Law. Em alguns casos observa-se melhora de até 75% do tempo necessário para reconstruir a rede após saída de parte dos nós.

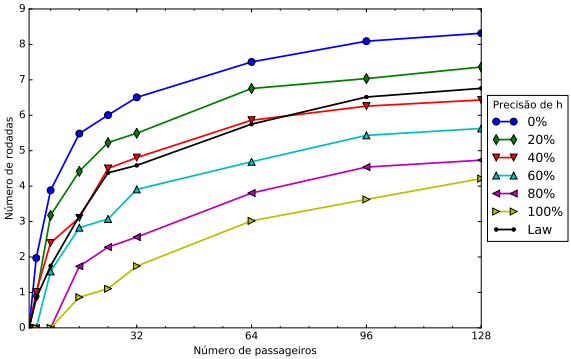
Quando h possui precisão menor que 50%, Bluemob aloca nas posições de mestres e pontes passageiros com alta propensão de desembarque. Com isso, há uma perda de desempenho em relação a Law, que posiciona os nós de forma aleatória na rede.

O número de rodadas necessárias para reconstruir a rede depende da porcentagem de nós que desembarcam da rede. Ao fixar o número de passageiros e variar o número de desembarques, chega-se a Figura 30.

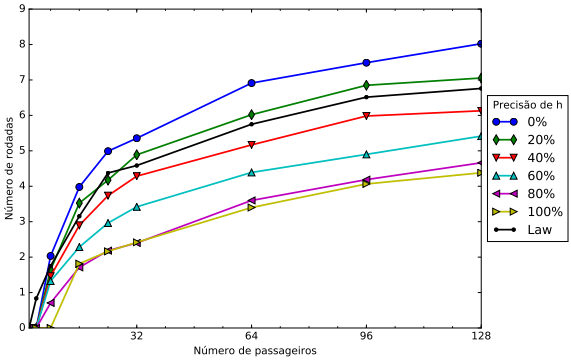
Um comportamento interessante de Bluemob é que o número máximo de rodadas necessárias para reconstruir a rede não cresce linearmente com a porcentagem de nós deixando a rede. Há um número máximo de rodadas atingido quando 50-60% dos nós deixam a rede. Após este limiar, o número de rodadas necessárias para reconstruir a rede começa a diminuir, pois o aumento no número de nós deixando a rede, ao invés de aumentar o número de componentes na rede devido ao particionamento, a reduz. Este comporta-



(a) saída de 25% dos nós da rede

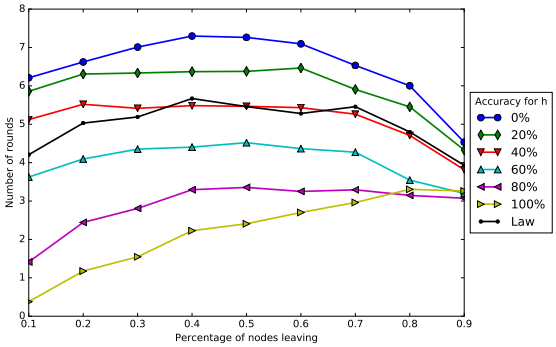


(b) saída de 50% dos nós da rede

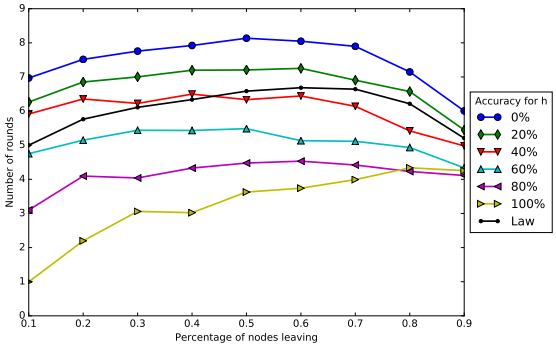


(c) saída de 75% dos nós da rede

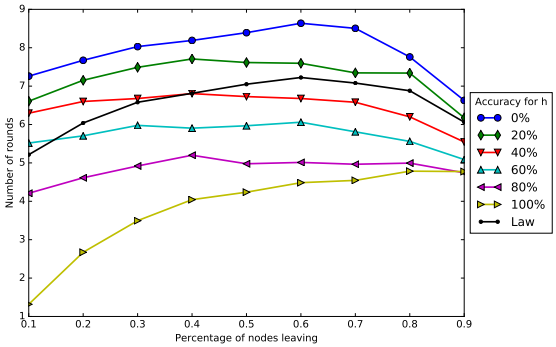
Figura 29: Experimento reconstruindo *Scatternet* após desembarque



(a) Rede inicialmente com 50 nós



(b) Rede inicialmente com 100 nós



(c) Rede inicialmente com 150 nós

Figura 30: Comportamento do algoritmo Bluemob em comparação com Law com aumento da taxa de saída de nós da rede

mento pode ser visto quando um mestre deixa a rede. Neste caso, todos os escravos do mestre se tornam um componente. Caso tanto um mestre como seu escravo deixe a rede, haverá um componente a menos para ser unido a rede. Logo, o número de reorganização começa a decrescer. Após o limiar de 50-60% dos nós deixando a rede, este fenômeno começa a predominar o que faz com que o número de rodadas para reconstrução da rede diminua.

5.3.3 Reformação da rede usando método mais conservador na atribuição de h

Conforme observado no experimento anterior, caso o valor atribuído a h não seja suficientemente preciso, Law apresenta melhores resultados que Bluemob. Entretanto, ao modificar a forma como os pesos h são atribuídos aos nós, é possível melhorar os resultados. Adotando uma metodologia de atribuição de h na qual se reduz o peso h daqueles nós que se tem certeza de que irão desembarcar em breve.

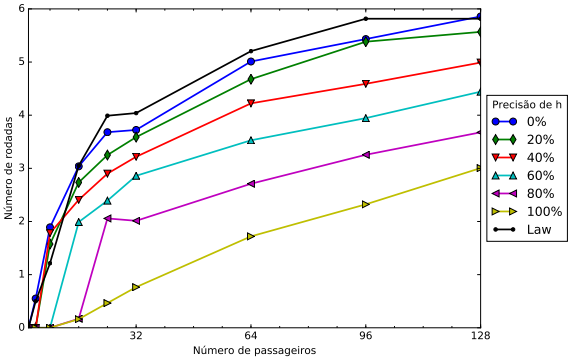
De forma a ilustrar as melhorias obtidas, realiza-se o mesmo experimento anterior, mas usando a nova metodologia de atribuição de pesos h aos nós. Neste experimento, os passageiros que desembarcam no ponto de parada dois possuem peso h igual a 2. Aqueles que desembarcam no ponto de parada um possuem $h = 1$. Erros indicam que uma porcentagem dos nós que desembarcam no ponto um terão seu peso $h = 2$.

Cada experimento é executado 400 vezes. O número médio de rodadas para que o algoritmo crie uma *Scatternet* conectada após o desembarque é mostrado na Figura 31. Foram executados três experimentos com 25% (Figura 31a), 50% (Figura 31b) e 75% (Figura 31c) dos nós deixando a rede.

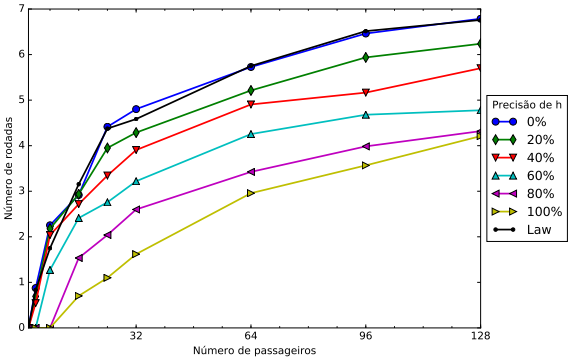
Como pode ser visto na Figura 31, o número de rodadas para a reconstrução da *Scatternet* apresentou melhoras em relação ao experimento anterior. Bluemob supera Law em todos os caso, exceto quando todos os nós possuem $h = 2$. Como antes, quanto mais passageiros possuem o peso correto, menor o número de rodadas para reconstrução da rede.

A adoção de uma estratégia mais conservadora na atribuição dos pesos h aos nós leva a uma melhora na performance de Bluemob. A metodologia conservadora evita subestimar pesos de h, atribuindo a h o número do ponto de desembarque apenas quando há um alto nível de confiança de o peso estar correto. Caso o nível de confiança seja baixo, atribui-se ao peso h valor igual ao número do último ponto de desembarque.

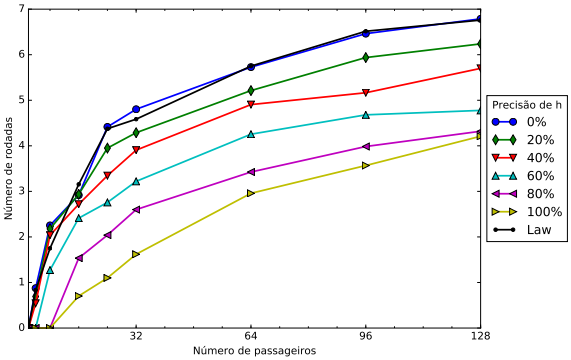
A ideia por trás desta metodologia é prevenir que Bluemob coloque em pontos chave da topologia nós que possuem alto nível de confiança que desembarcarão em breve. Isto é feito reduzindo o peso h destes nós.



(a) 25% dos nós desembarcam



(b) 50% dos nós desembarcam



(c) 75% dos nós desembarcam

Figura 31: Comportamento de Bluemob usando a metologia aperfeiçoada de atribuição de pesos *h*

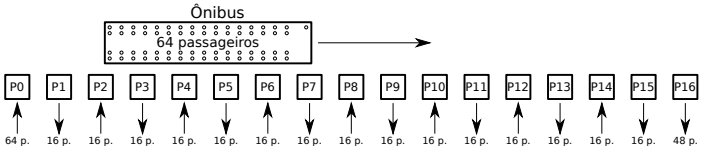


Figura 32: Trajetória de ônibus para teste de reformação da rede em itinerário

5.3.4 Reformação da rede durante um trajeto de ônibus

Este experimento tem como objetivo mostrar o comportamento do algoritmo ao longo da trajetória de um ônibus com uma série de embarques e desembarques. Para tanto, assume-se uma trajetória do ônibus conforme mostrado na Fig. 32. A trajetória do ônibus ocorre ao longo de 17 paradas, numeradas sequencialmente sendo a primeira parada a de número 0. Na parada zero ocorre o embarque de 64 passageiros. Nas paradas ímpares, há o desembarque de 16 passageiros e nas paradas pares embarcam novos 16. Os passageiros que desembarcam são aqueles embarcados a mais tempo, e o peso h é igual ao número do ponto estimado para desembarque. A precisão de h indica a porcentagem de passageiros com h correto.

Para Bluemob, a corretude de h corresponde a correção nas relações de ordem entre os pesos h. Portanto, o h do passageiro que desembarca em um ponto de parada está correto caso seja menor que o pesos dos passageiros que irão desembarcar posteriormente. Logo, atribui-se aos passageiros com h errados o número de parada menos o número da sua parada de embarque. Por exemplo, passageiros com h errado que embarcaram no ponto de parada 0 tem $h = 17$. Já passageiros que embarcam em 2, terão $h = 15$

Na Figura 33 é mostrado o efeito da saída de nós da rede quando há uma sequência de embarques e desembarques. Considerando o caso com h correto, duas fases distintas são observadas: (i) P0-P7; e (ii) P9-P15. Durante a fase (i), os 64 passageiros inicialmente embarcados em P0 desembarcam em grupos de 16 nas paradas ímpares. Enquanto isso, 16 novos passageiros embarcam no ônibus a cada parada par. Em P1, a maior parte dos 16 passageiros que desembarcam são escravos devido a seu baixo h (cujo valor é 1), resultando em um menor número de partições. Em P3 e P5 o número de partições aumenta devido ao desembarque de passageiros atuando como mestres e pontes na rede. O máximo em P5 é causado pela h relativamente alto (cujo valor é 5). Em P7, os últimos 16 nós embarcados em P0 deixam o ônibus causando poucas partições pois eles já foram deslocados para as bordas da rede devido aos nós embarcando em P2, P4 e P6 com maiores h.

Na fase (ii), os 16 nós que desembarcam em P9 ($h = 9$) embarcaram

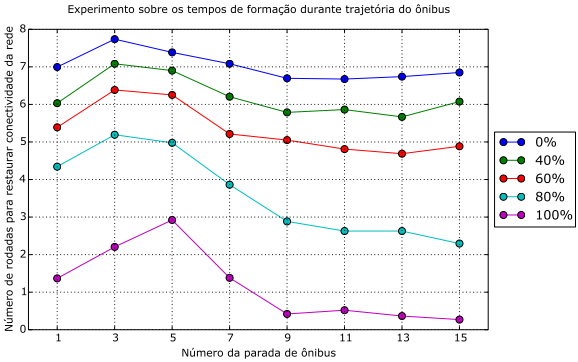


Figura 33: Efeito da saída de passageiros durante trajetória com 64 passageiros 16 desembarques seguidos de 16 embarques alternadamente

juntos em P2. Observa-se na simulação que em P2, Bluemob une estes 16 nós com o líder, não a misturando com outras *Piconets*. Portanto, estes 16 nós tendem a permanecer juntos em duas ou três *Piconets* interconectadas com nós com $h < 9$ por uma ponte. Então, em P9 poucas partições são feitas pois nós com $h = 9$ desembarcam juntos. A mesma análise se aplica aos desembarques em P11, P13 e P15. Note que durante a fase (ii), praticamente não há necessidade de reorganizações para reconectar a rede quando precisão de h é 100%.

5.4 CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentada um conjunto de experimentos com o objetivo de avaliar o algoritmo Bluemob, proposto no capítulo 4.

Os experimentos foram realizados usando dois ambientes de simulação: (1) com simulação da pilha *Bluetooth*; (2) sem simulação da pilha *Bluetooth*. O primeiro ambiente de simulação foi usado com o objetivo de determinar os tempos envolvidos para a conclusão de uma rodada do algoritmo Bluemob. Durante uma rodada são desenvolvidas duas atividades principais: busca por componentes e reorganização. A busca por componentes é caracterizado pelo processo de INQUIRY realizado pelo líder do componente, ou INQUIRY SCAN realizado ou pelo líder ou por um de seus escravos não compartilhados. A segunda atividade ocorre quando o processo de busca por um componente é concluída com sucesso. Os líderes dos componentes trocam informações entre si sobre a estrutura das componentes e decidem quais as modificações devem ser realizadas na topologia da rede para que o novo

componente seja válido. As atividades de busca e reorganização feitas por Bluemob dependem dos tempos envolvidos na comunicação entre os dispositivos.

Com base nos experimentos, o tempo máximo observado para finalizar as atividades de uma rodada foi de 7.75 s. Portanto, concluímos que o tempo para que uma nova rodada seja realizada deve ser igual ou maior a 7.75 s. Ao analisar o algoritmo de Law sob as mesmas condições, averiguamos que as reorganizações de Law necessitam de tempos semelhantes para serem concluídas. Note-se que 7.75 s é uma estimativa pessimista visto que uma série de otimizações podem ser implementadas.

O segundo ambiente de simulação foi usado para avaliar dois aspectos do algoritmo Bluemob: (1) características estruturais da *Scatternet* formada; (2) efeito do embarque e desembarque dos passageiros nos tempos de reorganização da rede.

As *Scatternets* formadas por Bluemob apresentaram uma clara melhora no posicionamento dos nós na rede quando comparadas às *Scatternets* formadas usando o algoritmo Law. Nós com maior peso *h* assumem posições chave na *Scatternet*: mestres e pontes. Estruturalmente, analisou-se o número de *Piconets* na *Scatternet*, e a média do diâmetro do grafo da *Scatternet*. Os dois parâmetros comportaram-se conforme esperado, apresentado um aumento linear com o aumento no número de nós da *Scatternet*. Ao comparar o comportamento com o algoritmo de Law observou-se que não há diferenças significativas entre os dois algoritmos.

Concluímos, portanto, que apesar de Bluemob apresentar características estruturais como número de *Piconets* e *Scatternet* comparáveis com outros algoritmo da literatura, como Law, há uma clara melhora na distribuição de papéis dentro da *Scatternet*. Isto permite adaptar o algoritmo de formação as necessidades especificas das aplicações que a utilizam.

Ao avaliar os experimentos envolvendo a entrada e saída de passageiros da *Scatternet*, e consequentemente do ônibus, observou-se melhora no tempo necessário para a reconstrução da rede após particionamentos causados pela saída de nós da rede. Por exemplo, em alguns cenários otimistas, houveram melhoras de até 75% nos tempos envolvidos quando comparados ao algoritmo Law. Esta melhora é fruto da melhor atribuição de papéis entre os nós, decorrente do uso do peso *h* por Bluemob.

A melhora no tempo de reorganização faz com que Bluemob seja bem adaptado para formação de redes entre passageiros, um dos objetivos desta tese. Ao incorporar conhecimento do comportamento dos passageiros ao processo de formação de rede, através de *h*, conseguiu-se reduzir os tempos de reconstrução da rede após desembarques. Segundo os experimentos, conseguiu-se formar, após o desembarque, uma rede no interior do veículo

dentro dos tempos estipulados para finalizar o processo de formação da ordem de 30 a 40 s ao usar um h suficientemente preciso. Por exemplo, no terceiro experimento, foi observado que com uma heurística suficientemente precisa (acima de 80%) é necessário menos de 38.5 s (5 rodadas multiplicado por $\delta = 7.75\ s$) para reestabelecer a rede após o desembarque de 64 passageiros. Há um grande número de passageiros (128 passageiros) e há o desembarque de 50% dos passageiros. Esta quantidade de desembarque, conforme visto no experimento 29c, o pior caso costuma situar-se quando entre 50% a 60% dos passageiros desembarca.

Por fim, mostramos que há um método de atribuição de pesos h aos passageiros tal que Bluemob apresenta tempos iguais ou melhores que aqueles observados ao usar Law. Desta forma, pode-se elaborar estratégias para codificação do conhecimento do comportamento dos passageiros em valores h de tal forma que a performance não seja degradada quando do uso de Bluemob. Este fato é relevante no cenário de operação proposto, pois oferece diretrizes para evitar que Bluemob tenha degradação de performance devido à atribuição errônea de valores aos pesos h dos passageiros.

Em resumo, neste capítulo mostramos que: (1) Bluemob é um algoritmo bem adaptado ao meio em que se propõe atuar; (2) as informações fornecidas pela aplicação influem positivamente no desempenho da rede, quando suficientemente precisas; (3) o tempo entre paradas é suficiente para reconstruir redes de até 128 passageiros após o desembarque, dado que exista um número suficiente de passageiros com pesos h precisos; (4) existe um método para atribuição de pesos h que permite que os tempos para reconstrução de Bluemob sejam inferiores ou igual aqueles observados com uso de Law.

6 AVALIAÇÃO OPERACIONAL DO ALGORITMO BLUEMOB EM CENÁRIO DE TRANSPORTE PÚBLICO

O algoritmo Bluemob foi validado no capítulo anterior do ponto de vista de métricas relacionadas aos aspectos de formação da rede entre passageiros. A validação realizada possui duas características principais: (1) a validação não abstrai o método de cálculo dos pesos *h* dos nós utilizando uma métrica de precisão na avaliação de Bluemob; e, (2) as características particulares de uma determinada linha e o padrão de embarques e desembarques não são levadas em consideração. Desta forma a validação se concentrou nas características estruturais da topologia da rede formada e tempos médios para formação após a saída de nós da rede mediante diferentes níveis de qualidade de *h* em comparação com outros algoritmos.

Neste capítulo, apresentamos um cenário de Transporte Público particular. Com base neste cenário, avaliamos Bluemob do ponto de vista operacional. O objetivo é validar o algoritmo Bluemob, mostrando que em um ambiente de transporte público real suas características são adequadas a uma aplicação de coleta de informações de trânsito urbano a exemplo do proposto no capítulo 2.

Para tanto, projetou-se um cenário baseado em uma linha de ônibus existente. Determinamos os tempos de deslocamento dos ônibus entre pontos de parada e o tempo para reconstruir a rede após os desembarques e embarques ao longo do trajeto. Com isso busca-se mostrar que Bluemob tem potencial para melhorar os tempos para reconstruir a rede entre os passageiros a bordo de um ônibus após os desembarques respeitando os limites temporais observados para que o ônibus transite de um ponto de parada a outro. A aplicação e performance de Bluemob depende da forma como o peso *h* são atribuídos aos nós. Logo, foram propostas e aplicadas algumas heurísticas que são usadas no momento do embarque do passageiro para determinação do peso *h*. Ao fim mostrou-se que, nesta linha em particular, usando Bluemob e as heurísticas propostas aliadas com conhecimento parcial sobre os hábitos dos passageiros é possível melhorar o tempo para reconstruir a rede após o desembarque.

6.1 MÉTODO DE AVALIAÇÃO EM CENÁRIO REAL

O cenário usado na avaliação de Bluemob é baseado na linha de ônibus Troncal-10 de Blumenau. Escolhemos esta linha de ônibus como base para a avaliação devido a: (1) familiaridade e conhecimento dos padrões de des-

locamentos dos passageiros desta linha; (2) viabilidade para modelagem e simulação do trânsito e das viagens dos ônibus da linha usando Aimsun (BARCELÓ; CASAS, 2005). O conhecimento da linha Troncal-10 foi obtida durante o desenvolvimento do projeto Sincrobus, finalizado em 2014, onde foi usada como linha piloto para experimentação de algoritmos de controle de tráfego e prioridade de ônibus. O simulador de trânsito Aimsun também foi escolhido para avaliar os tempos de deslocamentos do ônibus na linha pois já haviam resultados simulados da linha Troncal-10, e sua modelagem, nesta plataforma de simulação.

Determinada a linha de ônibus, capturou-se o conhecimento acerca dos padrões de deslocamento dos passageiros adquiridos pela equipe e nativos conhecedores da cidade de Blumenau em uma matriz Origem-Destino. Esta matriz captura de forma aproximada as viagens típicas praticadas pelos passageiros durante o período matutino.

No que segue serão usados dois simuladores: simulador *ad-hoc* desenvolvido por nós e descrito no capítulo 5 e o simulador Aimsun.

O simulador *ad-hoc* analisa os tempos envolvidos para reforma a rede de passageiros.

O simulador Aimsun é um simulador de tráfego urbano. Com ele determinamos os tempos de deslocamento de ônibus considerando o trânsito, semáforos, tempos necessário para embarque e desembarque de passageiros.

A avaliação de Bluemob do ponto de vista de uma aplicação de *crowdsensing* de informações de transporte público, idealmente, deve atender ao requisito de terminar de reconstruir a rede antes da próxima parada de ônibus ser realizada. Portanto, o objetivo é comparar os tempos necessários para reconstruir a rede de passageiros com o tempo de deslocamento do ônibus entre pontos de paradas.

