UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Ângelo dos Santos Melo

MÉTODO DE SENSORIAMENTO DE PASSAGEIROS DO TRANSPORTE PÚBLICO COM USO DE *SMARTPHONES* E BLUETOOTH

Florianópolis

2016

Ângelo dos Santos Melo

MÉTODO DE SENSORIAMENTO DE PASSAGEIROS DO TRANSPORTE PÚBLICO COM USO DE *SMARTPHONES* E BLUETOOTH

Dissertação submetida ao Programa de Pós-graduação em Engenharia de Automação e Sistemas para a obtenção do Grau de Mestre em Engenharia de Automação e Sistemas.
Orientador: Prof. Werner Kraus Jr, Dr.
Coorientador: Prof. Jean-Marie Alexandre Farines, Dr.

Florianópolis

2016

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

Melo, Ângelo dos Santos
Método de Sensoriamento de Passageiros do Transporte Público com Uso de smartphones e Bluetooth / Ângelo dos Santos Melo ; orientador, Werner Kraus Jr ; coorientador, Jean-Marie Farines. - Florianópolis, SC, 2016.
111 p.

Dissertação (mestrado) - 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. Sensoriamento Urbano. 3. Bluetooth. 4. Android. 5. Matriz O/D. 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.

Ângelo dos Santos Melo

MÉTODO DE SENSORIAMENTO DE PASSAGEIROS DO TRANSPORTE PÚBLICO COM USO DE *SMARTPHONES* E BLUETOOTH

Esta Dissertação foi julgada aprovada para a obtenção do Título de "Mestre em Engenharia de Automação e Sistemas", e aprovada em sua forma final pelo Programa de Pós-graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina.

Florianópolis, 17 de setembro 2016.

Prof. Werner Kraus Jr, Dr.
Orientador

Prof. Jean-Marie Alexandre Farines, Dr.
Coorientador

Prof. Daniel Ferreira Coutinho, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia de Automação e Sistemas

Banca Examinadora:

Prof. Werner Kraus Jr, Dr.
Presidente

Prof. Odilson Tadeu Valle, Dr. - IFSC/Florianópolis

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

Prof. Rodrigo Castelan Carlson, Dr. - DAS/UFSC

Às minhas avós, Maria Izabel da Silva Melo, primeira professora da comunidade do Rio Cubatão, Águas Mornas e Neide Zacchi dos Santos, *in memoriam*, professora de séries iniciais do grupo escolar de Palhoça

AGRADECIMENTOS

Agradeço em primeiro lugar a Deus *pois Dele, por Ele e para Ele são todas as coisas.*

Agradeço ao professor Werner Kraus Jr pela oportunidade de trabalharmos juntos e pela orientação realizada com grande otimismo e ao professor Jean-Marie Farines pela coorientação criteriosa, que elevou a qualidade deste trabalho.

Agradeço aos professores Carlos Montez, Rodrigo Carlson e Odilson Valle, que compuseram a banca examinadora. Suas sugestões e correções foram de grande valia!

Agradeço também ao professor Rômulo de Oliveira a acolhida recebida no retorno ao PGEAS.

Gostaria de agradecer também ao colega Giovani Pieri pelas assistências prestadas, especialmente sobre a tecnologia Bluetooth, que contribuíram grandemente na elaboração deste trabalho.

Por fim, um agradecimento especial à pessoa que me incentivou a realizar o mestrado e me apoiou durante todo este período de estudos, minha esposa Bruna.

Don't tell people how to do things, tell them what to do and let them surprise you with their results

(George S. Patton Jr)

RESUMO

Dados operacionais são fundamentais para o planejamento e gerenciamento de sistemas de transporte público coletivo. No entanto, o emprego de métodos tradicionais de levantamento de dados apresenta altos custos orçamentários e ineficiência quanto a obtenção de dados como origens e destinos de passageiros.

Observa-se, por outro lado, que a crescente popularidade de dispositivos móveis como *smartphones* possibilita um novo método de levantamento de dados de utilização de ônibus através de uma estratégia de *crowdsensing* que emprega os dispositivos móveis portados pelos passageiros como plataforma de aquisição de dados.

Dentre as tecnologias de comunicação disponibilizadas pelos dispositivos, Bluetooth provê suporte à comunicação de dados com baixo custo e baixo consumo de energia.

Neste trabalho, propõe-se um sistema de aquisição de dados operacionais do transporte coletivo através do uso de *smartphones* portados pelos passageiros. Com o propósito de avaliar a adequabilidade de tal tecnologia no problema de aquisição de dados de ônibus, implementa-se um protótipo, sobre o qual realiza-se uma série de testes de desempenho e consumo de energia. A estratégia de coleta de dados de interesse é avaliada através da utilização do protótipo do aplicativo em simulação de itinerários, onde são obtidos resultados que indicam a viabilidade da arquitetura proposta.

Palavras-chave: Bluetooth. Android. Aplicações para dispositivos móveis. *Crowdsensing*. *Participatory Sensing*. Sensoriamento Urbano. Mobilidade Urbana. Sistemas inteligentes de transporte. Estimativas de origens e destinos. Java SE. Bluecove.

ABSTRACT

Operational data are a key point on planning and maintenance of bus public transit systems. However, the employment of conventional gathering data methods presents high budget costs and inefficiency on gathering data such as origins and destinations of passengers.
On the other hand, we note that the growing popularity of mobile devices, such as smartphones, allows a new data gathering method of bus usage through a crowdsensing strategy, which employs the passengers's mobile devices as data gathering plataform.
Among the available smartphone communication technologies, Bluetooth provides data communication with low cost and low power.
In this work, we propose a data gathering system of public bus usage based on passenger's smartphones. In order to evaluate the proposal's suitability, a prototype is implemented for performance tests and power consumption evaluation. The proposed data acquisition strategy is evaluated through trials of the prototype in itinerary simulations, obtaining results that indicate the viability of our proposal.

Keywords: Bluetooth sensing. Android. Mobile Aplications. Crowdsensing. Participatory Sensing. Urban sensing. Urban mobility. Intelligent transportation systems. Origin-destination estimation. Java SE. Bluecove.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Sistemas de transporte público eficientes requerem a disponibilidade de dados que produzam informações sobre seu uso. Dados operacionais tais como números de passageiros transportados e noções sobre origens e destinos são informações valiosas que permitem medir o desempenho do sistema ou identificar eventuais ineficiências em sua operação. Neste contexto, mais especificamente, dados como lotações ao longo de um itinerário e as matrizes Origem/Destino (BEN-AKIVA; MORIKAWA, 1989) são de interesse primordial, pois permitem o levantamento de informações como o índice de renovação de rotas (ANTT, 2015) e a avaliação contínua da distribuição temporal e espacial da demanda.

Tais informações, embora importantes, são pouco utilizadas na prática devido à dificuldade de aquisição. Métodos tradicionais de contagem não permitem sua obtenção: a contagem de passageiros por catraca, por exemplo, não contabiliza desembarques (PIERI; KRAUS; FARINES, 2016). Outros dados importantes como os momentos de embarque e desembarque em cada ponto de parada são difíceis de obter, mesmo através de sistemas automáticos de contagem. O alto custo de implantação de sistemas de contagem automáticos é outro fator que contribui para a falta de interesse das empresas de transporte coletivo em obter dados de utilização das linhas de ônibus (PETKOVICS; FARKAS, 2014).

Por outro lado, o avanço tecnológico de dispositivos com capacidade computacional e sua crescente popularidade permite que o problema de aquisição de dados de transporte seja solucionado com o auxílio da tecnologia. O levantamento de dados comportamentais através do uso de dispositivos móveis é uma técnica recente de aquisição de dados e se tem mostrado promissora, conforme cresce o número de usuários deste tipo de tecnologia (CACERES; WIDEBERG; BENITEZ, 2007). Em se tratando de usuários de transporte coletivo, é notável o uso crescente de dispositivos móveis portados pelos passageiros, sendo os celulares os mais populares (ANATEL, 2015). Isto torna possível avaliar e propor uma abordagem de aquisição de dados que tire proveito da ubiquidade presente em tais ambientes para obter dados de utilização de ônibus.

1.2 OBJETIVO DO TRABALHO

O objetivo deste trabalho é o projeto e implementação de um sistema de aquisição de dados de utilização do transporte público coletivo em uma estratégia de *crowdsensing* baseada nos *smartphones* (dispositivos móveis com sistema operacional e suporte a aplicações) portados pelos passageiros. Neste contexto, emprega-se a tecnologia Bluetooth (2016) como principal plataforma de comunicação de dados entre dispositivos móveis a fim de preencher requisitos de baixo custo e de baixo consumo de energia.

O sistema proposto deve respeitar requisitos básicos de qualidade tais como robustez, desempenho e escalabilidade. O sistema deve ser robusto o suficiente para tratar falhas de comunicação, decorrentes de queda do canal ou dos processos dos componentes envolvidos. O sistema deve estar imune a interferências internas ou externas à aplicação. Por fim, o sistema deve apresentar um desempenho que permita seu funcionamento durante momentos de alta utilização dos ônibus, como ocorre em horários de pico.

1.3 MÉTODO DE DESENVOLVIMENTO

O escopo da proposta a ser apresentada restringe-se à implementação de um protótipo do sistema, englobando uma infraestrutura de bordo e um aplicativo para acessar a infraestrutura, chamado notificador. O aplicativo notificador foi desenvolvido para *smartphones* com sistema operacional Android (ANDROID, 2016). Apesar da arquitetura proposta ser aplicável a outros sistemas operacionais como iOS da Apple (iOS, 2016), utiliza-se o Android por questões de disponibilidade de equipamentos para realização de testes.

O protótipo foi projetado para identificar passageiros, servindo de base para inferir informações de lotações por trecho de itinerários e geração de matrizes O/D.

A avaliação do protótipo foi conduzida através de testes de bancada em ambiente controlado utilizando um simulador de rotas desenvolvido para este fim. Como não foram feitos testes de campo, torna-se válida a criação de um ambiente que permita simular a utilização dos aplicativos em rotas de ônibus previamente conhecidas. O simulador de rotas é integrado ao protótipo da infraestrutura de bordo. A infraestrutura de bordo foi implementada na forma de um servidor de dados Bluetooth, utilizando-se a API Bluetooth disponibilizada pela lingua-

gem Java *Standard Edition* e serviu para concentrar dados coletados nas simulações das linhas de ônibus. A construção deste componente prevê comportamentos que seriam utilizados em uma infraestrutura de bordo definitiva e serve como ferramenta para avaliação do desempenho e acurácia do sistema proposto.

Em termos de avaliação, restringe-se o escopo dos testes em três aspectos: autonomia, desempenho e corretude dos dados de interesse. Para a avaliação da autonomia, estimou-se através de testes e simulações o consumo médio de bateria durante a utilização do aplicativo de notificação a fim de determinar se a execução do aplicativo traria algum prejuizo ao portador do dispositivo.

A análise do desempenho do sistema visa avaliar o tempo de processamento e envio das mensagens de controle trocadas entre o servidor Bluetooth e os aplicativos de notificação. Para isto, foram realizados testes de bancada para determinar os tempos de processamento de mensagens e a capacidade de atendimento do servidor durante períodos de grande demanda.

Os dados de interesse inferidos a partir do mecanismo de notificação proposto se restringem à construção de matrizes O/D e relatórios de lotação por trecho apenas. Para se alcançar este objetivo, utilizaram-se dados simulados de uma linha de ônibus a fim de conferir as informações obtidas a partir de um comportamento de utilização da linha previamente definido.

Mecanismos de tolerância a faltas não foram abordados na implementação do protótipo, embora se realize uma análise dos principais casos de risco concernentes ao contexto da aplicação proposta, deixando-os para trabalhos futuros.

1.4 ORGANIZAÇÃO

Divide-se o presente trabalho em 7 capítulos. O capítulo 2 descreve os métodos convencionais de coleta de dados operacionais do transporte coletivo, suas características, vantagens e desvantagens. Ainda neste capítulo, aborda-se o conceito de *crowdsourcing*, com ênfase na utilização de dispositivos móveis como plataforma de levantamento de dados, concluindo o capítulo com os principais trabalhos relacionados ao tema de levantamento de dados de transporte utilizando dispositivos móveis.

No capítulo 3, descreve-se a tecnologia de comunicação sem fio Bluetooth, seus principais conceitos e aspectos necessários para a com-

preensão do sistema de comunicação de dados proposto.

O capítulo 4 apresenta a arquitetura do sistema proposto e o modelo de comunicação, com ênfase em aspectos tecnológicos relacionados à construção do sistema. O capítulo aborda ainda aspectos de tolerância a faltas relacionados aos requisitos de qualidade e confiabilidade do sistema.

O capítulo 5 descreve a implementação dos componentes necessários para a comunicação de dados sobre o Bluetooth, o aplicativo de notificação e o servidor de dados de ônibus.

O capítulo 6 apresenta a avaliação do sistema e discussões sobre os resultados obtidos em simulações em ambiente controlado. Neste capítulo, avaliam-se aspectos de autonomia e desempenho e a coleta das informações de interesse sobre a utilização de ônibus.

Finalizando, o capítulo 7 apresenta as considerações finais e conclusões obtidas na elaboração do presente trabalho. O capítulo traz também propostas de extensão do presente sistema em trabalhos futuros, com a utilização de novas arquiteturas da tecnologia Bluetooth, especificadas na versão mais recente porém ainda não disseminadas em dispositivos atuais.

2 TRABALHOS RELACIONADOS

Neste capítulo aborda-se a problemática da aquisição dos dados de interesse como lotações por trechos e matrizes O/D. Também é tratado o conceito de *crowdsoursing* e sua aplicação no âmbito deste trabalho. Por fim, descrevem-se os principais trabalhos relacionados ao tema de levantamento de dados de transporte público usando *smartphones* como plataforma de aquisição de dados.

2.1 COLETA DE DADOS DE TRANSPORTE

Atualmente existem várias soluções de coleta de dados operacionais de transporte público, que vão desde a contagem manual à utilização de contadores automáticos de passageiros (*Automatic Passenger Counters* - APCs), ou a uma combinação de ambos (BOYLE, 2008). Os principais dados de interesse para obtenção de estatísticas de utilização dos sistemas de transporte concentram-se em dois parâmetros principais: contagem de lotações e o movimento dos passageiros em termos de origem e destino.

2.1.1 Contagem de passageiros

Existem várias soluções que permitem estimar ou calcular com precisão o número de passageiros a bordo de veículos de transporte público. Abordagens tradicionais como a contagem manual são custosas e propensas a erros (KOSTAKOS, 2008). Uma alternativa para a contagem manual é a utilização de contadores de catraca, que permitem a obtenção de dados mais precisos (KOSTAKOS, 2008). Porém, este método também apresenta inexatidão na contagem, pois não contabiliza os passageiros que embarcam pelas portas de saída ou que desembarcam pelas portas de entrada.

Os APCs são dispositivos eletrônicos utilizados em veículos de transporte coletivo que permitem contar embarques e desembarques utilizando sensores presentes em todas as portas do veículo (CHENG; LIU; ZHAI, 2010). Desta forma, os APCs conseguem fornecer o número de passageiros a bordo em todos os trechos, ao contrário dos contadores de catraca (que não contabilizam desembarques). Com o uso de APCs é possível conhecer a lotação em qualquer momento da viagem,

permitindo calcular o índice de renovação da linha[1] de forma precisa. Apesar da precisão no levantamento de lotações ao longo do itinerário, APCs são incapazes de fornecer informações de origens e destinos dos passageiros e apresentam altos custos de aquisição e implantação (KOSTAKOS; CAMACHO; MANTERO, 2013), além de apresentarem resultados ineficientes em momentos de lotação extrema no veículo.

2.1.2 Observação do movimento de passageiros

Segundo Kostakos (2008), "geração de matrizes Origem/Destino é um requisito chave para o projeto e melhoria da eficiência das redes de transporte público". Matrizes O/D descrevem o fluxo de passageiros entre dois pontos de um itinerário ou linha de transporte público (ônibus ou metrôs) e permitem explorar o comportamento dos usuários em períodos de pico ou momentos de baixa utilização como em fins de semana e feriados. Ainda segundo Zhao (2006), informações de origem e destinos dos passageiros têm grande peso no projeto de novas linhas de ônibus e no escalonamento dos veículos e seus motoristas.

Ao contrário das abordagens de contagem de passageiros já consolidadas, os métodos de obtenção de matrizes O/D são custosos e resumem-se a observações e estimações através de sondagem. O levantamento por observação emprega indivíduos cuja a tarefa é observar e contar o fluxo de passageiros entre dois pontos da linha durante um período específico. Já o método de sondagem, passageiros são selecionados aleatoriamente para responderem a questionários com perguntas sobre a utilização da linha e ponto de destino. Estas abordagens permitem estimar dados para geração de matrizes origem/destino, porém são caras, pois demandam o emprego de profissionais especializados em sondagem. Outra desvantagem é que os dados levantados por sondagem tendem a ficar obsoletos em pouco tempo. O curto prazo de validade dos dados obtidos demanda atualização constante através de repetições da sondagem. Porém, devido aos altos custos financeiros, os dados são atualizados em intervalos mínimos de 6 meses a um ano (CEDER, 2007, p. 30).

Em contraste com os métodos tradicionais de estimação de origens e destinos, uma nova abordagem baseada em bilhetagem eletrônica tem se mostrado mais eficiente. Porém esta técnica é mais apropri-

[1] Índice de renovação da linha: lotação máxima ao longo do percurso dividida pela capacidade do veículo; o índice de renovação é um indicador importante da produtividade de uma linha de transporte coletivo.

ada a sistemas de metrôs, pois apresentam um sistema de bilhetagem ponto a ponto (KOSTAKOS, 2008), permitindo registrar pontos de entrada e saída do sistema de metrô. Casos reais de aplicação desta abordagem podem ser encontrados em alguns países da Europa como o sistema de metrô de Izmir na Turquia (NASIBOGLU et al., 2012) e de Londres (SEABORN; ATTANUCCI; WILSON, 2009) que permitem levantar informações dos fluxos dos passageiros através do cartão de pagamento. Infelizmente, esta técnica não é adequada a sistemas de transporte por ônibus, pois a maior parte dos sistemas de bilhetagem registram apenas o ponto de embarque. Além deste fator, a implantação de uma solução semelhante em um sistema de transporte público de ônibus que opera com altas taxas de lotação implicaria em atrasos grandes no desembarque, sem mencionar o custo de implantação dos recursos tecnológicos necessários.

Existem outros métodos inovadores tais como o uso de detectores de face e o rastreamento de celulares, que permitem o levantamento de informações sobre o fluxo de passageiros. Dentre estes, os mais promissores são baseados em dispositivos móveis em conjunto com estratégias de *crowdsourcing*, que é o objetivo deste estudo.

2.2 *CROWDSOURCING*

Crowdsourcing é uma técnica de solução de problemas ou levantamento de dados sobre uma determinada população através do uso do conhecimento coletivo presente na população (MISRA et al., 2014). Através de *crowdsourcing*, é possível coletar informações que permitam o levantamento de tendências comportamentais de uma determinada população. Ainda segundo Misra et al. (2014), *crowdsourcing* apresenta imenso potencial para substituir ou complementar abordagens tradicionais de levantamento de dados e é perfeitamente aplicável ao planejamento do transporte público visto que seus usuários formam uma grande população que compartilha as mesmas necessidades.

A implementação de sistemas de *crowdsourcing* depende de uma característica chave que é o recrutamento e gerenciamento de voluntários para contribuir com o projeto (MISRA et al., 2014). Para isto, são utilizadas estratégias baseadas em incentivos com o fim de estimular a participação e contribuição. Uma abordagem de *crowdsourcing* conhecida como *participatory sensing* ou *crowdsensing* é focada na delegação de tarefas a anônimos em troca de vantagens.

2.2.1 *Smartphones* como plataforma de *crowdsourcing*

O uso de *smartphones* como plataforma de aquisição de dados tem sido grandemente explorado nos últimos anos (CACERES; WIDEBERG; BENITEZ, 2007). A popularidade e aumento da capacidade computacional dos dispositivos móveis caracterizam os *smartphones* como ferramentas poderosas de coleta de dados a um baixo custo. O crescimento do número de usuários de *smartphones* permitiu empregar tais dispositivos como plataforma de *crowdsourcing* e tem sido tema frequente em trabalhos acadêmicos (WORK; BAYEN, 2008). No Brasil, observa-se o mesmo crescimento de usuários da telefonia móvel. Segundo a Anatel (ANATEL, 2015), existem mais de 284 milhões de linhas de telefonia móvel habilitadas no país, ou seja, mais de uma linha por habitante. Outro fato é a crescente utilização de dispositivos móveis por passageiros de transporte público. Aplicativos para acesso a redes sociais, envio de mensagens, leitura de texto, reprodução de músicas e jogos são usados pelos passageiros durante as viagens. O crescente avanço tecnológico aliado à popularidade do setor propiciou o surgimento de diversos aplicativos para os mais variados propósitos, que podem ser facilmente instalados por seus usuários a partir de um endereço confiável na Internet. Atualmente, dispositivos móveis do tipo *smartphone* possuem interfaces para localização por GPS e suporte à comunicação sem fio como Wi-Fi e Bluetooth.

