UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

GIONATTA MARCON MOCELLIN
GIUSEPE PIETRO NIQUELE

# DESENVOLVIMENTO DE APLICATIVO MÓVEL PARA ACOMPANHAMENTO DO TRANSPORTE COLETIVO DE CURITIBA

TRABALHO DE CONCLUSÃO DE CURSO

**CURITIBA**

**2015**

GIONATTA MARCON MOCELLIN
GIUSEPE PIETRO NIQUELE

# DESENVOLVIMENTO DE APLICATIVO MÓVEL PARA ACOMPANHAMENTO DO TRANSPORTE COLETIVO DE CURITIBA

Trabalho de Conclusão de Curso de Engenharia de Computação apresentado ao Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de "Engenheiro em Computação".

Orientadora: Prof$^a$. Dr$^a$. Ana Cristina Kochem Vendramin

Coorientadora: Prof$^a$. Dr$^a$. Anelise Munaretto Fonseca

**CURITIBA**

**2015**

## AGRADECIMENTOS

Primeiramente agradecemos à Deus por ter nos dado saúde e força para superar todas as dificuldades encontradas durante todos os anos de graduação.

Agradecemos aos nossos pais, irmãos e demais familiares que sempre nos incentivaram e nos apoiaram nos momentos mais difíceis, não apenas no desenvolvimento deste projeto, mas também na nossa formação como um todo.

Agradecemos às professoras Dr$^a$. Ana Cristina Kochem Vendramin e Dr$^a$. Anelise Munaretto Fonseca, orientadora e coorientadora, pelo apoio e orientação em cada etapa do projeto, sempre com empenho e dedicação.

Ao professor Dr. Mauro Sergio Pereira Fonseca que muito contribuiu para a realização deste projeto, propondo novas ideias e soluções para as dificuldades encontradas.

Agradecemos aos demais professores da universidade que ministraram suas aulas com dedicação e contribuíram, de algum modo, para nossa formação profissional.

À todos que, direta ou indiretamente, fizeram parte de nossa formação, nosso muito obrigado.

# RESUMO

MOCELLIN, Gionatta Marcon. NIQUELE, Giusepe Pietro. DESENVOLVIMENTO DE APLICATIVO MÓVEL PARA ACOMPANHAMENTO DO TRANSPORTE COLETIVO DE CURITIBA. 76 f. Trabalho de Conclusão de Curso – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná. Curitiba, 2015.

O incessante ritmo ditado pelos grandes centros urbanos gera na população a necessidade de minimizar o tempo gasto em seus deslocamentos, principalmente quando o transporte público é utilizado para este propósito. Contudo, é muito difícil determinar com precisão quando um ônibus chegará ao ponto ou ao terminal. Por vezes, isso causa longos períodos de espera, ou porque o ônibus acabou de partir, ou porque se atrasou devido a um imprevisto. Felizmente, a tecnologia evolui a tal ponto que os chamados *smartphones* se tornaram um artefato muito popular. Ainda, o advento de redes móveis e sem fio, tais como o 3G e o Wi-Fi, permite que pessoas fiquem conectadas a maior parte do tempo, independente do lugar em que estejam. Dentre os inúmeros benefícios que os *smartphones* fornecem, a possibilidade de troca de informações entre usuários é uma delas, permitindo um fenômeno chamado *crowdsourcing*. O *crowdsourcing*, aplicado em um âmbito maior - que englobe toda ou a maior parte da cidade - poderia ser um primeiro passo para elevá-la ao nível de uma *Smart City* ou "cidade inteligente", onde a tecnologia é utilizada pela população para a solução/mitigação de um determinado problema urbano, no caso, a mobilidade. Neste projeto é proposto o desenvolvimento de um aplicativo móvel que alie conceitos, tanto de *Smart Cities* quanto de *crowdsourcing*, no qual informações em tempo real sobre o transporte coletivo, especificamente no que compete o estado atual de uma linha de ônibus, sejam disponibilizadas aos usuários.

**Palavras-chave:** Dispositivos móveis, Wi-Fi Direct, Cidades inteligentes, *Crowdsourcing*, Acompanhamento de ônibus

# ABSTRACT

MOCELLIN, Gionatta Marcon. NIQUELE, Giusepe Pietro. . 76 f. Trabalho de Conclusão de Curso – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná. Curitiba, 2015.

The unceasing rhythm dictated by the large urban centers causes in the population the necessity of minimizing the time spent in their locomotion, especially when the public transport is used for this purpose. However, it's difficult to determine with precision when the bus will stop at the bus station. Fortunately, the technology evolved to a point where the smartphones are now a very popular artifact. Nevertheless, the mobile and wireless networks, such as 3G and Wi-Fi, allow the people to stay connected all the time, regardless the place they are. Amongst the severeal benefits brought by the smartphones, the information exchange between the users is one of them, making possible a phenomenon called crowdsourcing. The crowdsourcing, applied in a larger environment – covering the entire, or almost the entire, city – may be the first step to evolve it to a Smart City, where the technology is used by the population to solve/mitigate a specific problem, such as mobility. In this project it is proposed the development of a mobile application which integrates concepts from Smart Cities and crowdsourcing, where real time information concerning the public transport, specifically the actual state of a bus line, is available to users.

**Keywords:** Mobile devices, Wi-Fi Direct, Smart Cities, Crowdsourcing, Bus tracking

## LISTA DE SIGLAS

| | |
|---|---|
| AP | *Access Point* |
| API | *Application Programming Interface* |
| CVS | *Control Version System* |
| DHCP | *Dynamic Host Configuration Protocol* |
| DNS | *Domain Name System* |
| EF | *Environmental Factor* |
| GPS | *Global Positioning System* |
| ICS | *Internet Connection Sharing* |
| ICT | *Information and Communication Technology* |
| IP | *Internet Protocol* |
| JDK | *Java Development Kit* |
| JSP | *JavaServer Pages* |
| LAN | *Local Area Network* |
| MBTS | *Mobile Bus Tracking System* |
| P2P GO | *Peer-to-Peer Group Owner* |
| P2P | *Peer-to-Peer* |
| PAN | *Personal Area Network* |
| QoS | *Quality of Service* |
| RF | Radio Frequência |
| RIT | Rede Integrada de Transporte |
| S.O. | Sistema Operacional |
| SDK | *Software Development Kit* |
| SMS | *Short Message Service* |
| SSID | *Service Set Identifier* |
| TCF | *Technical Complexity Factor* |
| TCP | *Transmission Control Protocol* |
| UAW | *Unadjusted Actor Weight* |
| UCP | *Use Case Points* |
| UDP | *User Datagram Protocol* |
| UI | *User Interface* |
| URBS | Companhia de Urbanização e Saneamento de Curitiba |
| UUCP | *Unadjusted Use Case Point* |
| UUCW | *Unadjusted Use Case Weight* |
| WHN | *Wireless Hosted Network* |
| WLANs | *Wireless Local Area Networks* |
| WPS | *WiFi Protected Setup* |
| XML | *Extended Markup Language* |

**SUMÁRIO**

# 1 INTRODUÇÃO

O conceito de *Smart Cities* (Cidades Inteligentes, em tradução livre) é um sustentáculo elementar para o desenvolvimento urbano sustentável. De certa forma, ele poderia contribuir na solução de diversos problemas críticos, originados a partir da urbanização de grandes cidades como congestionamentos, poluição do meio ambiente e os limites dos recursos naturais. (PAN et al., 2013).

Um outro aspecto do conceito de cidades inteligentes a ser considerado é a importância crescente das tecnologias digitais para um futuro sustentável. Assim sendo, destacam-se seis principais áreas, nas quais essas inovações digitais tem a competência para fazer a diferença: vida inteligente, governança inteligente, economia inteligente, ambiente inteligente, pessoas inteligentes e mobilidade inteligente (SCHUURMAN et al., 2012).

Com o uso da tecnologia da informação e comunicação (do inglês ICT, *Information and Communication Technology*,) a análise e mineração de dados de sensoriamento das grandes cidades é um passo importante para considerá-las uma cidade inteligente. Por exemplo, informações sobre a mobilidade dos veículos e pessoas tornaram-se temas em voga nos últimos tempos devido à prevalência do GPS (*Global Positioning System*) e outras tecnologias de localização. Esse conhecimento sobre a cidade e seus habitantes poderá trazer benefícios nas áreas de transporte, planejamento urbano, saúde pública, segurança pública e comércio (PAN et al., 2013).

Embora as ações de sensoriamento e mineração dos dados seja de imprescindível importância, algo deve ser feito para transformar esses dados em informações. Nesse sentido, uma abordagem interessante para a resolução desta questão consiste no conceito de *crowdsourcing* (SCHUURMAN et al., 2012).

Howe (2006) caracteriza *crowdsourcing* como o fenômeno no qual várias pessoas comuns dedicam seu tempo livre na busca de soluções para um problema de interesse coletivo. Isto posto, a massificação dos chamados *smartphones* e o advento de redes móveis

e sem fio tais como o 3G e o WiFi e recentemente, o 4G, permitem que pessoas fiquem conectadas a maior parte do tempo, em qualquer lugar. Dentre os inúmeros benefícios que os *smartphones* fornecem, a possibilidade de troca de informações entre usuários é uma delas, bem como a aplicação do conceito de *crowdsourcing*.

Considera-se o seguinte cenário: uma pessoa que acabou de chegar em um ponto de ônibus, deseja saber onde o mesmo se encontra. Questões como "será que o ônibus está chegando?", "será que vai demorar?" ou "será que o ônibus estragou?" são bastante comuns. Mas como descobrir suas respostas? Infelizmente, não é possível respondê-las apenas com a tabela de horários de chegada dos ônibus, disponível nos terminais e no *site* da URBS (Companhia de Urbanização e Saneamento de Curitiba).

Nesse sentido, é proposto nesse documento uma solução possível para esse quesito. A ideia básica é a construção de um aplicativo que alie conceitos, tanto de *Smart Cities* quanto de *crowdsourcing*, no qual informações em tempo real sobre o transporte coletivo, especificamente no que compete o estado atual de uma linha de ônibus, sejam disponibilizadas aos usuários.

Alguns aplicativos para dispositivos móveis foram desenvolvidos com o objetivo de prover, ao usuário, informações referentes ao transporte público. Um dos aplicativos mais recentes é o chamado "Busão Curitibano", lançado em 2013[1]. Os desenvolvedores – e também os usuários do aplicativo – encontraram algumas resistências por parte da URBS, pois o aplicativo usava a base de dados do transporte coletivo. Devido aos inúmeros acessos, acabava derrubando o servidor da URBS e tornando o serviço de acompanhamento de ônibus indisponível.

Outro aplicativo semelhante foi desenvolvido por Sujatha et al. (2014). Os autores propuseram um sistema chamado MBTS (*Mobile Bus Tracking System*) que ajuda qualquer pessoa a obter informações de um determinado ônibus sem fazer ligações ou perturbar passageiros com mensagens SMS (*Short Message Service*). Um banco de dados é utilizado para armazenar informações sobre os ônibus (por exemplo, as coordenadas do veículo), as quais estarão disponíveis para acesso através de um aplicativo Android.

## 1.1 JUSTIFICATIVA

A utilização de uma arquitetura centralizada, estilo cliente–servidor, é um ponto comum nos dois trabalhos relacionados. Em ambos, nota-se que a arquitetura encontra-se

[1]Para mais informações sobre o aplicativo, consultar notícia da Gazeta do Povo, disponível em: http://www.gazetadopovo.com.br/vidaecidadania/conteudo.phtml?id=1377575

em uma relação n:1, ou seja, vários clientes para um único servidor. Alguns dos problemas que podem surgir nesse tipo de arquitetura é que o servidor pode ficar sobrecarregado e eventualmente ficar indisponível devido ao grande número de usuários utilizando o serviço, como foi o caso do "Busão Curitibano". Além disso, nesse caso, o servidor utilizado não é de propriedade dos desenvolvedores, mas sim da URBS. Caso a companhia deseje interromper o serviço de fornecimento de dados, o "Busão Curitibano" perde sua utilidade. Isso não constitui um problema para o MBTS, pois o mesmo concentra as informações em um servidor próprio, mantido pelos próprios desenvolvedores.

Sendo assim, a principal motivação para o desenvolvimento deste projeto é implementar um aplicativo de acompanhamento de ônibus para dispositivos móveis – *smartphones* – que utilize uma rede descentralizada. Nela, os próprios usuários são os nós, que compartilharão informações entre si. Para isso, utilizar como referência o aplicativo móvel "Waze"[2], que implementa uma espécie de comunidade para mapeamento do trânsito em tempo real e usufruir do GPS dos *smartphones* para fornecer aos usuários a posição dos veículos.

## 1.2 OBJETIVO GERAL

Desenvolver um aplicativo móvel que execute sobre o sistema operacional Android, aliando conceitos de *smart cities* e *crowdsourcing*, no qual informações em tempo real sobre o transporte coletivo sejam disponibilizadas aos usuários.