O método usado para esta avaliação consiste em: (1) determinar um conjunto de 4 matrizes-origem que apresentem as mesmas características que a matriz OD gerada pela equipe que conhece os padrões de deslocamento dos passageiros na cidade de Blumenau; (2) determinar o tempo de viagem entre paradas de um ônibus ao longo do itinerário para as 5 matrizes OD usando um simulador de tráfego urbano (Aimsun); (3) determinação do tempo médio para reconstruir a rede após cada embarque e desembarque para cada uma das 5 matrizes OD usando algoritmo de Law, e Bluemob com diferentes métodos para atribuição de pesos h; (4) avaliação dos resultados comparando os tempos observados nos passos (2) e (3).

A determinação das 4 matrizes-origem adicionais (passo 1) foram realizadas através de pequenas modificações introduzidas na matriz OD elaborada pelos conhecedores da linha Troncal-10. Chamaremos esta matriz de matriz semente. O método para introdução das modificações é como segue:

soma-se a cada elemento da matriz OD uma variável aleatória de média 0, sendo que cada elemento da matriz pode variar no máximo 20% em relação à matriz OD semente; e o somatório das linhas e colunas da matriz OD varia no máximo 10%. Apesar de apresentarem um mesmo padrão, viagens de ônibus não apresentam um mesmo número de embarques e desembarques em cada uma das paradas, havendo variabilidade entre viagens. Esta variabilidade afeta tanto os tempos para reconstruir a rede (observada no simulador *ad-hoc*), quanto o tempo gasto nas paradas de ônibus (Aimsun).

Através do simulador Aimsun determina-se os tempos que os ônibus levam para se deslocar de um ponto de ônibus até o próximo ponto (passo 2), levando em consideração o número de passageiros que realizaram o embarque/desembarque nas paradas e o tráfego. Estes tempos determinam o tempo hábil para reconstruir a rede entre passageiros até que o próximo embarque/-desembarque ocorra.

O simulador apresentado no capítulo 5 é usado para determinar o tempo médio gasto para reconstruir a rede ao longo do trajeto (passo 3). A simulação é feita usando matrizes OD de cada uma das viagens e o tempo médio para reconstrução da rede é determinado.

Em algumas situações é necessário acesso ao conhecimento histórico da linha para determinação dos pesos h dos nós. Nas simulações realizadas no passo 3 assumimos que a matriz OD histórica da linha é igual ao somatórios de todas as 5 matrizes OD determinados no passo 1.

Ao fim do processo, tem-se os tempos necessários para formação da rede usando Bluemob e os tempos que o ônibus leva para se deslocar. Ao compará-los (passo 4), busca-se mostrar que Bluemob tem potencial para melhorar os tempos para reconstruir a rede entre passageiros embarcados respeitando os limites temporais observados para que o ônibus transite de um ponto de parada a outro. Apresentamos para da uma das viagens simuladas: (1) o tempo médio para reconstruir a rede entre cada um dos trechos comparado ao tempo necessário para o ônibus percorrer o mesmo trecho; (2) o tempo médio para reconstruir a rede em cada uma das viagens; (3) tempo médio para reconstruir a rede em relação a Law ao usar dados individualizados de passageiros para determinação de h.

6.2 LINHA TRONCAL-10

A linha Troncal-10 de Blumenau é composta por 18 pontos de paradas. O itinerário da linha Troncal-10 é mostrado na Figura 34. O primeiro ponto de parada é chamado Fonte (localizado no extremo sul do itinerário), e o final Aterro (localizado no extremo norte do itinerário). Os pontos intermediários

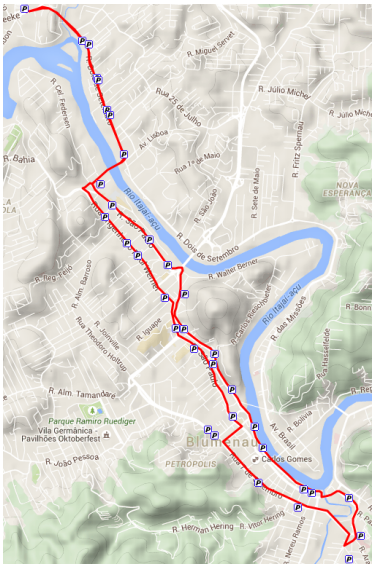


Figura 34: Itinerário da linha Troncal-10.

são: Siga 1, Siga 19, Dr. Blumenau, Carlos Gomes, Siga 22, Siga 23, Siga 24, Siga 25, Siga 26, Siga 27, Siga 28, Siga 29, Siga 30, Siga 31, Siga 32, Siga 33.

A linha Troncal-10 é uma linha de BRT, portanto sua velocidade média é alta quando comparada a outras linhas de ônibus. Esta linha apresenta como destino preferencial o Aterro, com em média 30% dos desembarques. Entretanto, outros 3 pontos apresentam um número significativos de desembarques: Dr. Blumenau (10%), Carlos Gomes (10%) e Siga 24 (8%). Entretanto embarques e desembarques são observados durante todo seu trajeto em menor grau. Os embarques por sua vez ocorrem em sua maioria na Fonte, com em média 45% dos embarques. O restante dos embarques são distribuídos entre os demais pontos sem haver grande concentração. Estas porcentagens são expressas na tabela OD semente gerada (Tabela 3).

6.3 SIMULAÇÃO DOS TEMPOS DE VIAGENS USANDO O SIMULADOR AIMSUN

O simulador Aimsun é um simulador microscópico de trânsito. Ou seja, as ações de cada veículo são simuladas individualmente com base em

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	-	8	6	17	16	1	6	6	2	1	-	-	6	1	1	-	-	14
2	-	-	-	1	1	-	-	2	-	-	-	-	-	-	-	-	-	1
3	-	-	-	-	1	-	-	1	-	-	-	-	1	-	-	-	-	1
4	-	-	-	-	-	2	2	3	-	-	-	-	1	-	1	-	-	3
5	-	-	-	-	-	1	2	2	-	-	2	-	2	-	-	-	-	3
6	-	-	-	-	-	-	-	1	-	-	-	-	1	-	-	-	-	1
7	-	-	-	-	-	-	-	-	3	1	1	-	1	1	-	-	-	3
8	-	-	-	-	-	-	-	-	-	2	1	1	-	-	-	-	-	1
9	-	-	-	-	-	-	-	-	-	-	-	2	-	1	-	-	-	-
10	-	-	-	-	-	-	-	-	-	-	-	2	-	1	1	-	-	1
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	2	4
12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	1	3
13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	2	3
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5
16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6
17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5
18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Tabela 3: Matriz OD semente usada para avaliação de Bluemob na linha Troncal-10, Blumenau, SC. Linhas representam origens, colunas os destinos.

Figura 35: Simulação Aimsun da linha Troncal-10.

informações locais. Na Figura 35 mostramos o Aimsun durante a simulação do trânsito na linha Troncal-10.

A simulação foi realizada com um ônibus saindo a cada 10 minutos. Foram simuladas 5 viagens de ônibus, sendo que para cada viagem foi gerada uma matriz OD.

Determinou-se ao fim da simulação os tempos de viagem entre pontos de parada para cada um dos ônibus. Os tempos obtidos a partir da simulação no Aimsun estão sumarizadas na tabela 4. Nota-se na tabela que há uma dispersão grande entre tempos observados entre viagens. Estes tempos são esperados e são ocasionados pelas sinaleiras entre os pontos. O impacto do trânsito sobre os tempos entre paradas é pequeno nesta linha pois os ônibus viajam em faixas exclusivas.

6.4 SIMULAÇÃO DO PROCESSO DE FORMAÇÃO DA REDE USANDO BLUEMOB

O objetivo deste estudo consiste na análise do tempo de reconstrução da rede comparativamente aos tempos de viagem dos ônibus e da influência de várias heurísticas. Para isto, determinamos o tempo necessário para reconstruir a rede entre os passageiros entre cada um dos trechos entre as paradas de ônibus e comparamos com o tempo de deslocamento do ônibus obtido pela simulação realizada no Aimsun.

O tempo necessário para reconstrução da rede pelo algoritmo Bluemob

		Ônibus							
		1	2	3	4	5	$E[t]$	$X[t]$	$\frac{E[t]}{X[t]}$
(1) Fonte	(2) Siga 1	69,0	99,7	39,7	72,0	103,5	76,8	26,0	3,0%
(2) Siga 1	(3) Siga 19	67,5	71,2	80,2	77,2	71,2	73,5	5,1	7,0%
(3) Siga 19	(4) Dr. Blum.	69,7	76,5	138,0	76,5	72,7	86,7	28,8	33,0%
(4) Dr. Blum.	(5) Crls Gmes	102,7	99,7	106,5	153,7	102,7	113,1	22,8	20,0%
(5) Crls Gmes	(6) Siga 22	98,2	88,5	98,2	122,2	79,5	97,4	16,0	16,0%
(6) Siga 22	(7) Siga 23	54,7	62,2	59,2	65,2	61,5	60,6	3,9	6,0%
(7) Siga 23	(8) Siga 24	57,0	60,0	66,7	54,7	45,7	56,9	7,7	13,0%
(8) Siga 24	(9) Siga 25	43,5	41,2	44,2	45,7	57,7	46,5	6,5	14,0%
(9) Siga 25	(10) Siga 26	48,0	49,5	54,0	53,2	51,7	51,3	2,5	5,0%
(10) Siga 26	(11) Siga 27	66,0	67,5	62,2	73,5	60,0	65,9	5,2	8,0%
(11) Siga 27	(12) Siga 28	54,0	55,5	54,7	54,0	48,0	53,3	3,0	6,0%
(12) Siga 28	(13) Siga 29	70,5	70,5	75,7	74,2	64,5	71,1	4,4	6,0%
(13) Siga 29	(14) Siga 30	111,7	94,5	98,2	154,5	95,2	110,9	25,4	23,0%
(14) Siga 30	(15) Siga 31	57,0	56,2	64,5	62,2	51,0	58,2	5,3	9,0%
(15) Siga 31	(16) Siga 32	60,7	60,7	62,2	63,7	90,7	67,7	13,0	19,0%
(16) Siga 32	(17) Siga 33	60,0	56,2	62,2	57,7	55,5	58,4	2,8	5,0%
(17) Siga 33	(18) Aterro	99,5	105,0	106,0	112,2	81,5	100,9	11,7	12,0%
Mínimo		43,5	41,2	44,2	45,7	45,7	41,2		
Máximo		111,7	105	138	154,5	102,7	154,5		
Total (s)		1190,0	1215,0	1273,0	1373,0	1193,0	6244,0		
Total (min)		19,8	20,3	21,2	22,9	19,9			

Tabela 4: Tempos de viagem entre paradas segundo simulação da linha Troncal-10 realizada no Aimsun

depende da forma como os pesos *h* são atribuídos a cada um dos passageiros. Portanto, elaborou-se um conjunto de heurísticas para serem usadas. As heurísticas utilizam informações como o ponto de embarque do passageiro e o conhecimento da matriz OD histórica da linha.

6.4.1 Heurísticas usadas na determinação dos pesos *h* dos nós

Nas simulações foram adotadas quatro heurísticas para determinação dos pesos *h* dos nós em conjunto com Bluemob. Assume-se que cada ponto de parada possui um número atribuído correspondente à ordem da parada no itinerário. Ou seja, a Fonte é o ponto 0, Siga 1 é o ponto 1, e assim por diante. As heurísticas usadas são as seguintes:

- Exata: a relação de ordem entre os pesos *h* atribuídos reflete a ordem dos embarques e desembarques dos passageiros durante a viagem do ônibus.

- Ponto de embarque: reflete a observação de que passageiros embarcados a mais tempo no ônibus tendem a desembarcar antes daqueles que embarcaram recentemente. Nesta heurística, *h* é igual ao número do ponto onde o nó embarcou.

- Ponto de desembarque mais frequente: a matriz histórica é usada para determinar o ponto de parada de desembarque mais frequente dentre aqueles que embarcam em um determinado ponto de parada. Neste caso *h* é igual ao número do ponto de parada onde há o maior número de desembarques dentre os nós que embarcaram neste ponto.

- Ponto de desembarque mais longínquo dentre os frequentes: Esta heurística visa escolher o ponto de desembarque mais longínquo dentre os destinos mais frequentes para quem embarca em um determinado ponto. Para isto, a matriz histórica é usada para classificar os pontos de paradas do desembarque do mais utilizado ao menos. Elimina-se os pontos menos usados da lista, ou seja, aqueles que são destino de menos de (100-K)% do destino mais usual. Dentre os pontos remanescente escolhe-se aquele localizado mais longe do ponto de embarque. Exemplo: considere K=10% e um embarque em P0, com três destinos: P1 com 150 desembarques, P2 com 100 desembarques e P3 com 140 desembarques. Primeiramente ordena-se os destinos ordenados por desembarque (P1=150, P3=140, P2=100), em seguida desconsidera-se aqueles que tem menos de 135 desembarques (150 - 10%) restando

P1=150 e P3=140. Por fim escolhe-se aquele mais longe do ponto de embarque, no caso P3.

6.5 EXPERIMENTOS EM CENÁRIOS DE TRANSPORTE

Conforme descrito no método adotado, foram realizadas simulações de 5 viagens de ônibus usando o Aimsun para determinar o tempo necessário para o ônibus se deslocar entre pontos de paradas.

Para cada uma destas viagens, usamos a mesma matriz OD para determinar o tempo para reconstruir a rede após o embarque e desembarque. Para isto, foi usado o simulador *ad-hoc*. Em cada uma das matrizes OD geradas, simulamos os tempos necessários para estabelecimento da rede obtidos pelo uso do algoritmo Law e Bluemob usando os métodos de atribuição de h (Exata, Ponto de embarque, Ponto de desembarque mais frequente, Ponto de desembarque mais longínquo dentre os frequentes com K=10%). Cada cenário foi simulado 400 vezes e o tempo médio para reconstruir a rede foi medido.

6.5.1 Experimento 1: análise de Bluemob com heurísticas que independem de informações individualizadas dos passageiros entre paradas

Apresentamos na Figura 36 os tempos médios para reconstruir a rede após os embarques e desembarques de passageiros, assim como o tempo gasto pelo ônibus para se deslocar entre os dois pontos de paradas.

Ao comparar o tempo para reconstruir a rede com o tempo para o ônibus se deslocar de uma parada a outra, nota-se que em alguns trechos não é possível reconstruir a rede antes do ônibus chegar na próxima parada, mesmo quando a estratégia exata é utilizada. O primeiro trecho onde isto ocorre é o trecho entre Fonte e Siga 1. Neste trecho, espera-se que não haja melhora pelo uso de Bluemob já que não há desembarques na Fonte pelo fato de este ser o ponto de parada inicial. O segundo trecho onde isto ocorre é o trecho entre os pontos Siga 24 e Siga 25. A distância entre os pontos de paradas é muito pequeno para que a rede entre passageiros consiga se reconstruir antes do ônibus chegar até a próxima parada.

A existência de trechos onde a rede não pode ser completamente reconstruída após o particionamento não inviabiliza a operação de um sistema de coleta. Entretanto, esta possibilidade deve ser levada em consideração durante o processamento e operação do sistema que utiliza a rede *ad-hoc* de passageiros.

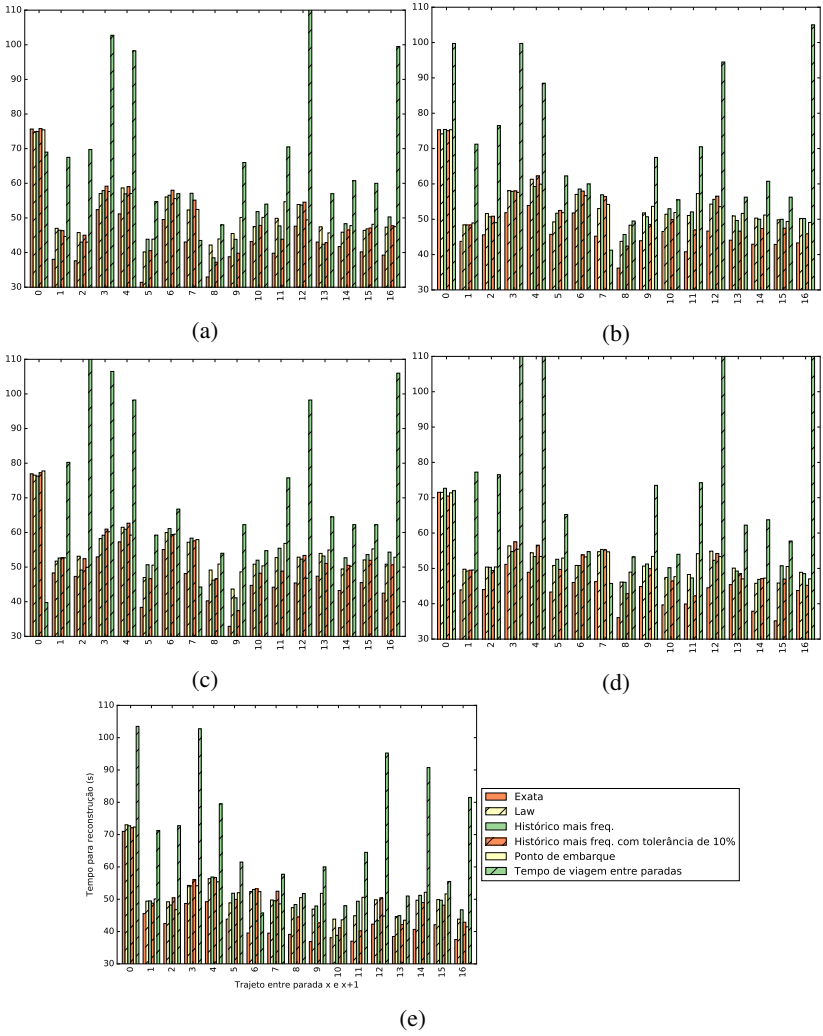


Figura 36: Tempo médio para reconstruir a rede (em segundos) durante (a) Viagem 1; (b) Viagem 2; (c) Viagem 3; (d) Viagem 4; (e) Viagem 5.

6.5.2 Experimento 2: comportamento médio de Bluemob heurísticas em relação a Law

A Figura 37 apresenta como cada uma das heurísticas propostas se comporta, em média, em relação a Law em cada uma das viagens simuladas pelo Aimsun. Ou seja, apresentamos para cada uma das heurísticas a média entre todos os trechos apresentados na figura 36 e em seguida relativizamos à Law.

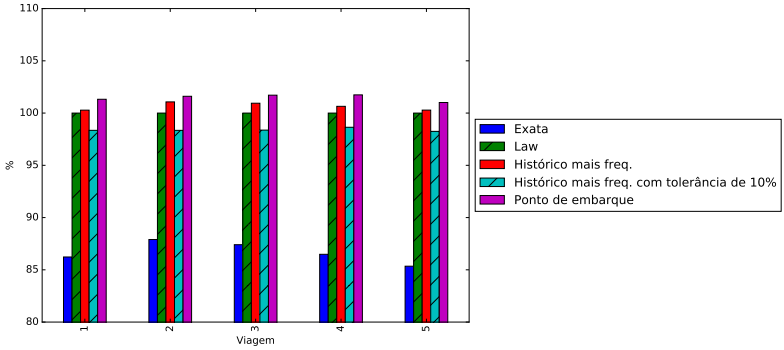


Figura 37: Tempo relativo para reconstruir a rede (Law = 100%).

As heurísticas propostas não obtiveram melhora expressiva em relação ao algoritmo Law, conforme pode ser visto na Figura 37. Isto ocorre pois os padrões de deslocamento de passageiros observados nesta linha, tornam especialmente difícil predizer o ponto de desembarque futuro dos passageiros no momento do embarque com base no ponto de parada e a matriz OD histórica. Por exemplo, no ponto onde a maior parte dos embarques ocorrem (Fonte) há 3 destinos prováveis (Dr. Blumenau, Carlos Gomes, Siga 24 e Aterro conforme pode ser visto na matriz OD da tabela 3), entretanto não é possível determinar qual passageiro se encaminhará a qual parada no momento do embarque. Entretanto, quando se consegue individualizar o ponto de desembarque de cada passageiro no momento do embarque, cenário representado pela estratégia exata, é possível obter melhoras significativas no tempo necessário para reconstruir a rede. Neste caso, Bluemob obtém tempos 12% a 15% menores que Law e demais heurísticas.

Além disso em determinados trechos, o grande número de embarques ocorridos aliados à proximidade entre paradas faz com que a rede não seja reformada antes de que o próximo desembarque ocorra. Isto indica que apesar de Bluemob ser capaz de melhorar os tempos de formação da rede após em-

barques e desembarques, é necessário que durante o processamento dos dados enviados à central de processamento seja levado em consideração a possibilidade de que se receba mais de um conjunto de passageiros embarcados em um mesmo ônibus, cada um referente a uma partição da rede de passageiros. Portanto, a central de processamento deverá perceber que 2 ou mais conjuntos de passageiros se referem a um mesmo ônibus e realizar a união dos conjuntos de passageiros para determinar os passageiros embarcados.

6.5.3 Experimento 3: Bluemob com heurísticas com informações individualizadas dos passageiros

Em determinadas linhas, como a Troncal-10, são necessárias informações individualizadas para que se consiga determinar o ponto de desembarque com precisão suficiente para Bluemob. Aplicativos para *smartphones* são capazes de capturar estas informações individualizadas ao usar o histórico e hábitos dos passageiros para auxiliar na determinação do ponto de desembarque no momento de embarque. Ao aliar estas informações àquelas já usadas pelas heurísticas é possível melhorar os tempos necessários para reconstruir a rede. Uma forma de usar esta informação consiste em usá-la sempre que disponível, usando uma segunda heurística qualquer para calcular o peso h quando não há histórico individualizado disponível para este trajeto.

A Figura 38 apresenta o tempo médio para reconstruir a rede em cada uma das viagens simuladas com Aimsun relativas a Law. Ao contrário do experimento anterior, considera-se que o aplicativo é capaz de capturar e fornecer informação sobre o ponto de desembarque dos passageiros usando o comportamento passado do passageiro. Parte-se do princípio que o aplicativo não possui sempre toda a informação necessária para determinar com precisão o ponto de desembarque do passageiro. Portanto traçamos experimentos variando a porcentagem de passageiros cujos aplicativos são capazes de determinar com precisão o ponto de desembarque. Caso o aplicativos não seja capaz de determinar no momento de embarque o ponto de desembarque do passageiro, usa-se a heurística "Ponto de desembarque mais longínquo dentre os frequentes" para determinação do ponto de desembarque.