Dentre os diversos projetos de *crowdsourcing* existentes, o StreetBump (2016) e Tiramisu Transit (ZIMMERMAN et al., 2011) são dois exemplos que empregam *smartphones* como plataforma de aquisição de dados (MISRA et al., 2014).

O projeto Street Bump baseia-se em um aplicativo para dispositivos móveis que emprega o acelerômetro para detectar automaticamente buracos e outros obstáculos nas rodovias na cidade de Boston, Massachusetts. Os dados coletados são automaticamente submetidos a uma central de armazenamento e permitem auxiliar o processo de restauração e pavimentação das rodovias. Esta abordagem permite a coleta de dados sem a intervenção do usuário cujo papel é apenas instalar e permitir a execução do aplicativo em seu dispositivo móvel. O projeto Tiramisu também é baseado em um aplicativo para *smartphone*, mas usa a estratégia de *feedback* do usuário, que passa a desempenhar um papel ativo na aquisição dos dados. O aplicativo fornece informações sobre próximas chegadas e lotações de transportes coletivos aos seus usuários baseadas em dados submetidos em tempo real por outros passageiros a bordo dos veículos. Para os passageiros que estão a bordo,

o aplicativo permite avaliar a qualidade do serviço prestado e fornece informações sobre o próximo ponto de parada.

A presença do GPS nos *smartphones* permitiu também o desenvolvimento de aplicativos de auxílio à navegação e tráfego de veículos, como o WAZE (2016), e de informações sobre o transporte público coletivo como o MOOVIT (2016). Ambos usam o conceito de *crowdsensing* para coletar dados sobre o estado de sistemas de transportes, fornecendo em troca informações valiosas aos usuários.

O aplicativo Waze é um sistema de navegação colaborativo que auxilia o usuário na escolha de rotas para um destino. O Waze fornece informações adicionais em tempo real como a presença de pistas obstruídas e acidentes nas rotas escolhidas. Estas informações são fornecidas por usuários, que recebem em troca como incentivo uma pontuação para cada contribuição realizada. Ao alcançar determinada pontuação, o usuário recebe acesso a novas ferramentas e informações de interesse, como por exemplo, o melhor preço de combustível nos postos presentes na rota trafegada. Outra característica importante é que o Waze fornece a estimativa da velocidade média de uma via, baseado em dados coletados automaticamente por sistemas Waze a bordo de outros veículos trafegando em tal rodovia.

O aplicativo Moovit é destinado aos usuários do transporte público coletivo. Sua principal função é traçar rotas considerando apenas meios de transporte públicos como ônibus, trens e metrôs. O aplicativo fornece outras informações úteis como o tempo previsto de chegada ao destino, o ponto de parada em que o usuário deve descer para ir a um determinado local e os horários estimados das próximas chegadas em um ponto de parada.

2.3 PRINCIPAIS TRABALHOS RELACIONADOS

A seguir, apresenta-se um conjunto de trabalhos que estão diretamente relacionados ao tema de aquisição de dados de transporte utilizando *smartphones* como plataforma de coleta de dados.

Caceres, Wideberg e Benitez (2007) propõem o rastreamento de celulares GSM para extrapolar os dados de deslocamento de passageiros. Esta abordagem possui a vantagem de dispensar o usuário de instalar ou executar qualquer aplicativo em seu dispositivo, porém, os dados obtidos precisam ser correlacionados com os horários de funcionamento e localização das linhas atendidas pelo sistema de transporte público. Porém, no contexto do presente trabalho, entende-se que se

torna mais adequado desenvolver um sistema no qual a aquisição de dados e seu armazenamento sejam de responsabilidade exclusiva do sistema, sem que haja delegação das atividades de aquisição a outras arquiteturas externas ao sistema.

Uma abordagem promissora no sensoriamento não-intrusivo de dispositivos móveis faz-se através do uso da interface Bluetooth presente nos dispositivos. Bluetooth é uma das tecnologias de comunicação sem fio que permite extrapolar informações de um indivíduo específico. O uso de Bluetooth tem sido amplamente explorado em vários trabalhos acadêmicos em aplicações como detectores de presença, monitoramento de fluxo e densidade de tráfego dentre outros. Bandara et al. (2004) propõem um método de sensoriamento de dispositivos Bluetooth em um ambiente, que utiliza vários sensores fixos para medir o indicador de força de sinal do rádio Bluetooth. Esta abordagem dispensa o uso de aplicativos e permite contabilizar o número de dispositivos Bluetooth no ambiente, porém, não provê identificação de usuários. Friesen e McLeod (2014) apresentam um sistema de monitoramento de tráfego baseado em observação de dispositivos Bluetooth habilitados e em modo visível, presentes em veículos ou portados por seus passageiros. O sistema é composto por uma rede de dispositivos monitores instalados em pontos chaves de uma rodovia, cuja tarefa é realizar temporariamente varreduras para descobrir dispositivos Bluetooth nas proximidades e com isso, levantar estimativas sobre o fluxo de veículos, com base no número de dispositivos Bluetooth encontrados.

Uma abordagem semelhante é proposta por Kostakos (2008), na qual utilizam-se dispositivos móveis com interface Bluetooth habilitada como mecanismo de apontamento de presença a bordo do ônibus. Este sistema emprega uma infraestrutura a bordo, cujo papel é realizar varreduras temporárias a procura de dispositivos Bluetooth visíveis no interior do ônibus. O protocolo de descoberta de dispositivos Bluetooth permite que dispositivos em estado visível respondam a requisições informando o endereço Bluetooth e a classe do dispositivo. Estas informações permitem identificar um indivíduo na população, tornando possível o levantamento de informações de origem e destino de um único passageiro. As informações coletadas nos processos de descoberta são armazenadas em um banco de dados juntamente com horário da descoberta do dispositivo. Esta abordagem apresenta a vantagem de dispensar a instalação e execução de aplicativos nos dispositivos dos passageiros, deixando-os livres de realizar qualquer papel no processo de aquisição de dados. Por outro lado, o número de usuários que habilitam a interface Bluetooth de seus dispositivos é muito pequena (KOSTAKOS,

2008). Outro fator negativo desta abordagem é a captura de ruídos, visto que qualquer dispositivo Bluetooth em modo visível nas proximidades torna-se um passageiro em potencial, mesmo que se encontre fora do ônibus.

Pieri, Kraus e Farines (2016) utilizam a interface Bluetooth presente em *smartphones* com o objetivo de coletar dados de utilização de ônibus. Esta proposta emprega uma abordagem de formação de *scatternets* (redes de dispositivos Bluetooth) durante as viagens, na qual as mudanças na topologia da rede permitem determinar embarques e desembarques de passageiros e assim gerar informações como matrizes O/D e lotações nos trechos do itinerário. A principal vantagem desta abordagem está em dispensar o uso de uma infraestrutura a bordo dos veículos. Por outro lado, a implementação desta abordagem, especificamente sobre dispositivos Android demanda a solução de alguns aspectos ainda não cobertos pelo sistema operacional, como a habilitação automática do modo de visibilidade do dispositivo, que atualmente exige intervenção do usuário. A detecção de quebras de conexões entre dispositivos é outro ponto necessário para a implementação desta abordagem. Embora o padrão Bluetooth cubra este ponto, não existe suporte do sistema operacional a nível de API. Portanto, inviabiliza-se a implementação de aplicativos que possam ser usados na prática no atual momento tecnológico, motivando a busca de soluções alternativas.

A Tabela 1 apresenta de forma resumida os principais trabalhos relacionados ao tema.

2.4 CONSIDERAÇÕES FINAIS

Os passageiros que portam dispositivos móveis tornam o ônibus um ambiente perfeito para a modelagem de um sistema de *crowdsensing*, pois todos compartilham um mesmo objetivo e podem participar de forma colaborativa ou não na aquisição de dados de utilização das linhas.

Conforme a utilização dos dispositivos móveis mostra-se cada vez mais presente no cotidiano da população, especificamente usuários de transporte coletivo, torna-se possível o surgimento de novos sistemas de *crowdsensing* para coleta de informações de interesse (CACERES; WIDEBERG; BENITEZ, 2007). Tais sistemas permitem levantar informações de forma colaborativa ou autônoma. Em uma abordagem colaborativa, o usuário desempenha voluntariamente algum papel na tarefa da coleta da informação. Sistemas que seguem o conceito de *participatory*

Tabela 1 – Resumo dos principais trabalhos relacionados

Autor	Escopo	Prós	Contras
Caceres, Wideberg e Benitez (2007)	Geração de matrizes O/D através de rastreamento GSM	Dispensa o uso de aplicativos e infraestruturas	Dificuldade na aquisição e correlação de dados
Bandara et al. (2004)	Sensoriamento de dispositivos através de indicador de força de sinal Bluetooth	Dispensa o uso de aplicativos	Necessário Bluetooth ativo nos dispositivos. Não identifica o usuário
Friesen e McLeod (2014)	Monitoramento de dispositivos Bluetooth visíveis	Dispensa o uso de aplicativos	Necessário Bluetooth visível. Arquitetura de implementação complexa e custosa
Kostakos (2008)	Monitoramento de dispositivos Bluetooth visíveis	Dispensa o uso de aplicativos	Necessário Bluetooth visível. Suscetibilidade a interferências externas
Pieri, Kraus e Farines (2016)	Monitoramento de passageiros através de formação de redes Bluetooth	Dispensa infraestrutura	Necessita uso de aplicativo. Aspectos de implementação do aplicativo de difícil solução

sensing geralmente oferecem em troca alguma recompensa conforme o usuário alimenta o sistema com informações, como por exemplo, ocorre com os aplicativos Waze e Moovit. Já os sistemas autônomos, como por exemplo o Street Bump (STREETBUMP, 2016) requerem apenas a execução de um aplicativo nos dispositivos dos usuários, deixando-os livre de qualquer tarefa na aquisição de dados.

Antes de se abordar a descrição do sistema de aquisição de dados proposto neste trabalho, apresenta-se no próximo capítulo um resumo da tecnologia Bluetooth, com o objetivo de esclarecer os principais conceitos e aspectos tecnológicos abordados na descrição da proposta.

3 BLUETOOTH

3.1 INTRODUÇÃO

Bluetooth é um padrão de comunicação sem fio, via rádio entre dispositivos a curta distância, com baixo custo e baixo consumo de energia (BLUETOOTH, 2016). Esta tecnologia foi idealizada pela fabricante de celulares Ericson em 1994, com o objetivo de dispensar a utilização de cabos físicos nas conexões entre dispositivos eletrônicos diversos. Outras empresas de comunicação juntaram-se à Ericson formando o *Special Interest Group* (SIG), um grupo de empresas responsável por especificar e tornar a tecnologia Bluetooth um novo padrão de mercado.

Atualmente a tecnologia Bluetooth está presente em grande parte dos aparelhos eletrônicos disponíveis no mercado, permitindo a conexão entre dispositivos eletrônicos diversos como computadores e periféricos e até mesmo eletrodomésticos. Além de servir como uma interface universal de conexão de dispositivos, Bluetooth também pode ser utilizado como plataforma para construção de diversas aplicações de comunicação de dados. Em termos de desenvolvimento de software, existem várias APIs disponíveis para as principais linguagens de programação como C++, Java e Python (HUANG; RUDOLPH, 2007). Diversas implementações da pilha de protocolos Bluetooth, dentre elas BlueZ, BlueDroid e Mac OS X estão disponíveis em sistemas operacionais como iOS, Linux, Android e Windows (BLUETOOTH, 2016).

3.2 CARACTERÍSTICAS DO MEIO FÍSICO

O rádio Bluetooth opera na faixa de frequência de 2,4 GHz, conhecida como *Industrial, Scientific and Medical* (ISM). A faixa ISM está disponível na maioria dos países e não requer autorização governamental ou licenciamento, porém, sua utilização deve empregar um mecanismo de proteção a interferências externas devido à grande quantidade de dispositivos eletrônicos que operam nesta faixa. Um método de acesso permite minimizar interferências externas no meio de transmissão. Nos Estados Unidos existe uma lei que obriga que fabricantes de componentes operando na faixa ISM apliquem um método de acesso baseado em saltos de frequência. Para o padrão Bluetooth foi escolhido o método de acesso de divisão de código por saltos de frequência *Frequency Hopping - Code-Division Multiple Access* (FH-CDMA). O

FH-CDMA faz a divisão da frequência em vários canais. Nos Estados Unidos e Europa é disponibilizada uma largura de banda de 83,5 MHz permitindo a definição de 79 canais de frequência contendo 1 MHz de espaçamento, com exceção da França onde são definidos apenas 23 canais. Durante a transmissão, um emissor realiza saltos de frequência, mudando de um canal para outro. Isto permite que seja utilizada uma pequena largura de banda da portadora diminuindo os riscos de interferência.

Para permitir a comunicação em modo *full-duplex* (no qual cada dispositivo funciona como emissor ou receptor) os canais Bluetooth usam o esquema *Frequency Hopping / Time Division Duplex* (FH/TDD) que permite transmissões e recepções intercaladas em *slots* de tempo de 625 ms. No FH/TDD, um diferente salto de frequência é usado por *slot*. Isto permite uma taxa nominal de 1.600 saltos por segundo, alternando em *slots* de transmissão e recepção sendo que um pacote de dados é transmitido por *slot*.

Em uma PICONET, todos os dispositivos compartilham o mesmo canal cujo estabelecimento é coordenado pelo dispositivo mestre. No Bluetooth, um canal é determinado pela sequência da frequência de saltos (única por PICONET) e pela fase nesta sequência. A sequência de saltos é determinada pelo endereço Bluetooth do dispositivo mestre e a fase é determinada pelo *clock* deste dispositivo. Assim os *slots* de tempo são sequenciados de acordo com o *clock* do dispositivo mestre. No esquema TDD, dispositivos mestre e escravos transmitem alternadamente, sendo que o mestre sempre inicia a transmissão nos *slots* de tempo de números pares e os escravos iniciam sua transmissão nos *slots* de tempo de números ímpares.

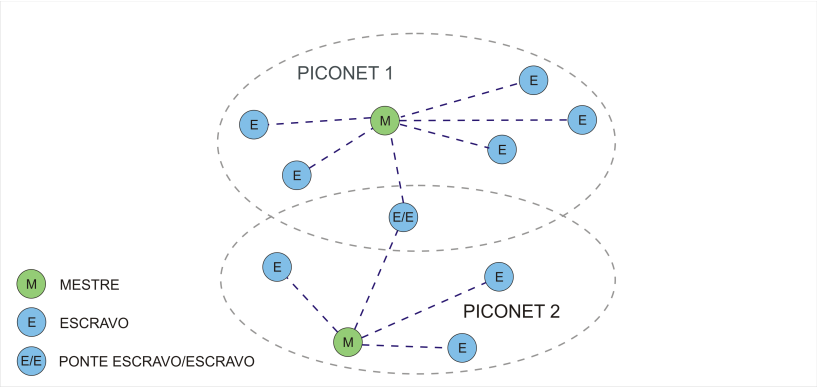
Dispositivos Bluetooth são divididos em quatro classes de operação conforme a Tabela 2 que define potência de sinal e alcance. Dispositivos de classes diferentes são compatíveis, porém, para que haja comunicação dentro do alcance de 100 metros é necessário que todos sejam

Tabela 2 – Classes de operação Bluetooth.

Classe	Potência máxima	Alcance típico
1	100 mW	100 m
2	2,5 mW	10 m
3	1 mW	1 m
4	0,5 mW	0,5 m

Fonte: Bluetooth (2016)

Figura 1 – Scatternet formada por duas piconets unidas por uma ponte escravo/escravo.



configurados como classe 1 enquanto que dispositivos dentro do alcance de até 10 metros, precisam estar configurados como classes 1 ou 2.

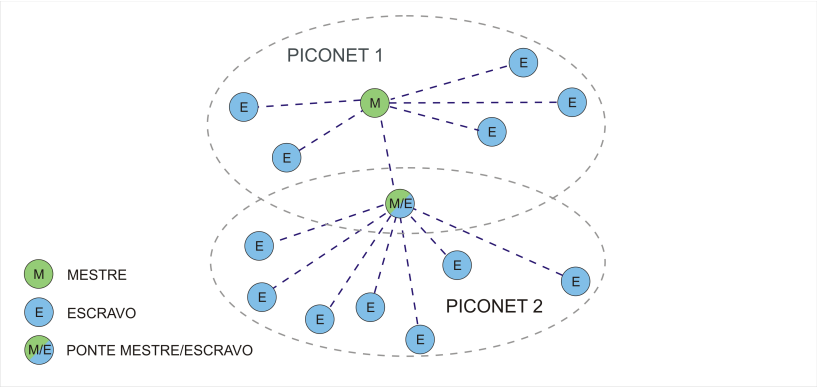
3.3 TOPOLOGIAS DE REDES BLUETOOTH

A conectividade de dispositivos Bluetooth permite a formação de redes ponto-a-ponto ou ponto-multiponto. No estabelecimento da conexão entre dois dispositivos, o nó que inicia a comunicação passa a desempenhar o papel de mestre enquanto que o nó que aceita uma requisição de conexão passa a desempenhar o papel de escravo. Conexões ponto-multiponto permitem a formação de PICONETS e SCATTERNETS.

PICONET é a topologia básica de uma rede de dispositivos Bluetooth que consiste em um dispositivo mestre e até sete dispositivos escravos. Em uma PICONET, o canal de comunicação entre os dispositivos é compartilhado por divisão de tempo. Configurações de multiplexação da PICONET como frequência de modulação e fase são obtidas a partir do *clock* e endereço Bluetooth do dispositivo mestre e são mantidas durante todo o tempo de duração da rede. O dispositivo mestre é sempre quem inicia a comunicação, comandando o processo de transmissão de dados e o sincronismo entre os dispositivos.

SCATTERNET é a sobreposição de PICONETS em um ambiente na qual duas ou mais PICONETS compartilham um mesmo nó. Em uma SCATTERNET, os nós que pertencem a mais de uma PICO-

Figura 2 – Scatternet formada por duas piconets unidas por uma ponte mestre/escravo.



NET são chamados pontes. Qualquer dispositivo de uma PICONET pode se comunicar com uma segunda PICONET desempenhando o papel de ponte, porém, um nó pode ser mestre em apenas uma PICONET.

Os nós que agem como pontes entre PICONETS são classificados conforme o seu papel em cada PICONET: quando um nó está conectado a dois mestres, este é chamado de Escravo/Escravo, ou seja, o nó age como escravo em ambas as PICONETS às quais este pertence, conforme a topologia da Figura 1; e Mestre/Escravo, quando um nó mestre de uma PICONET está conectado a um nó Mestre em outra PICONET, ou seja, o nó age como mestre em uma PICONET e escravo em outra (Figura 2).

3.4 ESTABELECIMENTO DE CONEXÕES

Conexões de dispositivos via Bluetooth envolvem os procedimentos de *inquiry* e *page*. Antes do estabelecimento da conexão, os dispositivos trocam informações que permitem a formação de uma nova PICONET ou a inclusão de um novo nó a uma rede já formada. Ambos os procedimentos *inquiry* e *page* são processos assimétricos (THAMRIN; SAHIB, 2009).

O procedimento de *inquiry* é utilizado para a descoberta de dispositivos Bluetooth passíveis de conexão, dentro do alcance operacional. Durante o *inquiry*, os nodos transmissores descobrem e coletam informações de dispositivos remotos na vizinhança. Para executar o

procedimento de varredura, um dispositivo deve entrar em estado de *inquiry*, durante o qual, o dispositivo emite pacotes de requisição chamados de *inquiry packets*, e aguarda o recebimento de respostas dos dispositivos encontrados. Dentre os dispositivos remotos, apenas aqueles em estado de *Inquiry scan* são passíveis de receber pacotes de requisição. Uma vez recebido um *inquiry packet*, o dispositivo remoto responde a requisição, emitindo pacotes de *inquiry reply* e migra para o estado *Inquiry response*. Cada pacote de *inquiry reply* contém informações sobre o dispositivo como nome, classe de operação e endereço Bluetooth.

O endereço Bluetooth ou *Bluetooth device address* (BD_ADDR) é um número de 48 bits utilizado como identificador único do dispositivo. Qualquer dispositivo Bluetooth possui um endereço Bluetooth gravado no chip durante a fabricação.

O procedimento de *page* é utilizado para configurar e estabelecer a conexão, através da sincronização dos dispositivos na PICONET. Para o estabelecimento de uma conexão, apenas o endereço Bluetooth é necessário. Porém, informações obtidas durante a etapa de *inquiry* como a estimativa do *clock* do dispositivo remoto pode acelerar o processo de sincronização. O dispositivo que requisita a conexão é quem executa o procedimento de *page* e será o mestre da PICONET.

A Figura 3 descreve o processo de conexão detalhando as etapas de *Inquiry* e *Page*:

1. O dispositivo LOCAL inicia o processo em estado *Inquiry*, enviando *inquiry packets*, a fim de descobrir um dispositivo passível de conexão no ambiente.

2. O dispositivo REMOTO, em estado de *Inquiry scan* recebe os pacotes de requisição, migra para o estado *page scan*, enviando um *inquiry reply*, contendo uma estimativa da frequência do seu *clock*, *Frequency Hopping Selection* (FHS), e endereço Bluetooth.

3. O dispositivo LOCAL recebe o *inquiry reply* e migra para o estado *page*, enviando *page packets* ao dispositivo remoto.