## 1.3 OBJETIVOS ESPECÍFICOS

- Buscar métodos para estabelecimento de comunicação entre dispositivos Android e de que forma implementá-los;
- Possibilitar usuários do aplicativo móvel a colaborarem entre si com informações sobre o transporte coletivo;
- Permitir que o aplicativo móvel funcione mesmo na ausência de conexão com a Internet;
- Implementar uma arquitetura de rede descentralizada, a fim de evitar problemas como sobrecarga de servidor.

[2]Uma análise do aplicativo pode ser encontrada em http://www.techtudo.com.br/tudo-sobre/waze.html

## 1.4 ESTRUTURA E ORGANIZAÇÃO

O presente documento encontra-se organizado da seguinte forma. O capítulo 1 apresenta a introdução, com a formulação do problema, justificativa do projeto, objetivo geral e específicos do mesmo. O capítulo 2 apresenta o levantamento bibliográfico, importante para fundamentar o projeto e fornecer um embasamento. A metodologia, que descreve as etapas de desenvolvimento pelas quais o projeto percorrerá, é apresentada no capítulo 3. Tópicos pertinentes ao desenvolvimento do aplicativo móvel, tais como APIs utilizadas e protocolo de comunicação, são apresentados no capítulo 4. No capítulo 5, discutem-se os testes realizados sobre o aplicativo desenvolvido e o capítulo 6 apresenta os resultados desses testes. Conclusões pertinentes ao desenvolvimento do projeto, tais como dificuldades encontradas e trabalhos futuros, são apresentados no capítulo 7. Por fim, o projeto de *software* encontra-se no Apêndice, com requisitos funcionais e não funcionais, diagramas de casos de uso, pontos de casos de uso, gerenciamento e análise de riscos.

# 2 LEVANTAMENTO BIBLIOGRÁFICO

Este capítulo apresenta uma breve introdução à *Smart Cities*, *Crowdsourcing*, contexto atual do desenvolvimento de programação voltada a aplicativos de transporte público e demais tópicos relativos ao desenvolvimento do trabalho.

## 2.1 *SMART CITIES*

O conceito de Cidades Inteligentes e o respectivo aumento no seu uso pode ser reconhecido como o resultado do alto crescimento das tecnologias digitais na intenção de garantir um futuro sustentável. Embora o conceito seja empregado normalmente com o objetivo de elaborar estratégias que objetivem melhorar a qualidade de vida dos cidadãos, criando um futuro sustentável, o conceito por si só continua sendo usado em diferentes contextos e assim permanece um tanto ambíguo. Dessa forma faz-se necessário estabelecer um critério muito importante que o diferencia das cidades-conceito, o aspecto colaborativo entre os diversos interessados, mais comumente os cidadãos (SCHUURMAN et al., 2012).

Um outro enfoque, mais empresarial, apresentado por Walravens (2012), menciona que o conceito também pode ser empregado para caracterizar um grupo de organizações que intentam por revolucionar algum aspecto em uma região, entre os quais estão parques empresariais, o nível de escolaridade de uma população, o uso de tecnologias em contextos urbanos, o aumento da eficácia de governos e especialmente aquelas com foco em ICT. Walravens argumenta também que, de maneira geral, o uso de serviços de telefonia móvel adquirem uma importância inevitável, já que é um sub aspecto de ICT e dessa forma grande importância na construção de uma Cidade Inteligente (WALRAVENS, 2012).

Em vista disso, sistemas operacionais desenvolvidos para *smartphones* inspiram desenvolvedores a conceber aplicativos e serviços que aperfeiçoam a vida nas cidades de diversas formas diferentes, como por exemplo, facilitando acesso à informação sobre o transporte público. Além disto, à medida que os *smartphones* se tornam mais acessíveis

e populares, cria-se o desejo de que se tornem eminentes ferramentas na busca de uma cidade mais inteligente (WALRAVENS, 2012).

## 2.2 *CROWDSOURCING*

Outro aspecto importante a ser mencionado é a caracterização de *crowdsourcing*, no que tange inteligência coletiva. À grosso modo, entende-se inteligência coletiva como quando indivíduos de um determinado grupo, que possui interesse por determinado assunto, combinam seus conhecimentos a fim de encontrar a solução para um determinado problema (SCHUURMAN et al., 2012). Através da interação social, o conhecimento individual é compartilhado, corrigido, processado, enriquecido e avaliado. Normalmente, os resultados colhidos são melhores que os obtidos por um único indivíduo. Talvez a aplicação mais difundida desse conceito seja a Wikipedia (SCHUURMAN et al., 2012).

Por conseguinte, a ponte que se estabelece entre *crowdsourcing* e *smart cities* acompanha o advento dos *smartphones*. Para Kanhere (2011), as melhorias no poder de processamento, sensoriamento e capacidades de armazenamento permite que telefones celulares sejam comparados a dispositivos de computação. Esse fenômeno abre margem ao surgimento de um novo paradigma, denominado pela literatura de sensoriamento participativo, cuja ideia principal gira em torno de capacitar um cidadão comum a coletar e compartilhar dados de sensoriamento do ambiente em que está inserido, a partir de seu telefone celular (KANHERE, 2011).

Ainda segundo Kanhere, sensoriamento participativo possui quatro vantagens principais sobre redes de sensores tradicionais (já que esta necessita de uma gama considerável de dispositivos sem fio, principalmente em áreas urbanizadas), sendo elas (KANHERE, 2011): custo de implementação baixo, já que usa telefones celulares e Wi-Fi; uma ampla mobilidade e cobertura proporcionada pelas operadoras de telefonia; economias de escala proporcionadas pelo uso de celulares; a facilidade assegurada pelas lojas de aplicativos e também as inúmeras formas de desenvolvimento de software disponíveis para sistemas operacionais de dispositivos móveis.

## 2.3 *PEER-TO-PEER*

A arquitetura *Peer-to-peer* (P2P) é capaz de fornecer recursos de rede com sobreposição e auto-organização distribuída, com o intuito de prover uma distribuição eficiente dos dados. O mecanismo básico de funcionamento consiste em pares, nos quais uma enti-

dade presta e, ao mesmo tempo, consome os recursos oferecidos por outras entidades que compõe o sistema. Outras características que estes sistemas apresentam, normalmente, são auto-organização, tolerância a falhas e escalabilidade (CACCIAPUOTI; CALEFFI; PAURA, 2011).

## 2.4 WI-FI DIRECT

O padrão IEEE 802.11, tornou-se uma das formas mais comuns para acessar a Internet. Porém mais de uma década depois de sua concepção inicial, a tecnologia precisa continuar sua evolução e abranger um conjunto maior de casos de uso (CAMPS-MUR; GARCIA-SAAVEDRA; SERRANO, 2013).

De acordo com Camps-Mur, Garcia-Saavedra e Serrano (2013), um caminho promissor nesse sentido é o investimento em tecnologias que não requeiram a presença de um ponto de acesso (do inglês AP, *Access Point*) ou seja, o alvo principal é conectividade de dispositivo-a-dispositivo. Com efeito, esta é a finalidade da recente tecnologia Wi-Fi Direct. Dada a ampla base de dispositivos com capacidades Wi-Fi e o fato de que o Wi-Fi Direct pode ser implementado totalmente em *software*, a comunicação dispositivo-a-dispositivo torna-se particularmente interessante (CAMPS-MUR; GARCIA-SAAVEDRA; SERRANO, 2013).

Em linhas gerais, o Wi-Fi Direct difere-se do tradicional modo *ad hoc* para estabelecer conexão entre dispositivos. Por ter sido construído sobre o IEEE 802.11 em modo infraestrutura, o Wi-Fi Direct herda mecanismos de QoS (*Quality of Service*), economia de energia e de segurança. No Wi-Fi Direct, os dispositivos negociam quem fará o papel de AP (CAMPS-MUR; GARCIA-SAAVEDRA; SERRANO, 2013).

Em uma rede Wi-Fi típica, os clientes descobrem e se associam com as WLANs (*Wireless Local Area Networks*), que são criadas e anunciadas pelos APs. Desta maneira, cada um dos dispositivos da rede assume um papel, quer como um AP ou como um cliente, com funções específicas. No Wi-Fi Direct esses papéis são dinâmicos e, portanto, a implementação do mesmo deve ser de tal maneira que um dispositivo seja capaz de assumir tanto o papel de um cliente quanto o de um AP (CAMPS-MUR; GARCIA-SAAVEDRA; SERRANO, 2013).

Dispositivos Wi-Fi Direct, formalmente conhecidos como dispositivos P2P, comunicam-se através da criação de grupos P2P. O dispositivo que realizar o papel de AP no grupo P2P, é referido como o P2P GO (*Peer-to-Peer Group Owner*) e, de forma análoga,

dispositivos na qualidade de clientes são conhecidos como clientes P2P. Uma vez que esses papéis não são estáticos, quando dois dispositivos P2P se descobrem, eles negociam seus papéis (cliente e *group owner*) para estabelecer um grupo P2P. Uma vez estabelecido o grupo P2P, outros clientes P2P podem se juntar ao grupo como em uma rede Wi-Fi tradicional (CAMPS-MUR; GARCIA-SAAVEDRA; SERRANO, 2013).

A Figura 1 ilustra um caso de uso da tecnologia Wi-Fi Direct, no qual um *smartphone* compartilha sua conexão 3G com dois *notebooks*. Nesse grupo, o *smartphone* atua como P2P GO ao passo que os dois *notebooks* atuam como P2P clientes. Para expandir a rede, um dos *notebooks* estabelece um segundo grupo P2P com uma impressora. Para este segundo grupo, o *notebook* atua como o P2P GO (CAMPS-MUR; GARCIA-SAAVEDRA; SERRANO, 2013).

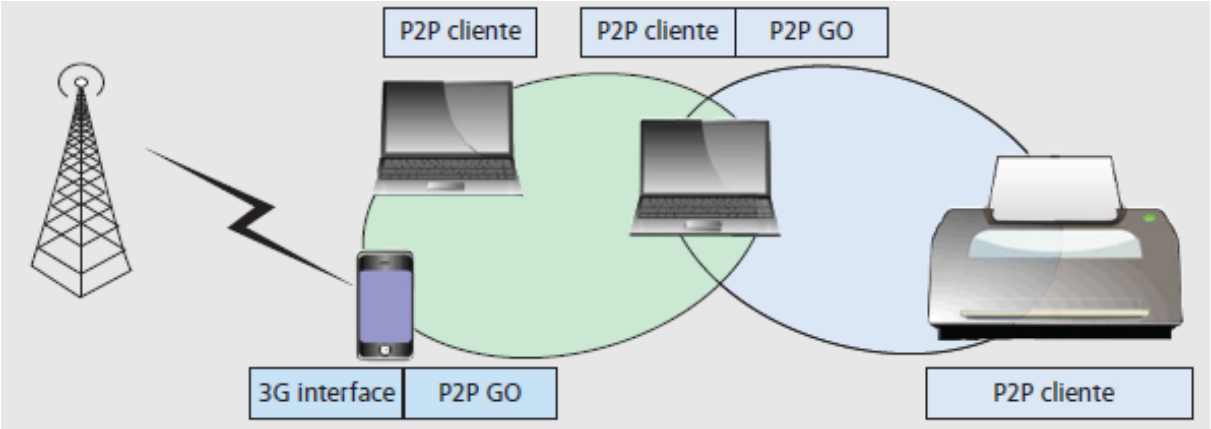



**Figura 1:** Casos de uso do Wi-Fi Direct.
Fonte: Adaptado de Camps-Mur, Garcia-Saavedra e Serrano (2013).

Como em um AP Wi-Fi tradicional, o P2P GO anuncia-se através de *beacons*[1]. O P2P GO também deve implementar um servidor DHCP (*Dynamic Host Configuration Protocol*) para fornecer aos clientes P2P endereços IP. O Wi-Fi Direct não permite transferir a função de P2P GO dentro de um Grupo P2P. Desta forma, se o P2P GO deixa o grupo P2P em seguida o grupo é desfeito, e deve ser restabelecido (CAMPS-MUR; GARCIA-SAAVEDRA; SERRANO, 2013).

Segundo Conti et al. (2013), a formação de grupos P2P no Wi-Fi Direct envolve dois passos: descoberta de dispositivos (*Device Discovery*) e formação de grupo (*Group Formation*). O primeiro passo é composto por duas fases: varredura e descoberta.

[1]De acordo com o padrão IEEE 802.11, cada ponto de acesso compatível envia periodicamente quadros de gerenciamento chamados *beacon frames*, ou quadros de sinalização. O objetivo dos quadros de sinalização é anunciar a presença de um AP em uma área, as suas capacidades, e algumas informações sobre configurações e segurança para os dispositivos situados como clientes (Cisco Systems Inc., 2015).

Durante a fase de varredura cada dispositivo colhe informações sobre os dispositivos ao redor, escaneando em todos os canais sem fio compatíveis (especificados pelo padrão IEEE 802.11) (CONTI et al., 2013).