Observa-se uma melhora linear nos tempos necessários para reconstruir a rede com Bluemob a medida que informações mais precisas são fornecidas pelo aplicativo. Esta melhora está na ordem de 3 a 4% a cada 25% de aumento no número de passageiros com histórico disponível. Caso o aplicativo consiga determinar com precisão o ponto de desembarque de 100% dos passageiros, ou seja, conseguesse chegar a heurística exata, consegue-se uma melhora aproximada de 15%.

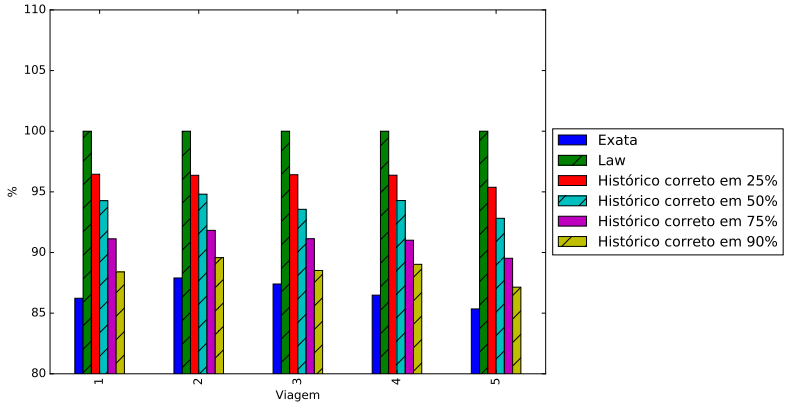


Figura 38: Tempo relativo para reconstruir a rede ao usar dados individualizados de passageiros (Law = 100%)

6.6 CONSIDERAÇÕES FINAIS

Neste capítulo observou-se o comportamento esperado de Bluemob em uma linha específica de ônibus. Isto foi feito simulando-se a linha Troncal-10 no Aimsun para determinar os tempos entre pontos de parada. Idealmente, o tempo entre paradas indicam o tempo máximo disponível para reconstruir a rede.

Observou-se que mesmo com uso de Bluemob, em determinados trechos, o grande número de embarques ocorridos aliados à proximidade entre paradas faz com que a rede não seja reformada antes de que o próximo desembarque ocorra. Isto indica que apesar de Bluemob ser capaz de melhorar o algoritmo, é necessário que a central de processamento leve em consideração a possibilidade de que se receba mais de um conjunto de passageiros embarcados referente a um mesmo ônibus.

Observa-se que a performance de Bluemob depende características particulares da linha de ônibus em análise. Além disso, melhoras nos tempos para reconstruir a rede dependem das heurísticas e do conhecimento do comportamento do passageiro durante a determinação do ponto de desembarque no momento de embarque dos passageiros.

A adoção de Bluemob na linha Troncal-10, em particular, mostrou-se complexa devido a baixa previsibilidade dos pontos de desembarque no momento de embarque. Usando heurísticas baseadas apenas no ponto onde os passageiros embarcaram, não houve melhoras em relação aos tempos observados usando Law. Entretanto, mostrou-se que com a adição de informações

individualizadas capturadas pelo aplicativo é possível obter melhoras consideráveis mesmos nestes cenários. Com informação individualizada de qualidade é possível atingir melhoras expressivas de até 15%, na média, em relação aos tempos observados em Law.

A linha Troncal-10 possui grande fluxo de passageiros com rotinas fixas, pois esta linha conecta regiões centrais e universidades. Portanto, seu uso é feito em grande parte por estudantes e trabalhadores para deslocamento de e para suas residências. Este fato leva a uma previsibilidade nos deslocamentos individuais sendo propicia a captura de informações pelo aplicativo.

Por fim, o algoritmo Bluemob apresenta potencial para uso em cenários reais como suporte para aplicativos de coleta de informações de sistemas de transporte público. No cenário em estudo, Bluemob foi capaz de atender o requisito temporal de reconstruir completamente a rede antes de chegar a próxima parada com exceção de alguns poucos trechos na Linha. Ressalta-se que a Linha Troncal-10 é uma linha de BRT, portanto sua velocidade média é alta quando comparada a outras linhas de ônibus. Portanto, a linha Troncal-10 é um caso que impõe ao algoritmo de formação de rede deadlines menores que o usual. Apesar desta limitação, mostramos o fato de não reconstruir a rede completamente em alguns trechos não inviabiliza o seu uso para aplicativos de coleta de informações de transporte público.

7 CONCLUSÃO

Nesta tese propomos uma forma para coleta de dados em sistemas de transporte público baseada em ônibus usando *Smartphones* e um sistema de *crowdsensing*. Esta proposta realiza a coleta de dados usando redes *Bluetooth ad-hoc* formadas entre os passageiros do ônibus. A pertinência do passageiro a rede *ad-hoc* é usada como critério de definição se um passageiro está ou não embarcado. Informações sobre a rede *ad-hoc* são enviadas a uma central de processamento onde dados de interesse a planejadores e operadores de transporte público são obtidas.

O funcionamento desta abordagem está diretamente condicionado a qualidade do algoritmo de formação da rede *ad-hoc* entre os passageiros. Foi observada uma potencial área de melhora neste aspecto, ao propor algoritmos que tomem informações específicas deste cenário para auxiliar a qualidade da formação.

Neste contexto, um algoritmo de formação de redes *Bluetooth*, chamado Bluemob, foi projetado para a construção de redes *Bluetooth ad-hoc* entre *smartphones* dos passageiros de ônibus.

O algoritmo BSF leva em consideração a propensão de um passageiro em permanecer no ônibus para influenciar a topologia da rede resultante. Passageiros com alta propensão em desembarcar são posicionados na borda da *Scatternet*. Deste modo, seu desembarque causa menor número de partições na rede. Passageiros com baixa propensão a deixar a rede são alocados em posições chave da topologia da rede: mestres e pontes.

Resultados experimentais obtidos por meio de simulação são apresentados. Eles mostram que o Bluemob melhora os tempos de reconstrução da rede após o desembarque com uso de uma boa medida de propensão de desembarque. Bluemob não introduz penalidades perceptíveis nos tempos e características da rede quando comparado a algoritmos da literatura, mas a distribuição dos nós na topologia da rede é melhorada significativamente. Obtendo-se tempos compatíveis com os tempos percebidos para o ônibus ir de um ponto de parada ao próximo.

Por fim, foi realizada a avaliação do algoritmo em um cenário mais próximo ao real usando como base a linha Troncal-10 de Blumenau em Santa Catarina. Matrizes Origem-Destino foram elaboradas usando conhecimento de habitantes da cidade sobre os padrões de deslocamento de passageiros observados nesta linha de ônibus. Então, os padrões de embarques e desembarques dos passageiros capturados nas Matrizes OD foram usados nas simulações para: (1) determinar os tempos de viagem entre pontos de paradas do itinerário; (2) determinar os tempos necessários para formar a rede de passa-

geiros mediante algumas possíveis heurísticas de determinação de propensão de desembarque de passageiros.

Observou-se que, nesta linha em particular, informação acerca dos pontos de embarque não é suficiente para determinação do ponto de desembarque, e consequentemente na determinação da propensão de desembarques entre passageiros. Entretanto, ao usar a capacidade do *smartphone* para recuperar o histórico de viagens individuais dos passageiros, é possível que Bluemob apresente melhora nos tempos médios quando comparados a Law. Mas, mesmo nestas situações, há casos onde os pontos de embarque e desembarque estão muito próximos e não é possível finalizar a formação da rede mesmo usando Bluemob em seus melhores cenários. Entretanto, ao adicionar processamento à central é possível reconstruir a matriz Origem-Destino mesmo nestes cenários.

Em suma, este trabalho traz as seguintes contribuições:

1. Algoritmo Bluemob para formação de redes espontâneas *Bluetooth* para uso ambientes pequenos (14 m x 14 m), onde quaisquer dois dispositivos possam estabelecer uma conexão *Bluetooth*), dinâmicos, e com alta volatilidade, que leva em consideração aspectos da aplicação na estruturação de redes *Bluetooth*;

2. Proposta para determinação dos passageiros embarcados entre paradas para determinação dos trajetos individuais de passageiros e informações correlatas (por exemplo, Matriz Origem-Destino) sem necessidade de instalação de equipamentos de infraestrutura nos ônibus ou na cidade;

3. Proposta de uso de redes de comunicação *ad-hoc* no interior do ônibus em sistema *crowdsensing*;

4. Prova de propriedades do Bluemob de redes espontâneas *Bluetooth* obtido.

5. Avaliação de Bluemob por simulação e comparativo com algoritmos encontrado na literatura em cenários fictícios e em um cenário de transporte público baseado na linha Troncal-10 de Blumenau.

Como trabalhos futuros e área de melhora, as seguintes questões podem ser investigadas:

- estender o algoritmo para reagir a mudanças nas estimativas da propensão de um passageiro deixar o ônibus que ocorram após a rede estar formada;

- estender o algoritmo para evoluir a rede em direção a uma distribuição ótima dos nós dentro da *Scatternet*, dada que a abordagem atual não aloca os nós em posições ótimas dentro da *Scatternet*;
- investigar e avaliar métodos para atribuir e comparar a propensão de um passageiro permanecer no ônibus;
- estender o algoritmo para diferenciar entre mestres e pontes durante as reorganizações, dado que o desembarque de mestres tendem a causar mais particionamentos que pontes;
- investigar outras áreas onde o algoritmo pode ser útil, tal como aplicações relacionadas a energia;
- comparar o tempo necessário e o consumo energético do Bluemob com a criação de uma rede *ad-hoc* WiFi. Alternativamente a manutenção de uma rede durante todo o trajeto avalia-se a possibilidade de estabelecer uma rede *ad-hoc* WiFi por poucos segundos apenas para realização da coleta;
- avaliar alternativas de formação de rede *ad-hoc* usando WiFi e comparar o consumo energético destas soluções com Bluemob.

REFERÊNCIAS

ARIAS, C. et al. **Manual de BRT: Guia de Planejamento (Bus Rapid Transit Manual)**. Brasília, DF, Brazil, 2008.

BARCELÓ, J.; CASAS, J. Dynamic network simulation with aimsun. In: **Simulation approaches in transportation analysis**. [S.l.]: Springer, 2005. p. 57–98.

BARRY, J. et al. Origin and destination estimation in new york city with automated fare system data. **Transportation Research Record: Journal of the Transportation Research Board**, v. 1817, n. -1, p. 183–187, jan. 2002.

BASAGNI, S. et al. Comparative performance evaluation of scatternet formation protocols for networks of bluetooth devices. **Wirel. Netw.**, v. 10, n. 2, p. 197–213, mar. 2004. ISSN 1022-0038.

BEN-AKIVA, M.; GARTNER, M.; WILSON, N. Methods to combine different data sources and estimate origin-destination matrices. In: **Proc. 10th International Syposium on Transportation and Traffic Theory**. [S.l.]: Elsevier, 1987. p. 459–481.

BEN-AKIVA, M.; MORIKAWA, T. Data fusion methods and their applications to origin-destination trip tables. In: **Selected Proceedings of The Fifth World Conference on Transport Research: Transport Policy, Management & Technology, Towards 2001**. [S.l.: s.n.], 1989. IV, p. 279–293.

Ben Moshe, B.; HADAS, Y.; LEVI, H. Energy-Efficient Framework for Indoor and Outdoor Tracking of Public Transit Passengers Using Bluetooth-Enabled Devices. In: **Transportation Research Board 93rd Annual Meeting**. Washington DC: [s.n.], 2014. p. 16–30. Disponível em: <http://trid.trb.org/view.aspx?id=1288331>.

Bluetooth SIG. **BLUETOOTH SPECIFICATION Version 4.0**. jun. 2010.

BURKE, J. et al. Participatory sensing. In: **Proc. Workshop on World-Sensor-Web (WSW'06): Mobile Device Centric Sensor Networks and Applications**. [S.l.: s.n.], 2006. p. 117–134.

CACERES, N.; WIDEBERG, J. P.; BENITEZ, F. G. Deriving origin destination data from a mobile phone network. **IET Intelligent Transport Systems**, v. 1, n. 1, p. 15–26, mar. 2007. ISSN 1751-956X.

CALABRESE, F. et al. Real-time urban monitoring using cell phones: A case study in rome. **IEEE Transactions on Intelligent Transportation Systems**, v. 12, n. 1, p. 141–151, mar. 2011. ISSN 1524-9050, 1558-0016.

CAMPBELL, A. T. et al. People-centric urban sensing. In: **Proc. 2nd Annual International Wireless Internet Conference (WICON '06)**. New York, NY, USA: ACM, 2006. p. 2–5. ISBN 1-59593-510-X.

CHEN, X. et al. Short-term forecasting of transit route OD matrix with smart card data. In: **2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)**. [S.l.]: IEEE, 2011. p. 1513–1518. ISBN 978-1-4577-2198-4.

CUI, A. **Bus passenger Origin-Destination Matrix estimation using Automated Data Collection systems**. Tese (Thesis), 2006.

CUOMO, F.; BACCO, G. D.; MELODIA, T. SHAPER: a self-healing algorithm producing multi-hop Bluetooth scatternets. **GLOBECOM '03. IEEE Global Telecommunications Conference (IEEE Cat. No.03CH37489)**, v. 1, p. 236–240, 2003.

DEMING, E.; STEPHAN, F. On a least squares adjustment of a sampled frequency table when the expected marginal totals are known. **Annals of Mathematical Statistics**, v. 11, n. 4, p. 427–444, 1940.

DUBHASHI, D. et al. Blue pleiades, a new solution for device discovery and scatternet formation in multi-hop bluetooth networks. **Wireless Networks**, v. 13, n. 1, p. 107–125, fev. 2007. ISSN 1022-0038, 1572-8196.

FURTH, P. et al. **Using Archived AVL-APC Data to Improve Transit Performance and Management**. Washington, DC, USA, 2006.

GANTI, R.; YE, F.; LEI, H. Mobile crowdsensing: current state and future challenges. **Communications Magazine, IEEE**, v. 49, n. 11, p. 32 –39, nov. 2011. ISSN 0163-6804.

GHOSH, J. et al. **BTSpin - Single Phase Distributed Bluetooth Scatternet Formation**. [S.l.], 2004.

GOHS, R. S.; GUNNARSSON, S. R.; GLENSTRUP, A. J. Beddernet: Application-level platform-agnostic MANETs. In: **Distributed Applications and Interoperable Systems**. [S.l.]: Springer Berlin Heidelberg, 2011. v. 6723, p. 165–178. ISBN 978-3-642-21386-1, 978-3-642-21387-8.

GORDILLO, F. **The value of automated fare collection data for transit planning : an example of rail transit OD matrix estimation**. Tese (Thesis) — Massachusetts Institute of Technology, Massachusetts, 2006. Thesis (S.M.)–Massachusetts Institute of Technology, Engineering Systems Division, Technology and Policy Program; and, (S.M.)–Massachusetts Institute of Technology, Dept. of Civil and Environmental Engineering, 2006.

HUANG, A. S.; RUDOLPH, L. **Bluetooth Essentials for Programmers**. Leiden: Cambridge Univ. Press, 2007.

JEDDA, A. et al. Bluetooth scatternet formation from a time-efficiency perspective. **Wireless Networks**, v. 20, n. 5, p. 1133–1156, nov. 2013. ISSN 1022-0038.

JIUH-BIING, S. A fuzzy clustering approach to real-time demand-responsive bus dispatching control. **Fuzzy Sets and Systems**, v. 150, n. 3, p. 437–455, mar. 2005. ISSN 0165-0114.

JOU, Y.-J. et al. Estimation of dynamic origin-destination by gaussian state space model with unknown transition matrix. In: . [S.l.]: IEEE. v. 1, p. 96–101. ISBN 978-0-7803-7952-7.

KOSTAKOS, V. **Using Bluetooth to capture passenger trips on public transport buses**. [S.l.], jun. 2008.

KOSTAKOS, V.; CAMACHO, T.; MANTERO, C. Wireless detection of end-to-end passenger trips on public transport buses. In: **Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on**. [S.l.: s.n.], 2010. p. 1795–1800. ISSN 2153-0009.

KUMAR, A. **BlueHoc manual**. [S.l.]: IBM Corporation, India Research Lab.

LAMPORT, L. Paxos made simple. **ACM SIGACT News**, v. 32, n. 4, p. 18–25, 2001.

LAW, C.; MEHTA, A. K.; SIU, K. Y. A New Bluetooth Scatternet Formation Protocol. **Mobile Networks and Applications**, v. 8, p. 485–498, 2003. ISSN 1383469X.

LI, D. et al. Estimating a transit passenger trip origin-destination matrix using automatic fare collection system. In: HUTCHISON, D. et al. (Ed.). **Database Systems for Adanced Applications**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. v. 6637, p. 502–513. ISBN 978-3-642-20243-8, 978-3-642-20244-5.

LIN, C.-Y. et al. A comfort measuring system for public transportation systems using participatory phone sensing. In: **Proc. of First International Workshop on Sensing for App Phones (PhoneSense'10)**. Zurich, Switzerland: [s.n.], 2010.

MADANAT, S.; HU, S.-R.; KROGMEIER, J. Dynamic estimation and prediction of freeway o-d matrices with route switching considerations and time-dependent model parameters. **Transportation Research Record: Journal of the Transportation Research Board**, v. 1537, n. -1, p. 98–105, jan. 1996.

McCord, M. et al. Iterative proportional fitting procedure to determine bus route passenger origin-destination flows. **Journal of the Transportation Research Board**, v. 2145, n. -1, p. 59–65, dez. 2010.

MOHAN, P.; PADMANABHAN, V. N.; RAMJEE, R. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In: **Proc. 6th ACM Conference on Embedded Networked Sensor Systems (SenSys '08)**. New York, NY, USA: [s.n.], 2008. p. 323–336. ISBN 978-1-59593-990-6.

Moovit. **Moovit**. 2014. Http://www.moovitapp.com/.

PARIKH, P.; KANABAR, M.; SIDHU, T. Opportunities and challenges of wireless communication technologies for smart grid applications. In: **2010 IEEE Power and Energy Society General Meeting**. [S.l.: s.n.], 2010. p. 1–7.

PEI, L. et al. Using inquiry-based bluetooth rssi probability distributions for indoor positioning. **Journal of Global Positioning Systems**, v. 9, n. 2, p. 122–130, 2010.

PETRIOLI, C.; BASAGNI, S. Degree-constrained multihop scatternet formation for Bluetooth networks. In: **IEEE Global Telecommunications Conference, 2002. GLOBECOM '02**. [S.l.: s.n.], 2002. v. 1, p. 222–226 vol.1.

PETRIOLI, C.; BASAGNI, S.; CHLAMTAC, I. Configuring bluestars: multihop scatternet formation for bluetooth networks. **IEEE Transactions on Computers**, v. 52, n. 6, p. 779–790, jun. 2003. ISSN 0018-9340.

PETRIOLI, C.; BASAGNI, S.; CHLAMTAC, I. BlueMesh: Degree-Constrained Multi-Hop Scatternet Formation for Bluetooth Networks. **Mobile Networks and Applications**, v. 9, p. 33–47, 2004. ISSN 1383469X.

REDDY, S. et al. Using mobile phones to determine transportation modes. **ACM Trans. Sen. Netw.**, v. 6, n. 2, p. 13:1–13:27, mar. 2010. ISSN 1550-4859.

SALONIDIS, T. et al. Distributed topology construction of Bluetooth wireless personal area networks. **IEEE Journal on Selected Areas in Communications**, v. 23, p. 633–643, 2005. ISSN 07338716.

SATO, T.; MASE, K. A scatternet operation protocol for bluetooth ad hoc networks. In: **The 5th International Symposium on Wireless Personal Multimedia Communications, 2002**. [S.l.: s.n.], 2002. v. 1, p. 223–227 vol.1.

SHAFIQUE, M. A.; HATO, E. Modelling of Accelerometer Data for Travel Mode Detection by Hierarchical Application of Binomial Logistic Regression. **Transportation Research Procedia**, v. 10, p. 236–244, 2015. ISSN 2352-1465.

SONG, C. et al. Limits of Predictability in Human Mobility. **Science**, v. 327, n. 5968, p. 1018–1021, fev. 2010. ISSN 0036-8075, 1095-9203.

STOJMENOVIC, I.; ZAGUIA, N. Bluetooth scatternet formation in ad hoc wireless networks. **Performance Modeling and Analysis of Bluetooth Networks**, Auerbach: Philadelphia, PA, p. 147–171, 2006.

SUGIYAMA, Y. et al. An approach for real-time estimation of railway passenger flow. v. 51, n. 2, p. 82–88. ISSN 0033-9008, 1880-1765. Disponível em: <http://joi.jlc.jst.go.jp/JST.JSTAGE/rtriqr/51.82?from=CrossRef>.

SUN, D.-h.; FU, Q.-s.; LI, Y.-f. Forecasting of public traffic passenger volume based on quantum neural network. v. 2, p. 021.

TAN, G. et al. An efficient scatternet formation algorithm for dynamic environments. In: **Proceedings of the IASTED Communications and Computer Networks (CCN)**. [S.l.]: Citeseer, 2002. v. 1.

The VINT Project. **The ns Manual**. Http://www.isi.edu/nsnam/ns/. Accessed: 2008-11-18.

THIAGARAJAN, A. et al. Cooperative transit tracking using smart-phones. In: **Proc. 8th ACM Conference on Embedded Networked Sensor Systems (SenSys '10)**. New York, NY, USA: [s.n.], 2010. p. 85–98. ISBN 978-1-4503-0344-6.

THIAGARAJAN, A. et al. VTrack: accurate, energy-aware road traffic delay estimation using mobile phones. In: **Proc. 7th ACM Conference on Embedded Networked Sensor Systems (SenSys '09)**. New York, NY, USA: [s.n.], 2009. p. 85–98. ISBN 978-1-60558-519-2.

WANG, Q.; AGRAWAL, D. **UCBT- Bluetooth extension for NS2 at the University of Cincinnati**. Http://www.ececs.uc.edu/ cdmc/ucbt/. Accessed: 2014-03-30.

WANG, Z. W. Z.; THOMAS, R.; HAAS, Z. Bluenet - a new scatternet formation scheme. **Proceedings of the 35th Annual Hawaii International Conference on System Sciences**, v. 00, n. c, p. 1–9, 2002.

Waze Mobile. **Waze**. 2014. Https://www.waze.com/.

Wei Wang; John P. Attanucci; Nigel H.M. Wilson. Bus passenger origin-destination estimation and related analyses using automated data collection systems. **Journal of Public Transportation**, v. 14, 2011.

YAO, X.-m.; ZHAO, P.; YU, D.-d. Real-time origin-destination matrices estimation for urban rail transit network based on structural state-space model. v. 22, n. 11, p. 4498–4506. ISSN 2095-2899, 2227-5223. Disponível em: <http://link.springer.com/10.1007/s11771-015-2998-4>.