4. O dispositivo REMOTO recebe o *page packet*, envia uma resposta ao dispositivo LOCAL e migra para o estado *Slave Response, Step 1.*

5. Ao receber a resposta, o dispositivo LOCAL envia um pacote FHS e migra para o estado *Master response, Step 1.*

6. O dispositivo REMOTO confirma o recebimento do pacote FHS enviando uma segunda resposta e migra para o estado *Slave Response, Step 2*.

7. O dispositivo LOCAL recebe a confirmação do envio do FHS, envia os parâmetros do canal ao dispositivo remoto e migra para o estado *Master response, Step 2*.

8. O dispositivo REMOTO recebe os parâmetros do canal, envia seus parâmetros de sincronização e migra para o estado *Slave Response, Step 3*.

9. Ambos os dispositivos agora compartilham os mesmos parâmetros de sincronização do canal. O dispositivo LOCAL migra para o estado *Master response, Step 3* e envia um pacote POOL ao dispositivo remoto, para garantir que este já esteja sincronizado com o mestre e que tenha migrado para a mesma frequência de saltos do canal.

10. O dispositivo REMOTO recebe o pacote POOL. Configurado como escravo, o dispositivo remoto termina o protocolo enviando qualquer mensagem ao dispositivo LOCAL e migra para o estado *Connection*.

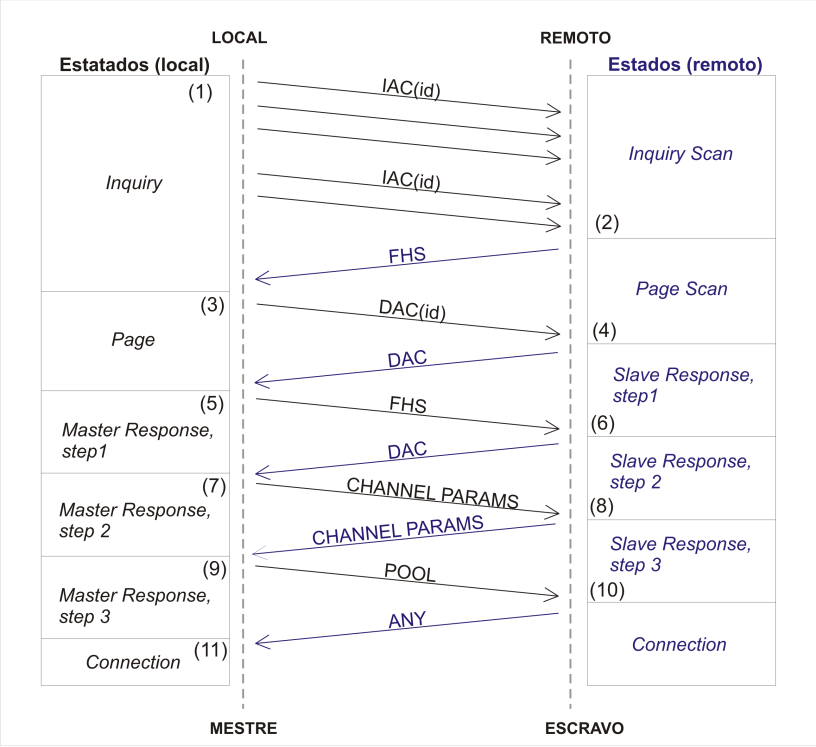
11. O dispositivo LOCAL, agora como mestre, recebe a resposta do escravo e migra para o estado *Connection*, concluindo o processo de *page*.

O processo padrão de conexão entre dispositivos geralmente envolve ambos os procedimentos *inquiry* e *page*. Porém, caso o endereço Bluetooth de um dispositivo remoto seja conhecido, o procedimento de *inquiry* pode ser suprimido.

Dispositivos em estado *connection* estão aptos a iniciar a transmissão. Após o encerramento da transmissão, os pares envolvidos podem permanecer no estado *connection* para, eventualmente continuar transmitindo. Caso a transmissão seja concluída, a conexão é encerrada e o dispositivo migra para o estado *stand by*.

O padrão Bluetooth define ainda outros três estados de baixo consumo. No estado *sniff*, o dispositivo utiliza um tempo menor de escuta do canal, em intervalos fixos programáveis. O estado *hold* define um modo de operação de espera, no qual as transmissões assíncronas são suspensas. O estado *park* define um modo de operação no qual o dispositivo emprega níveis mínimos de potência, ouvindo o canal em intervalos de tempos regulares para verificar eventuais transmissões.

Figura 3 – Procedimentos de *Inquiry* e *Page* durante o processo de conexão entre mestre e escravo.



Fonte: Thamrin e Sahib (2009)

3.5 SEGURANÇA

3.5.1 Pareamento

Pareamento é um mecanismo criado para prover segurança na conexão de dispositivos Bluetooth, mantendo a facilidade na formação de enlaces entre dispositivos. O pareamento avisa ao usuário que existe uma requisição de conexão em andamento caso o dispositivo remoto não seja conhecido. Caso o usuário aceite a conexão, ocorre a geração de uma autorização de conexão (*bond*) que passará a reconhecer o dispositivo remoto como seguro. O usuário intervém no estabelecimento da conexão apenas uma vez. Após realizar o pareamento, o *bond* é man-

tido, até que o usuário remova as autorizações de conexão. As conexões entre dispositivos conhecidos são realizadas de forma transparente ao usuário.

A maior parte dos dispositivos Bluetooth demanda a execução do pareamento antes de permitir o acesso aos seus serviços. Existem poucas exceções a esta regra como por exemplo algumas impressoras que permitem conectar-se a qualquer dispositivo sem prévia concessão de autorização.

3.5.2 Vulnerabilidade de sistemas Bluetooth

Choi et al. (2008) afirmam que redes sem fio apresentam maior risco de interceptação comparativamente às redes cabeadas. Em redes sem fio, devido a não existência de proteção física nos meios por onde as informações trafegam, como ocorre nas redes cabeadas, a suscetibilidade a ataques por interferência ou interceptação é um risco potencial. Segundo Rufino (2011) as maiores ameaças a um sistema de comunicação sem fio são as listadas a seguir:

- Interceptação de tráfego e escuta
- Negação de serviço
- Forja de identidade
- Configuração padrão e força bruta
- Acesso não autorizado

Além destas ameaças, existem ataques feitos por vírus construídos especificamente para aplicações Bluetooth, dentre os quais destacam-se:

- BlueBug (CALLAGHAN et al., 2006)- acesso não autorizado a informações contidas no calendário, lista telefônica e SMS. Consegue realizar e reencaminhar chamadas, enviar SMS, conectar a outras redes sem fio e até mesmo mudar de operadora.
- BlueSnarf (CALLAGHAN et al., 2006) - acesso não autorizado ao sistema de arquivos com direitos de leitura e escrita.
- Bluetracking (CALLAGHAN et al., 2006) - permite a obtenção do MAC de um dispositivo

Para evitar estes problemas, o SIG tem trabalhado em várias frentes a fim de que a tecnologia Bluetooth seja robusta o suficiente para evitar problemas gerados pela falta de segurança. Com a liberação da versão 2.1 foram sanados vários problemas de segurança. A partir desta versão, todas as *releases* têm contemplado algum aspecto de proteção aos ataques. Em termos gerais, os principais recursos que garantem a segurança do sistema estão listados a seguir:

- Adoção do método de acesso FH-CDMA, que permite isolar o meio de comunicação de interferências. Mesmo assim observa-se ataques baseados em *jamming*[1] ou geração de ruídos. Este tipo de ataque pode ser executado de duas maneiras: através da identificação da frequência de saltos utilizada pela PICONET ou pelo preenchimento de todo o espectro com tráfego de ruído(GRÉGIO, 2009).

- Utilização de *Personal Identification Number* (PIN) e pareamento, permitem aumentar o grau de segurança ao conectar dois dispositivos. O PIN pode ser definido na fabricação e alterado pelo usuário. No pareamento, o PIN é utilizado para gerar chaves que são usadas durante a execução do protocolo de *handshake*. Porém este recurso também é vulnerável a ataques de escuta com a utilização de *sniffers*. Na captura de tráfego o PIN é uma informação que pode ser obtida.

Apesar das ameaças existentes, a tecnologia Bluetooth é uma plataforma versátil para o desenvolvimento de aplicações não críticas. Usuários desta tecnologia podem usufruir dos benefícios de uma arquitetura de comunicação de dados gratuita e de baixo custo, desde que seja utilizada com as devidas precauções, em especial restringindo o acesso a dispositivos e aplicativos desconhecidos.

3.6 PILHA DE PROTOCOLOS BLUETOOTH

O padrão Bluetooth é especificado em um conjunto de protocolos propostos pelo SIG. Os protocolos definem mecanismos básicos da tecnologia, como descoberta de serviços, transporte, segmentação de pacotes e multiplexação. A pilha de protocolos Bluetooth é dividida em duas partes: *Controller stack* e *Host stack*. O *controller*, geralmente

[1] *Jamming*: Ataque baseado em negação de serviço utilizando quantidades massivas de tráfego ilegítimo ou malicioso (GIELOW; SANTOS; NOGUEIRA, 2012)

implementado em *hardware*, é composto pelo rádio Bluetooth e pelo microprocessador. O *host* é implementado em *software* e geralmente agrega parte do sistema operacional. *Controller* e *Host* são interligados pelo *Host Controller Interface* (HCI) conforme descrito na Figura 4. Os protocolos Bluetooth são divididos em três grupos: transporte, *middleware* e aplicação.

3.6.1 Grupo de transporte

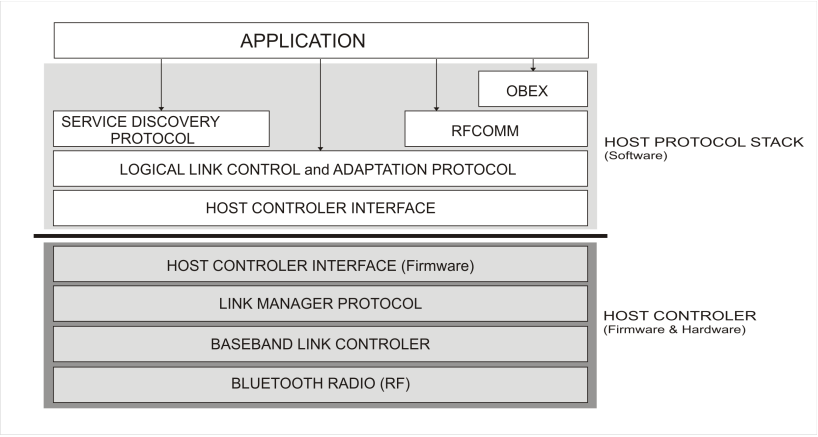
O grupo de transporte é composto por protocolos cuja tarefa é o controle do meio físico e enlace de dados. Estes protocolos desempenham a manutenção e gerenciamento de conexões entre dispositivos. Parte dos protocolos de transporte é implementada no hardware Bluetooth e parte no Host. As principais camadas deste grupo são:

RF - Esta é a camada mais baixa de toda a arquitetura Bluetooth. Nela estão contidas especificações para a transmissão em nível físico como frequência, potência e técnicas de modulação (ALBUQUERQUE; CAVALCANTE; URBANO, 2010). A principal tarefa desta camada é transmitir via radiofrequência os dados recebidos da camada superior (Baseband) bem como entregar os dados recebidos do meio físico para esta camada. A camada RF é implementada em hardware e é composta basicamente pelos seguintes componentes: transceiver, interface de radiofrequência e antena (ALCORN et al., 2011).

BASEBAND - Esta camada propicia o estabelecimento e manutenção de enlaces físicos de radiofrequência entre equipamentos Bluetooth que compoem uma PICONET (HARTE, 2009). A camada Baseband especifica detalhes do processo de conexão entre dispositivos como por exemplo a definição da sequência de saltos, sincronização e estabelecimento de papéis de mestre ou escravo na formação de PICONETS e implementa os processos de *inquiry* e *page*.

A camada Baseband também fornece métodos de enlaces síncronos e assíncronos. O método *Synchronous Connection-Oriented* (SCO) permite estabelecer um enlace sincronizado entre emissor e receptor. Este modo de operação é utilizado em aplicações de transmissão contínua de dados como voz, por exemplo e não permite retransmissão de pacotes perdidos. Já o método *Asynchronous Connection-Less* (ACL) estabelece um enlace assíncrono entre um dispositivo que inicia a comunicação e os demais dispositivos na sua rede, sem perda de informação, ou seja, permite o reenvio de pacotes de dados perdidos. Desta forma, ACL é o método indicado por exemplo para aplicações de transferência de

Figura 4 – Pilha de protocolos Bluetooth.



Fonte: Bluetooth (2016)

arquivos, pois garante a integridade da informação. Além dos métodos ACL e SCO, a camada Baseband permite também enlaces de *streaming*, utilizados para transferência contínua de dados. A camada Baseband também é contida no *hardware* Bluetooth.

LINK MANAGER - Esta camada é responsável pelo controle do enlace lógico entre dispositivos Bluetooth e fornece meios de autenticação e criptografia dos dados trafegados. As funcionalidades desta camada são implementadas pelo protocolo *Link Manager Protocol* (LMP) que gerencia as conexões entre os dispositivos, garantindo segurança, configuração e controle dos enlaces. O protocolo LMP também controla a alocação da largura de banda necessária para o tráfego de dados genéricos e de áudio, fornece suporte para autenticação e encriptação de dados, gerencia o tamanho dos pacotes da camada Baseband e o uso de energia ao controlar os modos de operação de baixo consumo. Esta camada também é implementada em *hardware*.

Host Controller Interface (HCI) é a interface entre o *chip* Bluetooth e o *Host*. O HCI permite a interoperabilidade entre o LMP e o protocolo L2CAP, descrito a seguir.

Logical link control and adaptation protocol (L2CAP) é o protocolo que estabelece e mantem conexões lógicas entre dispositivos Bluetooth, tornando possível a multiplexação de vários canais lógicos sobre um único canal físico. O L2CAP disponibiliza cerca de 15000 números de portas, chamados *Protocol Service Multiplexer* (PSM). Outra atri-

buição deste protocolo é a segmentação e reconstrução de pacotes de dados. Cada pacote L2CAP contém um identificador chamado *Connection Identifier* (CID) que permite relacionar um conjunto de pacotes a uma determinada conexão.

3.6.2 *Grupo de middleware*

Os protocolos de *middleware* provêm suporte às aplicações fornecendo mecanismos para acessar os serviços prestados pelos protocolos de transporte. Os principais protocolos de *middleware* são descritos a seguir.

Service Discovery Protocol (SDP) - Este protocolo especifica o mecanismo de descoberta de serviços disponibilizados por servidores Bluetooth e é utilizado no estabelecimento de conexões. A descoberta de um serviço fornece uma estrutura de dados chamada de *Service Record* contendo informações necessárias para a realização da conexão.

RFCOMM - Este protocolo simula uma conexão serial RS-232 e opera sobre o protocolo L2CAP. O RFCOMM disponibiliza 30 canais de comunicação por servidor e permite um canal de comunicação de dados entre dispositivos, baseado em *streaming*.

Object Exchange Protocol (OBEX) - Protocolo utilizado para trafegar objetos, fornecendo um modelo para o objeto e uma representação de operação.

Além destes protocolos, a especificação do Bluetooth permite a inclusão de outros protocolos de *middleware* para prover a integração com sistemas desenvolvidos sobre outros padrões estabelecidos no mercado, tais como TCS, PPP, TCP/IP/UDP e WAP.

3.6.3 Camada de aplicação

Cada aplicação deve definir um protocolo de comunicação conforme seus requisitos. Os protocolos de aplicação utilizam os serviços disponibilizados pelos protocolos SDP ou OBEX e podem também acessar diretamente serviços de alguns protocolos da camada de transporte como Audio e Link manager.

3.7 IMPLEMENTAÇÕES BLUETOOTH

Existem diversas implementações proprietárias da pilha de protocolos proposta pelo SIG. Estas implementações geralmente agregam parte do sistema operacional dos dispositivos que suportam a tecnologia Bluetooth. Dentre as implementações de propósito geral mais conhecidas, cita-se o BlueZ, BlueDroid, BlueSoleil, Mac OS X Bluetooth stack, WIDCOMM e Microsoft Windows Stack (WIKIPEDIA, 2016).

BlueZ (HOLTMANN; KRASNYANSKY et al., 2007) é a implementação da pilha Bluetooth de sistemas operacionais baseados em Linux. Versões do sistema operacional Android anteriores a 2012 utilizavam esta pilha. A partir da versão 4.1.2, o sistema Android substituiu o BlueZ pelo BlueDroid, uma implementação da pilha Bluetooth totalmente desenvolvida pela Google (ATTAM; MOISEENKO, 2013).

Dispositivos baseados em iOS utilizam o Mac OS X Bluetooth stack (APPLE, 2016), outra implementação proprietária da pilha Bluetooth desenvolvida pela Apple. Esta pilha provê suporte às tecnologias Bluetooth clássico e Bluetooth Low Energy e é considerada uma das mais completas implementações da pilha proposta pelo SIG.

Widcomm foi a primeira implementação da pilha Bluetooth desenvolvida pela Microsoft (WALEED et al., 2009). Atualmente, as versões do sistema operacional Windows utilizam uma implementação proprietária, chamada Microsoft Windows Stack. A partir da versão 8, o Windows provê suporte à tecnologia Bluetooth Low Energy (BLE) e inclui API para desenvolvimento de aplicações BLE utilizando o protocolo *Generic Attribute Profile* (GATT) ou RFCOMM.

Além das implementações no nível de sistema operacional, existem implementações da pilha a nível de API de linguagens de programação. A linguagem Java apresenta a especificação JSR-82 (KUMAR, 2010) na qual padroniza a primeira API Java para os protocolos Bluetooth, permitindo desenvolvedores construirem aplicações Bluetooth que funcionam em qualquer dispositivo. Dentre as implementações do padrão JSR-82 cita-se o BlueCove (2016), uma biblioteca Java compatível com as pilhas Mac OS X, WIDCOMM, BlueSoleil, Microsoft Bluetooth stack e BlueZ. Esta biblioteca provê suporte a vários perfis Bluetooth, tais como: SDAP (*Service Discovery Application Profile*), RFCOMM (protocolo de emulação de cabo serial), L2CAP, OBEX, e TCP. No momento, a biblioteca Bluecove não é compatível com a pilha BlueDroid.

3.8 CONSIDERAÇÕES FINAIS

Bluetooth é uma tecnologia de comunicação sem fio que permite compartilhar dados. Sistemas de *crowdsensing* geralmente utilizam como plataforma a Internet. Um sistema de comunicação sobre Bluetooth, embora limitado quanto ao alcance operacional, permite a comunicação sem o acesso a redes ou pacotes de dados. Os usuários ficam isentos de gastos financeiros ao utilizar o aplicativo. A limitação do alcance operacional justifica-se pois o ambiente interno dos ônibus permite a operação de dispositivos classe 2, que é de 10 metros de raio.

A comunicação entre dispositivos Bluetooth é feita com baixo consumo de energia. O protocolo de comunicação para formação de conexões é rápido o suficiente para garantir que a presença de um usuário seja registrada dentro de um intervalo de tempo entre duas paradas. O baixo consumo de energia também fornece a vantagem aos usuários de que a utilização do aplicativo não influencie na autonomia dos aparelhos.

Cada dispositivo Bluetooth possui um endereço único que permite identificar um usuário particular entre os passageiros. A possibilidade de diferenciar usuários possibilita relacionar informações sobre horários e pontos de embarque e desembarque, permitindo construir matrizes O/D de forma eficiente.

Bluetooth apresenta limitações em aspéctos de segurança. Porém, os riscos de invasão são reduzidos caso a comunicação seja realizada entre dispositivos conhecidos. Propõe-se a utilização de infraestrutura de bordo para dar suporte à comunicação. Esta abordagem limita as chances de falhas de segurança caso haja um mecanismo de certificação durante a conexão dos dispositivos dos passageiros à infraestrutura de bordo.

A utilização da interface Bluetooth presente em *smartphones* é possível e viável. A utilização desta tecnologia em um sistema de aquisição de dados do transporte coletivo é apresentada no capítulo seguinte.

4 DESCRIÇÃO DO SISTEMA DE COLETA DE DADOS

Neste capítulo descreve-se o sistema proposto para a coleta de dados do transporte coletivo. No decorrer do capítulo detalham-se os conceitos empregados na modelagem do sistema, a descrição da arquitetura e o modelo de comunicação entre os componentes.

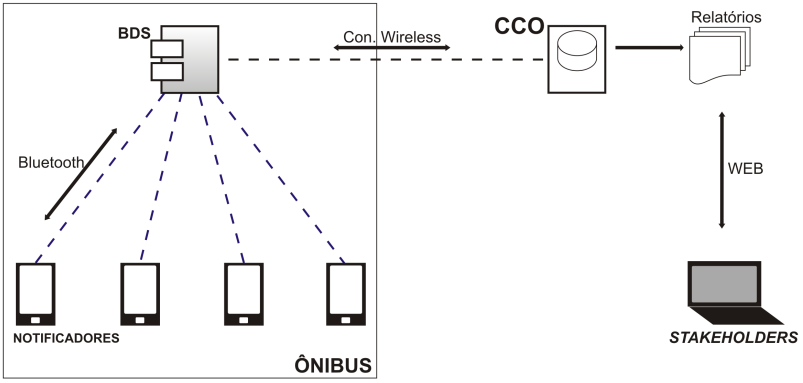
Aspectos de tolerância a faltas são identificados juntamente com propostas de tratamento dos respectivos pontos de falhas.