A fase de varredura tem uma duração pré-definida e é seguida pela fase de descoberta, em que os dispositivos P2P alternam entre estados de pesquisa e escuta nos canais sociais (ou seja, 1, 6, e 11 na banda de 2,4 GHz) em períodos de tempo aleatórios (CONTI et al., 2013). Nesta fase, um dispositivo no estado de busca envia "sondas" de requisição, a fim de descobrir outros dispositivos que estejam no estado de escuta no mesmo canal. Quando um dispositivo recebe uma "sonda" de requisição em um canal específico, responde com uma "sonda" de resposta, no mesmo canal, incluindo sua identificação e a que grupo pertence (se o dispositivo for um *group owner*) (CONTI et al., 2013).

O segundo passo estabelece os grupos P2P. Nele, o *group owner* é definido e os dispositivos estabelecem conexão. O Wi-Fi Direct define três procedimentos diferentes para a formação de grupos P2P (CONTI et al., 2013):

1. Padrão (*standard*): quando a negociação de *group owner* é dada através de *handshake* triplo[2].

2. Autônoma (*autonomous*): quando um dispositivo elege-se *group owner* e anuncia sua presença aos outros dispositivos através de mensagens *beacons*;

3. Persistente (*persistent*): quando os dispositivos envolvidos já participaram do mesmo grupo e são capazes de reconstruí-lo a partir de suas informações.

Uma vez que o *group owner* é definido, o Wi-Fi Direct implementa um procedimento de autenticação com base em WPS (*WiFi Protected Setup*), a fim de estabelecer uma conexão *wireless* segura. Segue-se então a fase de atribuição de endereços IP (através de um servidor DHCP executando no *group owner*) (CONTI et al., 2013).

[2]Handshake triplo é um processo de estabelecimento de conexão no qual três segmentos de dados são enviados entre dois *hosts*. O exemplo mais comum desse tipo de procedimento é o estabelecimento de conexão entre um cliente e um servidor TCP. A grosso modo, o cliente envia um segmento TCP com o bit SYN ajustado para 1 e um número de sequência aleatório; o servidor responde com outro segmento TCP chamado de SYNACK com o bit SYN com valor 1, o número de sequência do cliente + 1 e seu próprio número de sequência escolhido aleatóriamente também; por fim, o cliente responde novamente com um terceiro segmento, ajustando o valor de SYN para 0 e numero de sequência do servidor + 1 (KUROSE; ROSS, 2010).

## 2.5 REDES *AD HOC*

De acordo com Kurose e Ross (2010), uma estação base (tipicamente representada por pontos de acesso em uma LAN (*Local Area Network*), ou ainda, torres celulares) é uma parte fundamental da infraestrutura de rede sem fio. Uma estação base é responsável pelo envio e recebimento de dados (por exemplo, pacotes) de e para um hospedeiro sem fio que está associado com ela, a qual será responsável pela coordenação da transmissão de vários hospedeiros sem fio com os quais está associada (KUROSE; ROSS, 2010).

Quando hospedeiros estão associados a uma estação-base, em geral se diz que estão operando em modo infraestrutura, já que todos os serviços tradicionais de rede (por exemplo, atribuição de endereçamento e roteamento) são fornecidos pela rede com a qual estão conectados por meio da estação-base. Em redes *ad hoc*, hospedeiros sem fio não dispõem de nenhuma infraestrutura dese tipo com a qual se conectar. Na ausência de tal infraestrutura, os próprios hospedeiros devem prover serviços como roteamento, atribuição de endereço, tradução de endereços semelhantes ao DNS (*Domain Name System*) e outros (KUROSE; ROSS, 2010). Mais formalmente, Perkins (2008) define uma rede *ad hoc* como um envolvimento cooperativo de uma coleção de um nós móveis sem a necessidade de intervenção de qualquer ponto de acesso centralizado ou infraestrutura existente (PERKINS, 2008).

Complementando a ideia inicial apresentada por Kurose, Tanenbaum (2002) argumenta que a principal diferença entre as redes *ad hoc* e as redes fisicamente conectadas é que os roteadores podem ir e vir ou aparecer em novos lugares de um momento para o outro. Com uma rede fisicamente conectada, se um roteador tiver um caminho válido para algum destino, esse caminho continuará a ser válido indefinidamente (desde que não ocorra alguma falha em algum lugar do sistema). No caso de uma rede *ad hoc*, a topologia pode se alterar o tempo todo, e assim o interesse e até mesmo a validade dos caminhos podem se alterar de modo espontâneo, sem qualquer aviso. É desnecessário dizer que essas circunstâncias tornam o roteamento em redes *ad hoc* bem diferente do roteamento nas redes equivalentes fixas (TANENBAUM, 2002).

### 2.5.1 *WIRELESS HOSTED NETWORK* E *INTERNET CONNECTION SHARING*

Em que pese o uso de redes *ad hoc* no escopo do presente projeto, o sistema operacional Android não é capaz de fazer uso nativo de redes *ad hoc* em seu formato padrão (apenas em versões customizadas, nas quais é necessário realizar modificações nos

arquivos do *kernel* do Android) (Thinktube Inc., 2015a).

No entanto, a não compatibilidade de redes *ad hoc* por parte do Android pode ser substituída por uma alternativa simples, chamada "rede hospedada sem fio" (WHN, *Wireless Hosted Network*). Uma WHN, aliada ao recurso de compartilhamento de conexão com a Internet (ICS, *Internet Connection Sharing*), permite que um dispositivo se torne um AP "virtual" e compartilhe sua conexão com a Internet, caso a mesma esteja disponível. Os recursos WHN e ICS foram desenvolvidos pela Microsoft e estão disponíveis para serem configurados nas versões 7, 8 e 8.1 do sistema operacional Microsoft Windows (Microsoft, 2015). Qualquer dispositivo com uma interface *wireless* é capaz de se conectar à uma rede hospedada sem fio.

Uma WHN efetua duas funções principais: a virtualização de um adaptador *wireless* físico em mais de um adaptador *wireless* virtual, por vezes referido como *Virtual Wi-Fi*; um ponto de acesso *wireless* baseado em software por vezes referido como um *Soft-AP* que usa um adaptador virtual sem fio designado (Microsoft, 2015).

O ICS é fornecido através do *SharedAccess Service*. À grosso modo, o *SharedAccess* permite o compartilhamento de rede através de um computador onde o acesso à rede compartilhada não necessariamente fornece acesso à Internet. O WHN está intimamente ligado ao ICS para permitir tanto a rede sem fio de Área Pessoal (PAN - *Personal Area Network*) quanto os cenários de partilha de Internet (Microsoft, 2015).

Por fim, o recurso de rede hospedada sem fio foi utilizado para concepção de um modo alternativo de conexão por parte do aplicativo móvel, pois uma WHN é reconhecida pelo Android como uma interface de rede válida, ao contrário do observado com redes *ad hoc* convencionais.

## 2.6 ESTADO DA ARTE

Conforme relacionado no Capítulo 1, Sujatha et al. (2014) propuseram um sistema chamado MBTS para que qualquer pessoa obtenha informações de um determinado ônibus através de um aplicativo móvel.

Três são os atores do sistema: os passageiros, os chamados "coordenadores" do ônibus e as pessoas que estão aguardando em um ponto. Todos os atores necessitam de um *smartphone* com sistema operacional Android conectado à *Internet*. Continuamente, os passageiros e os "coordenadores" alimentam um banco de dados com informações sobre o ônibus. Essas informações podem ser as coordenadas do ônibus, obtidas através do GPS

integrado ao *smartphone*, ou até mesmo a "situação" do ônibus – se ele está lotado ou vazio, se houve algum acidente, se o ônibus está atrasado, entre outros. Essas informações ficam armazenadas em um banco de dados para serem acessadas, posteriormente, por alguém que esteja utilizando o aplicativo, principalmente usuários que estão aguardando um ônibus em algum ponto.

A arquitetura do MBTS pode ser observada na Figura 2. Basicamente, consiste de duas aplicações, uma Web e outra Android. A primeira permite o registro de usuários e ônibus e a segunda é utilizada para rastreio. A aplicação Web, desenvolvida em JSP (*JavaServer Pages*), é voltada para o administrador do sistema e também funciona como um *"middleware"* para que a aplicação Android se conecte ao banco de dados e armazene informações (SUJATHA et al., 2014).

Uma outra pesquisa neste área de acompanhamento de ônibus, envolve a utilização de RF (Radio frequência). Foi desenvolvida por Paradells et al. (2014).

Os autores propõe a utilização de uma rede de dados composta por transmissores e receptores RF. Os transmissores situam-se nos ônibus, os quais transmitem informações para os receptores, que situam-se nos pontos de ônibus que, basicamente, são os nós da rede. Assim que um ônibus se aproxima de um ponto de ônibus - nó - inicia a transmissão de informações relevantes, captadas por sensores atrelados ao veículo. Essas informações podem ser usadas/acessadas por um usuário que está aguardando no referido ponto de ônibus.

Os autores utilizaram receptores RF que são ativados a partir de uma distância de 30 m, ou seja, o veículo deverá estar a 30 m (ou menos) de um receptor para que a transmissão de dados ocorra. Para os autores, os testes foram satisfatórios com veículos desempenhando velocidades de até 40 km/h.

Porém, a 30 m de distância o veículo já não está tão longe do ponto de ônibus, e pode ser visualizado por um usuário sem o auxílio de um sistema específico para tal. Logo, essa distância de 30 m pode constituir um fator limitante ao projeto.

Alves, Martinez e Viegas (2012) apresentaram um sistema – chamado de *Trip-planner* – para planejar rotas em tempo real, para pessoas que utilizam transporte público em Lisboa. O sistema consegue informar quais são as melhores rotas e o tempo estimado de viagem para um determinado destino, baseados na estimativa de quantos veículos estão trafegando e quais são suas velocidades.

Informações de tempo-real são coletadas através do GPS equipado nos ônibus.

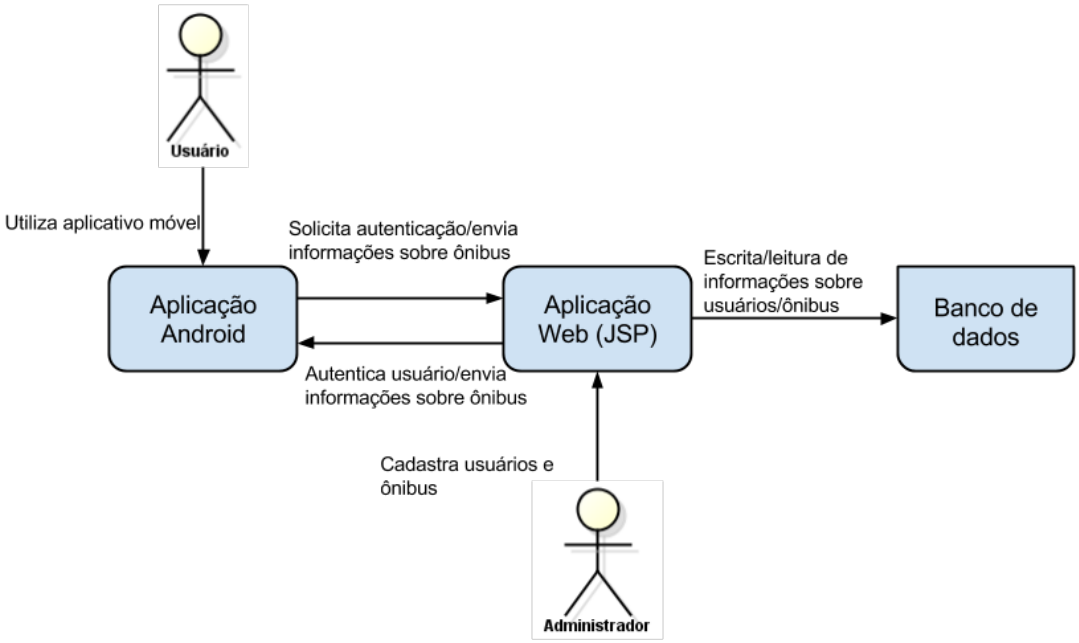



**Figura 2:** Arquitetura do MBTS.
Fonte: Adaptado de Sujatha et al. (2014).

Essas informações são utilizadas em um servidor (o que os autores chamam de *Data Center*) para atualizar históricos e melhorar as estimativas, uma vez que é utilizado um algoritmo para predição de tempos de viagem. Esses históricos se referem à relatórios de quatro meses, com informações sobre tempos de viagem e velocidades dos veículos. Essas informações são analisadas e passam por um classificador, e uma vez processadas, são repassadas para qualquer dispositivo móvel conectados em uma rede sem fio, através de *broadcast* (ALVES; MARTINEZ; VIEGAS, 2012).

A Figura 3 descreve de forma simplificada o sistema proposto por Alves, Martinez e Viegas (2012). Nota-se claramente que dados históricos e de tempo-real são coletados com o auxílio do GPS instalado no ônibus. Esses dados passam pelo *Data Center* e são utilizados como entrada para um algoritmo de predição. Uma vez processados, distribuem-se os dados para dispositivos móveis através de *broadcast.*

## 2.7 DISCUSSÕES

Conforme discutido no Capítulo 1, a principal motivação para o desenvolvimento deste projeto é implementar um aplicativo de acompanhamento de ônibus para dispositivos móveis - *smartphones* - que utilize uma rede descentralizada.