ZAGUIA, N.; DAADAA, Y.; STOJMENOVIC, I. Simplified bluetooth scatternet formation using maximal independent sets. **Proceedings - CISIS 2008: 2nd International Conference on Complex, Intelligent and Software Intensive Systems**, p. 443–448, 2008. ISSN 1069-2509.

ZHAO, J.; RAHBEE, A.; WILSON, N. H. M. Estimating a rail passenger trip origin–destination matrix using automatic data collection systems. **Computer Aided Civil and Infrastructure Engineering**, v. 22, n. 5, p. 376–387, jul. 2007. ISSN 1467-8667.

APÊNDICE A – Provas de corretude do algoritmo

Conforme descrito no capítulo 4, Bluemob forma *Scatternets* pela união de dois componentes, gerando um novo componente que respeita as seguintes seguintes propriedades:

(1)**Liderança única** - o componente possui um único líder;

(2)**Limite superior de escravos do líder** - o líder deve possuir no máximo seis escravos;

(3)**Limite superior de escravos** - os mestres devem possuir menos que sete escravos;

(4)**Disponibilidade para busca** - o líder deve possuir um escravo não-compartilhado, ou não possuir nenhum escravo;

(5)**Propriedade organizacional** - todo escravo não-compartilhado possui peso h menor ou igual ao peso h de seu mestre.

Ao assegurar que todos os componentes respeitam estas propriedades, a corretude e vivacidade de Bluemob são garantidas. No capítulo 4 já foram apresentados os lemas 1-5, e suas respectivas provas. Estes lemas demonstram que o resultado da união de dois componentes válidos no CASO 1 é um novo componente válido. Neste anexo, apresentamos as provas de que o resultado da união de dois componentes válidos por Bluemob respeitam estas propriedades também quando unidos nos CASOS 2 a 13.

Através dos lemas 1-5 e os demais lemas apresentados neste anexo, demonstra-se a validade do lema 67 (enunciado no capítulo 4, e demonstrado neste anexo) e, consequentemente, do teorema 1 demonstrado no capítulo 4.

A.1 CASO 2: $W = V \wedge |S(U)| < K \wedge H(V) > H(U)$

Lema 7. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 2, após a união (MERGE) o componente resultante da união possui um líder.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe mais de um líder ou não existe nenhum líder. Na linha 5 do algoritmo 3, u, líder do componente A (linha 17 do algoritmo 2), é aposentado. Antes da união, o componente A respeita todas as propriedades, e segundo a propriedade 1 pode-se concluir que após a finalização da reorganização, nenhum nó que pertencia a A é líder. Dado que: (1) não há nenhum líder pertencente a A; (2) a reorganização 2 não torna nenhum nó líder; e, (3) nenhum nó pertencente a B é aposentado; todos líderes de B são líderes do

Algoritmo 4 Algoritmo para seleção do caso de reorganização

1: **procedure** REORGANIZE(u, w, v, y)
2: **case** $w = v \wedge |S(u)| < K \wedge h(v) \leq h(u)$
3: CASE 1(u,w,v,y)
4: **case** $w = v \wedge |S(u)| < K \wedge h(v) > h(u)$
5: CASE 2(u,w,v,y)
6: **case** $w = v \wedge |S(u)| = K \wedge h(u) \geq h(v) \wedge h(y) \leq h(v)$
7: CASE 3(u,w,v,y)
8: **case** $w = v \wedge |S(u)| = K \wedge h(u) \geq h(v) \wedge h(y) > h(v)$
9: CASE 4(u,w,v,y)
10: **case** $w = v \wedge |S(u)| = K \wedge h(u) < h(v)$
11: CASE 5(u,w,v,y)
12: **case** $w \neq v \wedge |S(w)| + |S(u)| < K \wedge h(u) \geq h(w)$
13: CASE 6(u,w,v,y)
14: **case** $w \neq v \wedge |S(w)| + |S(u)| < K \wedge h(u) < h(w)$
15: CASE 7(u,w,v,y)
16: **case** $w \neq v \wedge |S(w)| + |S(u)| = K \wedge |S(u)| = 1 \wedge h(v) \geq h(u)$
17: CASE 8(u,w,v,y)
18: **case** $w \neq v \wedge |S(w)| + |S(u)| = K \wedge |S(u)| = 1 \wedge h(v) < h(u)$
19: CASE 9(u,w,v,y)
20: **case** $w \neq v \wedge |S(w)| + |S(u)| = K \wedge |S(u)| > 1 \wedge h(v) \geq h(u)$
21: CASE 10(u,w,v,y)
22: **case** $w \neq v \wedge |S(w)| + |S(u)| = K \wedge |S(u)| > 1 \wedge h(v) < h(u)$
23: CASE 11(u,w,v,y)
24: **case** $w \neq v \wedge |S(w)| + |S(u)| > K \wedge h(w) \geq h(u)$
25: CASE 12(u,w,v,y)
26: **case** $w \neq v \wedge |S(w)| + |S(u)| > K \wedge h(w) < h(u)$
27: CASE 13(u,w,v,y)
28: **end procedure**

componente após união. Como nenhum nó pertencente ao componente A é líder, todos os líderes do componente após a união são líderes do componente B. Logo, antes da reorganização o componente B não possui nenhum líder ou possui pelo menos 2 líderes. Isto é uma contradição, pois sabe-se devido a propriedade 1 que o componente B possui exatamente 1 líder. Conclui-se que a premissa é verdadeira e existe um líder após o término da reorganização. □

Lema 8. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 2, após a união (MERGE) o líder possuirá no máximo* $K-1=6$ *escravos.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união o líder possui mais de $K-1=6$ escravos. Sabe-se que, segundo lema 1 após a reorganização há somente um líder e que w não se aposentou. Portanto o líder do componente após a reorganização é w e $S(w) > K-1$. A ocorrência da reorganização CASO 2 exige que as condições da linha 4 do algoritmo 4 sejam satisfeitas, portanto $|S(u)| < K$ (equivalente a $|S(u)| \leq K-$

1) e que $v = w$. $v = w$ implica que w, líder do componente B, estava no estado de INQUIRY SCAN. Isto ocorre apenas se o componente B for composto por um único nó (linha 9 do algoritmo 2). Logo, antes da reorganização $S(w) = \emptyset$. Na linha 8 do algoritmo 2 $(S(u) - \{w\}) \cup \{u\}$ são feitos escravos de w. Sabe-se que a $|S(u)| \leq K - 1$ e $w \in S(u)$. Logo após a reorganização $|S(w)| = |m| - |\{w\}| + |\{u\}| = |S(w)| \leq K - 1$. Isto é uma contradição. Portanto, a premissa é verdadeira. □

Lema 9. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 2, não existe mestre com mais de $K = 7$ escravos.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre m com mais de $K = 7$ escravos. Sabe-se que $m \neq w$, pois segundo o lema 8 $|S(w)| \leq K - 1 \implies |S(w)| \leq K \implies m \neq w$.

A ocorrência da reorganização CASO 2 exige que as condições da linha 4 do algoritmo 4 sejam satisfeitas, portanto $|S(u)| < K$ e $v = w$. $v = w$ implica que w, líder do componente B, estava no estado de INQUIRY SCAN. Isto ocorre apenas se o componente B for composto por um único nó (linha 9 do algoritmo 2). Portanto, não há nenhum mestre em B. Como m é um mestre, m não estava em B antes da reorganização. Na linha 7 do algoritmo 3 a conexão entre u e seus escravos foi desfeita, logo u deixou de ser um mestre e $m \neq u$. Nenhum escravo foi desconectado ou novos escravos adicionados a nós diferentes de u e w durante a reorganização. Portanto, o conjunto de escravos de qualquer nó diferente de u e w é igual antes e depois da reorganização. Sabe-se que m não pertencia a B antes da reorganização, logo m pertence ao componente A. Como, o componente A respeita a propriedade 3, todo mestre em A possui menos de K escravos. Dado que m pertence a A, $S(m) < k$. Isto é uma contradição. Portanto a premissa é verdadeira. □

Lema 10. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 2, após a união (MERGE) o líder possui um escravo não compartilhado ou não possui nenhum escravo.*

*Demonstração.*Na linha 5 do algoritmo 3 u foi aposentado e w, líder de B (linhas 17 do algoritmo 2), não. Logo, o líder do componente após a união ser concluída é w.

Há dois casos: (1) u possui pelo menos um escravo, (2) u não possui escravos.

Assumindo que (1) seja verdadeiro. Sabe-se que antes da união dos componentes u é líder de A. Dado que a propriedade 4 é válida para o componente 2, um dos escravos de u é um escravo não-compartilhado. Na linha 7 e 8 do algoritmo 3, os escravos de u são desconectados e tornam-se escravos

de w. Portanto, o escravo não compartilhado de u se torna um escravo não compartilhado de w. Como o líder w possui um escravo não compartilhado, a proposição é verdadeira.

Assumindo que (2) seja verdadeiro. Sabe-se que antes de realizar a união, u é líder de A. Dado que A respeita a propriedade 4 antes da união, sabe-se que antes da união $M(u) = \emptyset$. Logo, A possui apenas um nó: u. Na linha 8 do algoritmo 3, u se torna escravo de w. Por definição, u é um escravo não compartilhado pois $S(u) = \emptyset \wedge M(u) = \{w\}$). Como o líder w possui um escravo não compartilhado, a proposição é verdadeira.

Dado que a proposição é verdadeira nos casos (1) e (2), conclui-se que a proposição é verdadeira. □

Lema 11. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 2, após a união (MERGE) todos os escravos não-compartilhados possuem h menor ou igual ao h de seu mestre.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre k com um escravo não compartilhado l tal que $h(l) > h(k)$.

A ocorrência da reorganização CASO 2 exige que as condições da linha 4 do algoritmo 4 sejam satisfeitas, portanto $h(v) > h(u)$.

Após a linha 8 do algoritmo 3 u se torna escravo de w. Logo $h(w) > h(u) \implies u \neq k$.

Dado que antes da união A e B respeitam a propriedade 5, os escravos não compartilhados de u possuem h menor ou igual a $h(u)$. Na linha 7 do algoritmo 2 todos os escravos de u são desconectados e reconectados a w (linha 8 do algoritmo 3). Como $h(w) > h(u)$, por transitividade, sabemos que $h(w)$ é maior que o h de qualquer escravo não compartilhado anteriormente de u. Portanto, $w \neq k$.

Para mestres diferentes de u e w: nenhum novo escravo lhes foi atribuído. Portanto um mestre diferente de u e w possui um escravo se, e somente se, ele possui um escravo antes da união. Dado que $k \neq u \wedge k \neq w$, antes da união k foi mestre de l em A ou B. Dado que A e B respeitam a propriedade 5, $h(k) \geq h(l)$.

Isto é uma contradição, e portanto a premissa é verdadeira. □

A.2 CASO 3: $W = V \wedge |S(U)| = K \wedge H(U) \geq H(V) \wedge H(Y) \leq H(V)$

Lema 12. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 3, após a união (MERGE)*

o componente resultante da união possui um líder.

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe mais de um líder ou não existe nenhum líder. Na linha 11 do algoritmo 3, u, líder do componente A (linha 17 do algoritmo 2), é aposentado. Antes da união, o componente A respeita todas as propriedades, e segundo a propriedade 1 pode-se concluir que após a finalização da reorganização, nenhum nó que pertencia a A é líder. Dado que: (1) não há nenhum líder pertencente a A; (2) a reorganização 2 não torna nenhum nó líder; e, (3) nenhum nó pertencente a B é aposentado; todos líderes de B são líderes do componente após união. Como nenhum nó pertencente ao componente A é líder, todos os líderes do componente após a união são líderes do componente B. Logo, antes da reorganização o componente B não possui nenhum líder ou possui pelo menos 2 líderes. Isto é uma contradição, pois sabe-se devido a propriedade 1 que o componente B possui exatamente 1 líder. Conclui-se que a premissa é verdadeira e existe um líder após o término da reorganização. □

Lema 13. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 1, após a união (MERGE) o líder possuirá no máximo* $K-1=6$ *escravos.*

*Demonstração.*A ocorrência da reorganização CASO 3 exige que as condições da linha 6 do algoritmo 4 sejam satisfeitas, portanto $v=w$. $v=w$ implica que w, líder do componente B, estava no estado de INQUIRY SCAN. Isto ocorre apenas se o componente B for composto por um único nó (linha 9 do algoritmo 2). Logo, $S(w)=\emptyset$. Na linha 13 do algoritmo 2 um escravo, y, é adicionado a w. Portanto, $S(w)=\{y\} \implies |S(w)|=1 \leq K-1$. □

Lema 14. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 3, não existe mestre com mais de* $K=7$ *escravos.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre m com mais de $K=7$ escravos. Sabe-se que $m \neq w$, pois segundo o lema 13 $|S(w)| \leq K-1 \implies |S(w)| \leq K \implies m \neq w$.

Antes da reorganização, componente A respeita a propriedade 2, ou seja, $|S(u)| \leq K-1$. Quando a conexão entre w e u é estabelecida (linha 16 no algoritmo 2) $|S(u)| \leq K$. Dado que nenhum novo escravo é adicionado a u durante a reorganização, após a reorganização o número de escravos de u permanece o mesmo. Logo, $|S(u)| \leq K \implies m \neq u$

A ocorrência da reorganização CASO 3 exige que as condições da linha 6 do algoritmo 4 sejam satisfeitas, portanto $v=w$ e $|S(u)|=K$.

Dado que $|S(u)| = K$ e u foi um líder no componente A, e A respeita a propriedade 4 sabe-se que se o líder possui escravos, então $Y_p(u) \neq \emptyset$ e y existe. Portanto, na linha 12 do algoritmo 3 y é desconectado de u e se torna um escravo de w. Como y é um escravo não compartilhado $m \neq y$.

$v = w$ implica que w, líder do componente B, estava no estado de INQUIRY SCAN. Isto ocorre apenas se o componente B for composto por um único nó (linha 9 do algoritmo 2). Portanto, como $m \neq w$, e w é o único nó em B, m não pertence a B. m pertence a A antes da união.

Para os outros mestres diferentes de u, w e y, nenhuma conexão é estabelecida ou destruída durante a reorganização. Portanto nós diferentes de u, y e w possuem um escravo após a reorganização se, e somente se, ele possui o escravo antes da reorganização. Como $m \notin \{u, w, y\}$ e depois da reorganização $|S(m)| > K$, então $|S(m)| > K$ antes da união $|S(m)| > K$. Dado que o componente A respeita a propriedade 3, antes da reorganização todos os mestres em A possuem no máximo K escravos. Como m estava em A antes da união, então $|S(m)| < K$.

Isto é uma contradição. Portanto a premissa é verdadeira. □

Lema 15. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 3, após a união (MERGE) o líder possui um escravo não compartilhado ou não possui nenhum escravo.*

*Demonstração.*Na linha 11 do algoritmo 3 u é aposentado e w, líder do componente B (linha 17 do algoritmo 2), não se aposenta. Portanto, após a conclusão da união o líder do componente é w.

A ocorrência da reorganização CASO 3 exige que as condições da linha 6 do algoritmo 4 sejam satisfeitas, portanto $v = w$ e $|S(u)| = K$. Dado que $|S(u)| = K$ e u foi um líder no componente A, e A respeita a propriedade 4 sabe-se que se o líder possui escravos, então $Y_p(u) \neq \emptyset$ e y existe. Portanto, na linha 12 do algoritmo 3 y é desconectado de u e se torna um escravo de w. Portanto, w possui pelo menos um escravo não compartilhado, y. □

Lema 16. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 3, após a união (MERGE) todos os escravos não-compartilhados possuem h menor ou igual ao h de seu mestre.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre k com um escravo não compartilhado l tal que $h(l) > h(k)$.

A ocorrência da reorganização CASO 3 exige que as condições da linha 6 do algoritmo 3 sejam satisfeitas, portanto $h(u) \geq h(w)$ e $h(y) \leq h(w)$.

Após a linha 13 do algoritmo 3 y se torna escravo de w. Logo $h(w) > h(u) \implies w \neq k$.

Dado que antes da união A e B respeitam a propriedade 5, os escravos não compartilhados de u possuem h menor ou igual a $h(u)$. $u \neq k$ pois os escravos não compartilhados de u após a reorganização diferentes de w já eram escravos de u antes da reorganização. E w é mestre de y após a reorganização, logo w não é um escravo não-compartilhado e $h(w)$ não é relevantes para u.

Para mestres diferentes de u e w: nenhum novo escravo lhes foi atribuído. Portanto um mestre diferente de u e w possui um escravo se, e somente se, ele possui um escravo antes da união. Dado que $k \neq u \wedge k \neq w$, antes da união k foi mestre de l em A ou B. Dado que A e B respeitam a propriedade 5, $h(k) \geq h(l)$.

Isto é uma contradição, e portanto a premissa é verdadeira. □

A.3 CASO 4: $W = V \wedge |S(U)| = K \wedge H(U) \geq H(V) \wedge H(Y) > H(V)$

Lema 17. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 4, após a união (MERGE) o componente resultante da união possui um líder.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe mais de um líder ou não existe nenhum líder.

Nas linhas 16 e 17 do algoritmo 3 são aposentados u e w, líderes dos componentes A (linha 17 do algoritmo 2) e B (linha 18 do algoritmo 2) respectivamente. Na linha 18 y é declarado um líder.

Antes da união, o componente A respeita todas as propriedades, e segundo a propriedade 1 u é o único líder em A. Portanto, dado que u foi aposentado, após a finalização da reorganização, nenhum nó diferente de y que pertence a A é líder.

Sabe que deve existir mais de um líder além de y. Como este líder não pertence a A, ele pertence a B antes da reorganização. Antes da reorganização, o componente B respeita todas as propriedades, e segundo a propriedade 1 o único líder em B é w, que foi aposentado. Portanto, nenhum nó que pertencia a B é líder após a reorganização ser concluída.

Isto é uma contradição. Conclui-se que a premissa é verdadeira e existe um líder após o término da reorganização. □

Lema 18. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 4, após a união (MERGE) o líder possuirá no máximo $K - 1 = 6$ escravos.*

*Demonstração.*Nas linhas 16 e 17 do algoritmo 3, u, líder do componente

A (linha 17 do algoritmo 2), e w, líder do componente B (linha 18 do algoritmo 2), são aposentados. E na linha 18 do algoritmo 3, y se torna o líder do novo componente.

É sabido que y é um escravo não compartilhado de u antes da reorganização. Portanto, antes da reorganização, $S(y) = \emptyset$. Na linha 20 w se torna escravo de y, e $|S(y)| = 1$.

w é o único escravo adicionado a y durante a reorganização. Portanto após a reorganização ser concluída, $|S(y)| = 1 \implies |S(y)| <= K - 1$. □

Lema 19. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 4, não existe mestre com mais de $K = 7$ escravos.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre m com mais de $K = 7$ escravos. Sabe-se que $m \neq y$, pois segundo o lema 18 $|S(y)| \leq K - 1 \implies |S(y)| \leq K \implies m \neq y$.

Antes da reorganização, componente A respeita a propriedade 2, ou seja, $|S(u)| \leq K - 1$. Quando a conexão entre w e u é estabelecida (linha 16 no algoritmo 2) $|S(u)| \leq K$. Dado que nenhum novo escravo é adicionado a u durante a reorganização, após a reorganização o número de escravos de u permanece o mesmo. Logo, $|S(u)| \leq K \implies m \neq u$

A ocorrência da reorganização CASO 4 exige que as condições da linha 8 do algoritmo 4 sejam satisfeitas, portanto $v = w$. $v = w$ implica que w, líder do componente B, estava no estado de INQUIRY SCAN. Isto ocorre apenas se o componente B for composto por um único nó (linha 9 do algoritmo 2). Portanto, como $m \neq w$, e w é o único nó em B, m não pertence a B. m pertence a A antes da união.

Para os outros mestres diferentes de u, w e y, nenhuma conexão é estabelecida ou destruída durante a reorganização. Portanto nós diferentes de u, y e w possuem um escravo após a reorganização se, e somente se, ele possui o escravo antes da reorganização. Como $m \notin \{u, w, y\}$ e depois da reorganização $|S(m)| > K$, então $|S(m)| > K$ antes da união $|S(m)| > K$. Dado que o componente A respeita a propriedade 3, antes da reorganização todos os mestres em A possuem no máximo K escravos. Como m estava em A antes da união, então $|S(m)| < K$.

Isto é uma contradição. Portanto a premissa é verdadeira. □

Lema 20. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 4, após a união (MERGE) o líder possui um escravo não compartilhado ou não possui nenhum escravo.*

*Demonstração.*Nas linhas 16 e 17 do algoritmo 3, u, líder do componente A (linha 17 do algoritmo 2), e w, líder do componente B (linha 18 do algo-

ritmo 2), são aposentados. E na linha 18 do algoritmo 3, y se torna o líder do novo componente.

A ocorrência da reorganização CASO 4 exige que as condições da linha 8 do algoritmo 4 sejam satisfeitas, portanto $v = w$. $v = w$ implica que w, líder do componente B, estava no estado de INQUIRY SCAN. Isto ocorre apenas se o componente B for composto por um único nó (linha 9 do algoritmo 2). Portanto, antes da reorganização, $S(w) = \emptyset$ e $M(w) = \emptyset$.

Na linha 19 w e u são desconectados, e na linha 20 w set torna escravo do líder y. Logo, após a reorganização $S(w) = \emptyset$ e $M(w) = \{y\}$. Ou seja, w é um escravo não compartilhado de y.

Dado que y é o novo líder, após terminada a reorganização, o líder y possui pelo menos um escravo não compartilhado: w. □

Lema 21. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 4, após a união (MERGE) todos os escravos não-compartilhados possuem h menor ou igual ao h de seu mestre.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre k com um escravo não compartilhado l tal que $h(l) > h(k)$.

A ocorrência da reorganização CASO 4 exige que as condições da linha 8 do algoritmo 4 sejam satisfeitas, portanto $h(u) \geq h(w)$ e $h(y) > h(w)$.

Após a linha 20 do algoritmo 3 y se torna mestre de w. Logo $h(y) > h(w) \implies y \neq k$.

Dado que antes da união A e B respeitam a propriedade 5, os escravos não compartilhados de u possuem h menor ou igual a $h(u)$. $u \neq k$ pois os escravos não compartilhados de u após a reorganização já eram escravos de u antes da reorganização.

Para mestres diferentes de u e y: nenhum novo escravo lhes foi atribuído. Portanto um mestre diferente de u e w possui um escravo se, e somente se, ele possui um escravo antes da união. Dado que $k \neq u \wedge k \neq y$, antes da união k foi mestre de l em A ou B. Dado que A e B respeitam a propriedade 5, $h(k) \geq h(l)$.