4.1 ESTRATÉGIA DE *CROWDSOURCING* PROPOSTA

Sistemas de *crowdsourcing* podem ser constituídos sobre diversas plataformas colaborativas que vão desde ferramentas disponíveis através da Internet, como formulários web, até o desenvolvimento de uma aplicação específica. O sistema proposto segue uma estratégia de *participatory sensing*, na qual um grupo de usuários, os passageiros do ônibus em um trecho do itinerário, alimenta o sistema voluntariamente com os dados necessários enquanto recebe informações de interesse sobre a linha, como os próximos pontos de parada e a lotação atual do ônibus. A estratégia de *participatory sensing* proposta demanda apenas que os participantes executem o aplicativo em seus dispositivos móveis. A comunicação entre os dispositivos e a habilitação da interface Bluetooth local é feita de forma autônoma, sem a intervenção do usuário. Propõe-se constituir um sistema autônomo para esta atividade a fim de liberar o usuário de tarefas adicionais. Uma estratégia que empregasse apontamentos em um formulário web simplificaria o desenvolvimento do mecanismo de levantamento dos dados, mas traria um peso maior ao usuário como necessidade de conexão com a Internet e a atenção do passageiro durante o apontamento, atividade extremamente difícil em horários de alta lotação.

A troca de informações entre os componentes do sistema é feita via radiofrequência utilizando a interface Bluetooth dos dispositivos. Esta abordagem permite o levantamento das informações de interesse utilizando uma tecnologia de comunicação sem fio eficiente, ao mesmo tempo em que dispensa o usuário de custos extras na utilização de dados móveis ou acesso à rede. Outro ponto positivo da interface de comunicação Bluetooth é o baixo consumo energético em relação a Wi-Fi (PERING et al., 2006), melhorando a autonomia dos aparelhos ao poupar recursos de bateria.

Figura 5 – Distribuição dos componentes do sistema.



Através da execução de um protocolo de comunicação, são coletados dados que permitem a estimação de matrizes Origem/Destino e lotações nos trechos do itinerário de uma linha.

4.2 DESCRIÇÃO DA ARQUITETURA DO SISTEMA

O sistema de aquisição de dados é formado por dois componentes principais: uma infraestrutura de bordo - *Bus Data Service* (BDS) e os aplicativos notificadores de presença. A Figura 5 apresenta uma visão geral da arquitetura do sistema.

O BDS é um servidor Bluetooth responsável por manter a comunicação com os notificadores e concentrar as informações coletadas. O BDS deve permanecer ativo e manter a interface Bluetooth habilitada durante toda a viagem a fim de possibilitar o recebimento de chamadas. Cada BDS é responsável por identificar apenas uma linha de ônibus. Este componente caracteriza-se em uma infraestrutura presente no ônibus, desempenhando função semelhante ao *scanner* Bluetooth proposto por Kostakos (2008). Porém, na presente abordagem, propõe-se que a infraestrutura de bordo não realize buscas por dispositivos Bluetooth habilitados e em modo visível no ambiente. A habilitação da visibilidade do dispositivo Bluetooth implica diversos fatores, além de questões de segurança. O usuário precisa deliberadamente permitir o modo de visibilidade em seu dispositivo, deixando o aparelho potencialmente vulnerável a ataques. Por esta razão, usuários de dis-

positivos Bluetooth não são encorajados a habilitar a visibilidade de seus dispositivos. Além disso, o suporte de software não é adequado para esta abordagem pois o modo de visibilidade por tempo indefinido efetivamente não é possível em dispositivos Android. Cada passageiro precisaria constantemente habilitar a visibilidade do seu dispositivo a fim de participar do processo de aquisição de dados.

Idealmente, sistemas de *crowdsensing* não requerem outro tipo de infraestrutura senão aquela provida por comunicação 3G/4G. Entretanto, para uso de Bluetooth, contornar a necessidade do uso de infraestrutura interna no ônibus apresenta dificuldades. Pieri, Kraus e Farines (2016) propõem uma abordagem de aquisição de dados via Bluetooth que dispensa o uso de uma infraestrutura de bordo. Porém, sua aplicação requer do usuário a habilitação dos modos de visibilidade e a confirmação de conexões durante a execução do aplicativo. Qualquer abordagem que torne o usuário livre da necessidade de interagir com o aplicativo implicaria em questões de segurança na comunicação entre os dispositivos via Bluetooth, inviabilizando o uso desta tecnologia.

Por outro lado, a presença de infraestrutura no ônibus simplifica a arquitetura do sistema e o protocolo de comunicação entre os componentes, visto que cada *smartphone* comunica-se apenas com o BDS, que é um dispositivo conhecido e seguro. Os participantes ficam liberados de tarefas como a habilitação da visibilidade do aparelho ou a confirmação de pareamentos.

Outro ponto positivo desta abordagem é a centralização do processamento das informações. O BDS concentra os dados sobre a viagem e realiza o envio das informações para uma central de dados, deixando os participantes liberados da tarefa do envio de dados para a central via Internet.

Em termos de custos de implantação, esta abordagem também se justifica, pois a infraestrutura neste caso pode ser implementada com um simples *smartphone*, talvez portado por um membro da tripulação do ônibus.

Os notificadores são aplicativos para celulares *smartphones* que desempenham o papel de clientes do BDS. A tarefa dos notificadores é realizar um processo de busca pela infraestrutura de bordo, realizar requisições de conexão e reportar temporariamente sua presença a bordo através de chamadas executadas ao longo da viagem. A cada requisição, uma conexão temporária é estabelecida e encerrada logo que a execução do protocolo de comunicação seja concluída. Durante as conexões, os notificadores informam seu endereço Bluetooth ao BDS, que armazena em um histórico o momento da chamada e o dispositivo

que realizou o apontamento. Com isto, é possível levantar informações sobre origem e destino de um passageiro bem como sumarizar o número de passageiros a bordo em cada trecho do itinerário.

Os dados coletados pelo sistema podem ser armazenados em um banco de dados de um servidor web. As informações contidas no servidor podem ser utilizadas por uma central de controle e operações cujo objetivo é fornecer informações de interesse, tanto para os participantes (passageiros) como para os *stakeholders* (operadores do sistema de transporte coletivo).

Os usuários do transporte coletivo podem ser beneficiados com informações em tempo real, tais como a previsão de chegadas em um ponto de ônibus e a lotação estimada das próximas chegadas. Tais informações podem ser obtidas através do acesso ao servidor de dados utilizando-se alguma tecnologia de acesso à rede como Wi-Fi, 3G ou 4G, e podem ser incluídas no aplicativo notificador. A liberação do acesso às informações de interesse pode tomar como base uma política de recompensa, pela participação voluntária no processo de coleta de dados.

Os *stakeholders* podem ter acesso a informações mais complexas, como a estimação de matrizes O/D, lotações ao longo dos trechos e relatórios de desempenho emitidos através da filtragem dos dados.

Os BDSs podem ser configurados para transmitirem, em tempo real, os dados adquiridos durante a viagem, utilizando-se alguma tecnologia de conexão *wireless*, como 3G, 4G ou Wi-Fi. Neste mesmo contexto, outras informações de interesse podem ser acessadas a partir da central de dados e fornecidas via BDS aos participantes à bordo, como por exemplo, a lotação e horários de outros ônibus, informações uteis para passageiros que necessitam realizar baldeação. Desta forma os passageiros conectados ao BDS podem ter acesso a informações em tempo real sem a necessidade de acesso à rede de dados.

Os componentes BDS e notificadores são localizados no interior dos ônibus. Os notificadores comunicam-se com o BDS através de Bluetooth, fornecem dados informando sua presença a bordo e recebem estimativa de próxima chamada e dados de interesse sobre a linha. O BDS comunica-se com a central de dados através de redes de dados ou Wi-Fi e faz a submissão dos dados coletados durante a viagem em tempo real. A partir das informações armazenadas na central de dados, os *stakeholders* podem acessar relatórios de desempenho a partir de um site na Internet.

4.3 MODELO DE COMUNICAÇÃO

O mecanismo de apontamento de presença a bordo é baseado em um protocolo de comunicação entre os notificadores e o BDS. Neste contexto, a comunicação entre os dispositivos Bluetooth é realizada em um processo que inclui a descoberta do servidor, a conexão, troca de mensagens de controle e o agendamento de uma nova conexão. O modelo de comunicação proposto baseia-se na troca de dois tipos de mensagens descritas abaixo:

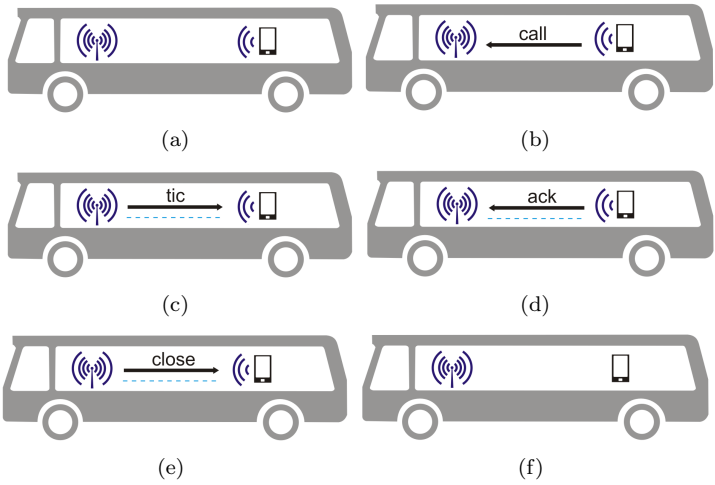
- TIC: mensagem enviada pelo BDS ao notificador. Informa ao notificador uma estimativa do horário para a programação da próxima chamada. Esta mensagem pode conter informações adicionais como trecho corrente no itinerário e a lotação corrente do ônibus.
- ACK: mensagem enviada pelo notificador para confirmar o recebimento do TIC. Informa o modo de operação, horário da descoberta do BDS e horário da última conexão (último recebimento de TIC).

O ciclo de vida da comunicação entre notificador e BDS é descrito na Figura 6. A comunicação é iniciada pelo notificador, ao executar o procedimento de varredura via radiofrequência (*inquiry*) para encontrar o BDS no ambiente. Uma vez encontrado o BDS (Figura 6(a)), o notificador faz um pedido de conexão informando seu endereço Bluetooth (Figura 6(b)). Em um momento oportuno, o BDS aceita a conexão com o notificador e envia uma mensagem de resposta informando um tempo de espera que o notificador deverá aguardar até realizar a próxima chamada (Figura 6(c)). O notificador confirma o recebimento da mensagem (Figura 6(d)). Ao receber a mensagem de confirmação, o BDS registra em histórico informações de interesse como a identificação do notificador e data e hora da chamada. O BDS encerra a conexão com o notificador (Figura 6(e)) a fim de poupar consumo de energia do aplicativo, que passa a operar em modo de espera até o momento da próxima chamada (Figura 6(f)).

Caso a próxima chamada ocorra após o desembarque do passageiro, a mesma não será estabelecida, pois o alcance máximo do rádio Bluetooth é de 10 metros para *smartphones*. A operação do notificador é concluída e o aplicativo pode ser desligado.

A estratégia de comunicação baseada em pequenas conexões temporárias permite que o BDS atenda o maior número possível de notifi-

Figura 6 – Ciclo de vida da comunicação entre BDS e notificador. (a): Notificador executa varredura; (b): Notificador chama BDS; (c): BDS envia *tic*; (d): Notificador confirma recebimento; (e): BDS encerra conexão; e (f): Notificador aguarda execução de nova chamada.



cadores em um curto espaço de tempo. Outra estratégia seria manter a conexão com o servidor Bluetooth até o momento do desembarque, quando a atenuação do sinal implicaria na quebra da conexão, trazendo a vantagem de registrar momentos exatos de embarque e desembarque.

Um dispositivo agindo como servidor desempenha o papel de escravo em uma PICONET. Embora a teoria confirme a possibilidade de um dispositivo ser escravo de múltiplos mestres, a prática não permite isto, pois a maior parte das pilhas de protocolo Bluetooth impede a conexão simultânea com mais de sete dispositivos. Desta forma, esta abordagem implicaria em trabalhar com formações de SCATTERNETS, o que aumentaria a complexidade do sistema. Na abordagem adotada não ocorre formação de redes Bluetooth se não aquelas formadas pelas conexões ponto a ponto entre BDS e notificadores.

4.4 COLETA DE DADOS

O histórico de chamadas mantido pelo BDS permite inferir informações de lotações durante os percursos e de origens e destinos dos

passageiros. Propõe-se que cada notificador realize um mínimo de duas chamadas em cada segmento do itinerário (trecho compreendido entre dois pontos de ônibus).

No recebimento de uma chamada, o BDS armazena a identificação do notificador, o horário e a identificação do segmento do itinerário no momento da chamada. Com estas informações, é possível levantar uma estimativa de lotação a bordo em um determinado segmento do itinerário a partir do número de notificadores presentes no histórico neste segmento. Neste procedimento, as chamadas redundantes recebidas de um mesmo notificador são descartadas pois uma chamada recebida por segmento é suficiente para contabilizar um passageiro a bordo.

As estimativas de origens e destinos dos passageiros são obtidas relacionando-se o endereço Bluetooth do notificador, o horário de recebimento da chamada e o identificador do segmento corrente durante a recepção da chamada. De forma semelhante, o processo de identificação da origem e do destino do passageiro elimina as chamadas redundantes recebidas no mesmo trecho. O ponto de origem do passageiro é determinado pelo identificador do segmento armazenado na primeira chamada recebida. Da mesma forma é possível obter o ponto no qual o passageiro desembarcou, em função da localização da última chamada registrada. A identificação do notificador possibilita relacionar a origem ao destino de um passageiro em particular durante a viagem, bem como o horário que estes eventos ocorreram. Com base nestas informações, torna-se possível a geração de matrizes origem/destino de forma precisa.

4.5 ASPECTOS DE TOLERÂNCIA A FALTAS

Falhas em sistemas de comunicação sem fio são conhecidas e documentadas na literatura. Especificamente em sistemas Bluetooth, a eficiência da comunicação e procedimentos de descoberta podem sofrer atenuações em ambientes densamente populados. Este fenômeno ocorre devido a interferências causadas pelos corpos das pessoas e outros obstáculos físicos (KARA; BERTONI, 2001) presentes no ambiente, fazendo com que a chance de encontrar dispositivos no ambiente durante o procedimento de *inquiry* seja diminuída. Falhas nos processos dos dispositivos envolvidos na comunicação podem acarretar atrasos e até mesmo a queda do canal de comunicação.

Em situações remotas, o procedimento de descoberta de dispositivos Bluetooth realizado pelos notificadores pode encontrar mais de

um BDS. Durante a parada no terminal ou nas proximidades de pontos de embarque existe possibilidade de haver outros ônibus próximos o suficiente para que seus BDS sejam detectados pelo notificador. Nestes casos surge um falso positivo em saber qual BDS descoberto pertence de fato a linha em que o usuário embarcou.

Por fim, um ponto que merece atenção é a interferência dos usuários na execução do aplicativo de monitoramento. O usuário pode inadvertidamente realizar ações que interfiram na execução do aplicativo. Com o objetivo de avaliar a viabilidade do sistema, identificam-se alguns casos prováveis de falhas na arquitetura proposta, os quais são descritos a seguir.

4.5.1 Queda do canal ou de processos

Em sistemas de comunicação de dados, a queda do canal de comunicação ou dos processos participantes pode impedir a comunicação entre as partes. A queda do desempenho dos processos pode interferir no protocolo de comunicação. Para contornar esta possibilidade de falha, propõe-se uma estratégia de comunicação baseada em múltiplas conexões temporárias de curta duração em um mesmo trecho. Esta redundância na execução de chamadas permite assegurar a efetividade do mecanismo de apontamento de presença, sem que se demande mecanismos mais complexos de tolerância a falhas de comunicação. Em caso de uma queda de desempenho temporária no servidor ou a quebra da conexão durante uma chamada, a presença do passageiro a bordo pode ser efetivada em outra chamada no mesmo segmento. Para fins de processamento da informação, apenas uma chamada por segmento é suficiente para indicar a presença a bordo. Para assegurar a informação do embarque, os notificadores armazenam e comunicam ao BDS a data e hora em que o servidor foi encontrado. Com isto é possível recuperar o momento e o trecho do embarque do passageiro.

4.5.2 Incapacidade de atender requisições por sobrecarga do servidor

Durante a execução do sistema, existe a possibilidade de o BDS estar conectado a múltiplos clientes simultaneamente. Caso o número de conexões simultâneas ultrapasse o limite de sete clientes, a pilha de protocolos Bluetooth rejeita o estabelecimento de novas conexões. Para

contornar possíveis erros causados por indisponibilidade do servidor, propõe-se uma estratégia de reagendamento de novas chamadas. Caso o BDS rejeite a requisição, o notificador reagenda uma nova chamada em um intervalo de tempo de 5 a 10 segundos. Adicionalmente, adota-se um método de estimação de novas chamadas, no qual o BDS inclui um valor aleatório às estimativas de tempo de novas chamadas para os dispositivos clientes, com a finalidade de distribuir temporalmente a sequência de chamadas e evitar colisões.

4.5.3 Descoberta de múltiplos BDS

Em situações remotas, existe a possibilidade de que o procedimento de varredura encontre mais de um BDS. Durante os trajetos podem ocorrer comboiamentos (quando mais de um ônibus compartilha o mesmo trecho do itinerário), parando nos pontos sequencialmente. Nestas situações, caso os ônibus estejam próximos o suficiente, existe a chance de um notificador entrar no raio de alcance de mais de um BDS. Esta situação impossibilita o notificador decidir qual BDS pertence realmente à linha que o passageiro embarcou e consequentemente pode incorrer em perda de informações ou geração de informações erradas. Para contornar tais situações, propõe-se um mecanismo que permita ao aplicativo notificador alterar o seu modo de operação. Na presença de múltiplos BDS, o aplicativo deixa de operar em modo padrão e passa a operar em modo duvidoso. O notificador reporta sua presença a todos os servidores encontrados, informando que opera em modo duvidoso. O notificador permanece em modo duvidoso até que a comunicação com todos os servidores externos seja interrompida. No momento em que o notificador estiver conectado a apenas um BDS, o modo de operação volta a ser legítimo e um aviso é enviado ao BDS para que as chamadas recebidas anteriormente em modo duvidoso sejam confirmadas. As chamadas recebidas em modo duvidoso e não confirmadas são descartadas após o término do percurso. Desta forma, evita-se a geração de informações ambíguas na contabilização das lotações dos trechos e na geração das matrizes O/D. As chamadas recebidas nas proximidades de pontos de paradas e durante o embarque no terminal de passageiros não são registradas, a fim de evitar erros causados pela descoberta de múltiplos BDS.

4.5.4 Interferência do usuário na execução do aplicativo

O usuário do aplicativo pode realizar inadvertidamente ações que causem a interferência ou até mesmo a interrupção da comunicação com o BDS. A desabilitação da interface Bluetooth, execução de aplicativos de alto consumo de recursos e até mesmo a interrupção do aplicativo notificador ou a reiniciação do dispositivo são ações possíveis e que podem acarretar na perda de informações ou falha no processo de aquisição de dados. A interrupção do aplicativo notificador durante o processo de comunicação pode resultar em falhas no histórico de chamadas deste notificador no BDS, interferindo na contabilização da lotação em determinados trechos ou até mesmo inviabilizando a informação de origem e destino deste usuário. Para contornar estas situações, propõe-se um mecanismo que permita recuperar informações sobre a presença do usuário nos trechos em que o dispositivo permaneceu desligado ou a comunicação com o servidor foi impossibilitada. Este mecanismo depende da informação registrada sobre o horário em que o BDS foi descoberto. Desta forma, a recuperação do histórico entre dois trechos só é concretizada se houver ao menos uma chamada registrada no BDS antes do usuário desligar o aplicativo e que o usuário volte a executar o aplicativo antes do desembarque.

4.6 CONSIDERAÇÕES FINAIS

Neste capítulo apresentou-se uma visão arquitetural do sistema de identificação de passageiros proposto. No próximo capítulo apresenta-se a implementação de um protótipo, em uma visão mais detalhista sobre a construção dos componentes do sistema, antes de se abordar a avaliação do sistema.

5 IMPLEMENTAÇÃO DO PROTÓTIPO

Neste capítulo descreve-se a implementação dos artefatos necessários para o método de aquisição de dados proposto. Detalham-se a construção do aplicativo de notificação e do servidor BDS, citando-se os principais desafios encontrados durante a implementação bem como os aspectos tecnológicos utilizados na construção de aplicativos para dispositivos Android e a utilização da interface Bluetooth disponibilizada pela API do Android. Por fim descreve-se a construção de um simulador utilizado como plataforma de testes para avaliar o comportamento do sistema.

5.1 O APLICATIVO DE NOTIFICAÇÃO

Conforme mencionado anteriomente, implementa-se o aplicativo de notificação sobre a plataforma Android por questões de disponibilidade de dispositivos. O mesmo conceito empregado pode ser aplicado a outras plataformas como iOS por exemplo.