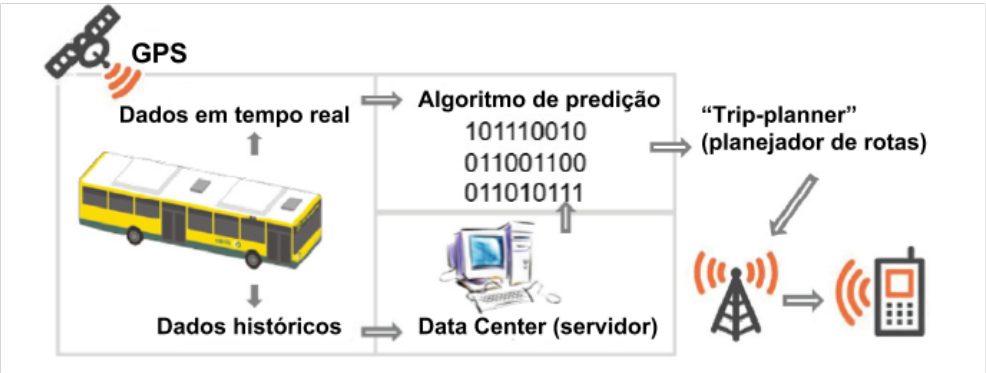



**Figura 3:** Arquitetura do *Trip-planner*.
Fonte: Adaptado de Alves, Martinez e Viegas (2012).

Espera-se que o aplicativo a ser desenvolvido seja de grande ajuda ao dia a dia dos passageiros do transporte coletivo e que, de alguma, forma traga informações de qualidade aos usuários, bem como lhes ofereça uma maior economia de tempo e torne o processo de decisão, sob qual ônibus tomar ou qual rota seguir, descomplicado.

Em termos de arquiteturas de rede, decidiu-se por utilizar neste projeto a arquitetura P2P, devido ao seu caráter "descentralizado". Além disso o P2P, aliado ao padrão Wi-Fi Direct, a ser discutido nos próximos capítulos, dispensa a necessidade de infra-estrutura de rede, constituindo um modelo muito interessante para ser utilizada por dispositivos que suportam redes móveis e sem fio, como *smartphones* e *tablets*.

# 3 METODOLOGIA

Neste capítulo apresenta-se a metodologia a ser empregada a fim de alcançar os desígnios do projeto. Para tanto, apresenta-se de que forma o desenvolvimento será abordado pela equipe e, na sequência, as tecnologias a serem utilizadas, tais como o ambiente de programação, a linguagem de programação, mecanismos para controle de versão e, por fim, a plataforma de *software*.

## 3.1 ABORDAGEM DA EQUIPE

Partindo de ideias de aplicativos já existentes para acompanhamento de tráfego (Waze) e acompanhamento de ônibus na cidade de Curitiba (Busão Curitibano), a equipe decidiu reuni-las e aperfeiçoá-las em um aplicativo único. O aperfeiçoamento será resultado do uso de Wi-Fi P2P (especificamente o Wi-Fi Direct) que dispensa centralização, se contrapondo à arquitetura cliente-servidor, na qual operam *softwares* (aplicativos) móveis existentes já citados nos Capítulos 1 e 2.

A arquitetura P2P permite que um usuário (que pode ser representado aqui por um nó da rede) adquira um papel simultâneo de "cliente" e "servidor". Em outras palavras, um usuário receberá/enviará informações de/para outros usuários, partindo do princípio que todos estejam utilizando o mesmo aplicativo móvel, a ser desenvolvido neste projeto.

## 3.2 TECNOLOGIAS

A presente seção refere-se às principais tecnologias a serem empregadas no desenvolvimento do projeto, tais como ambientes de desenvolvimento, linguagens de programação, plataforma de *software* e plataforma de *hardware*.

### 3.2.1 AMBIENTE DE DESENVOLVIMENTO

Pelo fato do projeto envolver um aplicativo para o sistema operacional Android, é necessário utilizar um ambiente de desenvolvimento específico para essa plataforma.

O *site* oficial para desenvolvimento de aplicativos Android, Android Developers[1], disponibiliza algumas ferramentas para a concepção e teste de aplicativos para a plataforma. Atualmente, a IDE oficial para desenvolvimento de aplicativos Android chama-se Android Studio, que pode ser baixada no site Android Developers. Os seguintes componentes são instalados juntamente com o Android Studio:

- Ambiente de desenvolvimento (IDE);
- Android SDK (*Software Development Kit*) correspondente à versão mais recente do sistema operacional Android;
- Ferramentas adicionais para a plataforma Android;
- Uma versão da imagem do sistema Android para o emulador.

O Android Studio exige o JDK (*Java Development Kit*), que pode ser baixado gratuitamente a partir do site oficial da Oracle: <http://oracle.com>. O JDK é necessário para a compilação de aplicativos na linguagem Java e por consequência, aplicativos Android.

Para testes de software será utilizada a emulação do dispositivo Android provida pela IDE, até onde for possível. É dito isso pois o projeto envolve o uso de GPS, Google Maps e compartilhamento de informações entre usuários (*crowdsourcing*), o que não é possível de ser feito apenas com a utilização de um emulador. Dessa forma, será necessário testar a aplicação em um ou mais dispositivos reais (*smartphone* ou *tablet*) que estejam executando o sistema operacional Android.

### 3.2.2 LINGUAGEM DE PROGRAMAÇÃO

Segundo Sims (2014), a linguagem de programação "oficial" para o desenvolvimento de aplicações Android é Java. Grande parte das aplicações são escritas em Java e suas API (*Application Programming Interface*) são projetadas para serem chamadas primeiramente por Java. É possível desenvolver utilizando linguagens como C e C++,

[1] O site pode ser acessado em http://developer.android.com/

mas isso é algo que a própria Google, desenvolvedora do Android, não incentiva (SIMS, 2014).

Por essas razões, o aplicativo será desenvolvido utilizando a linguagem de programação Java juntamente com as APIs (que contém bibliotecas e ferramentas) incluídas no Android SDK.

### 3.2.3 CONTROLE DE VERSÃO

Será utilizado um sistema de controle de versão (CVS, *Control Version System*) para o versionamento do código-fonte do aplicativo Android. Vários são os sistemas de versionamento de arquivos, dentre os quais os mais comuns são o SVN e Git. Para este último, existem repositórios de código-fonte gratuitos, como o GitHub (<https://github.com>) e o BitBucket (<https://bitbucket.org>).

O ideal seria manter um servidor próprio com o SVN ou o Git instalados. No entanto, isso poderia envolver custos adicionais, pois seria necessário deixar uma máquina dedicada para o controle de versão e acessível para todos os integrantes do projeto, além de realizar *backups* periódicos.

O GitHub permite que qualquer pessoa crie uma quantidade ilimitada de repositórios gratuitamente e compartilhe-o com quantos colaboradores for necessário. No entanto, o código-fonte fica disponível publicamente, para qualquer um acessar. Para projetos acadêmicos (como TCCs, teses de mestrado e doutorado), isso pode não ser indicado. O GitHub também oferece repositórios privados, com acesso restrito, mas apenas a partir da assinatura de um plano.

Em contrapartida, o BitBucket oferece uma quantidade ilimitada de repositórios privados gratuitamente, permitindo até cinco colaboradores em um projeto. Como o projeto é constituído de dois integrantes e levantou-se a necessidade de versionar o código-fonte sem disponibilizá-lo publicamente, optou-se por utilizar o BitBucket como repositório de código-fonte e o Git como como sistema de controle de versão.

### 3.2.4 PLATAFORMA DE *SOFTWARE*

Em relação ao sistema operacional Android, levanta-se outra questão. O Android, como sistema operacional, possui diversas versões desde o seu lançamento. Cada uma dessas versões acaba por incluir novas funcionalidades não apenas para o usuário final, mas também, para os desenvolvedores. É certo que existe a chamada retrocompatibilidade,

que permite que a grande maioria das aplicações desenvolvidas para uma versão mais recente do S.O. (Sistema Operacional), execute sem maiores problemas em uma versão mais antiga. No entanto, para que isso ocorra, muitas vezes a aplicação fica limitada, pois o desenvolvedor é forçado à desabilitar certos recursos que não são compatíveis, de forma alguma, com versões mais antigas da plataforma.

No momento da escrita deste documento, a versão mais recente do Android é a versão 5.1 (Lollipop). A maioria dos dispositivos disponíveis no mercado – *smartphones* e *tablets* – possuem uma versão do Android 4.0+ instalada.

A Tabela 1 fornece informações referentes ao *market share* das versões do Android, desde a 2.2 (Froyo) até a 4.4 (KitKat). Os dados foram coletados através da aplicação Google Play Store em um período de sete dias, finalizando no dia 03 de novembro de 2014. É possível verificar o percentual de quantos dispositivos executam uma determinada versão do sistema operacional. Essas informações são úteis para que o desenvolvedor determine qual versão da plataforma terá como "alvo".

**Tabela 1:** *Market Share* das versões do Android. Fonte: Android Developers (2014).

| **Versão** | **Codinome** | **API** | **Distribuição** |
|---|---|---|---|
| 2.2 | Froyo | 8 | 0,6% |
| 2.3.3 - 2.3.7 | Gingerbread | 10 | 9,7% |
| 4.0.3 - 4.0.4 | Ice Cream Sandwich | 15 | 8,5% |
| 4.1.x | Jelly Bean | 16 | 22,8% |
| 4.2.x | Jelly Bean | 17 | 20,8% |
| 4.3 | Jelly Bean | 18 | 7,3% |
| 4.4 | KitKat | 19 | 30,2% |

Observa-se na tabela que cerca de 10,3% usuários usa um dispositivo com uma versão antiga da plataforma, de codinome Froyo (2.2) ou Gingerbread (2.3.3 - 2.3.7). Uma parcela ainda menor utiliza as versões Ice Cream Sandwich (4.0.3 - 4.0.4), 8,5%. Grande parte dos usuários – cerca de 50% – utiliza o Android Jelly Bean (4.1.x - 4.3) e 30,2% utiliza a última versão lançada, KitKat (4.4).

Portanto, devido à popularidade da versão Jelly Bean (4.1.x - 4.3) do sistema operacional Android, o desenvolvimento do projeto terá como foco dispositivos que executam, no mínimo, esta versão da plataforma.

### 3.2.5 PLATAFORMA DE *HARDWARE*

Para o projeto em questão não será necessária a utilização de uma plataforma de *hardware*, pois a aplicação executará diretamente sobre um dispositivo móvel, tal como um *smartphone* ou um *tablet*.

## 3.3 ETAPAS DE DESENVOLVIMENTO

Inicialmente será feito um estudo das APIs disponíveis no Android SDK e, em seguida, a projeto de software com diagramas UML.

Com os diagramas UML já projetados, será preparado o ambiente de desenvolvimento, realizando-se o *download* e instalação do ambiente de desenvolvimento Android Studio, instalação de *drivers* para depuração em um dispositivo móvel real via cabo USB e configurações necessárias. Na sequência, dar-se-á início à implementação da aplicação.

Inicialmente, os testes serão realizados no emulador Android, disponibilizado pelo *bundle*, conforme citado na seção 3.2.1. Testes mais complexos serão feitos em um dispositivo real (*smartphone* e/ou *tablet*) que execute o sistema operacional Android com versão 4.1.x, pelo menos.

# 4 DESENVOLVIMENTO DO APLICATIVO MÓVEL

Este capítulo apresenta uma explicação sobre as tecnologias aplicadas no desenvolvimento do aplicativo móvel. Na primeira seção, discorre-se sobre as APIs do Android SDK, com ênfase no modo Wi-Fi *Peer-to-Peer*, Google Maps Android e Google Play Services Location. Na sequencia, analisa-se as evoluções do processo de comunicação entre os dispositivos. Por fim, na última seção, mostra-se o protocolo de comunicação da aplicação utilizado na troca de mensagens entre dispositivos.

## 4.1 APIS DO ANDROID SDK

Esta seção apresenta as APIs específicas que foram utilizadas para o projeto, inclusas no Android SDK. As APIs referentes à interface gráfica não serão discutidas aqui, por serem triviais. Desse modo, serão apresentadas as APIs relativas ao *framework* do Wi-Fi Direct, ao Google Maps (utilizado para mostrar a posição do ônibus, dadas as suas coordenadas) e ao serviço de localização da Google Play (utilizado para detectar a posição do dispositivo).