Isto é uma contradição, e portanto a premissa é verdadeira. □

A.4 CASO 5: $W = V \wedge |S(U)| = K \wedge H(U) < H(V)$

Lema 22. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 5, após a união (MERGE) o componente resultante da união possui um líder.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe mais de um líder ou não existe nenhum líder. Na linha 23 do algoritmo 3, u, líder do componente A (linha 17 do algoritmo 2), é aposentado. Antes da união, o componente A respeita todas as propriedades, e segundo a propriedade 1 pode-se concluir que após a finalização da reorganização, nenhum nó que pertencia a A é líder. Dado que: (1) não há nenhum líder pertencente a A; (2) a reorganização 2 não torna nenhum nó líder; e, (3) nenhum nó pertencente a B é aposentado; todos líderes de B são líderes do componente após união. Como nenhum nó pertencente ao componente A é líder, todos os líderes do componente após a união são líderes do componente B. Logo, antes da reorganização o componente B não possui nenhum líder ou possui pelo menos 2 líderes. Isto é uma contradição, pois sabe-se devido a propriedade 1 que o componente B possui exatamente 1 líder. Conclui-se que a premissa é verdadeira e existe um líder após o término da reorganização. □

Lema 23. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 5, após a união (MERGE) o líder possuirá no máximo* $K-1=6$ *escravos.*

*Demonstração.*A ocorrência da reorganização CASO 5 exige que as condições da linha 10 do algoritmo 4 sejam satisfeitas, portanto $v=w$. $v=w$ implica que w, líder do componente B, estava no estado de INQUIRY SCAN. Isto ocorre apenas se o componente B for composto por um único nó (linha 9 do algoritmo 2). Logo, $S(w)=\emptyset$. Na linha 25 do algoritmo 2 um escravo, y, é adicionado a w. Portanto, $S(w)=\{y\} \implies |S(w)|=1\leq K-1$. □

Lema 24. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 5, não existe mestre com mais de* $K=7$ *escravos.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre m com mais de $K=7$ escravos. Sabe-se que $m\neq w$, pois segundo o lema 23 $|S(w)|\leq K-1 \implies |S(w)|\leq K \implies m\neq w$.

Antes da reorganização, componente A respeita a propriedade 2, ou seja, $|S(u)|\leq K-1$. Quando a conexão entre w e u é estabelecida (linha 16 no algoritmo 2) $|S(u)|\leq K$. Dado que nenhum novo escravo é adicionado a u durante a reorganização, após a reorganização o número de escravos de u permanece o mesmo. Logo, $|S(u)|\leq K \implies m\neq u$

A ocorrência da reorganização CASO 3 exige que as condições da linha 10 do algoritmo 4 sejam satisfeitas, portanto $v=w$ e $|S(u)|=K$.

Dado que $|S(u)|=K$ e u foi um líder no componente A, e A respeita a propriedade 4 sabe-se que se o líder possui escravos, então $Y_p(u)\neq\emptyset$ e y

existe. Portanto, na linha 24 do algoritmo 3 y é desconectado de u e se torna um escravo de w. Como y é um escravo não compartilhado $m \neq y$.

$v = w$ implica que w, líder do componente B, estava no estado de INQUIRY SCAN. Isto ocorre apenas se o componente B for composto por um único nó (linha 9 do algoritmo 2). Portanto, como $m \neq w$, e w é o único nó em B, m não pertence a B. m pertence a A antes da união.

Para os outros mestres diferentes de u, w e y, nenhuma conexão é estabelecida ou destruída durante a reorganização. Portanto nós diferentes de u, y e w possuem um escravo após a reorganização se, e somente se, ele possui o escravo antes da reorganização. Como $m \notin \{u,w,y\}$ e depois da reorganização $|S(m)| > K$, então $|S(m)| > K$ antes da união $|S(m)| > K$. Dado que o componente A respeita a propriedade 3, antes da reorganização todos os mestres em A possuem no máximo K escravos. Como m estava em A antes da união, então $|S(m)| < K$.

Isto é uma contradição. Portanto a premissa é verdadeira. □

Lema 25. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 5, após a união (MERGE) o líder possui um escravo não compartilhado ou não possui nenhum escravo.*

*Demonstração.*Na linha 23 do algoritmo 3 u é aposentado e w, líder do componente B (linha 17 do algoritmo 2), não se aposenta. Portanto, após a conclusão da união o líder do componente é w.

A ocorrência da reorganização CASO 5 exige que as condições da linha 10 do algoritmo 4 sejam satisfeitas, portanto $v = w$ e $|S(u)| = K$. Dado que $|S(u)| = K$ e u foi um líder no componente A, e A respeita a propriedade 4 sabe-se que se o líder possui escravos, então $Y_p(u) \neq \emptyset$ e y existe. Portanto, na linha 24 do algoritmo 3 y é desconectado de u e se torna um escravo de w. Portanto, w possui pelo menos um escravo não compartilhado, y. □

Lema 26. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 5, após a união (MERGE) todos os escravos não-compartilhados possuem h menor ou igual ao h de seu mestre.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre k com um escravo não compartilhado l tal que $h(l) > h(k)$.

A ocorrência da reorganização CASO 5 exige que as condições da linha 10 do algoritmo 4 sejam satisfeitas, portanto $h(u) < h(w)$. Após a linha 25 do algoritmo 3 y se torna escravo de w. Sabe-se que u é mestre de y no componente A antes da reorganização. Dado que o componente A respeita a

propriedade 5 antes da reorganização $h(y) \leq h(u)$. Logo, $h(u) \leq h(w)$ implica que $h(y) \leq h(u) \leq h(w) \implies h(y) \leq h(w) \implies w \neq k$.

Dado que antes da união A e B respeitam a propriedade 5, os escravos não compartilhados de u possuem h menor ou igual a $h(u)$. $u \neq k$ pois os escravos não compartilhados de u após a reorganização diferentes de w já eram escravos de u antes da reorganização. E w é mestre de y após a reorganização, logo w não é um escravo não-compartilhado de u e $h(w)$ não é relevantes para u.

Para mestres diferentes de u e w: nenhum novo escravo lhes foi atribuído. Portanto um mestre diferente de u e w possui um escravo se, e somente se, ele possui um escravo antes da união. Dado que $k \neq u \wedge k \neq w$, antes da união k foi mestre de l em A ou B. Dado que A e B respeitam a propriedade 5, $h(k) \geq h(l)$.

Isto é uma contradição, e portanto a premissa é verdadeira. □

A.5 CASO 6: $W \neq V \wedge |S(W)| + |S(U)| < K \wedge H(U) \geq H(W)$

Lema 27. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 6, após a união (MERGE) o componente resultante da união possui um líder.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe mais de um líder ou não existe nenhum líder.

Na linha 28 do algoritmo 3, w, líder do componente B (linha 18 do algoritmo 2), é aposentado. Antes da união, o componente B respeita todas as propriedades, e segundo a propriedade 1 pode-se concluir que após a finalização da reorganização, nenhum nó que pertencia a B é líder. Dado que: (1) não há nenhum líder pertencente a B; (2) a reorganização 1 não torna nenhum nó líder; e, (3) nenhum nó pertencente a A é aposentado; todos líderes de A são líderes do componente após união. Como nenhum nó pertencente ao componente B é líder, todos os líderes do componente após a união são líderes do componente A. Logo, antes da reorganização o componente A não possui nenhum líder ou possui pelo menos 2 líderes. Isto é uma contradição, pois sabe-se devido a propriedade 1 que o componente A possui exatamente 1 líder. Conclui-se que a premissa é verdadeira e existe um líder após o término da reorganização. □

Lema 28. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 6, após a união (MERGE) o líder possuirá no máximo $K - 1 = 6$ escravos.*

*Demonstração.*A ocorrência da reorganização CASO 6 exige que as condi-

ções da linha 12 do algoritmo 4 sejam satisfeitas, portanto $|S(w)| + |S(u)| < K$.

Na linha 30 do algoritmo 3, todos os escravos de w são desconectados dele. Em seguida, na linha 31 do algoritmo 3, todos os escravos que pertenciam a w, inclusive w são conectados a u. Dado que $S(w) \cap S(u) = \{v\}$ quando $v \neq w$, v não é reconectado na linha 31 pois esta conexão já existe.

Considerando $S'(x)$ o conjunto de escravos de x na linha 12 do algoritmo 4. E $S(x)$ o conjunto de escravos de x após a conclusão da reorganização.

Logo:

$$\begin{aligned}
|S(u)| &= |(S'(u) \cup S'(w)) \cup \{w\}| \\
&= |S'(u) \cup S'(w)| + |\{w\}| - |(S'(u) \cup S'(w)) \cap \{w\}| \\
&= |S'(u)| + |S'(w)| - |S'(u) \cap S'(w)| + 1 - |\emptyset| \\
&= |S'(u)| + |S'(w)| - 1 + 1 \\
&= |S'(u)| + |S'(w)| \\
&< K
\end{aligned}$$

Logo, $|S(u)| < K \implies |S(u)| \leq K - 1$. □

Lema 29. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 6, não existe mestre com mais de $K = 7$ escravos.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre m com mais de $K = 7$ escravos. Sabe-se que $m \neq u$, pois segundo o lema 28 $|S(u)| \leq K - 1 \implies |S(u)| \leq K \implies m \neq u$.

$m \neq w$ pois $|S(w)| = 0$ após a linha 30 do algoritmo 3, e w não é mais um mestre.

$m \neq y$ pois, se existir, y é um escravo não-compartilhado de u.

Antes da união, v é um escravo não compartilhado de w em B. Como nenhum escravo foi adicionado a v durante a reorganização. Portanto, $S(v) = \emptyset \implies S(v) \leq K$ logo, $m \neq v$.

Para os outros mestres diferentes de u, w, y e v, nenhuma conexão é estabelecida ou destruída durante a reorganização. Portanto nós diferentes de u, w, y e v possuem um escravo após a reorganização se, e somente se, ele possui o escravo antes da reorganização. Como $m \notin \{u, w, v, y\}$ e depois da reorganização $|S(m)| > K$, então antes da união $|S(m)| > K$. Dado que os componentes A e B respeitam as propriedades 3, antes da reorganização todos os mestres em A e em B possuem no máximo K escravos. Como m estava em A ou B antes da união e não teve seu conjunto de escravos modificados, então

$|S(m)| < K$.

Isto é uma contradição. Portanto a premissa é verdadeira. □

Lema 30. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 6, após a união (MERGE) o líder possui um escravo não compartilhado ou não possui nenhum escravo.*

*Demonstração.*O líder do componente após a união é u, pois: (1) w, o líder de B, foi aposentado, (2) u, líder de A, não foi aposentado, e (3) nenhum outro nó foi transformado e um líder durante a reorganização.

Sabe-se que v é um escravo não compartilhado de w antes da união acontecer. Portanto antes da união $M(v) = \{w\}$ e $S(v) = \emptyset$. Ao iniciar o procedimento de reorganização, uma conexão é formada entre v e u. Logo, $M(v) = \{w,u\}$ e $S(v) = \emptyset$.

Dado que na linha 30 do algoritmo 3 v foi desconectado de w, então $M(v) = \{u\}$. Nenhuma outra modificação no conjunto de mestres e escravos de v ocorre durante a reorganização. Portanto, após a reorganização o líder do componente, u, possui um escravo não compartilhado, v. □

Lema 31. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 6, após a união (MERGE) todos os escravos não-compartilhados possuem h menor ou igual ao h de seu mestre.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre k com um escravo não compartilhado l tal que $h(l) > h(k)$.

w não é um mestre, pois todos seus escravos foram desconectados dele na linha 30 do algoritmo 3. Logo $k \neq w$.

Os escravos de u após concluída a reorganização podem ser separadas em três casos: (1) escravos que já eram escravos de u antes da união; (2) escravos que eram escravos de w antes da reorganização; (3) o ex-líder do componente B w.

Para nós em (1) a propriedade é verdadeira pois o componente A respeita a propriedade 5 antes da reorganização ocorrer.

Para nós em (3) a propriedade é verdadeira pois sabe-se que no caso 6 $h(u) \geq h(w)$, devido as condições da linha 12 no algoritmo 4.

Para nós em (2) a propriedade é verdadeiro por transitividade: sabe-se que B respeita a propriedade 5 antes da reorganização, logo sabe-se que todo os escravos não compartilhados n de w possuem $h(n) \leq h(w)$. Logo $h(w) \leq h(u) \implies h(n) \leq h(u)$.

Como todos os escravos não compartilhados de u possuem h menor ou igual a $h(u)$, então $k \neq u$.

Para mestres diferentes de u e w: nenhum novo escravo lhes foi atribuído. Portanto um mestre diferente de u e w possui um escravo se, e somente se, ele possui um escravo antes da união. Dado que $k \neq u \wedge k \neq w$, antes da união k foi mestre de l em A ou B. Dado que A e B respeitam a propriedade 5, $h(k) \geq h(l)$.

Isto é uma contradição, e portanto a premissa é verdadeira. □

A.6 CASO 7: $W \neq V \wedge |S(W)| + |S(U)| < K \wedge H(U) < H(W)$

Lema 32. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 7, após a união (MERGE) o componente resultante da união possui um líder.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe mais de um líder ou não existe nenhum líder.

Na linha 34 do algoritmo 3, u, líder do componente A (linha 17 do algoritmo 2), é aposentado. Antes da união, o componente B respeita todas as propriedades, e segundo a propriedade 1 pode-se concluir que após a finalização da reorganização, nenhum nó que pertencia a A é líder. Dado que: (1) não há nenhum líder pertencente a A; (2) a reorganização 1 não torna nenhum nó líder; e, (3) nenhum nó pertencente a B é aposentado; todos líderes de B são líderes do componente após união. Como nenhum nó pertencente ao componente A é líder, todos os líderes do componente após a união são líderes do componente B. Logo, antes da reorganização o componente B não possui nenhum líder ou possui pelo menos 2 líderes. Isto é uma contradição, pois sabe-se devido a propriedade 1 que o componente B possui exatamente 1 líder. Conclui-se que a premissa é verdadeira e existe um líder após o término da reorganização. □

Lema 33. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 7, após a união (MERGE) o líder possuirá no máximo $K - 1 = 6$ escravos.*

*Demonstração.*A ocorrência da reorganização CASO 7 exige que as condições da linha 14 do algoritmo 4 sejam satisfeitas, portanto $|S(w)| + |S(u)| < K$.

Na linha 36 do algoritmo 3, todos os escravos de u são desconectados dele. Em seguida, na linha 37 do algoritmo 3, todos os escravos que pertenciam a u, inclusive u são conectados a w. Dado que $S(w) \cap S(u) = \{v\}$ quando $v \neq w$, v não é reconectado na linha 37 pois esta conexão já existe.

Considerando $S'(x)$ o conjunto de escravos de x na linha 14 do algoritmo 4. E $S(x)$ o conjunto de escravos de x após a conclusão da reorganiza-

ção.

Logo:

$$\begin{aligned}
|S(w)| &= |(S'(u) \cup S'(w)) \cup \{w\}| \\
&= |S'(u) \cup S'(w)| + |\{w\}| - |(S'(u) \cup S'(w)) \cap \{w\}| \\
&= |S'(u)| + |S'(w)| - |S'(u) \cap S'(w)| + 1 - |\emptyset| \\
&= |S'(u)| + |S'(w)| - 1 + 1 \\
&= |S'(u)| + |S'(w)| \\
&< K
\end{aligned}$$

Logo, $|S(w)| < K \implies |S(w)| \leq K-1$. □

Lema 34. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 7, não existe mestre com mais de $K = 7$ escravos.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre m com mais de $K = 7$ escravos. Sabe-se que $m \neq u$, pois segundo o lema 33 $|S(w)| \leq K-1 \implies |S(w)| \leq K \implies m \neq w$.

$m \neq u$ pois $|S(u)| = 0$ após a linha 36 do algoritmo 3, e u não é mais um mestre.

$y \neq m$ pois, se existir, y é um escravo não-compartilhado de u.

Antes da união, v é um escravo não compartilhado de w em B. Como nenhum escravo foi adicionado a v durante a reorganização. Portanto, $S(v) = \emptyset \implies S(v) \leq K$ logo, $m \neq v$.

Para os outros mestres diferentes de u, w, y e v, nenhuma conexão é estabelecida ou destruída durante a reorganização. Portanto nós diferentes de u, w, y e v possuem um escravo após a reorganização se, e somente se, ele possui o escravo antes da reorganização. Como $m \notin \{u,w,v,y\}$ e depois da reorganização $|S(m)| > K$, então antes da união $|S(m)| > K$. Dado que os componentes A e B respeitam as propriedades 3, antes da reorganização todos os mestres em A e em B possuem no máximo K escravos. Como m estava em A ou B antes da união e não teve seu conjunto de escravos modificados, então $|S(m)| < K$.

Isto é uma contradição. Portanto a premissa é verdadeira. □

Lema 35. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 1, após a união (MERGE) o líder possui um escravo não compartilhado ou não possui nenhum escravo.*

*Demonstração.*O líder do componente após a união é w, pois: (1) u, o líder de A, foi aposentado, (2) w, líder de B, não foi aposentado, e (3) nenhum outro

nó foi transformado e um líder durante a reorganização.

Sabe-se que v é um escravo não compartilhado de w antes da união acontecer. Portanto antes da união $M(v) = \{w\}$ e $S(v) = \emptyset$. Ao iniciar o procedimento de reorganização, uma conexão é formada entre v e u. Logo, $M(v) = \{w, u\}$ e $S(v) = \emptyset$.

Dado que na linha 36 do algoritmo 3 v foi desconectado de u, então $M(v) = \{w\}$. Nenhuma outra modificação no conjunto de mestres e escravos de v ocorre durante a reorganização. Portanto, após a reorganização o líder do componente, w, possui um escravo não compartilhado, v. □

Lema 36. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 7, após a união (MERGE) todos os escravos não-compartilhados possuem h menor ou igual ao h de seu mestre.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre k com um escravo não compartilhado l tal que $h(l) > h(k)$.

u não é um mestre, pois todos seus escravos foram desconectados dele na linha 36 do algoritmo 3. Logo $k \neq u$.

Os escravos de w após concluída a reorganização podem ser separadas em três casos: (1) escravos que já eram escravos de w antes da união; (2) escravos que eram escravos de u antes da reorganização; (3) o ex-líder do componente A u.

Para nós em (1) a propriedade é verdadeira pois o componente A respeita a propriedade 5 antes da reorganização ocorrer.

Para nós em (3) a propriedade é verdadeira pois sabe-se que no caso 7 $h(u) < h(w)$, devido as condições da linha 14 no algoritmo 4.

Para nós em (2) a propriedade é verdadeira por transitividade: sabe-se que A respeita a propriedade 5 antes da reorganização, logo sabe-se que todo os escravos não compartilhados n de u possuem $h(n) \leq h(u)$. Logo $h(u) < h(w) \implies h(n) \leq h(w)$.

Como todos os escravos não compartilhados de w possuem h menor ou igual a $h(w)$, então $k \neq w$.

Para mestres diferentes de u e w: nenhum novo escravo lhes foi atribuído. Portanto um mestre diferente de u e w possui um escravo se, e somente se, ele possui um escravo antes da união. Dado que $k \neq u \wedge k \neq w$, antes da união k foi mestre de l em A ou B. Dado que A e B respeitam a propriedade 5, $h(k) \geq h(l)$.

Isto é uma contradição, e portanto a premissa é verdadeira. □

A.7 CASO 8: $W \neq V \wedge |S(W)| + |S(U)| = K \wedge |S(U)| = 1 \wedge H(V) \geq H(U)$

Lema 37. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 8, após a união (MERGE) o componente resultante da união possui um líder.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe mais de um líder ou não existe nenhum líder.

A ocorrência da reorganização CASO 8 exige que as condições da linha 16 do algoritmo 4 sejam satisfeitas, portanto $|S(u)| = 1$. Dado que $v \in S(u)$, antes da união o componente A é composto de apenas um nó: u.

u foi aposentado na linha 40 do algoritmo 3. Portanto, não existente nenhum líder no componente A.

Na linha 41, w, o líder do componente B, foi aposentado. E na linha 42 do algoritmo 3, v torna-se um líder.

Dado que não há líder no componente A, deve existir mais um líder que pertencia a B antes da reorganização além de u. Dado que u foi aposentado, deve existir um líder além de u e v em B. Como nenhum nó além de v foi tornado líder durante a reorganização, qualquer outro nó é líder após a reorganização se foi líder antes. Portanto, componente B possui um líder diferente de u e v antes da reorganização.

Dado que B respeita a propriedade 1, antes da reorganização existe apenas um líder em B: w. Isto é uma contradição. Conclui-se que a premissa é verdadeira e existe um líder após o término da reorganização. □

Lema 38. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 8, após a união (MERGE) o líder possuirá no máximo $K - 1 = 6$ escravos.*

*Demonstração.*Nas linhas 40 e 41 do algoritmo 3, u, líder do componente A (linha 17 do algoritmo 2), e w, líder do componente B (linha 18 do algoritmo 2), são aposentados. E na linha 42 do algoritmo 3, v se torna o líder do novo componente.

É sabido que v é um escravo não compartilhado de w antes da reorganização. Portanto, antes da reorganização, $S(w) = \emptyset$. Na linha 44 u se torna escravo de v, e $|S(y)| = 1$.

u é o único escravo adicionado a v durante a reorganização. Portanto após a reorganização ser concluída, $|S(v)| = 1 \implies |S(v)| <= K - 1$. □

Lema 39. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 8, não existe mestre com mais de $K = 7$ escravos.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre m com mais de $K = 7$ escravos. Sabe-se que $m \neq v$, pois segundo o lema 38 $|S(v)| \leq K-1 \implies |S(v)| \leq K \implies m \neq v$.

A ocorrência da reorganização CASO 8 exige que as condições da linha 16 do algoritmo 4 sejam satisfeitas, portanto $|S(u)| = 1$. Dado que $v \in S(u)$, antes da união o componente A é composto de apenas um nó: u.

$m \neq u$ pois como $S(u) = \{v\}$ e na linha 43 do algoritmo 3 v é desconectado de u e nenhum escravo é atribuído a u, após a reorganização $|S(u)| = 0 < K$.

Componente B respeita a propriedade 3, portanto antes da união $S(w) < K-1$. Como nenhum novo escravo é adicionado a w durante a reorganização, após a conclusão da união $|S(w)| \leq K$ e $m \neq w$.

Para os outros mestres diferentes de u, w e y, nenhuma conexão é estabelecida ou destruída durante a reorganização. Portanto nós diferentes de u, y e w possuem um escravo após a reorganização se, e somente se, ele possui o escravo antes da reorganização. Como $m \notin \{u,w,y\}$ e depois da reorganização $|S(m)| > K$, então $|S(m)| > K$ antes da união $|S(m)| > K$. Dado que o componente A e B respeitam a propriedade 3, antes da reorganização todos os mestres em A e B possuem no máximo K escravos. Como m estava em A ou B antes da união, então $|S(m)| < K$.