A API do Android disponibiliza uma biblioteca para o acesso à interface Bluetooth do dispositivo que permite o desenvolvimento de diversas aplicações de comunicação de dados. Em termos gerais, uma aplicação Bluetooth deve executar quatro processos essenciais:

1. Inicialização da pilha de protocolos
2. Procura de dispositivos Bluetooth visíveis
3. Estabelecimento de conexão
4. Troca de dados.

Cada API utiliza uma implementação da pilha de protocolos Bluetooth. Dentre estas implementações cita-se Bluecove, BlueZ e o PyBluez. O sistema operacional Android utiliza a pilha BlueZ até a versão 4.1.2. BlueZ provê suporte a GAP, SDP e RFCOMM (CAI et al., 2011). O processo de inicialização da pilha prepara o hardware para operar a comunicação e habilita os pontos de acesso para as aplicações clientes.

O processo de varredura ou *Inquiry* é um mecanismo utilizado para encontrar dispositivos Bluetooth habilitados para serem conectados e possibilitar a posterior conexão. A execução do procedimento

de *Inquiry* pode ser realizada pelo usuário a partir do sistema operacional do dispositivo ou através da execução de rotinas da biblioteca Bluetooth disponibilizada por API.

Para haver conexão entre dispositivos Bluetooth, um dispositivo deve agir como cliente e outro como servidor. O cliente inicia o processo com uma requisição ao servidor. Quando o servidor aceita uma requisição, ocorre a conexão entre os dispositivos. Ambos passam a compartilhar um mesmo canal RFCOMM, ou seja, existe um *socket* Bluetooth conectado para cada dispositivo no mesmo canal RFCOMM. O estabelecimento de um canal RFCOMM só é possível através da utilização de um serviço SDP disponibilizado pelo servidor. O SDP é um mecanismo que permite a aplicações clientes descobrir serviços oferecidos por um servidor Bluetooth através da utilização de um identificador chamado *Service ID*. Este identificador emprega um *Universally Unique Identifier* (UUID) de 128 bits (HOWES, 2008). Portanto, para que clientes utilizem o serviço BDS, o servidor deve publicar o serviço com a utilização de um UUID que deverá ser conhecido pelas aplicações clientes. Desta forma, em nível de implementação, a conexão depende unicamente do *Service ID* do serviço BDS.

A troca de dados é feita a partir de uma conexão estabelecida, quando é disponibilizado um *socket* para cada processo. A partir do *socket* é possível obter *streams* de dados que possibilitam a comunicação. Ambos os dispositivos podem enviar e receber dados.

5.1.1 Implementação do notificador em Android

Descrevem-se a seguir os detalhes da implementação do aplicativo de notificação utilizando a API do Android para desenvolvimento de aplicações Bluetooth. O aplicativo de notificação é um cliente de servidores Bluetooth que disponibilizam o serviço BDS. Ao ser executado, o aplicativo inicia a pilha de protocolos Bluetooth do dispositivo ao habilitar a interface Bluetooth via API do Android. O procedimento de inicialização da pilha é exibido no trecho de código a seguir:

```
...
//turnon bluetooth
if (BluetoothAdapter.getDefaultAdapter().isEnabled())
{
    BluetoothAdapter.getDefaultAdapter().disable();
}
BluetoothAdapter.getDefaultAdapter().enable();
...
```

O procedimento de inicialização da pilha é um processo assíncrono. Desta forma, deve-se empregar uma rotina de monitoramento de eventos para capturar a conclusão do processo antes de continuar com a aplicação. Isto faz-se com o uso de um objeto `BroadcastReceiver` configurado para ouvir o evento `BluetoothAdapter.ACTION_STATE_CHANGED`. No trecho de código a seguir, o `BroadcastReceiver` permite continuar a execução do aplicativo após a notificação do evento `BluetoothAdapter.STATE_ON`, indicando que a pilha Bluetooth foi iniciada com sucesso:

```
...
BroadcastReceiver bluetoothState = new BroadcastReceiver()
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        String stateExtra = BluetoothAdapter.EXTRA_STATE;
        int state = intent.getIntExtra(stateExtra, -1);
        switch(state)
        {
        ...
        case(BluetoothAdapter.STATE_ON):
        {
            manager.onBluetoothOn();
            break;
        }
        ...
        }
    };
};
...
```

O próximo passo é a busca pelo servidor BDS. O notificador registra o horário de início do procedimento de *Inquiry* e inicia a busca por dispositivos Bluetooth a partir do método `startDiscovery` da classe `BluetoothAdapter` (API Android).

```
...
//records start discovery
this.startDiscoveryTS = new Date();
//starts the inquiry procediure
btAdapter.startDiscovery();
...
```

Utiliza-se também um `BroadcastReceiver` para monitorar eventos de descoberta de dispositivos Bluetooth. Para isto, configura-se o `BroadcastReceiver` para ouvir o evento `BluetoothDevice.ACTION_FOUND`. O código do `BroadcastReceiver` utilizado para tratar o evento de descoberta de dispositivos remotos é apresentado a seguir:

```
...
BroadcastReceiver discoveryResult = new BroadcastReceiver()
{
    // onReceive is called for every device found on inquiry
    // procedure while the discovery is running
    @Override
    public void onReceive(Context context, Intent intent)
    {
        // get the remote BT device
        BluetoothDevice remoteDevice = intent
         .getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);

        manager.connect(remoteDevice);
    }
};
...
```

Ao descobrir um dispositivo, obtém-se o endereço Bluetooth e prossegue-se com a tentativa de conexão. É importante destacar que, neste contexto, qualquer dispositivo Bluetooth visível é tratado como um servidor BDS em potencial. Utiliza-se uma lista contendo os endereços Bluetooth de dispositivos BDS conhecidos a fim de validar os endereços descobertos pelo `BroadcastReceiver`. Caso o endereço obtido no procedimento de *Inquiry* seja de fato o de um servidor BDS, inicia-se o procedimento de conexão e registra-se o horário de descoberta do BDS para fins de relatórios de desempenho.

A conexão com o dispositivo remoto utiliza o endereço Bluetooth do servidor e o *Service ID* do serviço BDS para o estabelecimento de um canal RFCOMM. Emprega-se uma *thread* dedicada, para realizar a conexão com o servidor, a fim de não bloquear a execução do aplicativo. O processo de conexão via API do Android é detalhado no trecho de código a seguir:

```
...
// Use a temporary object that is later assigned to mmSocket
//, because mmSocket is final
BluetoothSocket tmp = null;
// try to create a RFCOMM chanel to server.
// MY_UUID is the BDS Service Id
tmp = device.createInsecureRfcommSocketToServiceRecord(
    BeaconDefaults.MY_UUID);
mmSocket = tmp;
...
```

O método `createInsecureRfcommSocketToServiceRecord` da classe `BluetoothDevice` permite obter um *socket* RFCOMM a partir de um serviço SDP definido no dispositivo remoto. Este método permite que

a conexão ocorra sem a necessidade de realizar o pareamento entre os dispositivos, permitindo transparência ao usuário durante a conexão com o servidor BDS. Após a obtenção do *socket*, o próximo passo é realização do pedido de conexão, que é detalhado no trecho de código a seguir:

```
...
try
{
    // Connect the device through the socket.
    //This will block until it succeeds or throws an exception
    mmSocket.connect();
}
catch (IOException connectException) { ...  }

try
{
    // Do work to manage the connection (in a separate thread)
    startTransmission(mmSocket);
}
catch(IOException e) { ...}
...
```

O método `connect` da classe `BluetoothSocket` realiza uma busca via SDP no dispositivo remoto com o *Service ID* fornecido. Caso a busca seja realizada com sucesso e o dispositivo remoto aceitar a conexão, ambos os dispositivos compartilham um canal RFCOMM para ser usado na conexão, concluindo-se a execução do método `connect`. O tempo de espera do método `connect` é de 12 s por padrão da API. Caso a requisição não seja aceita ou o método expirar, é lançada uma exceção.

Em caso de exceção na conexão, executa-se um procedimento que contabiliza o número de chamadas mal sucedidas ao servidor e programa-se uma nova chamada em 5 s:

```
...
// increments the missed call counter
missedCalls++;
//increments the call attemp counter
attempts++;

if(callWaiterThread != null)
{
    callWaiterThread.cancel();
}
//prepare a new call in 5 seconds
callWaiterThread = new CallWaiter(5000, true);
callWaiterThread.start();
...
```

Caso a conexão seja bem sucedida, registra-se o horário da aceitação da conexão e utiliza-se o `BluetoothSocket` para se obter os *Streams* de I/O, em uma *thread* de comunicação de dados:

```
...
InputStream tmpIn = null;
OutputStream tmpOut = null;
// Get the input and output streams, using temp objects
//because member streams are final
try
{
    tmpIn = socket.getInputStream();
    tmpOut = socket.getOutputStream();
}
catch (IOException e)
{
    mHandler.obtainMessage(MSG_TYPE_EXCEPTION, e)
      .sendToTarget();
}
mmInStream = tmpIn;
mmOutStream = tmpOut;
...
```

A *thread* de leitura e escrita permanece ativa enquanto durar o processo de notificação. Esta abordagem, embora utilize apenas as bibliotecas da interface Bluetooth fornecida pela API do Android, apresenta algumas deficiências. Em primeiro lugar, a necessidade de se empregar uma lista de dispositivos conhecidos para validar os dispositivos encontrados no procedimento de *Inquiry*, embora viável, implica em questões de segurança e manutenção do código. Outro ponto deficiente desta abordagem está na execução da requisição. O único método fornecido pela API realiza a busca por serviços SDP sempre que é chamado. Como a abordagem proposta é baseada em diversas conexões temporárias ao servidor BDS, este procedimento torna-se um *overhead* na execução do servidor. Por fim, os testes mostraram um desempenho deficiente na execução dos aplicativos desenvolvidos sobre esta abordagem e se verificou um grande número de erros no procedimento de obtenção do *socket*, causados por *timeout*. Estes fatores motivaram a se propor uma nova abordagem de implementação do aplicativo de monitoramento, utilizando-se a biblioteca Bluecove, que é descrita a seguir.

5.1.2 Implementação do notificador com Bluecove

Descreve-se a seguir os detalhes da implementação do aplicativo de notificação utilizando a biblioteca Bluecove em substituição às bibliotecas disponibilizadas pela API do Android para desenvolvimento de aplicações Bluetooth.

A habilitação da interface Bluetooth e a notificação de interface habilitada seguem os mesmos passos descritos na implementação do aplicativo de notificação Android. Após a inicialização da pilha prossegue-se para o procedimento de *Inquiry*, utilizando a biblioteca Bluecove. Para a execução da varredura utiliza-se o método de acesso *General/Unlimited Inquiry Access Code* (GIAC) para permitir a compatibilidade com o BDS, que utiliza o mesmo código de acesso. O trecho de código a seguir inicia o procedimento de varredura, informando o tipo de código de acesso (GIAC) e o tratador de eventos:

```
...
//start discovery
try
{
    LocalDevice.getLocalDevice().getDiscoveryAgent()
     .startInquiry(DiscoveryAgent.GIAC,
      this.discoveryListener);
}
...
```

O procedimento de varredura executa de forma assíncrona. Para o tratamento do *callback* é necessário empregar um objeto que implementa a interface `DiscoveryListener`, contida na API do Bluecoove. A descoberta de dispositivos é tratada pelo `DiscoveryListener` que recebe informações sobre o dispositivo remoto como o endereço Bluetooth, nome e classe de operação. Uma vez encontrado um dispositivo remoto inicia-se a procura por serviços registrados na base de dados do SDP do dispositivo remoto. Para realizar a busca, utiliza-se o *Service ID* do serviço BDS como chave:

```
...
UUID[] uuids = new UUID[1];
//STR_UUID is the BDS Service ID:
uuids[0] = new UUID(BeaconDefaults.STR_UUID, false);
try
{
    LocalDevice.getLocalDevice().getDiscoveryAgent()
     .searchServices(null, uuids, remoteDevice,
      this.discoveryListener);
}
...
```

A conclusão da busca por serviços a partir do `DiscoveryListener` retorna um array de *Service Records*, para cada serviço encontrado, com base na lista de UUIDs fornecida como parâmetro de busca. Caso o dispositivo seja realmente um BDS, o array deverá conter apenas o *Service Record* do serviço BDS. Cada *Service Record* é formado por uma lista de *Service Atributes* contendo informações sobre o serviço, como o endereço do servidor e a URL de conexão. O próximo passo é requisitar a conexão com o servidor BDS, utilizando a URL a partir do *Service Record* recebido:

```
...
//records the Bluetooth Address of remote device:
this.beaconDevice = servRecord[0].getHostDevice();
//get the connection URL for the service:
String connectionURL = servRecord[0].getConnectionURL(0, false);
this.beaconConnURL = connectionURL;
//request connection to remote device:
comunicationService.connect(connectionURL);
...
```

A conexão com o servidor é realizada com o auxílio de uma *thread*, a fim de evitar o bloqueio da *thread* principal. Após o estabelecimento, a conexão é encapsulada em um objeto `StreamConnection`, que disponibiliza um `InputStream` e um `OutputStream` para envio e recebimento de dados:

```
...
// Connect the device
stmConn = (StreamConnection)Connector.open(connectionUrl);
...
```

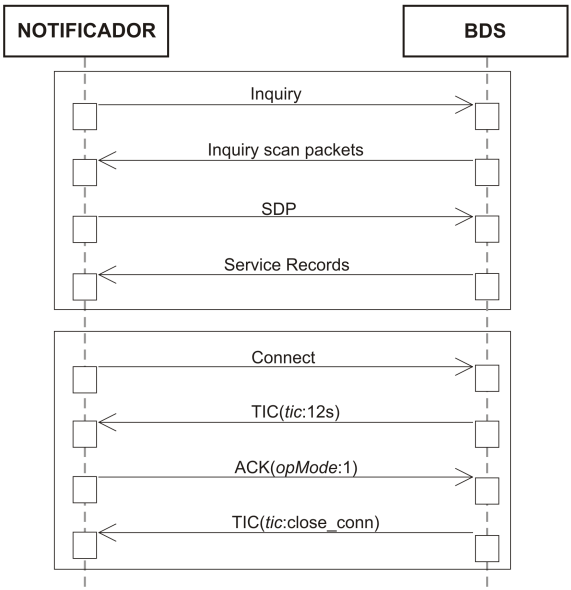
Após a abertura da conexão, registra-se o horário da aceitação da conexão com o BDS. Os *streams* de dados de entrada e saída são mantidos em uma *thread* que permanece ativa enquanto durar a chamada de notificação:

```
...
// Cancel any thread currently running a connection
if (mReadWriteThread != null)
    mReadWriteThread.cancel(); mReadWriteThread = null;
// Start thread to manage connection and perform transmissions
mReadWriteThread = new ReadWriteThread(stmConn);
mReadWriteThread.start();
//get the BDSś bluetooth address
RemoteDevice dev = RemoteDevice.getRemoteDevice(stmConn);
String remoteAdress = dev.getBluetoothAddress();
//records the connection acceptance time
mHandler.obtainMessage(MSG_TYPE_CONNECTED_TO_BEACON,
    remoteAdress).sendToTarget();
...
```

5.2 MÉTODO DE NOTIFICAÇÃO DE PRESENÇA A BORDO

A notificação consiste em uma série de trocas de mensagens entre o notificador e o BDS, conforme descrita no modelo de comunicação de dados. Uma forma simplificada da sequência de mensagens é apresentada na Figura 7. A primeira parte do diagrama engloba o processo de descoberta, sempre iniciado pelo notificador. Esta operação é executada apenas uma vez durante a execução do aplicativo de notificação. A segunda parte do diagrama descreve o processo de notificação. O notificador requisita a conexão com o BDS, que em momento oportuno aceita a conexão e envia a mensagem de TIC para o notificador. A conexão é mantida e o BDS aguarda a confirmação do recebimento. O notificador recebe a mensagem de TIC, programa a próxima chamada e confirma o recebimento, enviando uma mensagem de ACK ao BDS. Ao receber a mensagem de ACK, o BDS reconhece a presença do monitor registrando-o em histórico de chamadas recebidas e envia uma nova mensagem requisitando o encerramento da conexão. O monitor recebe

Figura 7 – Diagrama de sequência do processo de notificação.



a mensagem e encerra a conexão com o BDS, concluindo o processo de notificação.

A seguir, descrevem-se os detalhes da implementação do protocolo. No momento da aceitação da conexão, o BDS inicia o protocolo de comunicação enviando a mensagem TIC como resposta à requisição de conexão. Esta mensagem contém o intervalo de tempo que o notificador deverá aguardar até a próxima chamada. A mensagem TIC também contém dados sobre o itinerário servido pelo BDS, para efeitos de simulação.

Após o estabelecimento da conexão com o servidor, o notificador migra para um estado de espera pela mensagem TIC. Todas as mensagens trafegadas são codificadas no formato *Javascript Object Notation* (JSON). Uma vez recebida a mensagem, o notificador a decodifica, obtendo os valores dos campos *tic*, identificador de linha, nome da linha e último ponto. O campo *tic* armazena um valor numérico e pode conter a estimativa da próxima chamada em segundos, em caso de valor positivo. Em caso de valores negativos, o campo *tic* pode receber um comando para não reagendar nova chamada no caso de ter notificado com sucesso o último segmento do trajeto. Outro comando também disponível é para encerramento da conexão. O notificador registra o horário do recebimento da mensagem de TIC, prepara o agendamento da próxima chamada em *tic* segundos e envia uma mensagem de ACK ao BDS, confirmando o recebimento da mensagem:

```
...
int tic = json.getInt(BeaconDefaults.TIC_KEY');
int lineId = json.getInt(BeaconDefaults.TIC_LINEID_KEY);
String lineName = json.getString(BeaconDefaults.TIC_LINENM_KEY);
String lastStop =
json.getString(BeaconDefaults.TIC_LASTSTOPNM_KEY);
this.ui.showBeaconInfo("Following Beacon for line "+ lineName
+"("+ lineId + ")");
this.ui.showStopInfo(lastStop);
this.ui.showWarning();
if(tic == BeaconDefaults.INT_NO_RECALL)
{
    ui.showNextCallInfo("Final stop");
}
else
{
    ui.showNextCallInfo("Next call in "+ tic + "s");
    prepareNewCall(tic);
}
//send command to close this connection on the peer
sendAckMessage();
...
```

O notificador prepara a mensagem de ACK, registrando o horário do envio para fins de levantamento de indicadores de desempenho:

```
...
ConnectionPerformanceInfo info =
this.currentConnectedBeacons.get(
    beaconDevice.getBluetoothAddress());
info.setLastAckSentTs(new Date());
info.setMissedCalls(missedCalls);
String jsonString = BeaconDefaults.formatJson(
    LocalDevice.getLocalDevice().getBluetoothAddress(),
    getOppMode(), info);
comunicationService.sendMessage(jsonString);
...
```

A mensagem de ACK informa o endereço Bluetooth do notificador, o modo de operação (legítimo ou duvidoso) e os indicadores de desempenho coletados durante a execução do aplicativo.

O BDS, ao receber a mensagem de ACK, obtém os indicadores de desempenho do notificador, registra a presença do dispositivo no histórico de notificações, envia uma mensagem de TIC requisitando ao notificador o fechamento da conexão e encerra a comunicação com o notificador.

O notificador recebe uma mensagem de TIC contendo o comando para fechar a conexão. O notificador encerra a conexão com o BDS concluindo assim o protocolo de notificação.

```
...
int tic = json.getInt(BeaconDefaults.TIC_KEY);
if(tic == BeaconDefaults.INT_CLOSE_CONNECTION)
{
    comunicationService.shutDown();
}
...
```

O notificador continua sua execução em estado de espera até o momento de realização da próxima chamada, programada em função da estimativa fornecida pelo BDS. As estimativas de tempo podem variar em função do tempo médio de cobertura do segmento de itinerário. Segmentos mais longos resultam em períodos de espera maiores. Segmentos curtos demandam um tempo de espera menor. Segue-se uma estratégia que permita a execução de no mínimo três notificações por segmento. A única exceção recai no último segmento do trajeto, no qual uma única notificação é suficiente para contabilizar um passageiro que desembarcará no ponto final. Neste caso o BDS envia uma mensagem de TIC contendo um comando que informa ao notificador para não reagendar nova chamada:

```
...
//its not necessary to prepare new call when going to final stop
if(tic == BeaconDefaults.INT_NO_RECALL)
{
    ui.showNextCallInfo("Final stop");
}
...
```

5.3 IMPLEMENTAÇÃO DO SERVIDOR BDS

Um servidor Bluetooth deve atender dois requisitos básicos: deve ser visível a aplicações clientes e deve conter uma *thread* dedicada para tratar as requisições. A API Android para desenvolvimento de aplicações Bluetooth disponibiliza um mecanismo para habilitar a visibilidade do dispositivo através do uso de um objeto `Intent` para a ação `ACTION_REQUEST_DISCOVERABLE`. No trecho de código a seguir é feita uma requisição para a habilitação da visibilidade do dispositivo por 300 s:

```
...
Intent discIntent = new Intent(
    BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
discIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION,
    300);
startActivity(discIntent);
...
```

De acordo com a documentação, o valor 0 configura o adaptador local para deixar o dispositivo sempre visível. A habilitação da visibilidade permite apenas que o dispositivo seja descoberto por outros dispositivos, ou seja, coloca o dispositivo em estado de *Inquiry Scan*. Para permitir o estabelecimento de conexões é necessário que a aplicação disponibilize um serviço utilizando uma *thread* para tratar as requisições a este serviço. A API do Android provê a classe `BluetoothServerSocket` para a publicação de um serviço e configuração da *thread* de escuta. No trecho de código a seguir, utiliza-se o método `listenUsingRfcommWithServiceRecord` da classe `BluetoothAdapter` para obter-se um *server socket* para um determinado serviço.

```
...
BluetoothServerSocket tmp = null;
try {
    tmp = btAdapter.listenUsingRfcommWithServiceRecord(
        NAME, MY_UUID);
} catch (IOException e) { }
mmServerSocket = tmp;
...
```

A obtenção do *socket* utiliza o nome do serviço e o *Service ID*, que também será usado pelas aplicações clientes na busca do serviço. Internamente, a chamada deste código cria um novo registro no banco de dados SDP do servidor, permitindo a execução do protocolo de descoberta de serviço durante as requisições do serviço.