Antes de apresentar as APIs utilizadas, é importante esclarecer alguns termos específicos ao Android SDK, tais como:

- *Activity*: classe utilizada para desenvolver uma UI (*User Interface*) para que o usuário possa interagir com o aplicativo. Toda *Activity* tem um ciclo de vida definido, que envolve desde sua criação até sua destruição e perpassa por estados como *Running* (executando), *Paused* (pausada) e *Stopped* (parada) (Android Developers, 2015a);
- *Fragment*: classe que representa uma seção modular de uma *Activity*, e que recebe seus próprios eventos (de entrada, por exemplo). Um *Fragment* tem seu ciclo de vida próprio e pode ser adicionado ou removido de uma *Activity*, mesmo que ela esteja executando (Android Developers, 2015e);
- *Service*: classe utilizada para executar operações em segundo-plano (*background*)

e que não provê uma UI. Um *Service*, ao ser iniciado por um aplicativo, continua executando mesmo que o usuário abra outro aplicativo. Exemplos de aplicação de *Service* incluem transações de rede, *player* de música e operações de entrada/saída (Android Developers, 2015i);

- *AsyncTask*: permite o uso da *thread* de interface gráfica (UI *thread*) de forma apropriada. Uma classe que herda a classe *AsyncTask* pode executar operações em segundo-plano e atualizar a UI conforme a necessidade, ou seja, *AsyncTask* é útil nos casos em que a UI precisa ser atualizada enquanto uma operação (geralmente demorada) está executando (Android Developers, 2015c). Ressalta-se que a classe *Thread* (do *Java Development Kit*) não pode fazer alterações diretamente na UI *thread* do Android;

- *Intent*: uma descrição abstrata de uma operação que deve ser executada. Pode ser utilizada para iniciar uma *Activity* ou um *Service*, ou enviada à um *BroadcastReceiver* registrado no aplicativo (Android Developers, 2015g);

- *Broadcast Receiver*: classe que receberá Intents e tomará ações a partir dos *Intents* recebidos (Android Developers, 2015d).

### 4.1.1 WI-FI PEER-TO-PEER

O modo Wi-Fi *Peer-to-Peer* (Wi-Fi P2P) permite que dispositivos com Android 4.0 ou mais recente se conectem a outros dispositivos diretamente, utilizando apenas o Wi-Fi e sem a necessidade de um AP. Esse modo segue o padrão Wi-Fi Direct, estabelecido pela W-Fi Alliance. Utilizando o *framework* Wi-Fi P2P, é possível procurar e se conectar à outros dispositivos através de uma conexão mais rápida (10 Mbps) e em um alcance de comunicação muito maior (100 m), se comparado à tecnologia *Bluetooth* (2 Mbps e 10 m) (Android Developers, 2015j).

A API é constituída por três partes principais (Android Developers, 2015j):

- A classe *WifiP2pManager* que fornece métodos para encontrar outros dispositivos, requisitar conexões entre dois dispositivos e conectá-los;

- *Listeners* para notificar, de forma assíncrona, quando um método da classe *WifiP2pManager* obteve êxito ou falhou;

- *Intents* para notificações acerca de eventos específicos. Por exemplo, quando um novo *peer* é descoberto ou quando alguém se desconecta.

Ao se utilizar o *framework* Wi-Fi P2P do Android, os seguintes métodos estão disponíveis ao desenvolvedor (Android Developers, 2015j):

**Tabela 2:** Métodos do *framework* Wi-Fi P2P. Fonte: Android Developers (2015j).

| Método | Descrição |
|---|---|
| initialize() | Registra o aplicativo com o *framework* Wi-Fi P2P. Esse método tem que ser chamado antes de chamar qualquer outro método relativo ao *framework*. |
| connect() | Inicializa uma conexão *peer-to-peer* com um dispositivo. |
| cancelConnect() | Cancela qualquer negociação de um grupo *peer-to-peer* que esteja em andamento. |
| requestConnectInfo() | Requisita informações sobre conexão relativas à um dispositivo. |
| createGroup() | Cria um grupo *peer-to-peer*. O dispositivo no qual o método é chamado será o *"group owner"*. |
| removeGroup() | Remove o grupo atual. |
| requestGroupInfo() | Requisita informações sobre um grupo *peer-to-peer*. |
| discoverPeers() | Inicializa a procura por *peers*. |
| requestPeers() | Requisita a lista atual de *peers* encontrados. |

Conforme citado, cada um dos métodos da classe *WifiP2pManager* permite que seja passado um *Listener* para que o *framework* possa notificar, de forma assíncrona, sobre o resultado de uma operação (se obteve êxito ou falhou). As interfaces *listeners* disponíveis e seu correspondente método da classe *WifiP2pManager* são mostradas na Tabela 3 (Android Developers, 2015j):

**Tabela 3:** *Listeners* para o *framework* Wi-Fi P2P. Fonte: Android Developers (2015j).

| Interface | Métodos associados |
|---|---|
| WifiP2pManager.ActionListener | connect(), cancelConnect(), createGroup(), removeGroup() e discoverPeers() |
| WifiP2pManager.ChannelListener | initialize() |
| WifiP2pManager.ConnectionInfoListener | requestConnectInfo() |
| WifiP2pManager.GroupInfoListener | requestGroupInfo() |
| WifiP2pManager.PeerListListener | requestPeers() |

O *framework* Wi-Fi P2P permite quatro *Intents* para notificar o aplicativo acerca de eventos específicos. Os *Intents* disponíveis podem ser observados na Tabela 4:

Para utilizar a API do Wi-Fi P2P é necessário, também, alterar o arquivo AndroidManifest no diretório raiz do projeto. O AndroidManifest é um arquivo no formato

**Tabela 4:** *Intents* para o *framework* Wi-Fi P2P. Fonte: Android Developers (2015j).

| Intent | Evento |
|---|---|
| WIFI_P2P_CONNECTION_CHANGED_ACTION | Ocorre quando o estado da conexão W-Fi do dispositivo se altera. |
| WIFI_P2P_PEERS_CHANGED_ACTION | Ocorre quando há uma chamada ao método discoverPeers(). Assim que esse evento dispara, é possível chamar o método requestPeers() para obter uma lista atualizada de *peers* descobertos. |
| WIFI_P2P_STATE_CHANGED_ACTION | Ocorre quando o Wi-Fi P2P (Wi-Fi Direct) é habilitado ou desabilitado no dispositivo. |
| WIFI_P2P_THIS_DEVICE_CHANGED_ACTION | Ocorre quando informações sobre o dispositivo (por exemplo, seu nome) são alteradas. |

XML (*Extended Markup Language*) com algumas diretrizes, e que todas as aplicações Android devem ter. O *manifest* informa ao sistema operacional quais são os requisitos necessários para executar o aplicativo corretamente. Nele, por exemplo, é indicado o nome do pacote Java, são descritas todas as *Activities*, *Services* e *Broadcast Receivers* do aplicativo e, principalmente, as permissões que o aplicativo precisa obter para acessar componentes protegidos do sistema operacional (Android Developers, 2015b).

O *framework* Wi-Fi P2P exige que as seguintes permissões de aplicativo sejam incluídas no arquivo *manifest* do projeto (Android Developers, 2015h):

- ACCESS_NETWORK_STATE: permite que o aplicativo acesse informações sobre redes;
- ACCESS_WIFI_STATE: permite que o aplicativo acesse informações sobre Wi-Fi;
- CHANGE_NETWORK_STATE: permite que o aplicativo altere o estado de conectividade da rede;
- CHANGE_WIFI_STATE: permite que o aplicativo altere o estado de conectividade Wi-Fi;

- INTERNET: permite que o aplicativo abra *sockets*.

### 4.1.2 GOOGLE MAPS ANDROID API

Para utilizar o Google Maps Android API é necessário, além do Android SDK, uma chave Google Maps Android API v2, obtida gratuitamente. Ela efetuará a validação nos servidores da Google e, sem ela, não é possível carregar mapas nos aplicativos desenvolvidos (Google Developers, 2015a).

A obtenção dessa chave envolve os seguintes passos, que serão descritos resumidamente em seguida:

1. Obtenção do SHA-1 *fingerprint* do certificado de *Debug* (ou *Release*) do Android SDK;
2. Criação de um projeto "API" no Google Developers Console (<https://console.developers.google.com/>);
3. Obtenção da chave Android API;
4. Inserção da chave Android API no arquivo *manifest* do aplicativo desenvolvido.

O SHA-1 pode ser obtido tanto de um certificado de *Debug* quanto de um certificado de *Release*. O certificado de *Debug* é gerado automaticamente no momento em que o aplicativo que está sendo desenvolvido e depurado. O certificado de *Debug* só pode ser utilizada para testes, ou seja, não é possível publicar aplicativos utilizando o SHA-1 de *Debug*. No caso em que o aplicativo seja publicado, o SHA-1 de *Release* se faz necessário (Google Developers, 2015a).

O certificado de *Debug* encontra-se no diretório raiz da instalação do Android SDK e seu SHA-1 *fingerprint* pode ser acessado através do executável *keytool*, instalado automaticamente durante a instalação do JDK (Google Developers, 2015a).

Com o SHA-1 em mãos, é necessário acessar o site Google Developers Console, criar um projeto e nele habilitar o Google Maps Android API. Em seguida, criar uma chave Android API, especificando o SHA-1 gerado e o nome do pacote do aplicativo em desenvolvimento. Uma vez que isso é feito, o Google Developers Console retorna uma chave de quarenta dígitos, no seguinte formato: AIzaSyBdVl-cTICSwYKrZ95SuvNw7dbMuDt1KG0 (Google Developers, 2015a). Essa chave deve ser incluída no atributo *"value"* do elemento *"meta-data"* do arquivo AndroidManifest.xml, já mencionado na seção 4.1.1:

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="API_KEY"/>
```

Além disso, é necessário especificar no *manifest* a versão do Google Play Services SDK:

```
<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
```

Além das permissões ACCESS_NETWORK_STATE e INTERNET, explicadas na seção 4.1.1, a API do Google Maps Android requer também (Android Developers, 2015h):

- ACCESS_COARSE_LOCATION: permite que o aplicativo acesse a localização aproximada do dispositivo, utilizando fontes como torres de telefonia e Wi-Fi;
- ACCESS_FINE_LOCATION: permite que o aplicativo acesse a localização precisa do dispositivo, utilizando fontes como GPS, torres de telefonia e Wi-Fi;
- WRITE_EXTERNAL_STORAGE: permite que o aplicativo escreva no dispositivo de armazenamento externo.

A API ainda utiliza o OpenGL ES 2 para renderizar o mapa, e isso também deve ser especificado no *manifest*:

```
<uses-feature
        android:glEsVersion="0x00020000"
        android:required="true"/>
```

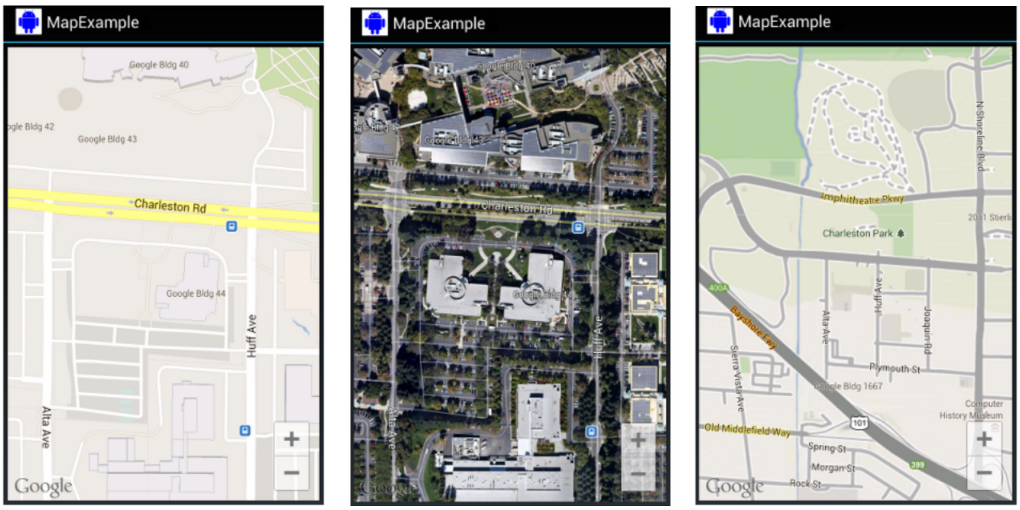
Após isso, a API pode ser utilizada no aplicativo. O mapa é exibido em um *MapFragment*, e uma classe que herda a classe *FragmentActivity* deve ser criada para que o desenvolvedor possa manipular o mapa. A classe *MapFragment* nada mais é do que um contêiner para o mapa e que provê acesso à classe *GoogleMap* (Google Developers, 2015c).

A classe *GoogleMap* modela o objeto de mapa no aplicativo, que será representado por um *MapFragment*. Além disso, a classe *GoogleMap* executa, de forma transparente ao desenvolvedor, as seguintes operações relacionadas à mapas (Google Developers, 2015c):

- Conexão ao serviço do Google Maps;

- *Download* do mapa;
- Exibição do mapa na tela do dispositivo;
- Exibição de controles interativos, como *zoom* e *scroll*.

A Google Maps Android API fornece diferentes modos de mapa, que distinguem em como o mesmo será exibido ao usuário final. O modo normal (Figura 4(a)), exibe ruas, construções e recursos naturais (como rios e vegetação); o modo *hybrid* (Figura 4(b)) exibe ruas com algumas imagens de satélite; o modo *satellite* exibe apenas imagens de satélite (ruas não são visíveis) e, por fim, o modo *terrain* (Figura 4(c)) exibe a topografia de uma região (Google Developers, 2015c). O modo do mapa pode ser alterado chamando-se o método setMapType() da classe *GoogleMap*, ou alterando o XML que descreve o *Fragment* do mapa (Google Developers, 2015c).




**(a)** Modo Normal **(b)** Modo *Hybrid* **(c)** Modo *Terrain*

**Figura 4:** Diferenças entre os modos de mapa.
Fonte: Google Developers (2015c).

É possível deixar o mapa com uma configuração inicial, de acordo com as necessidades do desenvolvedor. Posicionar a centroide em uma região específica (cidade de Curitiba, por exemplo), ao invés de mostrar o mapa com todos os continentes. Essas configurações contribuem para a boa usabilidade do aplicativo e podem ser realizadas no arquivo XML que descreve o *Fragment* do mapa. As seguintes configurações estão disponíveis ao desenvolvedor (Google Developers, 2015c):

- *mapType*: permite especificar o tipo de mapa a ser utilizado. Valores permitidos: "normal", *"hybrid"*, *"satellite"*, *"terrain"* e *"none"*.

- *cameraTargetLat*, *cameraTargetLng*, *cameraZoom*, *cameraBearing*, *cameraTilt*: permitem especificar a posição inicial da câmera.
- *uiZoomControls*, *uiCompass*: permitem especificar se os controles de *zoom* e bússola serão mostrados no mapa.
- *uiZoomGestures*, *uiScrollGestures*, *uiRotateGestures*, *uiTiltGestures*: permitem especificar quais gestos estão habilitados ou desabilitados para interação no mapa.
- *zOrderOnTop*: permite especificar se a superfície do mapa será mostrada à frente de outros controles.
- *useViewLifecycle*: permite especificar se o ciclo de vida do mapa será associado ao *Fragment*.
- *liteMode*: permite especificar se o *"lite mode"* será utilizado. O *"lite mode"* consiste em apenas uma imagem *bitmap* do mapa que suporta apenas parte das funcionalidades providas pela API. Por exemplo, no *"lite mode"*, não é possível interagir com o mapa (através de gestos, *zoom* e *scroll*).

### 4.1.3 GOOGLE PLAY SERVICES LOCATION API

A localização de um dispositivo Android pode ser obtida por meio da API Google Play Services Location. Ela foi utilizada no projeto para obter as coordenadas (latitude e longitude) do dispositivo no qual uma informação sobre uma linha de ônibus está sendo registrada. Caso o usuário que está registrando a informação esteja dentro do ônibus, pode-se dizer que as coordenadas do dispositivo deste usuário correspondem, aproximadamente, às coordenadas do veículo.

Um dos recursos fornecidos pela API é obter a "última posição conhecida", que geralmente equivale à posição atual do dispositivo (Android Developers, 2015f). A API permite, também, que a posição do dispositivo seja coletada periodicamente.

A API requer que o Google Play Services seja incluído no projeto. Essa configuração varia com a IDE que está sendo utilizada para desenvolver o aplicativo.

Assim como as demais APIs apresentadas, o *manifest* do projeto tem que ser alterado para permitir que o aplicativo acesse a localização do dispositivo no qual está executando. A Google Play Services Location API fornece dois tipos diferentes de permissões de localização: ACCESS_COARSE_LOCATION e ACCESS_FINE_LOCATION.

Essas permissões são requeridas também pela API do Google Maps Android e foram explicadas na seção 4.1.2.

Para utilizar a API, é necessário criar uma instância da classe *GoogleApiClient.Builder*, que representa o cliente *Google Play Services*. Uma vez instanciado, chama-se o método addApi(), passando o atributo estático "API" da classe *LocationServices* (*LocationServics.API*) por parâmetro. A classe *Builder* permite que métodos de *callback* sejam chamados de forma assíncrona quando algum evento ocorre. Por exemplo, uma vez que a conexão com o serviço é estabelecida, o *callback* onConnected() é disparado (Android Developers, 2015f).

Quando isso ocorre, é possível requisitar ao *Google Play Services* a "última posição conhecida", equivalente à posição atual do dispositivo. Para tal, chama-se o método getLastLocation(), passando por parâmetro a instância da classe *GoogleApiClient*. Esse método retorna um objeto da classe *Location*, que contém as coordenadas (latitude e longitude) da posição obtida (Android Developers, 2015f).

## 4.2 COMUNICAÇÃO ENTRE OS DISPOSITIVOS

Durante a implementação do aplicativo, foram testadas diversas formas de estabelecer a comunicação entre os dispositivos. Primeiramente foram utilizados *sockets* TCP (*Transmission Control Protocol*) e, em seguida, utilizou-se *broadcast* UDP (*User Datagram Protocol*) para transmissão de mensagens. Mas a versão final do aplicativo utiliza *multicast* (UDP) para transmitir mensagens entre dispositivos, por razões explicadas nas próximas seções.

### 4.2.1 COMUNICAÇÃO COM *SOCKETS* TCP

No início da implementação do aplicativo, decidiu-se por usar *sockets* TCP para a troca de mensagens entre os dispositivos, utilizando um modelo "cliente-servidor". Basicamente, o dispositivo que possuísse informações faria o papel de "servidor", enquanto o dispositivo que estivesse requisitando informações faria o papel de "cliente".

Para a utilização deste método seria necessária a implementação de um servidor *Multi-thread*, dado o grande número de usuários que estariam requisitando informações em um determinado momento. Além disso, uma vez que o protocolo TCP é confiável, o uso da rede ficaria maior, devido aos controles estabelecidos pelo próprio protocolo.

Após reuniões com os orientadores de projeto, constatou-se que a arquitetura

"cliente-servidor" estabelecida por este método não era adequada ao mesmo. Até porque um dos objetivos deste trabalho é, justamente, utilizar uma arquitetura descentralizada, que desvincule um dispositivo de outro. Além disso, os dispositivos teriam que ficar trocando de papéis dependendo da situação: ora seriam clientes, ora seriam servidores. Isso aumentaria muito a complexidade da implementação, desnecessariamente. Por essas razões, este método foi descartado.

### 4.2.2 COMUNICAÇÃO POR *BROADCAST* UDP

A utilização de *broadcast* para transmissão das mensagens dispensaria a necessidade de um dispositivo "servidor" atendendo à vários "clientes". A mensagem simplesmente seria enviada para o endereço de *broadcast* da rede, em uma determinada porta, e, todos os dispositivos que estivessem escutando essa porta na rede iriam receber as mensagens. O endereço de *broadcast* seria obtido dinamicamente em tempo de execução, através de uma análise do IP atribuído ao dispositivo, quando o mesmo estivesse conectado à uma rede.

O fato de utilizar UDP ao invés de TCP não traria problemas ao projeto porque supõe-se que um grande número de dispositivos estaria escutando na porta atrelada ao endereço de *broadcast*. Se um pacote destinado à um dispositivo fosse perdido, outro provavelmente o receberia. Uma requisição poderia ser retransmitida também, caso nenhuma resposta fosse recebida após um determinado tempo.

O problema de se utilizar a comunicação por *broadcast* é que o endereço IP (*Internet Protocol*) poderia mudar assim que o dispositivo entrasse em outra rede. Isso faria com que dispositivos parassem de receber mensagens de *broadcast*, por estarem simplesmente escutando a porta de um endereço *broadcast* diferente. A solução seria utilizar um IP fixo (*hard-coded*), conhecido por todos os dispositivos que estivessem utilizando o aplicativo móvel. Por essa razão, a comunicação por *broadcast* foi substituída pela comunicação por *multicast*.

### 4.2.3 COMUNICAÇÃO POR *MULTICAST* UDP

A comunicação por *multicast* funciona de modo muito semelhante à comunicação por *broadcast*. A diferença, conforme já mencionado na seção anterior, é que o endereço IP *multicast* não seria obtido dinamicamente como era feito no modelo *broadcast*; seria definido no código-fonte do aplicativo, "fixo", e dessa forma conhecido por todos os dispositivos que estivessem executando o aplicativo móvel.

Os endereços *multicast* encontram-se no intervalo 224.0.0.0 a 239.255.255.255, com alguns endereços reservados neste intervalo. Optou-se por utilizar, no aplicativo, o endereço não-reservado 225.4.5.6. Assim, todos os dispositivos que estivessem em uma mesma rede e executando o aplicativo móvel, "entrariam" no grupo representado pelo endereço IP 225.4.5.6 e enviariam e receberiam as mensagens através desse endereço, independente de qual fosse o endereço de *broadcast* da rede.

Pelas razões apresentadas nesta seção e nas seções anteriores, constatou-se que a comunicação por *multicast* seria ideal para o projeto. Ela foi aplicada tanto para a comunicação utilizando Wi-Fi Direct (que nada mais é que uma rede virtual) quanto para a comunicação utilizando um AP.

## 4.3 PROTOCOLO DE COMUNICAÇÃO

O protocolo de comunicação desenvolvido para o aplicativo consiste em uma mensagem de requisição e uma mensagem de informação. A primeira contém a requisição de informações de uma determinada linha de ônibus e a segunda contém informações sobre uma linha solicitada via mensagem de requisição.