Isto é uma contradição. Portanto a premissa é verdadeira. □

Lema 40. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 8, após a união (MERGE) o líder possui um escravo não compartilhado ou não possui nenhum escravo.*

*Demonstração.*Nas linhas 40 e 41 do algoritmo 3, u, líder do componente A (linha 17 do algoritmo 2), e w, líder do componente B (linha 18 do algoritmo 2), são aposentados. E na linha 42 do algoritmo 3, v se torna o líder do novo componente.

A ocorrência da reorganização CASO 8 exige que as condições da linha 16 do algoritmo 4 sejam satisfeitas, portanto $|S(u)| = 1$. Logo, $S(u) = \{v\}$. Na linha 43 do algoritmo 3 v é desconectado de u e nenhum escravo é adicionado a u. Portanto, $S(u) = \emptyset$ e $M(u) = \emptyset$.

Na linha 44, u torna-se escravo do líder v. Portanto, $S(u) = \emptyset$ e $M(u) = \{v\}$. Ou seja, u é um escravo não compartilhado do líder v. □

Lema 41. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 8, após a união (MERGE) todos os escravos não-compartilhados possuem h menor ou igual ao h de seu mestre.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre k com um escravo não compartilhado l tal que $h(l) > h(k)$.

$k \neq u$ pois seu único escravo v foi desconectado dele.

O líder do novo componente é v (linha 42 no algoritmo 3). v é, antes da união, um escravo não compartilhado de w no componente B. Portanto, antes da união $S(v) = \emptyset$. Na linha 44 do algoritmo 3 u se torna escravo de v, logo $S(v) = \{u\}$. A ocorrência da reorganização CASO 8 exige que as condições da linha 16 do algoritmo 4 sejam satisfeitas, portanto $h(v) \geq h(u)$. Logo, $k \neq v$.

Para mestres diferentes de u e v: nenhum novo escravo lhes foi atribuído. Portanto um mestre diferente de u e v possui um escravo se, e somente se, ele possui um escravo antes da união. Dado que $k \neq u \wedge k \neq v$, antes da união k foi mestre de l em A ou B. Dado que A e B respeitam a propriedade 5, $h(k) \geq h(l)$.

Isto é uma contradição, e portanto a premissa é verdadeira. □

A.8 CASO 9: $W \neq V \wedge |S(W)| + |S(U)| = K \wedge |S(U)| = 1 \wedge H(V) < H(U)$

Lema 42. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 9, após a união (MERGE) o componente resultante da união possui um líder.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe mais de um líder ou não existe nenhum líder. Na linha 47 do algoritmo 3, w, líder do componente B (linha 18 do algoritmo 2), é aposentado. Antes da união, o componente B respeita todas as propriedades, e segundo a propriedade 1 pode-se concluir que após a finalização da reorganização, nenhum nó que pertencia a B é líder. Dado que: (1) não há nenhum líder pertencente a B; (2) a reorganização não torna nenhum nó líder; e, (3) nenhum nó pertencente a A é aposentado; todos líderes de A são líderes do componente após união. Como nenhum nó pertencente ao componente B é líder, todos os líderes do componente após a união são líderes do componente A. Logo, antes da reorganização o componente A não possui nenhum líder ou possui pelo menos 2 líderes. Isto é uma contradição, pois sabe-se devido a propriedade 1 que o componente A possui exatamente 1 líder. Conclui-se que a premissa é verdadeira e existe um líder após o término da reorganização. □

Lema 43. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 9, após a união (MERGE)*

o líder possuirá no máximo $K-1=6$ escravos.

*Demonstração.*O líder do componente após completada a união é u, pois apenas o líder w é aposentado na linha 47 do algoritmo 3.

A ocorrência da reorganização CASO 9 exige que as condições da linha 18 do algoritmo 4 sejam satisfeitas, portanto $|S(u)|=1$.

Dado que nenhum escravo é adicionado ou removido de u, após a reorganização ser finalizada, $S(u)=1 \implies |S(u) \leq K-1|$. □

Lema 44. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 9, não existe mestre com mais de $K=7$ escravos.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre m com mais de $K=7$ escravos. Sabe-se que $m \neq u$, pois segundo o lema 43 $|S(u)| \leq K-1 \implies |S(u)| \leq K \implies m \neq u$.

A ocorrência da reorganização CASO 8 exige que as condições da linha 16 do algoritmo 4 sejam satisfeitas, portanto $|S(u)|=1$. Dado que $v \in S(u)$, antes da união o componente A é composto de apenas um nó: u.

$m \neq u$ pois como $S(u)=\{v\}$ e na linha 43 do algoritmo 3 v é desconectado de u e nenhum escravo é atribuído a u, após a reorganização $|S(u)|=0<K$.

Componente B respeita a propriedade 3, portanto antes da união $S(w)<K-1$. Na linha 48 do algoritmo 3, o nó v é removido do conjunto de escravos de w, e na linha 49 do algoritmo 3 o nó u torna-se escravo de w. Portanto, o número de escravos permanece o mesmo após a reorganização ser concluída. Logo, $|slave(w)| \leq K$ e $m \neq w$

No início do procedimento de reorganização, v é escravo de w e u e não possui escravos. Como nenhum escravo foi adicionado a v durante a reorganização, v não é mestre e $m \neq v$

Para os outros mestres diferentes de u, w e v, nenhuma conexão é estabelecida ou destruída durante a reorganização. Portanto nós diferentes de u, v e w possuem um escravo após a reorganização se, e somente se, ele possui o escravo antes da reorganização. Como $m \notin \{u,w,v\}$ e depois da reorganização $|S(m)|>K$, então $|S(m)|>K$ antes da união $|S(m)|>K$. Dado que o componente A e B respeitam a propriedade 3, antes da reorganização todos os mestres em A e B possuem no máximo K escravos. Como m estava em A ou B antes da união, então $|S(m)|<K$.

Isto é uma contradição. Portanto a premissa é verdadeira. □

Lema 45. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 9, após a união (MERGE) o líder possui um escravo não compartilhado ou não possui nenhum escravo.*

*Demonstração.*Na linha 47 do algoritmo 3, w, o líder do componente B (linha 17 do algoritmo 2) é aposentado. Portanto, u permanece um líder após a reorganização.

No início do procedimento de reorganização, v é escravo de w e u ($M(v) = \{u, w\}$) e não possui escravos ($S(v) = \emptyset$). Na linha 48, v é desconectado de w. Logo, $M(v) = \{u\}$, tonando-se um escravo não compartilhado do líder u. □

Lema 46. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 9, após a união (MERGE) todos os escravos não-compartilhados possuem h menor ou igual ao h de seu mestre.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre k com um escravo não compartilhado l tal que $h(l) > h(k)$.

A ocorrência da reorganização CASO 9 exige que as condições da linha 18 do algoritmo 4 sejam satisfeitas, portanto $h(v) < h(u)$ e $|S(u)| = 1$.

Dado que $v \in S(u)$, então $S(u) = \{v\}$ e $h(v) < h(u) \implies k \neq u$.

$k \neq u$ pois seu único escravo v foi desconectado dele.

Considerando $S'(x)$ o conjunto de escravos de x na linha 18 do algoritmo 4.

Os escravos de w após o término da reorganização são: $S(w) = S'(w) - \{v\} \cup \{u\}$. Antes da união o componente B respeita a propriedade 5. Logo, os escravos não compartilhados em $S(w)$ possuem h menor ou igual a $h(w)$ $S(u) \neq \emptyset$, portanto u não é um escravo não compartilhado de w, e seu valor h não é relevante. Portanto, todos os escravos não compartilhados de w possuem h menor ou igual a w. Logo, $k \neq w$

Para mestres diferentes de u e w: nenhum novo escravo lhes foi atribuído. Portanto um mestre diferente de u e w possui um escravo se, e somente se, ele possui um escravo antes da união. Dado que $k \neq u \wedge k \neq w$, antes da união k foi mestre de l em A ou B. Dado que A e B respeitam a propriedade 5, $h(k) \geq h(l)$.

Isto é uma contradição, e portanto a premissa é verdadeira. □

A.9 CASO 10: $W \neq V \wedge |S(W)| + |S(U)| = K \wedge |S(U)| > 1 \wedge H(V) \geq H(U)$

Lema 47. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 10, após a união (MERGE) o componente resultante da união possui um líder.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa:

após a união existe mais de um líder ou não existe nenhum líder.

u foi aposentado na linha 52 do algoritmo 3. Portanto, como o componente A respeita a propriedade 1 antes da reorganização e nenhum nó de A foi feito líder, não existente nenhum líder no componente A.

Na linha 53, w, o líder do componente B, foi aposentado. E na linha 54 do algoritmo 3, v torna-se um líder.

Dado que não há líder no componente A, deve existir mais um líder que pertencia a B antes da reorganização além de u. Dado que u foi aposentado, deve existir um líder além de u e v em B. Como nenhum nó além de v foi tornado líder durante a reorganização, qualquer outro nó é líder após a reorganização se foi líder antes. Portanto, componente B possui um líder diferente de u e v antes da reorganização.

Dado que B respeita a propriedade 1, antes da reorganização existe apenas um líder em B: w. Isto é uma contradição. Conclui-se que a premissa é verdadeira e existe um líder após o término da reorganização. □

Lema 48. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 10, após a união (MERGE) o líder possuirá no máximo* $K-1=6$ *escravos.*

*Demonstração.*Nas linhas 52 e 53 do algoritmo 3, u, líder do componente A (linha 17 do algoritmo 2), e w, líder do componente B (linha 18 do algoritmo 2), são aposentados. E na linha 54 do algoritmo 3, v se torna o líder.

A ocorrência da reorganização CASO 10 exige que as condições da linha 20 do algoritmo 4 sejam satisfeitas, portanto $|S(u)|>1$. $S(u)>1$ e a propriedade 4 implicam que y possui um escravo não compartilhado, e portanto y é definido.

Sabe-se que antes da reorganização v era um escravo não compartilhado de w no componente B. Logo, antes da união, $S(v)=\emptyset$. Na linha 58, v se torna mestre de u e y, logo $|S(v)|=2$.

u e y são os únicos escravos adicionados a v durante a reorganização. Portanto, após a reorganização ser finalizada $|S(v)|=2 \implies |S(v)|<=K-1$. □

Lema 49. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 10, não existe mestre com mais de* $K=7$ *escravos.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre m com mais de $K=7$ escravos. Sabe-se que $m \neq v$, pois segundo o lema 38 $|S(v)| \leq K-1 \implies |S(v)| \leq K \implies m \neq v$.

Na linha 56 do algoritmo 3 todos os escravos de u são desconectados dele. Portanto, $S(u)=\emptyset \implies |S(u)|<k$ e $m \neq u$.

A ocorrência da reorganização CASO 10 exige que as condições da linha 20 do algoritmo 4 sejam satisfeitas, portanto $|S(w)| + |S(u)| = K$ e $|S(u)| > 1$. $S(u) > 1$ e a propriedade 4 implicam que y possui um escravo não compartilhado, e portanto y é definido.

Na linha 57 do algoritmo 3 todos os escravos que eram de u, exceto y, tornam-se escravos de w.

Considerando $S'(x)$ o conjunto de escravos de x na linha 12 do algoritmo 4.

Após a reorganização ser finalizada $S(w) = S'(w) \cup (S'(u) - \{y, v\})$. Logo:

$$\begin{aligned}
|S(w)| &= |S'(w) \cup (S'(u) - \{y,v\})| \\
&= |S'(w)| + |(S'(u) - \{y,v\})| - |S'(w) \cap (S'(u) - \{y,v\})| \\
&= |S'(w)| + |S'(u) - \{y,v\}| - |\emptyset| \\
&= |S'(w)| + |S'(u)| - |S'(u) \cap \{y,v\}| \\
&= |S'(w)| + |S'(u)| - 2 \\
&= K - 2 \\
&\leq K.
\end{aligned}$$

Logo, $m \neq w$.

Para os outros mestres diferentes de u, w e v, nenhuma conexão é estabelecida ou destruída durante a reorganização. Portanto nós diferentes de u, v e w possuem um escravo após a reorganização se, e somente se, ele possui o escravo antes da reorganização. Como $m \notin \{u, w, v\}$ e depois da reorganização $|S(m)| > K$, então $|S(m)| > K$ antes da união $|S(m)| > K$. Dado que o componente A e B respeitam a propriedade 3, antes da reorganização todos os mestres em A e B possuem no máximo K escravos. Como m estava em A ou B antes da união, então $|S(m)| < K$.

Isto é uma contradição. Portanto a premissa é verdadeira. □

Lema 50. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 10, após a união (MERGE) o líder possui um escravo não compartilhado ou não possui nenhum escravo.*

*Demonstração.*Nas linhas 52 e 53 do algoritmo 3, u, líder do componente A (linha 17 do algoritmo 2), e w, líder do componente B (linha 18 do algoritmo 2), são aposentados. E na linha 54 do algoritmo 3, v se torna o líder do novo componente.

A ocorrência da reorganização CASO 10 exige que as condições da linha 20 do algoritmo 4 sejam satisfeitas, portanto $|S(u)| > 1$. $S(u) > 1$ e a

propriedade 4 implicam que y possui um escravo não compartilhado, e portanto y é definido.

Na linha 58 v se torna mestre de u e y. No início da reorganização, y é um escravo não compartilhado pois pertence a $Y_p(u)$ (linha 26 do algoritmo 2) que, por definição, contém apenas escravos não compartilhados.

Portanto, o líder v possui pelo menos um escravo não compartilhado y. □

Lema 51. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 10, após a união (MERGE) todos os escravos não-compartilhados possuem h menor ou igual ao h de seu mestre.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre k com um escravo não compartilhado l tal que $h(l) > h(k)$.

$k \neq u$ pois seus escravos foram desconectados dele na linha 56 do algoritmo 3.

A ocorrência da reorganização CASO 10 exige que as condições da linha 20 do algoritmo 4 sejam satisfeitas, portanto $|S(u)| > 1$ e $h(v) \geq h(u)$. $S(u) > 1$ e a propriedade 4 implicam que y possui um escravo não compartilhado, e portanto y é definido.

O líder do novo componente é v (linha 54 no algoritmo 3). v é, antes da união, um escravo não compartilhado de w no componente B. Portanto, antes da união $S(v) = \emptyset$. Na linha 58 do algoritmo 3 u e y se tornam escravos de v, logo $S(v) = \{u, y\}$. Dado que $h(v) \geq h(u)$ e y era um escravo não compartilhado de u em A, e A respeita a propriedade 5 antes da reorganização, então $h(u) \geq h(y) \implies h(v) \geq h(y)$. Logo, $k \neq v$.

Na linha 57 do algoritmo 3 todos os escravos de u, exceto y e v, tornam-se escravos de w. Dado que antes da união v era um escravo não-compartilhado de w no componente B, e que o componente B respeita a propriedade 5, então $h(w) \geq h(v) \implies h(w) \geq h(u)$. Já que componente A respeita a propriedade 5, conclui-se que $\forall s \in slavesOf(u) : h(w) \geq h(u) \geq h(s) \vee \neg isFreeSlave(s)$.

Os escravos de w que não foram conectados na linha 57, eram escravos no componente B antes da união. Devido a propriedade 26, estes escravos possuem h menor ou igual ao $h(w)$. Logo, todos os escravos não compartilhados de w possuem h menor ou igual a $h(w)$. Portanto, $m \neq w$.

Para mestres diferentes de u e v: nenhum novo escravo lhes foi atribuído. Portanto um mestre diferente de u e v possui um escravo se, e somente se, ele possui um escravo antes da união. Dado que $k \neq u \wedge k \neq v$, antes da união k foi mestre de l em A ou B. Dado que A e B respeitam a propriedade

5, $h(k) \geq h(l)$.

Isto é uma contradição, e portanto a premissa é verdadeira. □

A.10 CASO 11: $W \neq V \wedge |S(W)| + |S(U)| = K \wedge |S(U)| > 1 \wedge H(V) < H(U)$

No caso 11 u realiza o processo de INQUIRY, descobre o nó v e estabelece a conexão entre u e v representada pela linha tracejada. O mestre de v é w, líder do componente. O peso de v é maior ou igual ao peso de u ($h(v) \geq h(u)$).

Nesta reorganização, não é possível unificar as *Piconets* de u e w em uma, pois a *Piconet* do líder possuiria 7 nós. Com isso, a propriedade 2 seria violada. Mas ao contrário das reorganização 8 e 9, na reorganização 11 u possui escravos. logo, existe um nó y.

Nesta situação, como $h(v) < h(u)$. Isso permite mover todos os escravos de w para u, deixando apenas v para que o novo líder w tenha um escravo não compartilhado.

Lema 52. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 11, após a união (MERGE) o componente resultante da união possui um líder.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe mais de um líder ou não existe nenhum líder.

Na linha 61 do algoritmo 3, u, líder do componente A (linha 17 do algoritmo 2), é aposentado. Antes da união, o componente B respeita todas as propriedades, e segundo a propriedade 1 pode-se concluir que após a finalização da reorganização, nenhum nó que pertencia a A é líder. Dado que: (1) não há nenhum líder pertencente a A; (2) a reorganização 1 não torna nenhum nó líder; e, (3) nenhum nó pertencente a B é aposentado; todos líderes de B são líderes do componente após união. Como nenhum nó pertencente ao componente A é líder, todos os líderes do componente após a união são líderes do componente B. Logo, antes da reorganização o componente B não possui nenhum líder ou possui pelo menos 2 líderes. Isto é uma contradição, pois sabe-se devido a propriedade 1 que o componente B possui exatamente 1 líder. Conclui-se que a premissa é verdadeira e existe um líder após o término da reorganização. □

Lema 53. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 11, após a união (MERGE) o líder possuirá no máximo $K-1=6$ escravos.*

*Demonstração.*Na linha 61 do algoritmo 3, u, o líder do componente A (linha 17 do algoritmo 2) foi aposentado. Logo, w é o líder do componente após

a reorganização ser concluída.

Na linha 65 todos os escravos de w são desconectados de w, exceto v. Portanto, $|S(w)| = 1 \implies |S(w)| \leq K-1$. □

Lema 54. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 11, não existe mestre com mais de $K = 7$ escravos.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre m com mais de $K = 7$ escravos. Sabe-se que $m \neq w$, pois segundo o lema 53 $|S(w)| \leq K-1 \implies |S(w)| \leq K \implies m \neq w$.

$m \neq v$ pois nenhum escravo foi adicionado a v. Sendo v um escravo não compartilhado de w, $S(v) = \emptyset$.

A ocorrência da reorganização CASO 11 exige que as condições da linha 22 do algoritmo 4 sejam satisfeitas, portanto $|S(w)| + |S(u)| = K$ e $|S(u)| > 1$. $S(u) > 1$ e a propriedade 4 implicam que y possui um escravo não compartilhado, e portanto y é definido.

Na linha 65 do algoritmo 3 todos os escravos de w, exceto v, são desconectados de w. E na linha 66, todos os antigos escravos de w tornam-se escravos de u.

Considerando $S'(x)$ o conjunto de escravos de x na linha 22 do algoritmo 4.

Após a conclusão da reorganização $S(u) = (S'(u) - \{v\}) \cup (S'(w) - \{v\}) \cup \{w\}$. Logo:

$$\begin{aligned}
|S(u)| &= |(S'(u) \cup \{w\} - \{v\}) \cup (S'(w) - \{v\})| \\
&= |S'(u) \cup \{w\} - \{v\}| + |S'(u) - \{v\}| - |(S'(u) - \{v\}) \cap (S'(w) - \{v\})| \\
&= |S'(u)| + |\{w\}| - |\{v\}| + |S'(u)| - |\{v\}| - |\emptyset| \\
&= |S'(u)| + |S'(u)| - 1 \\
&= K - 1 \\
&\leq K.
\end{aligned}$$

Portanto, conclui-se que $m \neq u$.

Para os outros mestres diferentes de u, w e v, nenhuma conexão é estabelecida ou destruída durante a reorganização. Portanto nós diferentes de u, v e w possuem um escravo após a reorganização se, e somente se, ele possui o escravo antes da reorganização. Como $m \notin \{u, w, v\}$ e depois da reorganização $|S(m)| > K$, então $|S(m)| > K$ antes da união $|S(m)| > K$. Dado que o componente A e B respeitam a propriedade 3, antes da reorganização todos os mestres em A e B possuem no máximo K escravos. Como m estava em A ou B antes da união, então $|S(m)| < K$.

Isto é uma contradição. Portanto a premissa é verdadeira. □

Lema 55. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 1, após a união (MERGE) o líder possui um escravo não compartilhado ou não possui nenhum escravo.*

*Demonstração.*Na linha 61 do algoritmo 3, u, o líder do componente A (linha 17 do algoritmo 2) foi aposentado. Logo, w é o líder do componente após a reorganização ser concluída.

Na linha 65 todos os escravos de w são desconectados de w, exceto v. Portanto, $S(w) = \{v\}$.

Antes da união, v é um escravo não-compartilhado de w no componente B ($M(v) = \emptyset \wedge S(v) = \{w\}$). Quando u, o líder de A, descobre o componente B uma conexão é estabelecida e v se tornou um escravo de u ($M(v) = \{u,w\} \wedge S(v) = \emptyset$).

Na linha 62, v foi desconectado de u, logo $M(v) = \{w\} \wedge slavesOf(v) = \emptyset$. Ou seja, v é o escravo não compartilhado do líder w. □

Lema 56. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 11, após a união (MERGE) todos os escravos não-compartilhados possuem h menor ou igual ao h de seu mestre.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre k com um escravo não compartilhado l tal que $h(l) > h(k)$.

$k \neq u$ pois seus escravos foram desconectados dele na linha 56 do algoritmo 3.

A ocorrência da reorganização CASO 11 exige que as condições da linha 22 do algoritmo 4 sejam satisfeitas, portanto $h(v) < h(u)$ e $|S(u)| > 1$. $S(u) > 1$ e a propriedade 4 implicam que y possui um escravo não compartilhado, e portanto y é definido.

Todos os escravos de w, exceto v, são desconectados de w na linha 65 do algoritmo 3 e nenhum escravo é adicionado a w. Dado que antes da reorganização v é um escravo não compartilhado de w no componente B, e B respeita a propriedade 5, então $h(v) \leq h(w)$. Logo, o único escravo de w respeita a propriedade e $k \neq w$.

$k \neq v$, pois v é um escravo não compartilhado do líder w após o término da reorganização.

Na linha 66 do algoritmo 3 todos os escravos de w, e o nó w tornam-se escravos de u. v é escolhido dentro os escravos não compartilhados de w ($Y_p(w)$) para INQUIRY SCAN na linha 11 do algoritmo 2 . Portanto, por definição, antes da reorganização $\forall s \in S(w) : h(v) \geq h(s) \vee \not{s}UnsharedSlave(s)$.