Após a publicação do serviço prossegue-se para a escuta por requisições:

```
...
BluetoothSocket socket = null;
// Keep listening until exception occurs or a socket is returned
while (true) {
    try {
        socket = mmServerSocket.accept();
    } catch (IOException e) { break; }
    config(socket);
}
...
```

Neste exemplo fornecido pela documentação do Android, a *thread* permanece em espera ocupada até o momento da chegada de uma requisição, quando um *socket* é obtido através do método `accept` da classe `BluetoothServerSocket`. A conexão pode ser configurada utilizando-se uma *thread* para manter os *streams* de I/O com o cliente.

Para implementação do protótipo, optou-se por desenvolver o servidor BDS em um notebook com sistema operacional Windows. O desenvolvimento do BDS sobre a plataforma Android, apesar de viável, apresentou algumas questões de incompatibilidade, em especial a configuração da visibilidade do dispositivo. Embora a documentação afirme que o valor 0 configure o dispositivo para estar sempre visível, sua utilização prática não é possível. Para o desenvolvimento do BDS em Windows, utilizou-se a biblioteca Bluecove, versão 2.1.1 para Java.

As conexões via Bluecove são realizadas em dois passos. Em um primeiro momento realiza-se uma busca por dispositivos visíveis no ambiente. Ao encontrar um dispositivo visível, realiza-se uma busca por serviços SDP específicos deste dispositivo.

Esta abordagem possibilita duas vantagens na implementação do servidor BDS. A possibilidade de consultar serviços elimina a necessidade de realizar tentativas de conexões em qualquer dispositivo potencial. A tentativa de conexão é realizada apenas em dispositivos com o serviço BDS.

Outra vantagem é a possibilidade de se reaproveitar a URL de conexão. Como o serviço de notificação realiza várias chamadas, a busca pelo serviço é realizada apenas na primeira requisição, quando se

obtém a URL para conexão com o BDS. As demais chamadas dispensam a necessidade de consultar o serviço a cada conexão, como ocorre no Android.

Por fim, outro ponto positivo da utilização do Bluecove é a possibilidade de avaliar a comunicação de dados sobre Bluetooth a partir de diferentes plataformas.

Apresentam-se a seguir os detalhes da implementação do servidor BDS em Java e Bluecove. Na execução da aplicação inicia-se o serviço BDS. Para isto, obtém-se o controlador Bluetooth local do dispositivo e habilita-se a visibilidade utilizando o código de acesso *General/Unlimited Inquiry Access Code* (GIAC). O código de acesso GIAC permite habilitar a visibilidade de um dispositivo por um intervalo de tempo indefinido, já o código de acesso *Limited Inquiry Access Code* (LIAC) é usado quando o dispositivo será visível por um período limitado de tempo (SHAHRIYAR et al., 2008). Em seguida configura-se um objeto `StreamConnectionNotifier` utilizando o *uuid* do serviço BDS:

```
...
// retrieve the local Bluetooth device object
LocalDevice local = null;
StreamConnectionNotifier notifier;
StreamConnection connection = null;
// setup the server to listen for connection
try
{
    local = LocalDevice.getLocalDevice();
    // lets the device be discoverable by General/Unlimited
Inquiry Access Code.  All remote devices will be able to find
the BDS
    local.setDiscoverable(DiscoveryAgent.GIAC);
    //setup the url of beacon service
    String url = "btspp://localhost:"+ uuid.toString() +
";name="+ NAME;
    notifier = (StreamConnectionNotifier)Connector.open(url);
}
...
```

O objeto `StreamConnectionNotifier` age da mesma forma que um objeto `BluetoothServerSocket` da API Android, ou seja, representa um *socket* de servidor aberto para ouvir requisições de clientes.

Após a ativação do objeto `StreamConnectionNotifier`, inicia-se a execução da *thread* de escuta. Esta *thread* é responsável por aguardar requisições de conexão. Ao surgir uma requisição de um cliente, recebe-se um `StreamConnection` a partir do método `acceptAndOpen` da classe `StreamConnectionNotifier`. O trecho de código a seguir apresenta a *thread* de escuta:

```
...
while(this.running)
{
    try
    {
        //waits for incoming connections
        //acceptAndOpen will block the server until a connection
        //request is received
        connection = notifier.acceptAndOpen();
        //client connected:  start transmission thread
        new SetUpTransmissionThread(connection).start();
    }
    catch (Exception e) { ...  }
}
...
```

O objeto `StreamConnection` encapsula dados do dispositivo remoto e *streams* de I/O, que são utilizados para estabelecer o canal de comunicação entre os dois dispositivos com o auxílio de uma *thread* dedicada para leitura e escrita. O método de configuração de *threads* de leitura e escrita cria uma nova *thread* e a acrescenta em uma lista de *threads* ativas. Em seguida o servidor envia a mensagem de TIC para o notificador:

```
...
//get the Bluetooth address of remote device
RemoteDevice dev = RemoteDevice.getRemoteDevice(connection);
String remoteAddress = dev.getBluetoothAddress();
...
//configure new thread for R/W
ReadWriteThread mReadWriteThread = new ReadWriteThread(
    connection, remoteAddress);
mReadWriteThread.start();
//adds thread to list
addRWThread(remoteAddress, mReadWriteThread);
//sends TIC message
String msgTic = getTicMessage(remoteAddress);
sendMessage(remoteAddress, msgTic);
...
```

Para montagem da mensagem de TIC, utiliza-se a estimativa em segundos para a próxima chamada. Caso o segmento atual seja o último do itinerário, atribui-se à estimativa o valor da constante `INT_NO_RECALL`, informando ao notificador que uma nova chamada não é mais necessária. A mensagem de TIC é completada com os campos *lineId*, *lineName* e *lastStop*, informando respectivamente o identificador da linha, o nome da linha e nome do último ponto de parada. O código para montagem da mensagem de TIC é apresentado a seguir:

```
...
int secs = getTicTimeMin() + new Random()
    .nextInt(getTicTimeOffset());
if(this.currentTripSegmentInfo.isLast())
{
    secs = BeaconDefaults.INT_NO_RECALL;
}
int lineId = this.lineInfo.getLineId();
String lineName = this.lineInfo.getLineNm();
String lastStop = "Next stop:  "+ this.currentTripSegmentInfo
    .getTripSegmentDestination();
if(this.currentTripSegmentInfo.isStop())
{
    lastStop = this.currentTripSegmentInfo
        .getTripSegmentOrign();
}
ticMsg = BeaconDefaults.getTicJson(secs, lineId,
    lineName, lastStop);
...
```

Após o envio da mensagem de TIC, o servidor aguarda o recebimento da mensagem de ACK no canal RFCOMM estabelecido com o notificador. Ao receber a mensagem de ACK, o BDS verifica primeiramente se o ônibus se encontra nas proximidades de um ponto de parada. Em caso positivo, a notificação não é registrada e a mensagem é descartada. Em seguida obtém-se os indicadores de desempenho contidos na mensagem e registra-se a chamada:

```
...
JSONObject json = new JSONObject(msg);
String remoteMac = json.getString(BeaconDefaults.MAC_KEY);
//register only if not on a stop
if(!onStop())
{
    //get performance infos
    Date beaconFoundTs = BeaconDefaults.getBeaconFoundTs(json);
    ...
    int missedCalls = BeaconDefaults.getMissedCalls(json);
    Date lastTicTs = BeaconDefaults.getLastTicReceivedTs(json);
    Date lastAckSentTs = BeaconDefaults.getLastAckSentTs(json);
    //get the operation mode
    int oppMode = BeaconDefaults.getOppMode(msg);
    //save the call
    registerCall(remoteMac, oppMode, startDiscoveryTs,
     beaconFoundTs, firstConnectionAcceptanceTs,
     lastConnectionRequestTs, lastAuthenticConnectionRequestTs,
     lastConnectionAcceptanceTs, lastTicTs, lastAckSentTs,
     missedCalls);
}
...
```

O registro da chamada cria um novo histórico de chamadas para o notificador, caso seja a primeira chamada recebida. Caso o notificador já tenha realizado alguma chamada, acrescenta-se um novo registro ao seu histórico de chamadas:

```
...
if(this.deviceCalls.containsKey(mac))
{
    //get the stored list
    CallHistoric callHistoric = deviceCalls.get(mac);
    callHistoric.getCalls().add(callInfo);
}
else
{
    //add a new historic
    CallHistoric callHistoric = new CallHistoric();
    callHistoric.setMac(mac);
    callHistoric.setStartDiscoveryTs(startDiscoveryTs);
    callHistoric.setBeaconFoundTs(beaconFoundTs);
    callHistoric.setFirstConnectionAcceptanceTs(fstConnAcTs);
    callHistoric.getCalls().add(callInfo);
    this.deviceCalls.put(mac, callHistoric);
}
...
```

Um registro de chamada contém informações sobre o notificador como o endereço Bluetooth, e modo de operação, informações sobre o contexto durante o recebimento da chamada como identificador do segmento de itinerário, origem e destino do segmento e data/hora do recebimento da chamada. O registro armazena ainda os indicadores de desempenho recebidos do notificador para fins de emissão de relatórios e avaliação do desempenho do sistema nas simulações.

```
...
CallInfo callInfo = new CallInfo();
callInfo.setCallTimeStamp(timeStamp);
callInfo.setCurrentTripSegmentId(tripSegmentId());
callInfo.setCurrentTripSegmentOrign(orign);
callInfo.setCurrentTripSegmentDestination(dest);
callInfo.setCurrentTripSegmentTs(tripSegmentStartTs);
callInfo.setOppMode(oppMode);
callInfo.setLastConnectionRequestTs(lastConnRequestTs);
callInfo.setLastAuthenticConnectionRequestTs(lastAutConnReqTs);
callInfo.setLastConnectionAcceptanceTs(lastConnAcceptanceTs);
callInfo.setLastTicTs(lastTicTs);
callInfo.setLastAckSentTs(lastAckSentTs);
callInfo.setMissedCalls(missedCalls);
...
```

Após registrar a chamada, o BDS envia como resposta uma mensagem de TIC, na qual atribui-se ao campo *tic* o valor da constante INT_CLOSE_CONNECTION, informando ao cliente que os *sockets* de comunicação devem ser fechados. Por fim, o BDS encerra a comunicação com o cliente, terminando o canal RFCOMM:

```
...
// request the client to close connection:
sendMessage(remoteMac, "{"+ BeaconDefaults.TIC_KEY + ":"+
BeaconDefaults.INT_CLOSE_CONNECTION + "}");
// finish communication to client:
stopComunicationThread(remoteMac);
...
```

5.4 CONSIDERAÇÕES FINAIS

Conforme as etapas de implementação apresentadas neste capítulo, foi possível construir um prototipo funcional do sistema, com o qual se pode avaliar a efetividade do mecanismo de notificação, bem como o funcionamento dos componentes da arquitetura. Uma avaliação mais criteriosa do sistema é realizada através de uma série de testes de bancada, cujos resultados são apresentados no próximo capítulo.

6 AVALIAÇÃO

Neste capítulo, avalia-se a viabilidade do sistema proposto através de testes realizados em ambiente controlado dos três aspectos de qualidade propostos: impacto da execução do aplicativo ao usuário, adequação da plataforma Bluetooth na aquisição dos dados e a acurácia dos dados coletados a partir do processo de notificação.

6.1 TESTES DE CONSUMO DE BATERIA

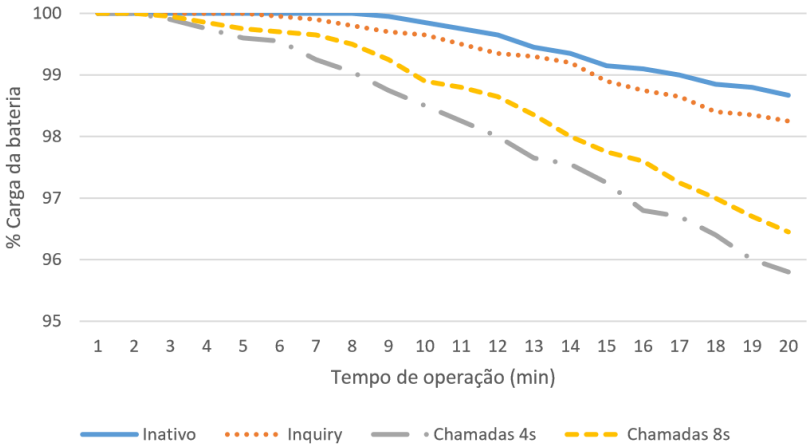
Com o objetivo de avaliar o consumo de energia dos dispositivos Android durante a execução do aplicativo de notificação, propõe-se uma estratégia de testes que permita comparar o consumo de energia do dispositivo durante períodos de inatividade e durante a utilização da interface Bluetooth em situações diversas. Para alcançar este objetivo, dividem-se os testes de consumo de bateria em quatro fases descritas abaixo.

- Inativo: Durante o período de testes, o aplicativo é executado sem acionar a interface Bluetooth;
- *Inquiry*: O aplicativo realiza o procedimento de *Inquiry* durante o período de teste;
- Chamadas, fase 1: Dispositivos realizam chamadas de notificação a cada 4 s;
- Chamadas, fase 2: Dispositivos realizam chamadas de notificação a cada 8 s;

Para a realização dos testes, emprega-se um conjunto de 4 dispositivos modelo Samsung Galaxy SII e sistema operacional Android 4.1.2. Importante destacar que cada aparelho possui a mesma data de fabricação e aquisição. Outro ponto importante é o tempo de vida das baterias. Como os dispositivos têm mais de 4 anos de utilização, aspectos de autonomia e tempo de recarga podem apresentar discrepâncias em relação ao desempenho de baterias de aparelhos recém fabricados.

Cada dispositivo tem sua bateria totalmente carregada antes da execução de cada rodada de testes. As fases de testes têm a duração de 20 min e foram repetidas cinco vezes. Os resultados são obtidos através

Figura 8 – Gráfico resultante do teste de consumo de bateria.



da geração de um arquivo contendo as seguintes informações: nível de bateria, temperatura e *time stamp* da amostragem.

Os níveis de bateria são amostrados com a utilização de um `BroadcastReceiver` configurado para ouvir o evento `ACTION_BATTERY_CHANGED`. Portanto, os pontos de amostragem não seguem um intervalo de tempo fixo, pois o evento monitorado é assimétrico, ou seja, ocorre quando existe mudança no nível da bateria.

Os resultados experimentais de consumo de bateria estão sumarizados na Figura 8, a qual apresenta o gráfico de consumo médio de bateria em cada fase de testes, ao longo de 20 min de operação. Os testes realizados em modo inativo resultaram um consumo médio de 1,33% da carga inicial da bateria, em 20 min de execução. Como a interface Bluetooth permaneceu inativa durante os ensaios, atribui-se o consumo de bateria à manutenção da interface gráfica do aplicativo e outros processos executados pelo sistema operacional do dispositivo.

Os testes realizados com dispositivos executando a operação de *inquiry* resultaram nos valores apresentados na Tabela 3. Observa-se que, em 20 min de execução, obteve-se um consumo médio de 1,75% de bateria, pouco acima do consumo obtido na fase de testes anterior.

Os testes realizados durante a execução do aplicativo de notificação apresentaram valores maiores de consumo em relação às fases de testes anteriores. A execução do aplicativo com uma taxa de chamadas de 4 s apresentou uma média de 4,2% de consumo de bateria, enquanto que a execução com taxa de chamadas de 8 s apresentou

Tabela 3 – Consumo médio de bateria por etapa

Dispositivo	1	2	3	4	Média
Inativo:	1,7 %	0,8 %	1,4 %	1,4 %	1,3 %
Inquiry:	2,0 %	1,4 %	1,8 %	1,8 %	1,8 %
8 s:	3,4%	2,8 %	3,8 %	4,2 %	3,6 %
4 s:	5,2%	3,0 %	4,8 %	3,8 %	4,2 %

uma média de 3,55%, o que permite concluir que a taxa de conexões Bluetooth influencia no consumo de bateria.

A realização dos testes permite inferir um consumo médio de 4,2% de carga, durante vinte minutos de execução do aplicativo em dispositivos com bateria totalmente carregada. Considerando-se que o consumo médio de 1,33% obtido nos testes em modo inativo seja em grande parte decorrente da interface gráfica de usuário, pode-se inferir que o consumo de energia dispensado restritamente com a interface Bluetooth está entre 2,22% e 2,87% em 20 min de execução.

Considerando-se a execução do aplicativo em condições reais de funcionamento, a taxa de chamadas deve ficar acima de uma chamada a cada 20 s. Nestas condições, o consumo médio ficaria dentro da faixa de 0,42% e 2,22%. Assumindo-se um tempo médio de permanência a bordo de ônibus em viagens em torno de 50 min, pode-se afirmar que o aplicativo pode ser executado sem influenciar a autonomia de dispositivos com carga plena.

6.2 TESTES DE DESEMPENHO DOS NOTIFICADORES ANDROID

Avalia-se o desempenho do sistema através da execução de testes de bancada baseados em simulações de itinerários.

Para a realização desta etapa de testes, contam-se um total de 10 dispositivos com arquiteturas e versões do sistema operacional Android diferentes: quatro *smartphones* Samsung Galaxy SII, três *smartphones* Samsung Galaxy SIII e três *tablets* Samsung Galaxy com versões do Android entre 4.0 e 4.3. Divide-se esta fase de testes em duas etapas, variando-se o número de notificadores empregados nas simulações.

A primeira etapa emprega 5 notificadores e é conduzida com simulações de itinerários com a seguinte configuração:

- Tempo de parada: 5 s;
- Tempo do trajeto entre dois pontos: 45 s;
- Intervalo de tempo entre chamadas: entre 13 e 20 s.

A configuração proposta permite que o intervalo de tempo entre chamadas fornecido pelo BDS seja sorteado aleatoriamente dentro do intervalo de 13 e 20 s. Esta abordagem visa diminuir a chance de colisões durante as chamadas de notificação. A comunicação via Bluetooth utiliza canais RFCOMM, o que faz com que o BDS esteja conectado a apenas um notificador por vez, impossibilitando o recebimento de novas notificações enquanto uma chamada corrente não for concluída. A faixa de valores escolhidos no intervalo de 13 a 20 s permite a realização de ao menos duas chamadas por dispositivo em cada trecho do itinerário, visto que o tempo de trajeto entre dois pontos é de 45 s. Esta etapa de testes é repetida 5 vezes e resulta em uma média de 150 chamadas a cada execução. Os resultados obtidos nesta etapa de testes são apresentados na Tabela 4.

A segunda etapa de testes de desempenho do sistema com o aplicativo Android emprega 10 dispositivos. Como no experimento anterior, os testes são repetidos cinco vezes. Para esta etapa de testes a configuração das simulações de itinerários é a seguinte:

- Tempo de parada: 5 s;
- Tempo do trajeto entre dois pontos: 75 s;
- Intervalo de tempo entre chamadas: entre 20 e 25 s.

Os valores escolhidos entre 20 e 25 s permitem a realização de ao menos duas chamadas por dispositivo em cada trecho do itinerário, que nesta etapa tem a duração de 75 s. A configuração utilizada para as simulações desta fase resulta em uma média de 200 chamadas por execução.

Os resultados da segunda etapa de testes de desempenho do sistema com o aplicativo Android são apresentados na Tabela 5.

Tabela 4 – Resultados dos testes de desempenho dos notificadores Android (5 dispositivos)

Tempos por etapa (ms)	Média	Mediana	Mínimo	Máximo
Descoberta	1356	1231	61	2838
Aceite (1ª conexão)	3458	2745	1417	7161
Aceite de conexões	2537	2280	1269	7418
Execução do protocolo	25	21	2	166

6.2.1 Discussão sobre o desempenho dos notificadores Android

A análise dos dados apresentados nas tabelas 4 e 5 permite inferir que, dobrando o número de notificadores, o tempo médio de descoberta aumenta cerca de 55%, o tempo de aceitação da 1ª conexão cai 12% e os tempos de aceitação das conexões posteriores aumentam apenas 2,5%. O tempo de execução do protocolo permaneceu praticamente o mesmo. O aumento mais expressivo é verificado no pior caso do tempo de descoberta, quase 300% mais longo. Ainda assim, o tempo total (descoberta + 1ª conexão) fica abaixo dos 20 s, bastante inferior ao tempo de viagem entre dois pontos de parada sucessivos.