E $h(v) < h(u)$ implica que $\forall s \in S(w) : h(u) > h(v) \geq h(s) \vee$ *isUnsharedSlave*(s). Logo, todos os escravos de w podem ser escravos de u sem violar a propriedade 5. w não é um escravo não compartilhado, portanto o valor $h(w)$ é irrelevante e ele pode se tornar escravo de u. Os demais escravos de u após o término da reorganizam, já o eram antes da união ocorrer no componente A. Dado que componente A respeita a propriedade 5 antes da reorganização, estes escravos continuam a manter a propriedade 5 após a conclusão da reorganização. Assim sendo, todos os escravos de u respeitam a propriedade 5, e $k \neq u$.

Para mestres diferentes de u, v e w: nenhum novo escravo lhes foi atribuído. Portanto um mestre diferente de u, v e w possui um escravo se, e somente se, ele possui um escravo antes da união. Dado que $k \neq u \wedge k \neq w \wedge k \neq v$, antes da união k foi mestre de l em A ou B. Dado que A e B respeitam a propriedade 5, $h(k) \geq h(l)$.

Isto é uma contradição, e portanto a premissa é verdadeira. □

A.11 CASO 12: $W \neq V \wedge |S(W)| + |S(U)| > K \wedge H(W) \geq H(U)$

No caso 12, u realiza o processo de INQUIRY, descobre o nó v e estabelece a conexão entre u e v representada pela linha tracejada. O mestre de v é w, líder do componente.

Nesta reorganização, não é possível unificar as *Piconets* de u e w em uma, pois há mais de 8 nós juntando-se as duas *Piconets*.

Na reorganização 12 $h(w) \geq h(u)$. Isso permite realocar os escravos de u para w, esvaziando a *Piconet* do novo líder u. v é mantido como ponte entre u e w.

Lema 57. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 12, após a união (MERGE) o componente resultante da união possui um líder.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe mais de um líder ou não existe nenhum líder. Na linha 69 do algoritmo 3, w, líder do componente B (linha 18 do algoritmo 2), é aposentado. Antes da união, o componente B respeita todas as propriedades, e segundo a propriedade 1 pode-se concluir que após a finalização da reorganização, nenhum nó que pertencia a B é líder. Dado que: (1) não há nenhum líder pertencente a B; (2) a reorganização não torna nenhum nó líder; e, (3) nenhum nó pertencente a A é aposentado; todos líderes de A são líderes do componente após união. Como nenhum nó pertencente ao componente B é líder, todos os líderes do componente após a união são líderes do componente

A. Logo, antes da reorganização o componente A não possui nenhum líder ou possui pelo menos 2 líderes. Isto é uma contradição, pois sabe-se devido a propriedade 1 que o componente A possui exatamente 1 líder. Conclui-se que a premissa é verdadeira e existe um líder após o término da reorganização. □

Lema 58. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 12, após a união (MERGE) o líder possuirá no máximo* $K-1=6$ *escravos.*

*Demonstração.*O líder após finalizada a reorganização é u, pois entre os dois líderes u e w, apenas w é aposentado na linha 69 do algoritmo 3.

Na linha 74 escravos são desconectados de u e reconectados a w na linha 75. O número de escravos movidos de u para w é igual ao menor valor entre as variáveis *availablesSlots* (70) e *availableSlaves* (75)).

Considerando $S'(x)$ o conjunto de escravos de x na linha 24 do algoritmo 4. E $S(x)$ o conjunto de escravos de x após a conclusão da reorganização. Após a reorganização o número de escravo de w é:

$$\begin{aligned}|S(u)| &= |S'(u) - slavesToMove| \\ &= |S'(u)| - |slavesToMove| \\ &= |S'(u)| - min(availableSlots, availableSlaves) \\ &= |S'(u)| - min(K - |S'(w)|, |S(u)| - 2)\end{aligned}$$

E segue que:

1. $|S'(u)| < K$ Pois u era líder do componente A
2. $|S'(w)| < K$ Pois w era líder do componente B
3. $|S'(w)| > 1$ Pois $w \neq v$
4. $|S'(w)| + |S'(u)| > K$ Devido a condição da linha 20 do algoritmo 4
5. $K - |S'(w)| > 0$
6. $|S'(u)| > K - |S'(w)|$
7. $|S'(u)| > 1$
8. $1 < |S'(w)| < K$
9. $-1 > -|S'(w)| > -K$

10. $K-1 > K-|S'(w)| > K-K$

11. $K-1 > K-|S'(w)| > 0$

12. $0 < K-|S'(w)| < K-1$

13. $1 < |S'(u)| < K$

14. $1-2 < |S'(u)|-2 < K-2$

15. $-1 < |S'(u)|-2 < K-2$

16. $0 \leq |S'(u)|-2 < K-2$

17. $0 \leq min(K-|S'(w)|, |S'(u)|-2) < K-2$

18. $0 \geq -min(K-|S'(w)|, |S'(u)|-2) > -K+2$

19. $-K+2 < -min(K-|S'(w)|, |S'(u)|-2) \leq 0$

20. $|S'(u)|-K+2 < |S'(u)|-min(K-|S'(w)|, |S'(u)|-2) \leq |S'(u)|$

21. $|S'(u)|-min(K-|S'(w)|, |S'(u)|-2) \leq |S'(u)| \leq K-1$

22. $|S'(u)|-min(K-|S'(w)|, |S'(u)|-2) \leq K-1$

Portanto, conclui-se que $|S(u)| \leq K-1$. □

Lema 59. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 12, não existe mestre com mais de $K = 7$ escravos.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre m com mais de $K = 7$ escravos. Sabe-se que $m \neq u$, pois segundo o lema 58 $|S(u)| \leq K-1 \implies |S(u)| \leq K \implies m \neq u$.

$m \neq v$ pois nenhum escravo foi adicionado a v. Sendo v um escravo não compartilhado de w, $S(v) = \emptyset$.

Considerando $S'(x)$ o conjunto de escravos de x na linha 22 do algoritmo 4.

Na linha 75 do algoritmo 3 alguns escravos do líder u são desconectados de u e tornam-se escravos de w. E na linha 75, todos os antigos escravos de w tornam-se escravos de u. Após a reconexão o número de escravos de w é:

$$
\begin{aligned}
|S(w)| &= |S'(w) \cup slavesToMove| \\
&= |S'(w)| + |slavesToMove| \\
&= |S'(w)| + min(availableSlots, availableSlaves) \\
&= |S'(w)| + min(K - |S'(w)|, |S(u)| - 2) \\
&\leq |S'(w)| + K - |S'(w)| \\
&\leq K
\end{aligned}
$$

Dado que $|S(w)| \leq K$, $m \neq w$.

Para os outros mestres diferentes de u, w e v, nenhuma conexão é estabelecida ou destruída durante a reorganização. Portanto nós diferentes de u, v e w possuem um escravo após a reorganização se, e somente se, ele possui o escravo antes da reorganização. Como $m \notin \{u, w, v\}$ e depois da reorganização $|S(m)| > K$, então $|S(m)| > K$ antes da união $|S(m)| > K$. Dado que o componente A e B respeitam a propriedade 3, antes da reorganização todos os mestres em A e B possuem no máximo K escravos. Como m estava em A ou B antes da união, então $|S(m)| < K$.

Isto é uma contradição. Portanto a premissa é verdadeira. □

Lema 60. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 12, após a união (MERGE) o líder possui um escravo não compartilhado ou não possui nenhum escravo.*

*Demonstração.*Na linha 69 do algoritmo 3, w, o líder do componente B (linha 17 do algoritmo 2) é aposentado. Logo, u permanece como líder do componente após a conclusão da reorganização.

Na linha 73 do algoritmo 3 é definido o conjunto de nós *slavesToMove* que são desconectado do líder u na linha 74. y é o nó selecionado do conjunto $Y_p(u)$ na linha 26 do algoritmo 2. Por definição, todos os nós em $Y_p(u)$ são escravos não compartilhados de u. Portanto, após a reorganização o líder u possui pelo menos um escravos não compartilhado y. □

Lema 61. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 12, após a união (MERGE) todos os escravos não-compartilhados possuem h menor ou igual ao h de seu mestre.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre k com um escravo não compartilhado l tal que $h(l) > h(k)$.

$k \neq u$ pois nenhum escravo foi atribuído ao líder u a exceção de v. Portanto, um nó diferente de v é um escravo de u se, e somente se, ele é escravo de u antes da reorganização. Dado que o componente A respeita a propriedade 5 antes da reorganização, sabe-se que todos os escravos não compartilhados de u antes da reorganização possuem h menor ou igual a $h(u)$. Quanto a v, não é um escravo não-compartilhado de u.

A ocorrência da reorganização CASO 11 exige que as condições da linha 22 do algoritmo 4 sejam satisfeitas, portanto $h(v) < h(u)$ e $|S(u)| > 1$. $S(u) > 1$ e a propriedade 4 implicam que y possui um escravo não compartilhado, e portanto y é definido.

Todos os escravos de w, exceto v, são desconectados de w na linha 65 do algoritmo 3 e nenhum escravo é adicionado a w. Dado que antes da reorganização v é um escravo não compartilhado de w no componente B, e B respeita a propriedade 5, então $h(v) \leq h(w)$. Logo, o único escravo de w respeita a propriedade e $k \neq w$.

Os escravos de w após a união podem ser agrupados em: (1) escravos que eram escravos de w antes da união no componente A; (2) escravos que eram escravos de u antes da união no componente A.

Para nós em (1) a propriedade é verdadeira pois o componente B respeitava a propriedade 5 antes da união.

Para nós em (2) a propriedade é verdadeira pois a ocorrência do caso 12 exige que as condição na linha 24 do algoritmo 4 sejam satisfeitas. Logo $h(w) \geq h(u)$. Dado que o componente A respeita a propriedade 5, todos os escravos não compartilhados n de u possuem $h(n) \leq h(u)$. Por transitividade $h(w) \geq h(u) \implies h(w) \leq h(n)$.

Dado que todos os escravos não compartilhados de w possuem h menor ou igual à $h(w)$, $k \neq w$.

Para mestres diferentes de u, v e w: nenhum novo escravo lhes foi atribuído. Portanto um mestre diferente de u, v e w possui um escravo se, e somente se, ele possui um escravo antes da união. Dado que $k \neq u \wedge k \neq w \wedge k \neq v$, antes da união k foi mestre de l em A ou B. Dado que A e B respeitam a propriedade 5, $h(k) \geq h(l)$.

Isto é uma contradição, e portanto a premissa é verdadeira. □

A.12 CASO 13: $W \neq V \wedge |S(W)| + |S(U)| > K \wedge H(W) < H(U)$

No caso 13 u realiza o processo de INQUIRY, descobre o nó v e estabelece a conexão entre u e v representada pela linha tracejada. O mestre de v é w, líder do componente.

Não é possível unificar as *Piconets* de u e w em uma, pois há mais de

8 nós juntando-se as duas *Piconets*.

Ao contrário da reorganização 12 $h(w) < h(u)$. Portanto não é possível realocar os escravos de u para w, mas o contrário é. Neste caso, v não é mantido como ponte entre u e w. O nó v é deixado como escravo não compartilhado de w. w torna-se escravo de u.

Lema 62. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 13, após a união (MERGE) o componente resultante da união possui um líder.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe mais de um líder ou não existe nenhum líder.

Na linha 78 do algoritmo 3, u, líder do componente A (linha 17 do algoritmo 2), é aposentado. Antes da união, o componente B respeita todas as propriedades, e segundo a propriedade 1 pode-se concluir que após a finalização da reorganização, nenhum nó que pertencia a A é líder. Dado que: (1) não há nenhum líder pertencente a A; (2) a reorganização 1 não torna nenhum nó líder; e, (3) nenhum nó pertencente a B é aposentado; todos líderes de B são líderes do componente após união. Como nenhum nó pertencente ao componente A é líder, todos os líderes do componente após a união são líderes do componente B. Logo, antes da reorganização o componente B não possui nenhum líder ou possui pelo menos 2 líderes. Isto é uma contradição, pois sabe-se devido a propriedade 1 que o componente B possui exatamente 1 líder. Conclui-se que a premissa é verdadeira e existe um líder após o término da reorganização. □

Lema 63. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 13, após a união (MERGE) o líder possuirá no máximo $K-1=6$ escravos.*

*Demonstração.*O líder após finalizada a reorganização é w, pois entre os dois líderes u e w, apenas u é aposentado na linha 78 do algoritmo 3.

Na linha 83 escravos são desconectados de w e reconectados a u na linha 84. O número de escravos movidos de u para w é igual ao menor valor entre as variáveis *availablesSlots* (79) e *availableSlaves* (84)).

Considerando $S'(x)$ o conjunto de escravos de x na linha 24 do algoritmo 4. E $S(x)$ o conjunto de escravos de x após a conclusão da reorganização. Após a reorganização o número de escravo de w é:

$$\begin{aligned}
|S(w)| &= |S'(w) - slavesToMove| \\
&= |S'(w)| - |slavesToMove| \\
&= |S'(w)| - min(availableSlots, availableSlaves) \\
&= |S'(w)| - min(K - |S'(w)|, |S(u)| - 2)
\end{aligned}$$

E segue que:

1. $|S'(u)| < K$ Pois u era líder do componente A
2. $|S'(w)| < K$ Pois w era líder do componente B
3. $|S'(w)| > 1$ Pois $w \neq v$
4. $|S'(w)| + |S'(u)| > K$ Devido a condição da linha 26 do algoritmo 4
5. $K - |S'(w)| > 0$
6. $|S'(u)| > K - |S'(w)|$
7. $|S'(u)| > 1$
8. $1 < |S'(u)| < K$
9. $-1 > -|S'(u)| > -K$
10. $K - 1 > K - |S'(u)| > K - K$
11. $K - 1 > K - |S'(u)| > 0$
12. $0 < K - |S'(u)| < K - 1$
13. $1 < |S'(w)| < K$
14. $1 - 2 < |S'(w)| - 2 < K - 2$
15. $-1 < |S'(w)| - 2 < K - 2$
16. $0 \leq |S'(w)| - 2 < K - 2$
17. $0 \leq min(K - |S'(u)|, |S'(k)| - 2) < K - 2$
18. $0 \geq -min(K - |S'(u)|, |S'(k)| - 2) > -K + 2$
19. $-K + 2 < -min(K - |S'(u)|, |S'(k)| - 2) \leq 0$
20. $|S'(w)| - K + 2 < |S'(w)| - min(K - |S'(u)|, |S'(k)| - 2) \leq |S'(w)|$
21. $|S'(w)| - min(K - |S'(u)|, |S'(k)| - 2) \leq |S'(w)| \leq K - 1$
22. $|S'(w)| - min(K - |S'(u)|, |S'(k)| - 2) \leq K - 1$

Portanto, $|S(w)| \leq K - 1$. □

Lema 64. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 13, não existe mestre com mais de $K = 7$ escravos.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre m com mais de $K = 7$ escravos. Sabe-se que $m \neq w$, pois segundo o lema 58 $|S(w)| \leq K-1 \implies |S(w)| \leq K \implies m \neq w$.

$m \neq v$ pois nenhum escravo foi adicionado a v. Sendo v um escravo não compartilhado de w, $S(v) = \emptyset$.

Considerando $S'(x)$ o conjunto de escravos de x na linha 26 do algoritmo 4.

Na linha 84 do algoritmo 3 alguns escravos do líder u são desconectados de u e tornam-se escravos de w. E na linha 84, todos os antigos escravos de w tornam-se escravos de u. Após a reconexão o número de escravos de u é:

$$\begin{aligned}
|S(u)| &= |S'(u) \cup slavesToMove \cup \{w\} - \{v\}| \\
&= |S'(u)| + |slavesToMove| + |\{w\}| - |\{v\}| \\
&= |S'(u)| + min(availableSlots, availableSlaves) \\
&= |S'(u)| + min(K - |S'(u)|, |S(k)| - 2) \\
&\leq |S'(u)| + K - |S'(u)| \\
&\leq K
\end{aligned}$$

Dado que $|S(w)| \leq K$, $m \neq u$.

Para os outros mestres diferentes de u, w e v, nenhuma conexão é estabelecida ou destruída durante a reorganização. Portanto nós diferentes de u, v e w possuem um escravo após a reorganização se, e somente se, ele possui o escravo antes da reorganização. Como $m \notin \{u, w, v\}$ e depois da reorganização $|S(m)| > K$, então $|S(m)| > K$ antes da união $|S(m)| > K$. Dado que o componente A e B respeitam a propriedade 3, antes da reorganização todos os mestres em A e B possuem no máximo K escravos. Como m estava em A ou B antes da união, então $|S(m)| < K$.

Isto é uma contradição. Portanto a premissa é verdadeira. □

Lema 65. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 13, após a união (MERGE) o líder possui um escravo não compartilhado ou não possui nenhum escravo.*

*Demonstração.*Na linha 78 do algoritmo 3, u, o líder do componente A (linha 17 do algoritmo 2) é aposentado. Portanto, w permanece como líder do componente após o término da reorganização.

No início do procedimento de reorganização v é escravo de w e u ($M(v) = \{u, w\}$) e não possui escravos ($S(v) = \emptyset$). Na linha 85, v é desco-

nectado de u. Logo, $M(v) = \{w\}$. Ou seja, v é escravo não compartilhado do líder w. □

Lema 66. *Dados dois componentes A e B que respeitam as propriedades antes da união. Se ocorrer uma reorganização CASO 13, após a união (MERGE) todos os escravos não-compartilhados possuem h menor ou igual ao h de seu mestre.*

*Demonstração.*A prova é por contradição. Assume-se que a premissa é falsa: após a união existe um mestre k com um escravo não compartilhado l tal que $h(l) > h(k)$.

$k \neq w$ pois nenhum escravo foi atribuído ao líder w. Portanto, um nó é um escravo de w se, e somente se, ele é escravo de w antes da reorganização. Dado que o componente B respeita a propriedade 5 antes da reorganização, sabe-se que todos os escravos não compartilhados de w antes da reorganização possuem h menor ou igual a $h(w)$.

A ocorrência da reorganização CASO 11 exige que as condições da linha 22 do algoritmo 4 sejam satisfeitas, portanto $h(v) < h(u)$ e $|S(u)| > 1$. $S(u) > 1$ e a propriedade 4 implicam que y possui um escravo não compartilhado, e portanto y é definido.

Todos os escravos de w, exceto v, são desconectados de w na linha 65 do algoritmo 3 e nenhum escravo é adicionado a w. Dado que antes da reorganização v é um escravo não compartilhado de w no componente B, e B respeita a propriedade 5, então $h(v) \leq h(w)$. Logo, o único escravo de w respeita a propriedade e $k \neq w$.

Os escravos de u após a união podem ser agrupados em: (1) escravos que eram escravos de u antes da união no componente A; (2) escravos que eram escravos de w antes da união no componente A; e, (3) w.

Para nós em (1) a propriedade é verdadeira pois o componente A respeitava a propriedade 5 antes da união.

Para nós em (2) a propriedade é verdadeira pois a ocorrência do caso 13 exige que as condição na linha 26 do algoritmo 4 sejam satisfeitas. Logo $h(w) < h(u)$. Dado que o componente B respeita a propriedade 5, todos os escravos não compartilhados n de w possuem $h(w) > h(n)$. Por transitividade $h(u) > h(w) \implies h(u) \geq h(n)$.

Para o caso (3), w não é um escravo não compartilhado. Portanto $h(w)$ é irrelevante para a propriedade 5.

Dado que todos os escravos não compartilhados de w possuem h menor ou igual à $h(w)$, $k \neq w$.

Para mestres diferentes de u, v e w: nenhum novo escravo lhes foi atribuído. Portanto um mestre diferente de u, v e w possui um escravo se, e somente se, ele possui um escravo antes da união. Dado que $k \neq u \wedge k \neq$

$w \wedge k \neq v$, antes da união k foi mestre de l em A ou B. Dado que A e B respeitam a propriedade 5, $h(k) \geq h(l)$.

Isto é uma contradição, e portanto a premissa é verdadeira. □

A.13 PROPRIEDADES DO ALGORITMO

As organizações são projetadas de tal forma que as propriedades são mantidas após as reorganizações serem concluídas. Isto permite que as uniões de componentes ocorram indefinidamente.

Lema 67. *Dado que dois componentes A e B respeitam as propriedades antes da união, segue que: (1) o componente resultante possui um único líder; (2) o líder do componente resultante possui no máximo seis escravos; (3) não existe nenhum mestre com mais de sete escravos no componente resultante; (4) o líder do componente resultante possui pelo menos um escravo não-compartilhado; (5) todos os escravos do componente resultante que não são pontes possuem h menor ou igual ao h de seu mestre.*

*Demonstração.*Propriedade 1 é válida para todas as 13 reorganizações possíveis em uma união em decorrência dos lemas 1, 7, 12, 17, 22, 27, 32, 37, 42, 47, 52, 57, 62.

Propriedade 2 é válida para todas as 13 reorganizações possíveis em uma união em decorrência dos lemas 2, 8, 13, 18, 23, 28, 33, 38, 43, 48, 53, 58, 63.

Propriedade 3 é válida para todas as 13 reorganizações possíveis em uma união em decorrência dos lemas 3, 9, 14, 19, 24, 29, 34, 39, 44, 49, 54, 59, 64.

Propriedade 4 é válida para todas as 13 reorganizações possíveis em uma união em decorrência dos lemas 4, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65.

Propriedade 5 é válida para todas as 13 reorganizações possíveis em uma união em decorrência dos lemas 5, 11, 16, 21, 26, 31, 36, 41, 46, 51, 56, 61, 66. □

APÊNDICE B – Protótipo e Experimentação do Sistema de Coleta de Matrizes OD

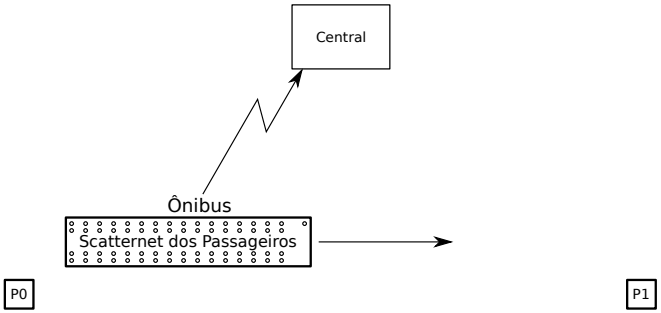


Figura 39: Ilustração do sistema de coleta de informações em ônibus baseado em *crowdsensing*.

Conforme descrito no capítulo 2, algumas informações sobre os passageiros em sistemas de transporte coletivo são de difícil obtenção usando tecnologias e equipamentos atualmente disponíveis. Em especial, informações detalhadas sobre o trajeto de passageiros em sistemas de transporte coletivo baseados em ônibus são de difícil obtenção. Ou seja, operadores e planejadores de sistemas de transportes não tem acesso, usando tecnologias atualmente disponíveis, a dados sobre matrizes OD.