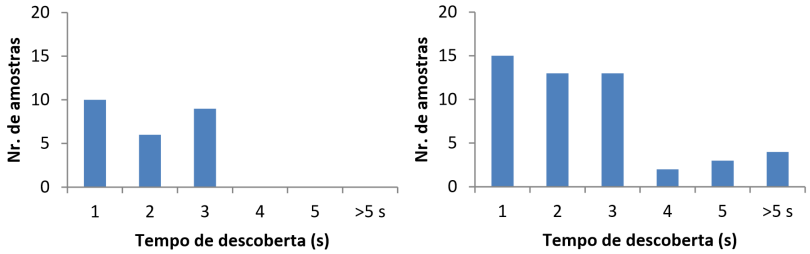
Os histogramas com as distribuições de frequência dos tempos coletados são mostrados nas figuras 9, 10 e 11. Como se observa na Figura 9, a maioria dos notificadores descobre o servidor antes de 3 s; no caso de 10 notificadores, há ocorrências acima de 5 s até o máximo de 11,2 s, conforme consta na Tabela 5.

A Figura 10 mostra os tempos da aceitação das conexões. Conforme observa-se na Figura 10(a), não ocorrem tempos inferiores a 2 s. Já com 10 monitores, aparecem eventos com tempos menores, conforme

Tabela 5 – Resultados dos testes de desempenho dos notificadores Android (10 dispositivos)

Tempos por etapa (ms)	Média	Mediana	Mínimo	Máximo
Descoberta	2114	1660	165	11233
Aceite (1ª conexão)	3045	2571	733	7926
Aceite de conexões	2600	2425	102	7926
Execução do protocolo	25	21	2	175

Figura 9 – Distribuição de frequência dos tempos de descoberta do BDS. a: 5 dispositivos. b: 10 dispositivos (tempos em segundos)

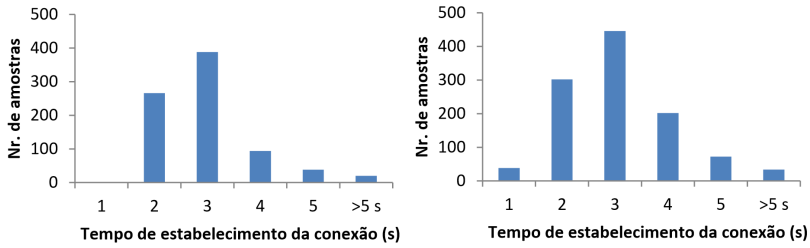


(a) (b)

já indicado na Tabela 5 (tempo mínimo de 0,73 s). Observa-se, ainda, que a maioria das conexões é formada com menos de 3 s, com rápido decaimento de ocorrências maiores que 5 s. A mediana, neste caso, é de 2,57 s conforme apresentado na Tabela 5. Uma vez estabelecidas as conexões, o tempo de execução do protocolo para troca de informações de identidade e localização entre BDS e notificador é bastante baixo, conforme se observa na Figura 11. A distribuição de frequências destes tempos parece não ser afetada pelo aumento no número de notificadores.

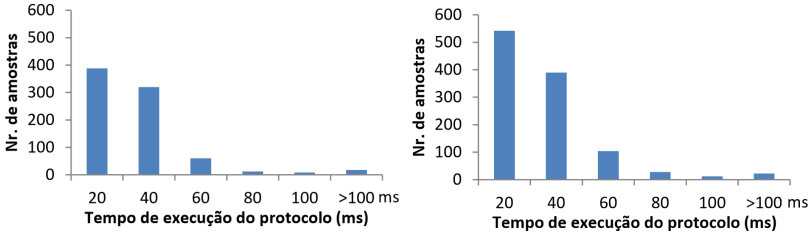
Em todos os casos analisados, os tempos envolvidos na descoberta, conexão e execução do protocolo estão compatíveis com o tempo de percurso de um ônibus entre dois pontos de parada.

Figura 10 – Distribuição de frequência dos tempos de estabelecimento de conexões. a: 5 dispositivos. b: 10 dispositivos (tempos em segundos)



(a) (b)

Figura 11 – Distribuição de frequência dos tempos de execução do protocolo a: 5 dispositivos. b: 10 dispositivos (tempos em ms)



(a) (b)

6.3 BENCHMARK DAS VERSÕES DO NOTIFICADOR

Esta fase de testes visa comparar o desempenho das duas implementações do aplicativo de notificação. Emprega-se a versão desenvolvida sobre a API Bluecove utilizando cinco dispositivos Android com sistema operacional 4.1.2, sendo 4 Samsung Galaxy SII e um *tablet* Samsung Galaxy Note 10.1. Conforme apresentado no Capítulo 3, o Android 4.1.2 é a última versão a utilizar o pilha de protocolos BlueZ, sendo a última versão compatível com a biblioteca Bluecove. Para a condução dos testes, utiliza-se a mesma configuração da rodada de testes com 5 notificadores Android.

6.3.1 Discussão do benchmark

Os resultados dos testes de desempenho com 5 notificadores Bluecove são apresentados na Tabela 6. Conforme apresentado na tabela, observa-se que a versão Bluecove apresenta um tempo médio de descoberta do servidor maior que a versão Android. A Figura 12 compara as distribuições de frequências dos tempos de descoberta do BDS de ambas as implementações.

Enquanto a versão Android resulta tempos de descoberta do servidor nas faixas de 1 a 3 s, os testes com o aplicativo Bluecove resultaram tempos dentro das faixas de 11 e 12 s, cerca de 10 s a mais que a implementação Android.

Comparando-se os tempos de aceitação das conexões, a versão Bluecove apresentou ligeira melhora em relação à versão Android, como

Tabela 6 – Resultados dos testes de desempenho dos notificadores Bluecove (5 dispostivos)

Tempos por etapa (ms)	Média	Mediana	Mínimo	Máximo
Descoberta	11159	10689	11948	11096
Aceite (1a conexão)	1003	914	841	1797
Aceite de conexões	2003	1709	817	11838
Execução do protocolo	31	24	2	417

se pode verificar na Figura 13, o maior número de ocorrências está dentro da faixa de 2 s.

Para a versão Android, a maior concentração se dá no bloco de 3 s. A mediana resultante desta fase de testes é 1709 ms para a versão Bluecove, enquanto que a versão Android resultou uma mediana de 2280 ms. Existem ocorrências de 12 s no tempo de aceitação de conexões conforme apresentado na Tabela 6.

O desempenho da execução do protocolo para a versão Bluecove apresenta uma média de 31 ms e mediana de 24 ms, conforme apresentado na Tabela 6, ligeiramente maior que a versão Android, que apresentou valores de 25 ms e 21 ms para estes parâmetros (Tabela 4). Porém, levando-se em conta o pequeno tempo de execução do protocolo, pode-se considerar que o desempenho de ambas as imple-

Figura 12 – Distribuição de frequências dos tempos de descoberta do BDS nas duas versões do aplicativo (tempos em segundos).

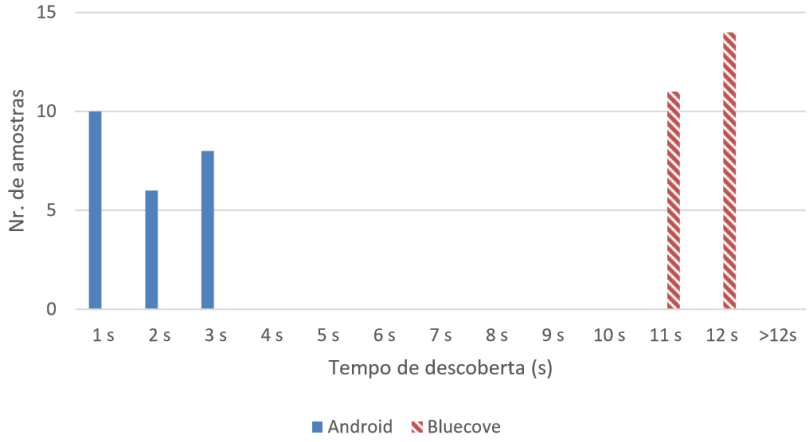
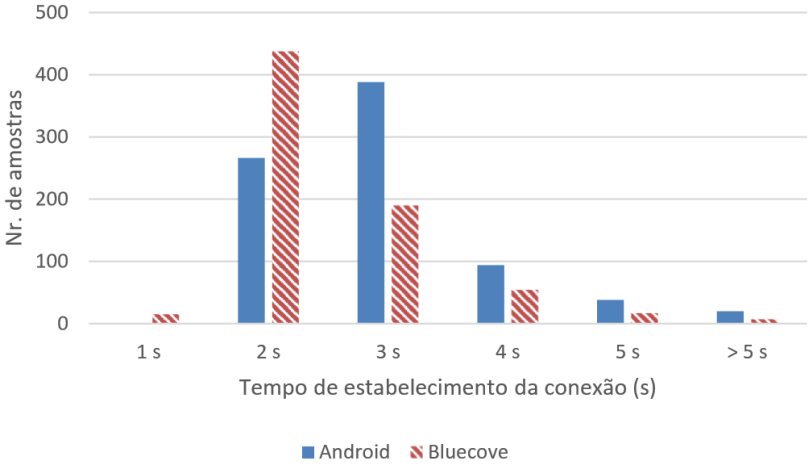


Figura 13 – Distribuição de frequências dos tempos de estabelecimento de conexões de ambas as implementações (tempos em segundos)



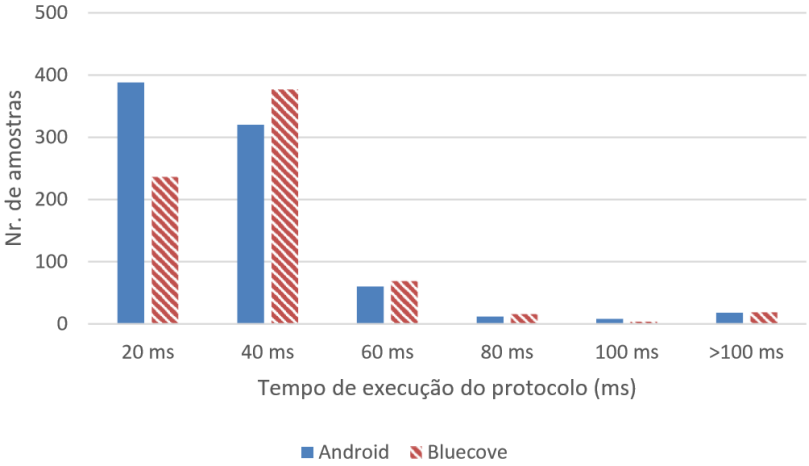
mentações permaneceu equivalente quanto à execução do protocolo. A Figura 14 apresenta a comparação das distribuições de frequências dos tempos de execução do protocolo para ambas as implementações. De forma geral, a implementação sobre a API Bluecove apresentou pior desempenho nos tempos de descoberta, uma ligeira melhora nos tempos de aceitação de conexões e manteve o desempenho nos tempos de execução do protocolo.

Observa-se que os tempos de descoberta do servidor não exercem grande peso no desempenho final do sistema, visto que a operação de descoberta é realizada apenas uma vez durante a execução dos aplicativos de notificação. Desta forma, levando-se em conta os eventos de aceitação de conexão e execução do protocolo, conclui-se que o desempenho da implementação Bluecove é ligeiramente melhor que a implementação Android.

6.4 AVALIAÇÃO DO BDS

Objetiva-se avaliar a implementação do servidor BDS em termos de escalabilidade e capacidade de atender requisições. Objetiva-se também determinar a taxa ótima de recebimento e processamento de requisições por segundo. Pretende-se simular momentos de alta ocupação do servidor, dispondo um pequeno número de notificadores. Para au-

Figura 14 – Distribuição de frequências dos tempos de execução do protocolo de ambas as implementações (tempos em ms)



mentar a taxa de recebimento de mensagens no servidor, utilizam-se intervalos de tempo entre chamadas inferiores a 1 s, em cada dispositivo. Os testes são conduzidos com o emprego de 6 dispositivos Android com sistema operacional 4.1.2, sendo 4 Samsung Galaxy SII, um *tablet* Samsung Galaxy TAB 8.9 (gtp-7300) e um *tablet* Samsung Galaxy Note 10.1 rodando a versão do aplicativo de notificação desenvolvido sobre a biblioteca Bluecove.

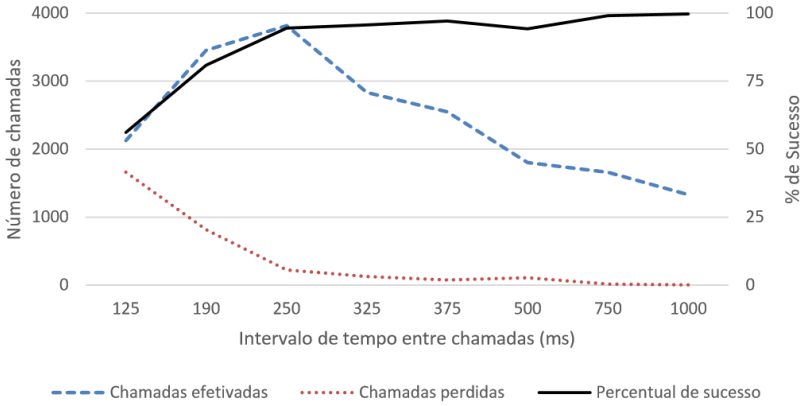
Divide-se esta fase de testes em oito rodadas, variando-se o intervalo de tempo entre chamadas, com valores entre 125 ms e 1 s. Cada rodada é repetida cinco vezes, empregando os 6 notificadores em um percurso de cinco minutos.

6.4.1 Discussão sobre a avaliação do BDS

A Figura 15 apresenta um gráfico contendo a quantidade de chamadas efetivadas e perdidas em função do tempo de espera entre chamadas utilizado em cada rodada de testes. Por questões de simplicidade, serão discutidos apenas os resultados das rodadas 1, 3 e 5.

A primeira rodada de testes de avaliação do BDS utilizou um tempo de espera entre chamadas de 1 s e resultou uma taxa média de 4,6 chamadas/s. Vale observar que, teoricamente, esta configuração

Figura 15 – Quantidade de chamadas efetivadas e perdidas e percentual de sucesso em função do intervalo de tempo entre chamadas. Resultados obtidos em um ensaio com 5 minutos de duração



deveria resultar em uma taxa média de 6 chamadas/s, assumindo-se que 6 notificadores realizam uma requisição a cada segundo. Observou-se que as perdas se deram devido a atrasos na execução do processo de *inquiry* e em atrasos no processamento de algumas chamadas. Esta rodada de testes gerou uma média de 1330 chamadas efetivadas e uma média de 4 chamadas perdidas, resultando um percentual de sucesso (chamadas efetivadas/total de chamadas) de 99,7%.

Para a terceira rodada de testes de testes de avaliação do BDS, utilizou-se um tempo de espera entre chamadas de 500 ms. Comparando-se os resultados obtidos na primeira rodada, observa-se que, ao se diminuir o tempo de espera entre chamadas pela metade, houve redução no percentual de sucesso, que resultou em 94,2%. Houve um aumento na taxa de chamadas efetivas, que resultou uma média de 6,3 chamadas/s.

A sexta rodada de testes de avaliação do BDS utiliza um tempo de espera entre chamadas de 250 ms. Comparando-se os resultados obtidos na terceira rodada, obteve-se neste ensaio uma taxa média de chamadas efetivadas de 13 chamadas/s, equivalente ao dobro obtido na terceira rodada. O percentual de sucesso obtido com esta configuração foi de 94,5%.

Conforme apresentado na Figura 15, observa-se que, ao se testar tempos de espera entre chamadas menores que 250 ms ocorre decaimento no desempenho. Esta degradação se dá devido à grande quantidade de chamadas perdidas, decorrentes da crescente ocorrência de

Tabela 7 – Desempenho do protocolo e estabelecimento de conexões

Taxa	Protocolo (média)	Protocolo (mediana)	Aceite (média)	Aceite (mediana)
1 s	25	16	92	65
500 ms	27	16	139	74
250 ms	26	17	104	86
Médias:	26	17	112	75

colisões no tratamento de requisições no servidor e indicando a ineficiência deste em atender taxas acima de 13 chamadas/s com apenas 6 dispositivos. O pior caso ocorre com a utilização de um intervalo de espera de 125 ms. Esta configuração, apesar de resultar uma taxa de 7,34 chamadas/s, resultou o menor valor de percentual de sucesso, neste caso de 56,1%, ou seja, 44% das chamadas realizadas foram perdidas devido a alta taxa de colisões.

A Tabela 7 apresenta os tempos médios obtidos na execução dos testes para as rodadas executadas com intervalo de espera de 1 s, 500 ms e 250 ms, que totalizam 6948 chamadas realizadas. Observa-se que o tempo médio de execução do protocolo é de 26 ms. Já o tempo médio de estabelecimento de conexões é de 112 ms. Estes valores resultam um tempo médio de duração de uma chamada em 138 ms. Durante este intervalo de tempo, o servidor mantém uma conexão exclusiva com um cliente e fica impossibilitado de aceitar novas requisições.

6.5 SIMULAÇÕES E COLETA DE DADOS DE INTERESSE

Visando preparar o protótipo para testes de campo, realizam-se simulações de itinerário a fim de extrair as informações de interesse sobre utilização da linha em ambiente controlado. Para alcançar este objetivo, implementa-se um simulador de linhas de ônibus que é integrado à aplicação BDS, desenvolvida em Java. A partir do uso do simulador, podem-se obter informações mais detalhadas sobre o contexto das chamadas recebidas pelo BDS. Os seguintes parâmetros podem ser configurados no simulador de linhas:

- Tempo de parada: simula o tempo em segundos que o ônibus aguarda em um ponto;
- Tempo de segmento: configura o tempo no segmento entre duas

paradas;

- Tempo base de chamada: configura o tempo mínimo em segundos que o notificador deve aguardar para realizar uma nova notificação;
- *Offset* do tempo de chamada: permite configurar um intervalo aleatório entre 1 e *offset* segundos, que é adicionado ao tempo de espera da notificação;
- Número de paradas: Configura o número de pontos de ônibus a ser utilizado na simulação do itinerário;
- Atraso no atendimento da chamada: Permite configurar um tempo em segundos que o servidor deve aguardar para processar a mensagem de ACK e registrar a chamada em histórico;

Durante a execução da simulação, o BDS fornece aos notificadores informações sobre a linha como o trecho onde se encontram. O BDS desempenha também o papel de centralizador das conexões Bluetooth e de concentrador de dados. O número de passageiros a bordo em um dado momento é fornecido pelo BDS aos notificadores presentes em cada trecho de itinerário.

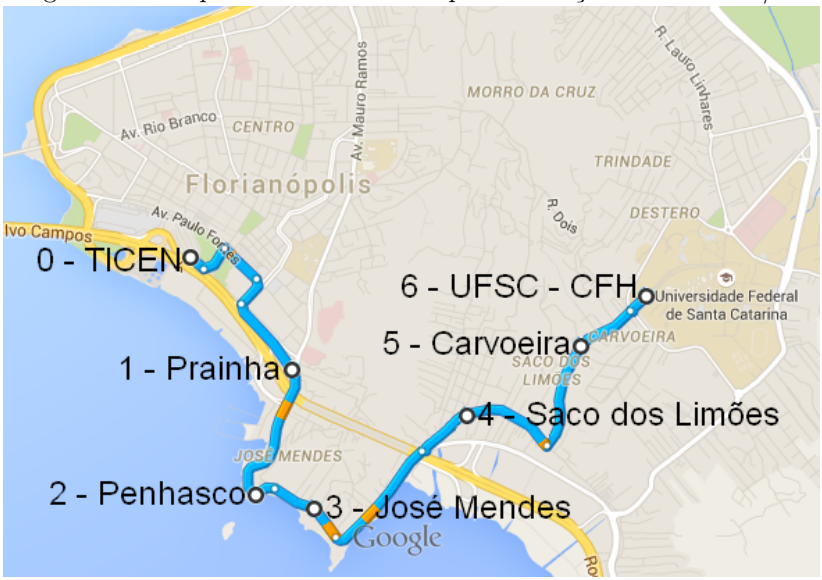
A execução dos testes por simulação emprega um itinerário baseado em uma das linhas de ônibus que atende a Universidade Federal de Santa Catarina com origem no terminal central de Florianópolis. Este itinerário é dividido em 6 trechos e contém 7 pontos de parada descritos na Figura 16. A simulação emprega tempos fixos de parada nos pontos e de percurso nos trechos de itinerário. A simulação de embarques e desembarques é feita através da ativação manual do aplicativo de notificação em dispositivos pré determinados.

6.5.1 Teste de geração da matriz O/D

Realiza-se o teste de geração de matrizes O/D sobre o itinerário proposto empregando-se 5 dispositivos que simularam o seguinte comportamento:

- 3 passageiros embarcam no terminal central.
- O primeiro passageiro desembarca no ponto 4 (Saco dos Limões)
- O segundo passageiro desembarca no ponto 5 (Carvoeira)

Figura 16 – Mapa da linha utilizada para obtenção da matriz O/D.