Neste anexo descrevemos o protótipo desenvolvido para o testar o sistema para coleta de dados e levantamento de matrizes OD e informações sobre passageiros descrito no capítulo 2.

B.1 SISTEMA DE COLETA BASEADO EM *CROWDSENSING*

O sistema de coleta de informações sobre passageiros de ônibus, esquematizado na Figura 39, é um sistema distribuído composto por um número qualquer de *smartphones* e uma central de processamento. Os *smartphones* se comunicam através da tecnologia *Bluetooth* e usam a rede celular para envio da informação à uma central.

Os *smartphones* dos passageiros são usados para determinar os passageiros embarcados entre paradas. Conforme demonstrado no capítulo 2, a partir da informação sobre os passageiros embarcados entre paradas é possível determinar um conjunto de informações úteis a operadores e planejadores de sistemas de transporte coletivo. Por exemplo, matrizes origem-destino ou número de passageiros embarcados. *Smartphones* são flexíveis e ubíquos. Seu poder de processamento aliado aos sensores lhes permitem analisar o ambiente no qual são inseridos, a partir dos dados de seus sensores e pelas

potencialidades em comunicar com *smartphones* vizinhos e em executar algoritmos distribuídos.

O conjunto de passageiros embarcados muda a cada embarque e desembarque, que ocorrem nas paradas de ônibus. Portanto, há uma restrição temporal para que a coleta de informações dos passageiros no ônibus seja realizada. O tempo para finalizar a determinação do conjunto é igual ao tempo que o ônibus leva para se deslocar de um ponto de parada ao próximo.

A adoção de *smartphones* possibilita que o sistema de coleta não dependa de equipamentos instalados na infraestrutura ou nos ônibus. Isto visa facilitar a implantação e manutenção do sistema.

O levantamento colaborativo do conjunto de passageiros no interior do ônibus é feito em três etapas: (1) formação e manutenção automática de uma rede de comunicação entre os *smartphones* dos passageiros embarcados; (2) determinação do conjunto de passageiros no ônibus; (3) envio do conjunto de passageiros à central para processamento e determinação da matriz OD.

B.1.0.0.1 Formação automática da rede ad-hoc:

A utilização de redes *ad-hoc* dos *smartphones* a bordo permite identificar os passageiros sem que haja necessidade de infraestrutura de rede nos ônibus. O primeiro passo é determinar se o *smartphone* está no interior de um ônibus. Os trabalhos de (REDDY et al., 2010; THIAGARAJAN et al., 2010; SHAFIQUE; HATO, 2015) apresentam métodos de classificação do meio de transporte no qual o *smartphone* se encontra a partir de dados de seus sensores.

Uma vez que a classificação indica que um *smartphone* está em um ônibus, a interface *Bluetooth* é ativada e uma rede entre os passageiros embarcados é formada.

B.1.0.0.2 Determinação do conjunto de passageiros no ônibus:

Uma vez que a rede espontânea entre os passageiros é estabelecida é possível fazer um paralelo entre estar embarcado em um ônibus e a pertinência à rede espontânea formada no interior do veículo. O ingresso e saída de um *smartphones* na rede *ad-hoc* implica no ingresso e saída do passageiro do ônibus.

Através desta analogia, a determinação do conjunto de passageiros embarcados no interior do veículo, resume-se a determinação dos *smartphones* em uma rede *ad-hoc*.

Dado que o objetivo é identificar os passageiros embarcados entre paradas, o sistema deve determinar se ônibus está em processo de embarque e desembarque ou se ele está se deslocando entre paradas.

B.1.0.0.3 Envio das informações à central:

Uma vez que os passageiros foram identificados, esta informação deve ser enviada à central. Usualmente, em sistemas baseados em *crowdsensing* descritos na literatura, cada *smartphone* realiza o envio da informação. Entretanto, devido a existência de uma rede de comunicação entre os passageiros, é possível ao nosso sistema otimizar o envio evitando que a mesma informação seja enviada múltiplas vezes à central. Um dentre os passageiros é escolhido para enviar a mensagem a central.

A central é responsável por receber, processar, armazenar e disponibilizar as informações enviados pelos *smartphones* aos interessados em um formato útil. Aos planejadores do sistema de transporte coletivo, o principal objetivo do protótipo está na determinação do número de passageiros embarcados e matriz Origem-Destino. A central deve, ao receber os integrantes, utilizar as observações acerca da reconstrução da matriz OD a partir da identificação dos passageiros embarcados feitas no capítulo 2.

B.2 PROTÓTIPO DESENVOLVIDO

O protótipo é formado por duas partes complementares: um aplicativo instalado em cada *smartphone* que permite a criação da rede sobre *Bluetooth*; e, um servidor HTTP que coordena e coleta dados dos experimentos através de uma conexão WiFi com os *apps*.

Em cada *smartphone*, há uma camada composta pelos protocolos do Beddernet que possuem, entre outros, o algoritmo de roteamento DSDV. No protótipo, as funcionalidades do protocolo Beddernet foram estendidos e melhoradas. As principais mudanças são:

- **Sequenciamento das operação na pilha *Bluetooth*:** Há operação na pilha *Bluetooth* que devem ser executadas em passos mutuamente exclusivos. Por exemplo, a busca e estabelecimento de conexões não devem ocorrer simultaneamente. O protocolo Beddernet permite a ocorrência simultânea destas operações em alguns casos. Para evitar este comportamento, o gerenciamento das conexões *Bluetooth* foi modificado de tal forma que a implementação respeita as restrições das operações impostas pela especificação *Bluetooth*.

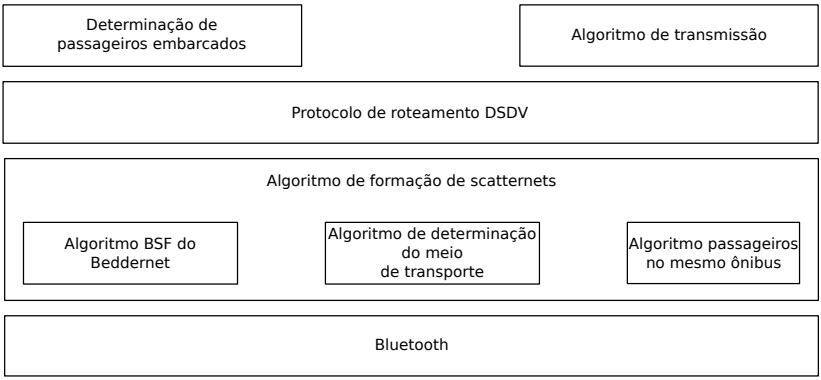


Figura 40: Estrutura em camadas do aplicativo operando no *smartphones*.

- **Notificações de eventos:** Mecanismos foram adicionados para notificar *apps* de eventos ocorridos na rede *ad-hoc* na qual o dispositivos se insere. Os *apps* são notificados da conexão de novos dispositivos, mudanças na tabela de roteamento do *smartphone*, e falha no canal de comunicação com outro dispositivo. O sistema de notificação foi implementado usando o mecanismos de Intents fornecido pelo Android para comunicação e notificação entre diferentes *apps*.

O aplicativo de cada *smartphone* é organizado em três componentes, conforme Figura 40: (1) **Formação da rede *ad-hoc* entre *smartphones* -** responsável por formar a rede *ad-hoc* entre os passageiros embarcados; (2) **Estimador -** responsável pela análise dos sensores e da rede *ad-hoc* para identificar um conjunto de passageiros embarcados entre paradas consecutivas; (3) **Protocolo de envio -** responsável pela transmissão a central, do conjunto de passageiros acordado.

B.2.1 Formação da rede *ad-hoc* entre *smartphones*

Utilizou-se o *middleware* Beddernet (GOHS; GUNNARSSON; GLENSTRUP, 2011) para a formação e manutenção da rede *ad-hoc* entre os dispositivos. Beddernet é um software open-source para Android que busca o estabelecimento de um rede *ad-hoc Bluetooth* para aplicações. Além dos algoritmos para formação de rede, Beddernet fornece algoritmo de roteamento DSDV.

O objetivo deste bloco é a formação de uma rede entre passageiros. Portanto, antes de permitir o ingresso de um *smartphone* na rede, realiza-se alguns testes de modo a determinar se os passageiros estão juntos no interior

de um mesmo ônibus. Esta operação é realizada em duas etapas. Em um primeiro momento, o meio de transporte no qual o *smartphone* está operando é determinado usando o algoritmo descrito em (REDDY et al., 2010).

O aspecto de determinação se passageiros estão embarcados em um mesmo veículo foge do escopo da tese e por este motivo não foi mais explorado. Trabalhos futuros se fazem necessários para avaliar se a correlação é suficientemente precisa para os fins, e determinação de métodos alternativos.

B.2.2 Determinação do conjunto de passageiros no ônibus

O Estimador é responsável por identificar os passageiros embarcados no ônibus entre paradas. O algoritmo usado no protótipo parte da observação que o conjunto de passageiros do ônibus muda apenas durante o embarque e desembarque. Consequentemente, a rede *ad-hoc* formada pelos *smartphones* dos passageiros é modificada apenas durante embarques e desembarques. Logo, entre paradas, a tabela de roteamento do DSDV do Beddernet não é modificado.

O algoritmo monitora a tabela de roteamento para identificar os períodos nas quais a tabela não está sendo modificada. Utiliza-se um temporizador com tempo de estabilização mínimo ϕ que é um parâmetro de ajuste do algoritmo. A cada vez que uma mudança na tabela de roteamento é identificada o temporizador é reiniciado. Quando o temporizador atinge o tempo ϕ sem reinícios, os nós contidos na tabela de roteamento DSDV são classificados como passageiros. Da mesma forma, quando mudanças ocorrem identifica-se o início do embarque e desembarque.

O algoritmo usado no protótipo considera que apenas *smartphones* de passageiros de um mesmo ônibus conseguem ingressar na rede *ad-hoc*.

B.2.3 Protocolo de envio

O protocolo de envio é responsável pela transmissão do conjunto de passageiros à central de processamento.

A existência de uma rede de comunicação entre os passageiros permite otimizar o número de mensagens enviadas à central de processamento através de um processo de eleição de líder. Este líder é responsável pelo envio.

Além da informação do conjunto de passageiros outras informações são anexadas à mensagem. Por exemplo, o traço de GPS do ônibus desde a última parada, e estampilha de tempo do momento em que o conjunto de passageiros foi determinado.

B.2.4 Central de processamento

A central de processamento foi prototipada usando a tecnologia NodeJS e o protocolo HTTP para transmissão e recepção dos dados. Ao receber o conjunto de endereços *Bluetooth* dos participantes, o servidor o armazena junto com o horário de recebimento.

A matriz OD é gerada usando a premissa que cada conjunto de passageiros embarcados recebido corresponde a um trecho entre as paradas.

A central de processamento do protótipo foi desenvolvida para prova de conceito e não implementa as funções de tratamento de dados avançadas que seriam necessárias em sistema real. Por exemplo, a premissa anterior impede que o sistema colete informações de dois ônibus simultaneamente.

Em uma implementação real do sistema, deve ser possível coletar informações de ônibus distintos em uma mesma linha, e em linhas diferentes. Além disso, os dados devem ser tratados e analisados para tratar situações excepcionais. Por exemplo, quando há mais de uma rede *ad-hoc* dentro do mesmo veículo. Esta situação pode ocorrer quando o algoritmo não consegue conectar todos os passageiros a tempo de chegar a próxima parada. Devido a natureza distribuída do processo, cada rede acredita ser a única no interior do veículo já que não conseguiu detectar informações acerca da outra. Logo a central receberá dois conjuntos de passageiros pertencentes ao mesmo veículo. Idealmente a central é capaz de determinar que estes conjuntos correspondem ao mesmo veículo baseado em dados de geolocalização e tempo enviados à ele pelos *smartphones*.

B.3 RESULTADOS E OBSERVAÇÕES EM EXPERIMENTOS COM O PROTÓTIPO

Testes foram conduzidos com até doze dispositivos (8 *smartphones* e 4 tablets) rodando o sistema operacional Android. Os testes foram conduzidos em laboratório. Os *smartphones* foram distribuídos ao longo de um corredor com largura e comprimento similares a de um ônibus. O servidor HTTP atua como a Central coletando os dados enviados pelos *smartphones*.

Além do conjunto de passageiros determinado pelos *smartphones* estarem embarcados no veículo, os *smartphones* registram a ocorrência de eventos junto a uma estampilha de tempo. Estes eventos descrevem mudanças no estado do sistema. Por exemplo, eventos de estabelecimento de conexões *Bluetooth* ou o recebimento de uma atualização da tabela DSDV são registrados.

Um sistema para auxiliar a execução da simulação foi desenvolvido em um servidor HTTP. Sua tarefa é sincronizar o início do experimento e

posteriormente coletar e armazenar os eventos registrados pelos *smartphones*.

Cada *smartphone* possui um identificador único atribuído pelo servidor. Este identificador é usado para demarcar a origem dos eventos recebidos. Além dos eventos, o servidor também recebe o conjunto de passageiros determinados pelos *smartphones* embarcados. Este é conjunto de identificadores atribuídos ao servidor.

O objetivo do experimento é analisar os tempos envolvidos no processo de formação de uma rede no interior do ônibus. Uma viagem foi simulada em um ambiente controlado com número variável de passageiros. O cenário foi projetado de forma que todos os passageiros estão inicialmente desconectados. Ao iniciar o experimento, os dispositivos são informados que estão no interior de um ônibus e iniciam a atividade de estabelecimento da rede.

Estes experimentos pressupõem que todos os dispositivos são visíveis e pareados um com outro. Isto é necessário devido a restrições tecnológicas impostar pela plataforma Android que requer a existência de um pareamento entre os dispositivos participantes para permitir o estabelecimento de conexões *Bluetooth*. O procedimento de pareamento *Bluetooth* requer a confirmação do usuário para ser completada. Na prática, esta restrição impede que a formação automática de *Scatternets Bluetooth* entre dispositivos não pareados. Visibilidade impacta o protocolo de formação de *Scatternets* dado que dispositivos *Bluetooth* só descobrem dispositivos próximos que estão visíveis. Uma confirmação do usuário é requerida pela plataforma para tornar o dispositivo visível. Portanto, uma implementação real do sistema requer o uso de um sistema operacional que permita à aplicações autorizadas pelo usuário a estabelecer conexões *Bluetooth* sem intervenção do usuário.

O início do experimento é sincronizado, de modo a melhor avaliar os resultados. Todos os *smartphones* iniciam a execução do seu protocolo aproximadamente ao mesmo tempo. Isto é feito usando o servidor como mediador para o início das atividades. Primeiramente, o número de *smartphones* participantes do experimento é registrado no servidor HTTP. Então o servidor entra no estado de espera por conexões. Cada *smartphone* é ligado para dar início ao experimento. Neste instante, uma requisição é feita ao servidor informando que o dispositivo está pronto para iniciar. Após o envio da mensagem, o *smartphone* para suas atividades e aguarda o sinal de início do servidor HTTP. No momento em que o servidor recebe uma conexão de cada um dos participantes, uma resposta é enviado a todos os participantes simultaneamente para iniciar a simulação. Este método não é perfeito, pois os há atrasos e perdas causadas pela transmissão de pacotes através da rede de comunicação WiFi usada na comunicação entre os *smartphones* e o servidor HTTP. Logo, dispositivos não recebem a mensagem do servidor HTTP exa-

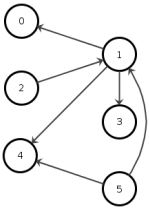


Figura 41: *Scatternet* formada após execução do experimento com 6 dispositivos usando o protótipo

tamente ao mesmo tempo. Entretanto, para fins deste experimento a precisão alcançada com esta sincronização é suficiente. As diferenças entre o horário de recebimento das mensagens é na casa de milissegundos, enquanto o tempo envolvido nos procedimentos de estabelecimento de conexão *Bluetooth* estão na casa dos segundos.

Além da sincronização, os relógios dos *smartphones* são sincronizados via NTP; considera-se o drift do relógio irrelevante e a sincronização suficientemente precisa para determinar a ordem da ocorrência dos eventos entre *smartphones* diferentes.

A medida que o tempo passa, passageiros se conectam um ao outro, até que todos os passageiros estejam conectados. O tempo necessário pelo algoritmo para formar uma *Scatternet* com todos os passageiros é medido nos experimentos.

No que segue, descreve-se os eventos ocorridos durante um experimento realizado com 6 passageiros embarcados. Cada dispositivo recebe um número entre 0 e 5. Nesta execução em particular, o passageiro 1 iniciou a busca por vizinhos e conectou-se aos passageiros 2 e 5 após 13 segundos, formando uma *Piconet*. 3 segundos depois, o passageiro 3 encontrou e conectou-se ao passageiro 1 formando outra *Piconet*. Neste ponto, um *Scatternet* é formado contendo duas *Piconets* com os passageiros 1 e 3 desempenhando o papel de mestres das *Piconets*. Este processo de descoberta e conexão entre passageiros durou 21 segundos, quando a *Scatternet* tornou-se totalmente conectada.

A topologia resultante desta execução é ilustrada na Figura 41. Nesta figura, cada círculo representa um *smartphone* e as setas representam as conexões *Bluetooth* entre os *smartphones*. A orientação da seta representa a relação mestre e escravo. A seta parte do escravo apontando para o mestre. A direção não representa uma comunicação mão única; pelo contrário, nós adjacentes podem tanto transmitir dados quanto receber.

A Figura 42 mostra o tempo necessário para formar a *Scatternet* em

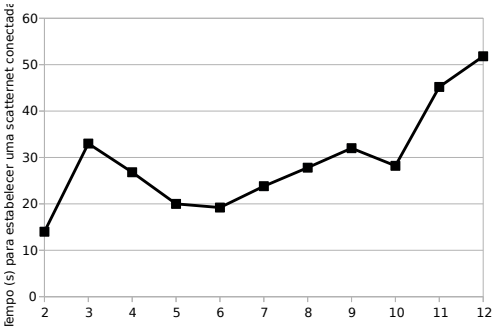


Figura 42: Tempo médio necessário para que o experimento com o protótipo forme uma *Scatternet* conectada

relação ao número de *smartphones* embarcados, de 2 até o máximo de 12 dispositivos. Cada experimento foi repetido 5 vezes e as médias calculadas. Salientamos que em alguns experimentos o tempo para formação da rede superaram os 3 minutos sem que a rede tenha sido completamente formada. Estes casos foram considerados como falhas do experimento e ignorados. Estes casos ocorrem com certa frequência com o aumento no número de *smartphones*, em especial durante os experimentos com doze *smartphones*. A formação da *Scatternet* mais veloz ocorre quanto dois dispositivos estão presentes. Com alguns *smartphones* a mais, a natureza probabilista do algoritmo de formação de *Scatternet* usada pelo Beddernet, aumenta as chance dos *smartphones* entrarem em estado de espera, impedindo-os de realizar buscas por dispositivos próximos. O processo de descoberta é diretamente impactado pela proporção entre os *smartphones* que ingressaram no estado de espera, e dispositivos que ingressaram no estado de descoberta. Já que dispositivos que se encontram no estado de descoberta só são capazes de encontrar dispositivos em espera devido a limitações impostas pelo protocolo *Bluetooth*. Isto explica a diminuição no tempo necessário para formar a rede apesar do aumento no número de passageiros embarcados. Tempos necessário para estabelecer a rede diminui com o aumento no número de *smartphones*, até que o número de 6 dispositivos seja atingido. Após este ponto, o aumento no número de conexões necessário para construir a rede torna-se dominante. E o tempo para formação da *Scatternet* começa a aumentar. Outro fator relevante que impacta o algoritmo é que o número de dispositivos que precisam ser descobertos usando *Bluetooth*. O tempo necessário para formar uma rede aumenta linearmente com o número de dispositivos.

Ao fim dos experimentos, a central de processamento recebeu o conjunto de passageiros enviado pelos *smartphones*. Todos as vezes que o sis-

tema foi executado, os dispositivos foram corretamente identificados. Devido ao experimento ser executado em ambiente controlado, o mecanismo de detecção de modo de locomoção e se dispositivos estão embarcados no mesmo veículo foram desabilitados. Em um experimento real, estes dispositivos introduzirão erros. Estes erros se refletirão nos valores recebidos pela central. Estudos adicionais sobre o impacto destes erros para os operadores do sistema de transporte coletivo, e mecanismos para minimizá-los se fazem necessários.

B.4 CONSIDERAÇÕES FINAIS

Neste capítulo apresentamos uma solução inédita para o problema de levantamento de matrizes OD e informações sobre passageiros no interior do ônibus usando o paradigma de *crowdsensing*.

O aplicativo atua usando como base uma rede de comunicação espontânea formada entre passageiros. Através da implementação de um protótipo mostramos que a solução é viável. Os experimentos realizados com o protótipo demonstraram que para a atuação efetiva do sistema, um algoritmo de formação de redes *Bluetooth* mais eficiente deve ser adotado para formação de redes no interior do veículo. Nos experimentos realizados levou-se aproximadamente 1 minuto para estabelecer a rede entre 12 passageiros. Caso contrário, é maior a possibilidade de a rede entre os passageiros não ser completamente estabelecida entre paradas, o que levaria a uma queda de performance do sistema e uma maior complexidade do sistema de tratamento de dados da central. A central precisa nestes casos identificar que grupos de passageiros próximos pertencem ao mesmo ônibus e mesclá-los para apresentar dados corretos aos operadores do sistema de transporte coletivo.

Estudos futuros são necessários para viabilizar uma implementação prática do protótipo. Experimentos realizados em ambientes reais de operação, no interior de ônibus, são necessário para averiguar aspectos de corretude e erro introduzidos pelos algoritmos de estimação de passageiros.

Além disso, a natureza distribuída da computação abre margem a alguns problema de consistência de dados entre passageiros no interior do veículo. O dispositivo de um passageiro pode ter uma visão diferenciada do conjunto de passageiros embarcados que outro passageiro devido a diferença no conjunto de mensagens recebidas. Estas diferenças ocorrem naturalmente devido a atrasos e perdas de pacotes. Portanto estudos mais detalhados sobre como tratar este caso se fazem necessário.

Durante o desenvolvimento desta tese, avaliou-se a possibilidade da inclusão de uma etapa de consenso após cada dispositivo determinar o conjunto de nós. Neste fase, dispositivos trocam informações dos passageiros

embarcados e determinam um conjunto único entre todos para envio à central. Simulações foram realizadas para avaliar o impacto da abordagem no sistema em termos de número de mensagens trocadas. Chegou-se a conclusão que os benefícios obtidos no uso do consenso não se justificam frente ao custo em termos de número de mensagens trocadas. Algoritmo de paxos (LAMPORT, 2001), por exemplo, possuem complexidade quadrática do número de mensagens. E devido a estrutura de comunicação *Bluetooth* baseada em *Piconets* e comunicação mestre-escravo, o número de mensagens que podem ser trocadas simultaneamente é limitado.