- O terceiro passageiro desembarca no ponto 6 (UFSC - CFH).
- Um passageiro embarca no ponto 2 (Prainha) e desembarca no ponto 5 (Carvoeira).
- O último passageiro embarca no ponto 3 (Penhasco) e desembarca no ponto 6 (UFSC - CFH).

O experimento realizado produz a matriz O/D e o relatório de lotações por trecho do itinerário que podem ser vistos nas tabelas 8 e 9. Embora o ensaio seja de pequena dimensão, os resultados obtidos mostram a correta execução dos algoritmos usados na reconstituição da

Tabela 8 – Matriz O/D resultante da simulação

O/D	Prainha	Penhasco	J. Mendes	S. dos Limões	Carvoeira	UFSC
TICEN				1	1	1
Prainha					1	
Penhasco						1

Tabela 9 – Relatório de lotações por trecho do itinerário.

Segmento	Hora	Lotação
TICEN - Prainha	19:22:40	3
Prainha - Penhasco	19:23:15	4
Penhasco - José Mendes	19:23:50	5
José Mendes - Saco dos Limões	19:24:25	5
Saco dos Limões - Carvoeira	19:25:00	4
Carvoeira - UFSC	19:25:35	2

matriz O/D e totalização dos passageiros a bordo.

6.6 CONSIDERAÇÕES FINAIS

Os testes de desempenho dos notificadores foram aplicados com as duas implementações do aplicativo de notificação. Embora o tempo de descoberta seja maior no aplicativo desenvolvido com Bluecove, esta versão apresenta um desempenho melhor na execução do mecanismo de notificação, pois apresenta um tempo de estabelecimento de conexões menor.

Os testes de escalabilidade permitiram determinar o tempo médio de execução de uma notificação em 138 ms. Também foi possível determinar a taxa máxima de processamento de chamadas do BDS, resultando em torno de 13 chamadas/s.

Assumindo-se um pior caso de utilização do servidor em termos de aplicação real do sistema, considera-se um trecho curto de itinerário, com a duração de 20 segundos e um veículo com sua capacidade máxima, em torno de 120 passageiros a bordo. A arquitetura apresentada propõe a realização de pelo menos duas chamadas por trecho do itinerário, consequentemente os notificadores realizariam uma chamada a cada 10 segundos, o que resultaria uma taxa média de emissão de 12 mensagens por segundo. Por outro lado, entende-se que num conjunto de 120 dispositivos dificilmente haverá ausência de colisões, que, conforme discutido anteriormente, acarretam em atrasos e consequente queda de desempenho. Uma alternativa para contornar a incapacidade de recebimento de mensagens é o emprego de multiplos servidores por veículo. Esta abordagem traria maior robustez ao mecanismo de aquisição de dados, permitindo a obtenção de amostras mais significativas, porém, para sua aplicação, tornam-se necessários mecanismos de

sincronização e tolerância a faltas em sistemas distribuídos.

Por fim, ensaios de utilização do aplicativo utilizando o simulador de rotas comprovam a produção eficiente de relatórios de utilização da linha, como as matrizes O/D e os relatórios de lotação a bordo. Apesar do pequeno número de dispositivos Android disponíveis para realização dos testes, os resultados indicam a viabilidade do método proposto.

7 CONCLUSÃO

7.1 CONSIDERAÇÕES FINAIS

Neste trabalho, propôs-se um sistema de aquisição de dados de utilização do transporte público urbano que emprega dispositivos móveis populares e um meio de comunicação de dados de baixo custo e consumo. Realizou-se o projeto e implementação de um sistema baseado na tecnologia Bluetooth disponível em *smartphones* com sistema operacional Android, seguindo uma estratégia de *participatory sensing*. Realizaram-se testes do sistema em ambiente controlado com o auxílio de um simulador de itinerários para a obtenção de dados sobre a rota percorrida. Estes dados permitiram a extração de indicativos de interesse tais como matrizes O/D e relatórios de lotação ao longo do itinerário. Com base nos resultados obtidos em simulação, pode-se afirmar que o sistema proposto fornece indicadores fidedignos de utilização de ônibus.

Os testes realizados permitiram também avaliar o impacto da execução do aplicativo de notificação na autonomia dos dispositivos e resultaram um consumo médio de bateria entre 0,42% e 2,22%, durante vinte minutos de execução, comprovando o baixo consumo de energia da tecnologia Bluetooth.

Avaliou-se a capacidade de processamento de chamadas do servidor através de testes de escalabilidade. A execução dos testes permitiu identificar uma taxa máxima de atendimento de chamadas de 13 chamadas/s o que, teoricamente possibilitaria identificar em torno de 120 passageiros durante um trecho do percurso com duração de 20s (abstraindo-se as perdas de chamadas por colisões).

Entende-se que na prática, os dados coletados refletirão o comportamento de apenas uma amostra da população presente à bordo. A qualidade dos dados obtidos dependerá da significância da amostra coletada. Conforme o uso de um aplicativo deste tipo torna-se popular, a tendência é que as amostras sejam cada vez mais significativas, melhorando gradativamente a qualidade dos dados.

O método proposto poderia ser agregado a um aplicativo de auxílio à utilização do transporte público já estabelecido no mercado, como o Moovit por exemplo. Desta forma, os usuários poderiam fornecer informações ao sistema de forma autônoma ao mesmo tempo em que se beneficiam de informações obtidas em tempo real. Usuários aguardando no ponto poderiam receber informações como a previsão das

próximas chegadas e as respectivas estimativas de lotação dos ônibus.

Um resumo dos resultados obtidos neste trabalho foi publicado no 29° congresso da Associação Nacional de Pesquisa e Ensino em Transportes em 2015 :

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.

7.2 TRABALHOS FUTUROS

A versão 4.0 do Bluetooth introduziu uma nova tecnologia de transmissão chamada Bluetooth Low Energy (BLE), que apresenta um consumo de energia 17 vezes menor que o Bluetooth clássico. BLE permitiu o surgimento de uma nova tecnologia de guiamento baseado em sensores fixos(KOÜHNE; SIECK, 2014), dentre os quais citam-se o padrão iBeacon da Apple (iBEACON, 2016) e o Eddystone (EDDYSTONE, 2016), lançado em julho de 2015 pela Google.

A tecnologia BLE ainda não possui suporte integral para desenvolvimento sobre a plataforma Android no momento tecnológico atual, porém, oferece grande potencial para estender o sistema de identificação de passageiros em trabalhos futuros. O hardware BLE é utilizado com mais parcimônia, devido ao pequeno tamanho das mensagens trafegadas. Em termos de taxa de transmissão, a quantidade de informações que é suportada pelo padrão BLE não é um fator limitante, visto que o protocolo de identificação de passageiros trafega mensagens de pequeno tamanho. Com a utilização de BLE, o processo de varredura não é necessário, pois é possível alertar de forma automática a presença do BDS dentro do ônibus. Por fim, o baixo custo de componentes BLE apresenta outra vantagem, permitindo às empresas de transporte um custo orçamentário ainda mais baixo na implementação de um sistema de monitoramento de viagens que traria vários benefícios aos seus usuários e gestores.

REFERÊNCIAS BIBLIOGRÁFICAS

ALBUQUERQUE, H.; CAVALCANTE, S.; URBANO, P. **Um estudo sobre pilhas Bluetooth e suas limitações em sistemas embarcados**. 2010. Trabalho de Graduação em Engenharia da Computação. Centro de Informática. Universidade Federal de Pernambuco.

ALCORN, E. et al. **Bluetooth Architecture Overview**. [S.l.], 3 2011. Windows Embedded Compact 7. Disponível em: `https://pt.scribd.com/document/54515508/Bluetooth-Architecture-Overview`.

ANATEL. 2 2015. Telefonia Móvel - Acessos. Disponível em: `http://www.anatel.gov.br/dados/index.php?option=com_content&view=article&id=270`. Acesso em: 17/07/2015.

ANDROID. 2016. Página oficial. Disponível em: `https://www.android.com/`. Acesso em: 17/08/2016.

ANTT. 2015. Agência Nacional de Transportes Terrestres. Glossário. Disponível em: `http://www.antt.gov.br/html/objects/_downloadblob.php?cod_blob=6529`. Acesso em: 12/05/2015.

APPLE. 2016. OS X Bluetooth Stack. Disponível em: `https://developer.apple.com/bluetooth/`. Acesso em: 18/08/2016.

ATTAM, A.; MOISEENKO, I. **NDNBlue: NDN over Bluetooth**. [S.l.], 11 2013. Relatório técnico.

BANDARA, U. et al. Design and implementation of a Bluetooth signal strength based location sensing system. In: IEEE. **Radio and Wireless Conference, 2004 IEEE**. [S.l.], 2004. p. 319–322.

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

BLUECOVE. 2016. Página oficial. Disponível em: `http://bluecove.org/`. Acesso em: 17/08/2016.

BLUETOOTH. 2016. Página oficial. Disponível em: http://www.bluetooth.com. Acesso em: 10/05/2016.

BOYLE, D. K. **Passenger counting systems - A synthesis of transit practice**. Washington, DC, USA, 2008. Relatório técnico. TCRP Synthesis 77. Disponível em: http://onlinepubs.trb.org/onlinepubs/tcrp/tcrp_syn_77.pdf.

CACERES, N.; WIDEBERG, J.; BENITEZ, F. Deriving origin destination data from a mobile phone network. **Intelligent Transport Systems, IET**, IET, v. 1, n. 1, p. 15–26, 2007.

CAI, J. et al. A Bluetooth toy car control realization by Android equipment. In: IEEE. [S.l.], 2011. p. 2429–2432. Transportation, Mechanical, and Electrical Engineering (TMEE), 2011 International Conference on.

CALLAGHAN, M. et al. Case study on the Bluetooth vulnerabilities in mobile devices. **IJCSNS International Journal of Computer Science and Network Security**, Digital Library of Academic Research, v. 6, n. 4, p. 125–129, 2006.

CEDER, A. **Public Transit Planning and Operation: Modeling, Practice and Behavior**. [S.l.]: CRC Press, 2007.

CHENG, S.; LIU, B.; ZHAI, B. Bus arrival time prediction model based on APC data. **Transportation of China (AFTC 2010), 6th Advanced Forum**, p. 165–169, 2010.

CHOI, M.-k. et al. Wireless network security: Vulnerabilities, threats and countermeasures. **International Journal of Multimedia and Ubiquitous Engineering**, Citeseer, v. 3, n. 3, p. 77–86, 2008.

EDDYSTONE. 2016. Página oficial. Disponível em: https://developers.google.com/beacons/. Acesso em: 10/05/2016.

FRIESEN, M.; MCLEOD, R. Bluetooth in intelligent transportation systems: A survey. **International Journal of Intelligent Transportation Systems Research**, Springer, p. 1–11, 2014.

GIELOW, F. H.; SANTOS, A.; NOGUEIRA, M. **Uma Análise Experimental de Ataques *Jamming* no Acesso ao Meio em Redes de Sensores Sem Fio**. Curitiba - PR - Brasil, 2012. Nucleo de Redes Sem Fio e Redes Avançadas - DINF - Universidade Federal do Paraná.

GRÉGIO, A. **Tecnologia Bluetooth e Aspectos de Segurança**. 2009. Relatório Técnico. Instituto de Computação - UNICAMP - Brasil. Disponível em: `http://www.ic.unicamp.br/~ducatte/mo401/1s2009/T2/079779-t2.pdf`.

HARTE, L. **Introduction to Bluetooth**. 2. ed. Fuquay Varina, NC, USA: Althos, 2009. ISBN 1932813721.

HOLTMANN, M.; KRASNYANSKY, M. et al. **BlueZ**. 2007. Official Linux Bluetooth protocol stack. Disponível em: `http://www.bluez.org`.

HOWES, T. **Discovery Whitepaper: Service Discovery Applications**. 2008. Bluetooth Special Interest Group. Manual. Disponível em: `http://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?docid=144841`.

HUANG, A. S.; RUDOLPH, L. **Bluetooth essentials for programmers**. Cambridge, UK: Cambridge University Press, 2007.

iBEACON. 2016. Página oficial. Disponível em: `https://developer.apple.com/ibeacon/`. Acesso em: 17/08/2016.

iOS. 2016. Página oficial. Disponível em: `https://www.apple.com/br/ios/`. Acesso em: 17/08/2016.

KARA, A.; BERTONI, H. L. Blockage/shadowing and polarization measurements at 2.45 GHz for interference evaluation between Bluetooth and IEEE 802.11 WLAN. **Antennas and Propagation Society International Symposium, 2001. IEEE**, v. 3, p. 376–379, 2001.

KOSTAKOS, V. **Using Bluetooth to capture passenger trips on public transport buses**. Lab:USE, University of Madeira, Human Computer Interaction Institute, Carnegie Mellon University, 2008. Paper.

KOSTAKOS, V.; CAMACHO, T.; MANTERO, C. Towards proximity-based passenger sensing on public transport buses. **Personal and ubiquitous computing**, Springer-Verlag, v. 17, n. 8, p. 1807–1816, 2013.

KOÜHNE, M.; SIECK, J. Location-based services with iBeacon technology. In: IEEE. **Artificial Intelligence, Modelling and Simulation (AIMS), 2014 2nd International Conference on**. [S.l.], 2014. p. 315–321.

KUMAR, B. **Jsr-82: Java APIis for Bluetooth**. 2010. Java Community Process. Disponível em `http://www.jcp.org/jsr/detail/82.prt`.

MISRA, A. et al. Crowdsourcing and its application to transportation data collection and management. **Transportation Research Record: Journal of the Transportation Research Board**, Transportation Research Board of the National Academies, n. 2414, p. 1–8, 2014.

MOOVIT. 2016. Página oficial. Disponível em: `http://moovitapp.com/pt-br`. Acesso em: 20/07/2015.

NASIBOGLU, E. et al. Origin-destination matrix generation using smart card data: Case study for Izmir. **Problems of Cybernetics and Informatics (PCI), 2012 IV International Conference**, p. 1–4, 2012.

PERING, T. et al. Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces. **Proceedings of the 4th international conference on Mobile systems, applications and services**, p. 220–232, 2006.

PETKOVICS, Á.; FARKAS, K. Efficient event detection in public transport tracking. **Telecommunications and Multimedia (TEMU), 2014 International Conference on**, p. 74–79, 2014.

PIERI, G.; KRAUS, W.; FARINES, J.-M. **Bluemob: Algoritmo Dinâmico para Formação de Redes Bluetooth em Aplicações de Sensoriamento Urbano**. Tese (Doutorado) — Universidade Federal de Santa Catarina, Florianópolis - SC - Brasil, 2016.

RUFINO, N. M. d. O. **Segurança em Redes sem Fio**. São Paulo: Novatec, 2011. 18–20 p. ISBN 8575220705.

SEABORN, C.; ATTANUCCI, J.; WILSON, N. Analyzing multimodal public transport journeys in London with smart card fare payment data. **Transportation Research Record: Journal of the Transportation Research Board**, Transportation Research Board of the National Academies, n. 2121, p. 55–62, 2009.

SHAHRIYAR, R. et al. Remote controlling of home appliances using mobile telephony. **International Journal of Smart Home**, v. 2, n. 3, p. 37–54, 2008.

STREETBUMP. 2016. Página oficial. Disponível em: http://www.streetbump.org. Acesso em: 25/08/2015.

THAMRIN, T.; SAHIB, S. The Inquiry and Page Procedure in Bluetooth Connection. In: **Soft Computing and Pattern Recognition, 2009. SOCPAR '09. International Conference of**. [S.l.: s.n.], 2009. p. 218–222.

WALEED, G. M. et al. Bluetooth wireless network authentication using radio frequency communication protocol. **Journal of Computer Science**, Citeseer, v. 5, n. 9, p. 646, 2009.

WAZE. 2016. Página oficial. Disponível em: http://www.waze.com/pt-BR. Acesso em: 20/07/2015.

WIKIPEDIA. 2016. Bluetooth. Disponível em: https://en.wikipedia.org/wiki/Bluetooth. Acesso em: 10/05/2016.

WORK, D.; BAYEN, A. Impacts of the mobile internet on transportation cyberphysical systems: traffic monitoring using smartphones. In: **National Workshop for Research on High-Confidence Transportation Cyber-Physical Systems: Automotive, Aviation, & Rail**. [S.l.: s.n.], 2008. p. 18–20.

ZHAO, L. A heuristic method for analyzing driver scheduling problem. **Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on**, IEEE, v. 36, n. 3, p. 521–531, 2006.

ZIMMERMAN, J. et al. Field trial of Tiramisu: crowd-sourcing bus arrival times to spur co-design. In: ACM. **Proceedings of the SIGCHI Conference on Human Factors in Computing Systems**. [S.l.], 2011. p. 1677–1686.

APÊNDICE A – Histórico de Versões do Bluetooth

A.1 BLUETOOTH 1.0 E 1.0B

São as primeiras especificações da tecnologia Bluetooth. Foram encontrados vários problemas de interoperabilidade entre dispositivos.

A.2 BLUETOOTH 1.1

Esta versão, lançada em fevereiro de 2001 estabeleceu a tecnologia de comunicação como o padrão IEEE 802.15.1 WPAN (*Wireless Personal Area Network*). A versão 1.1 corrige vários problemas encontrados nas primeiras versões e acrescenta suporte a RSSI.

A.3 BLUETOOTH 1.2

A versão 1.2 aperfeiçoa o suporte a formação de scatternets, apresenta maior robustez contra interferências. Esta versão aprimorou o estabelecimento de conexões e o processo de descoberta e trouxe também atualizações no processamento de voz. Versão lançada em novembro de 2003.

A.4 BLUETOOTH 2.0

A versão 2.0 introduziu o EDR (*Enhanced Data Rate*) que permitiu um aumento da taxa de transmissão para 3.0 Mb/s com um consumo de energia mais baixo. Esta versão também trouxe várias correções de falhas da versão 1.2 e melhoria na comunicação entre dispositivos.

A.5 BLUETOOTH 2.1

Lançada em agosto de 2007, a versão 2.1 trouxe melhorias no processo de busca de dispositivos, acrescentando mais informações nos sinais de Inquiry com a introdução do EIR (*Extended inquiry response*), permitindo uma melhor seleção de dispositivos antes de realizar a conexão. Esta versão também trouxe melhorias no gerenciamento do consumo de energia e em aspectos de segurança, como a adição de novos recursos de criptografia e proteção do enlace de dados.

A.6 BLUETOOTH 3.0 + HS

Bluetooth Plus High Speed, nesta versão a principal evolução foi feita em termos de desempenho que contempla uma taxa de transferência de dados que de até 24 Mb/s. Para atingir esta velocidade foi necessário adotar um link 802.11 como suporte de transferência. A principal funcionalidade é a AMP (*Alternative MAC/PHY*) que permite a utilização do link 802.11 e alta velocidade. A comunicação via radio Bluetooth ainda é utilizada na descoberta de dispositivos, conexão inicial e configurações de perfil. Assim, as conexões de baixo consumo Bluetooth são usados apenas quando o sistema estiver ocioso, e links de alta velocidade são utilizados apenas quando grandes quantidades de dados precisam ser enviadas. A versão 3.0 também trás melhorias no controle de potência dos aparelhos.

A.7 BLUETOOTH 4.0

Esta versão, também chamada de Bluetooth Smart, trás várias melhorias em relação a versão 3.1, sendo a principal a introdução de uma pilha de protocolos totalmente dedicada para transmissõs de baixo consumo de energia, BLE (*Bluetooth Low Energy*). A tecnologia BLE incorpora um controle mais inteligente de consumo de energia, objetivando evitar desperdícios que ocorriam com transferência de dados de pouca informação em alguns tipos de dispositivos como relógios ou monitores cardíacos. Esta versão também incorpora melhorias em termos de velocidade de transmissão e segurança da informação.

A.8 BLUETOOTH 4.1

A versão 4.1 é a mais recente e foi lançada em dezembro de 2013. Esta versão é uma atualização da versão anterior, acrescentando novas funcionalidades com a finalidade de melhorar a usabilidade do usuário final e aumentar o interesse dos desenvolvedores de aplicativos. Dentre as melhorias estão:

- Suporte a interoperabilidade com rádios LTE, cuja coordenação entre as duas tecnologias acontece agora de maneira transparente;
- Melhoria nas conexões: o intervalo de tempo para reconexões agora é flexível e variável, melhorando a usabilidade ao permitir

a reconexão sem a intervenção do usuário;

- Transferência de dados foi melhorada permitindo que dados em massa sejam transmitidos.

- Melhoria do ambiente de desenvolvimento: Bluetooth Smart permite agora a criação de aplicações que podem assumir múltiplas funções: um dispositivo pode desempenhar por exemplo os papeis de um periférico Bluetooth Smart e um hub Bluetooth Smart ao mesmo tempo.

ANEXO A – Código-fonte do protótipo

Todo código-fonte desenvolvido neste projeto está disponível on line no repositório GitHub. O acesso aos fontes, por artefato é feito através do acesso dos endereços abaixo:

Aplicativo de notificação:
https://github.com/AngeloMelo/BeaconMonitor

Aplicativo de notificação (Bluecove):
https://github.com/AngeloMelo/BluecoveBeaconMonitor

BDS e simulador de linhas:
https://github.com/AngeloMelo/PCBeacon