A mensagem de requisição é composta apenas pelo nome de uma linha de ônibus (por exemplo, "ABRANCHES"). Já a mensagem de informação é composta por dados sobre uma linha que foi requisitada como, por exemplo, sua situação (lotada, atrasada), posição do veículo da respectiva linha (latitude e longitude) e horário em que foram registrados esses dados. A mensagem de informação sempre inicia com um *token* (REQINFO), para diferenciá-la da mensagem de requisição. As informações contidas na mensagem de informação foram registradas previamente por um usuário no aplicativo móvel.

Quando mais de um usuário possui informações sobre uma determinada linha e envia-as para a rede, o aplicativo móvel considerará apenas a mensagem mais recente (selecionada pelo horário de registro de informação) e descartará as demais.

Nos casos em que nenhum usuário possui informações sobre uma linha requisitada, simplesmente não será enviada a mensagem de informação.

Mais detalhes sobre cada uma das mensagens do protocolo são apresentados a seguir:

1. Mensagem de requisição:

   Formato: "<LINHA DE ÔNIBUS>"

Aplicação: Quando um usuário está procurando informações sobre uma linha de ônibus, esta mensagem é enviada em *multicast* para todos os usuários do aplicativo que estejam na mesma rede.

Exemplos: “ABRANCHES”, “PINHEIRINHO”, “STA. CÂNDIDA / CAPÃO RASO”.

2. Mensagem de informação:

   Formato: “REQINFO; <LINHA DE ÔNIBUS>; <LAT>; <LONG>; <SITUAÇÂO>; <HORÁRIO>”

   Aplicação: Quando um usuário possui informações sobre uma linha de ônibus requisitada esta mensagem é enviada em *multicast* para todos os usuários do aplicativo que estejam na mesma rede.

   Exemplo: “REQINFO; ABRANCHES; -25.52345; -49.80134; LOTADO; 16:30:00 01/06/2015”.

# 5 TESTES

O presente capítulo discorre sobre os testes que foram realizados no aplicativo móvel desenvolvido, durante a implementação do mesmo.

## 5.1 INTERFACE GRÁFICA

Uma interface simples e *"clean"* foi pensada para o aplicativo móvel, de modo que ofereça todas as funcionalidades providas pelo mesmo em poucos toques de tela, para que o usuário final não tenha dificuldades em compartilhar ou procurar por informações sobre o transporte coletivo. A interface gráfica é composta por diversas partes (onde cada uma delas é descrita por uma classe *Activity* do Android), que serão apresentadas na sequência.

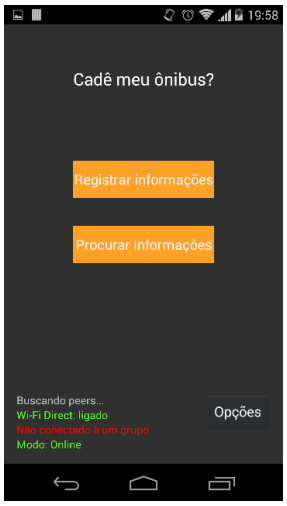
### 5.1.1 MENU: PRINCIPAL

O primeiro contato que o usuário tem com o aplicativo é através do Menu principal. A partir dele, o usuário pode registrar informações sobre uma determinada linha de ônibus, procurar informações sobre uma linha ou configurar opções do aplicativo. Além disso, o Menu principal apresenta algumas informações relevantes que variam de acordo com o modo de operação do aplicativo (Wi-Fi Direct ou AP).

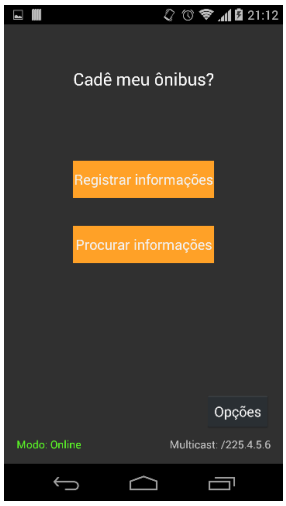
A escolha dos modos de operação é realizada automaticamente através de uma busca das redes Wi-Fi disponíveis. Se a rede Wi-Fi de SSID (*Service Set Identifier*) "Cade meu onibus?" está disponível para conexão no momento em que o aplicativo está sendo iniciado, o mesmo entrará no modo AP. Caso contrário, entrará no modo Wi-Fi Direct.

Assim que o modo de operação foi selecionado, o aplicativo verifica conexão com a *Internet*. Caso haja conexão, o aplicativo mostrará uma *label* "Modo: Online" na cor verde (Figuras 5(a) e 5(b)); caso contrário, a *label* será "Modo: Offline", na cor vermelha (Figuras 5(c) e 5(d)). Esse procedimento é realizado tanto no modo de operação Wi-Fi Direct (Figuras 5(a) e 5(c)) quanto no modo AP (Figuras 5(b) e 5(d)). No modo *online*, o Google Maps é utilizado para mostrar as informações ao usuário. Já no modo *offline*,

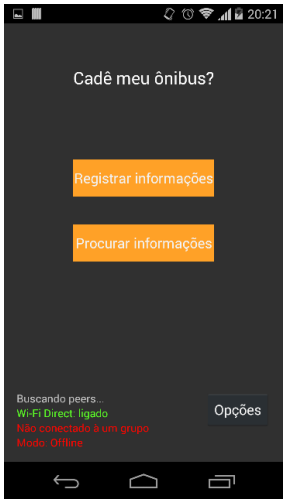
as mesmas são dispostas em modo texto.



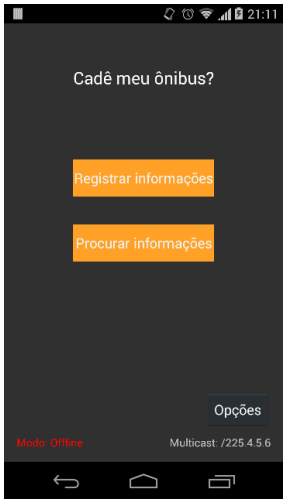

**(a)** Wi-Fi Direct no modo *online*




**(b)** AP no modo *online*




**(c)** Wi-Fi Direct no modo *offline*




**(d)** AP no modo *offline*

**Figura 5:** Menu principal indicando modo *online*/*offline* para os modos de operação Wi-Fi Direct e AP.

Caso o modo Wi-Fi Direct esteja em operação, informações adicionais como a quantidade de *peers* encontrados nas proximidades, se o usuário está conectado à um grupo Wi-Fi Direct no momento e o *status* do Wi-Fi Direct (ligado ou desligado) serão mostradas no Menu principal (Figura 6).

Outra informação disposta no Menu principal é o endereço IP *multicast*. No modo de operação Wi-Fi Direct, o endereço IP é mostrado tão logo o usuário se conecte à um grupo Wi-Fi Direct. Já no modo AP, o endereço IP é mostrado assim que o aplicativo se conecta à rede Wi-Fi "Cade meu onibus?".

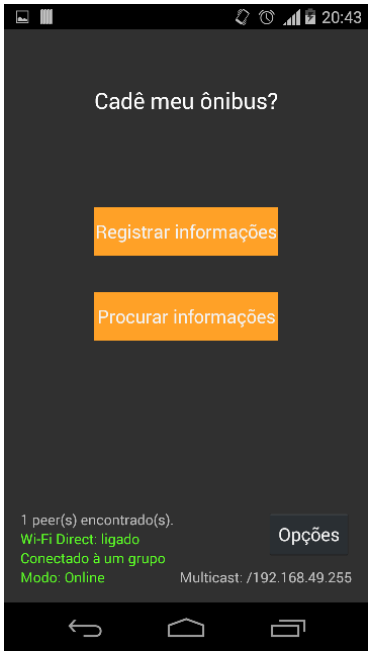


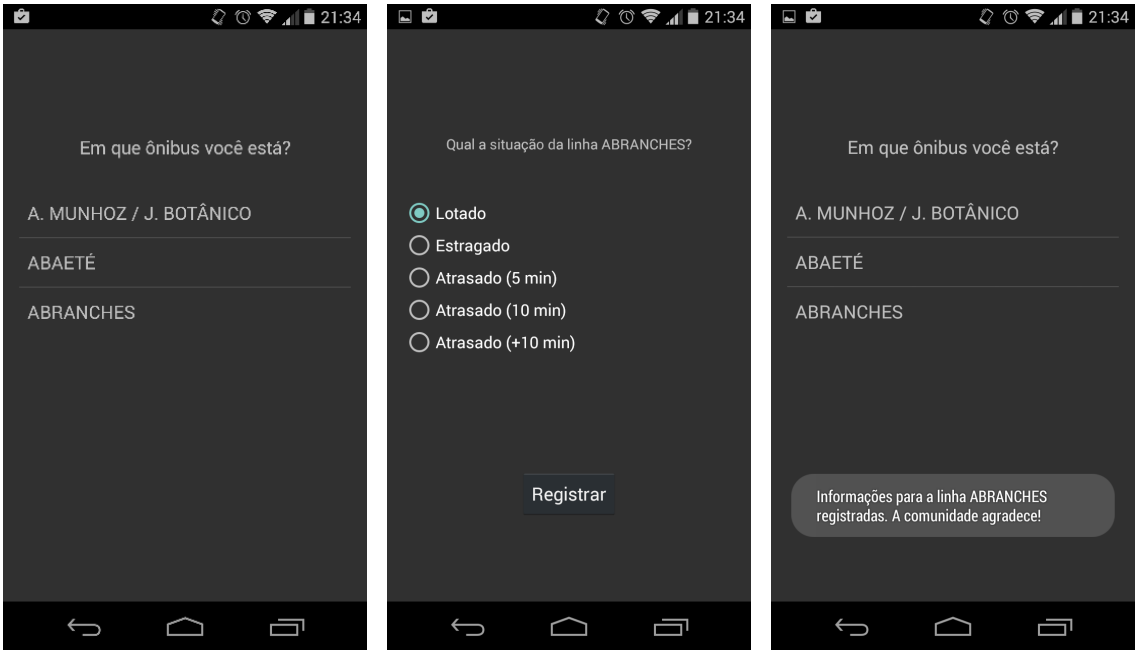

**Figura 6:** Aplicativo no modo Wi-Fi Direct indicando que um *peer* foi encontrado, que o Wi-Fi Direct está ligado e que o usuário está conectado à um grupo.

### 5.1.2 MENU: REGISTRAR INFORMAÇÕES

Ao tocar no botão "Registrar informações" presente no Menu principal, o usuário será levado para outra tela (Figura 7(a)). Esta tela conterá linhas de ônibus referentes ao transporte coletivo de Curitiba. No momento, apenas três linhas foram registradas no aplicativo.

Ao selecionar qualquer uma das linhas de ônibus, o usuário será levado para uma tela onde registrará o "estado" da linha selecionada (Figura 7(b)). As opções disponíveis são "Lotado", "Estragado", "Atrasado (5 min)", "Atrasado (10 min)" e "Atrasado (+10 min)". Apenas uma das opções pode ser selecionada.

Uma vez que o usuário seleciona o atual estado da linha de ônibus, ele pode tocar no botão "Registrar" para salvar a informação na memória do dispositivo. Uma mensagem será mostrada confirmando o registro da informação (Figura 7(c)). Uma vez registrada, ela estará disponível para outro usuário que esteja requisitando informações sobre essa linha. A informação da linha será apagada se o usuário fechar o aplicativo.




**(a)** Seleção de linha de ônibus **(b)** Opções disponíveis para uma linha de ônibus **(c)** Mensagem confirmando o registro de uma informação

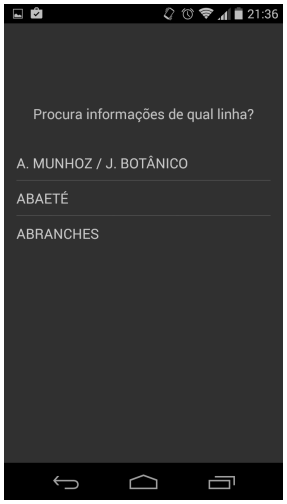
**Figura 7:** Menu: registrar informações.

### 5.1.3 MENU: PROCURAR INFORMAÇÕES

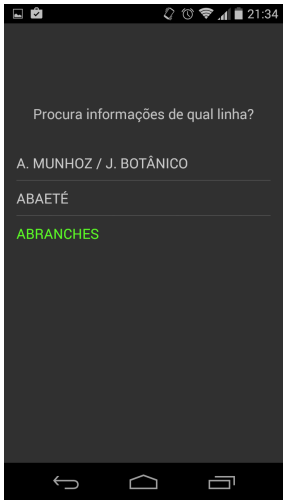
Assim que o usuário toca no botão "Procurar informações", disponível no Menu principal, uma tela semelhante à tela de registro de informações será mostrada, com as linhas de ônibus disponíveis no aplicativo. Se nenhuma informação estiver disponível para a linha, o nome dela será mostrado na cor cinza (Figura 8(a)). Ao selecionar uma dessas linhas com informações indisponíveis, uma mensagem de requisição será enviada imediatamente para o endereço de *multicast*.

No entanto, se o usuário já possuir informações sobre a linha, a mesma será mostrada na cor verde (Figura 8(b)). Nesse caso, as informações serão mostradas de acordo com o estado de conexão com a *Internet*. Se o aplicativo estiver conectado à *Internet*, um mapa contendo informações sobre a linha será mostrado (Figura 8(c)); a linha será indicada utilizando as coordenadas do dispositivo no qual as informações foram registradas.

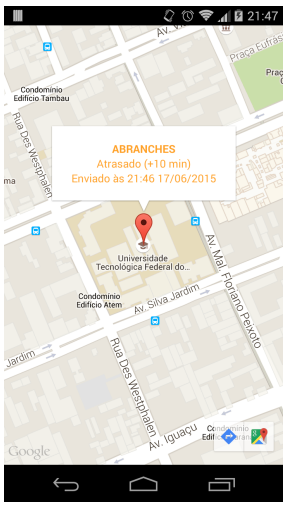
Caso não haja conexão com a *Internet*, informações sobre as linhas continuam sendo mostradas, mas em modo texto (Figura 8(d)).



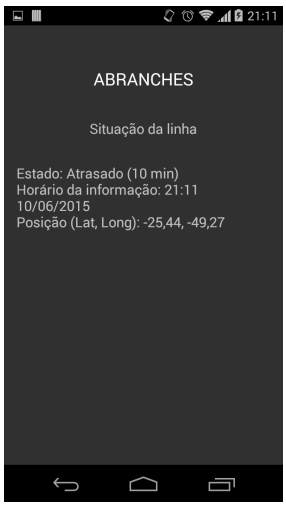

**(a)** Seleção de linha de ônibus, nenhuma informação disponível




**(b)** Seleção de linha de ônibus, informações disponíveis para a linha "Abranches"




**(c)** Informações mostradas em mapa




**(d)** Informações mostradas em texto

**Figura 8:** Menu: procurar informações e disposição das informações em mapa e texto.

### 5.1.4 MENU: OPÇÕES

Ao tocar no botão "Opções", também no Menu principal, o usuário pode alterar alguns parâmetros do aplicativo (Figura 9). Através da opção "Qtd. de retransmissões", é possível especificar quantas vezes uma mensagem de requisição (por exemplo, "Abranches") será enviada por *multicast*. Os valores permitidos estão no intervalo [4, 10], sendo 4 o valor *default*. Por exemplo, se o valor *default* está sendo utilizado, a mensagem de requisição será enviada para o endereço de *multicast* até 4 vezes (em intervalos de 1 segundo), caso o usuário não receba nenhuma resposta para a requisição enviada. Isso é útil nos casos em que está ocorrendo muita perda de pacotes, para que o usuário não tenha

que reenviar a solicitação manualmente.

No Menu opções o usuário pode, também, habilitar ou desabilitar as notificações do Android que aparecem quando informações sobre uma determinada linha estão disponíveis. As notificações são habilitadas por *default*, mas é possível desabilitá-las através do *switch* "Notificações".

Para salvar qualquer alteração feita na tela de opções, deve-se tocar no botão "Aplicar".

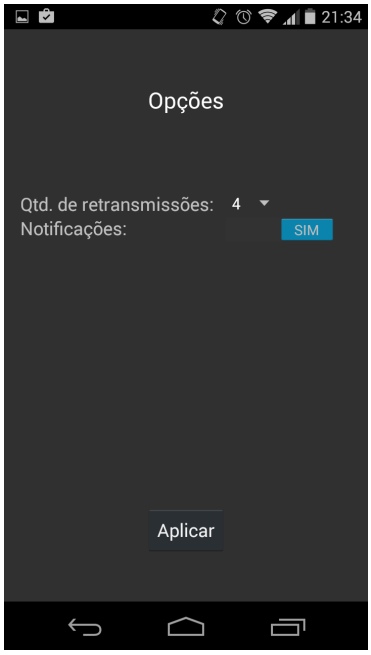



**Figura 9:** Menu opções.

### 5.1.5 NOTIFICAÇÃO

Um recurso interessante fornecido pelo Android SDK é a possibilidade de implementar notificação dos aplicativos. A notificação serve para que o usuário seja informado sobre determinados eventos que ocorrem no aplicativo, mesmo que ele esteja rodando em *background*.

Quando novas informações sobre uma determinada linha de ônibus são recebidas pelo aplicativo, uma notificação é mostrada (item intitulado "Cadê meu ônibus?", na Figura 10). Ao selecionar essa notificação, o usuário é levado diretamente à tela correspondente ao menu "Procurar informações" onde pode acessar, de imediato, as novas informações que chegaram recentemente.

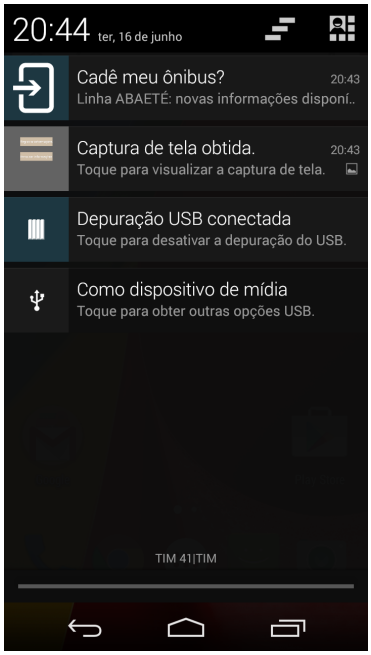



**Figura 10:** Notificação mostrada quando uma nova informação está disponível no aplicativo.

## 5.2 TESTES NO MODO WI-FI DIRECT

Conforme discutido no capítulo anterior, o *framework* Wi-Fi P2P foi utilizado para que dispositivos se conectem entre eles apenas utilizando Wi-Fi, sem a necessidade de um AP.

A utilização da API Wi-Fi P2P foi relativamente tranquila, uma vez que há muita documentação acerca das APIs do Android SDK. Além da documentação de classes e métodos (estilo javadoc) existem, também, tutoriais e exemplos de aplicativos que podem ser carregados e testados imediatamente em um dispositivo real.

O processo de conexão e transmissão de informações entre dois dispositivos utilizando Wi-Fi Direct envolve, basicamente, quatro passos:

1. Busca por dispositivos (*peers*) próximos. O alcance em que os outros dispositivos serão encontrados é determinado pelo alcance do próprio Wi-Fi (100 m) (Wi-Fi Alliance, 2015);

2. Solicitação de conexão: se um dispositivo A vai se conectar à um dispositivo B, B precisa aceitar o "convite" enviado por A. Processo semelhante pode ser observado quando o *Bluetooth* está sendo utilizado para pareamento entre dois dispositivos;

3. Aceitar/recusar solicitação: o dispositivo B pode tanto aceitar quanto recusar a so-

licitação enviada por A. No primeiro caso, a conexão será estabelecida, ao passo que no segundo caso, nenhuma conexão será estabelecida entre os dispositivos envolvidos;

4. Compartilhamento: uma vez que B aceita o convite de A, ambos podem compartilhar informações, ou até mesmo arquivos (assim como no *Bluetooth*), entre si.

No primeiro passo, os dispositivos nas proximidades que estão com Wi-Fi Direct habilitado, se encontrados, são colocados em uma lista de dispositivos. Essa lista é dinâmica pois, dispositivos podem ser colocados e retirados dessa lista de acordo com seu estado Wi-Fi Direct (habilitado/desabilitado) e sua proximidade (um dispositivo que sai do raio de comunicação de outro, provavelmente será retirado da lista).

Um dispositivo pode solicitar uma conexão (conforme indica o segundo passo) com qualquer um dos dispositivos que foram encontrados e colocados na lista. O dispositivo que foi solicitado precisa aceitar o "convite", que foi enviado por outro dispositivo, para que a conexão seja estabelecida (Figura 11). Uma vez que esse convite foi aceito, uma espécie de grupo é formado com um dos dispositivos sendo o *"group owner"* (proprietário do grupo). O *group owner* faz o papel de AP, e uma vez que o *group owner* é "escolhido" para um determinado grupo, não é possível mudar de *group owner*. Além disso, existe apenas um *group owner* por grupo formado. Outros dispositivos que queiram entrar em um grupo formado podem apenas realizar a solicitação ao *group owner*. Por essas razões, quando o grupo está formado, apenas o *group owner* fica visível à outros dispositivos que não estejam no grupo (dispositivos desconhecidos).

Um dos principais problemas observados ao se utilizar o Wi-Fi Direct é, justamente, o processo de aceitar o convite (a solicitação) enviado por dispositivos "desconhecidos" para que a conexão seja estabelecida e para que os mesmos participem do grupo. Supondo que o dispositivo A seja um *group owner* e existem cinco dispositivos desconhecidos que estão nas proximidades de A e solicitam conexão ao mesmo. O dispositivo A receberá cinco solicitações diferentes (uma para cada dispositivo desconhecido) e deverá aceitar cada uma delas.

Esse processo de aceitar (ou recusar) uma solicitação prejudica muito a usabilidade do aplicativo. É tolerável para poucas solicitações mas, no escopo do projeto, muitas solicitações seriam enviadas à um usuário que estivesse utilizando o aplicativo móvel, uma vez que informações sobre o transporte coletivo estariam sendo compartilhadas. Seria inviável que, em um determinado momento, um usuário (*group owner)* tivesse que aceitar 50 ou 100 solicitações de outros usuários (dispositivos).


**Figura 11:** Convite enviado por um dispositivo solicitando conexão via Wi-Fi Direct.

Acredita-se que esse processo manual de aceitar solicitações esteja muito relacionado com questões de segurança. Porque a solicitação está sendo enviada por um dispositivo desconhecido. Como determinar quais são as verdadeiras intenções do dispositivo solicitante? Alguém poderia utilizar o Wi-Fi Direct para acessar arquivos pessoais (como fotos, por exemplo) ou, quem sabe, utilizar um *sniffer* e capturar informações sigilosas (como senhas ou até mesmo conversas estabelecidas por meio de outros aplicativos).

Outro problema advindo da relação 1:n (um *group owner* para vários dispositivos) é que, se o *group owner* se desconectar, toda a rede formada pelo mesmo se desfaz. Ou seja, se o AP deixar de existir, todos os dispositivos que estavam conectados à ele se desconectam e o compartilhamento de informações cessa. É necessário refazer o grupo (com um novo *group owner*) de modo que os dispositivos possam se conectar novamente. Contudo, a recriação de grupo não é um processo automático ou transparente ao usuário. O *group owner* deste novo grupo deverá aceitar cada uma das solicitações enviadas à ele.

No entanto, uma vez formado o grupo, os dispositivos pertencentes à esse grupo podem solicitar conexão ao *group owner* sem que o mesmo receba os convites. Os grupos ficam registrados nos dispositivos que fazem parte do grupo (independente de serem *group owner* ou não) e a conexão passa a ficar mais transparente nesse caso. Supondo que A e B fazem parte de um grupo, sendo A o *group owner*, e este grupo está salvo em ambos os dispositivos. Se, mais tarde, B desejar se conectar ao dispositivo A, esse não receberá mais o convite e a conexão será estabelecida automaticamente. Esse comportamento é

comumente chamado de “grupos persistentes”.

Foram observados alguns problemas na utilização do recurso de grupos persistentes. Por algum motivo, a conexão só funcionava bem na primeira vez que os dispositivos se conectavam. Momentos mais tarde, quando uma tentativa de conexão era estabelecida entre os dois dispositivos, o aplicativo móvel se comportava de modo estranho. Por exemplo, por mais que os dispositivos estivessem conectados, não havia mais compartilhamento entre os mesmos (mensagens não eram mais enviadas ou recebidas). Era necessário entrar nas configurações de Wi-Fi Direct do sistema operacional, remover os grupos Wi-Fi Direct persistentes e recriar o grupo através do aplicativo móvel (enviando a solicitação de um dispositivo para outro). O compartilhamento de informações só voltava a funcionar após esse procedimento.

## 5.3 TESTES NO MODO AP

O modo AP “resolve” uma das principais limitações do modo Wi-Fi Direct: a necessidade de enviar ou aceitar um convite/solicitação. É necessário que o usuário se conecte apenas uma vez à uma determinada rede que poderá (ou não) estar protegida por senha. Todos os usuários conectados ao AP podem trocar mensagens sem a necessidade de ficar aceitando convites.

Foi criado um AP “virtual” através do recurso de *Hosted Networks* existente no sistema operacional Microsoft Windows 8/8.1. Para tal, o dispositivo no qual a *Hosted Network* está sendo criada tem que possuir uma placa de rede *wireless* (Wi-Fi).

Como o AP centraliza as conexões, se o mesmo cair, todos os usuários conectados ao mesmo se desconectam. Fato semelhante ocorre quando o *group owner* do Wi-Fi Direct cai, uma vez que ele faz o papel de AP. Numa aplicação real para o aplicativo, diversos APs deveriam ser instalados pela cidade de Curitiba, nos pontos e terminais (e também ônibus) com o mesmo SSID (“URBS” ou “Cade meu onibus?”, por exemplo) e a mesma senha de acesso (ou sem nenhuma senha). Assim, essa “troca” de AP seria transparente aos usuários. Por exemplo, um usuário que está num terminal A e conectado à rede deste terminal, se desconectará ao sair do alcance desta rede; no entanto, chegando em um ponto de ônibus (ou outro terminal) com um AP disponível, logo estará conectado de novo e voltará a trocar informações sobre o transporte coletivo.

## 5.4 TESTES GERAIS DO APLICATIVO MÓVEL

Foi possível realizar testes do aplicativo móvel em quatro dispositivos – dois *smartphones* e dois *tablets* – com diferentes versões de sistema operacional. Os modelos de cada um dos dispositivos, bem como sua versão do Android e a informação de que suportam Wi-Fi Direct, são listados na Tabela 5.

**Tabela 5:** Dispositivos utilizados nos testes do aplicativo móvel.

| **Modelo** | **Versão do S.O.** | **Suporta Wi-Fi Direct?** |
|---|---|---|
| Motorola Moto G (2ª Geração) | 4.4.4 (KitKat) | Sim |
| Motorola Moto E (2ª Geração) | 5.0.2 (Lollipop) | Sim |
| Genesis Tab GT-7305 | 4.2.1 (Jelly Bean) | Sim |
| Samsung Galaxy Tab 2 | 4.1.2 (Jelly Bean) | Sim |

Nenhum dos modelos testados ofereceu problemas relacionados à interface gráfica, que se ajustou adequadamente independentemente do tamanho da tela do dispositivo ou se o mesmo estava sendo utilizado vertical ou horizontalmente.

Em relação à comunicação e troca de mensagens, todos os modelos forneceram resultados satisfatórios tanto em modo Wi-Fi Direct quanto em modo AP, com exceção do *tablet* Samsung Galaxy Tab 2.

No modo Wi-Fi Direct, foi possível trocar informações com os dispositivos situados a uma distância de 40 m a 50 m uns dos outros. Foram utilizadas quatro retransmissões de requisições. Acima de 50 m, observou-se que muitas mensagens são perdidas, fato esse que pode ser atenuado aumentando-se a quantidade de retransmissões. Mas o problema maior ocorre quando um dos dispositivos acaba saindo do grupo (se desconectando) por estar muito longe do *group owner*.

No modo AP, a comunicação e troca de mensagens ocorreu sem maiores problemas, com a condição de que os dispositivos conectados ao AP estejam no alcance do mesmo.

Alguns problemas foram observados no *tablet* da Samsung. O primeiro deles está relacionado com o Wi-Fi Direct: apesar da busca por *peers* finalizar com sucesso, o *tablet* não conseguia "entrar" em nenhum grupo P2P. As requisições eram enviadas pelo *tablet*, mas nenhum dos demais dispositivos (principalmente o *group owner*) as recebia. Outro problema ocorreu quando um AP "Cade meu onibus?" estava disponível. O *tablet* não conseguia se conectar ao AP e o aplicativo era finalizado com erro.

Como o *tablet* testado já tinha apresentado, previamente, alguns problemas relacionados com sua interface Wi-Fi, os testes não-sucedidos foram atribuídos à este fato, e não ao aplicativo móvel desenvolvido.

# 6 RESULTADOS E DISCUSSÕES

Este capítulo apresenta discussões sobre os resultados obtidos após concluída a fase de desenvolvimento e testes. Em linhas gerais, são debatidos temas que causaram contrariedades no decorrer do presente empreendimento. Em um primeiro momento, argumenta-se sobre a possibilidade de substituição das redes *ad hoc* pelo Wi-fi Direct, no Android. Logo após, a discussão volta-se para a utilização de um AP, seguido das diversas imposições infligidas pela API do Google Maps. Finalmente, argumenta-se sobre a solução encontrada para o uso do aplicativo em modo *offline*, ou seja, quando não há conexão com a Internet.

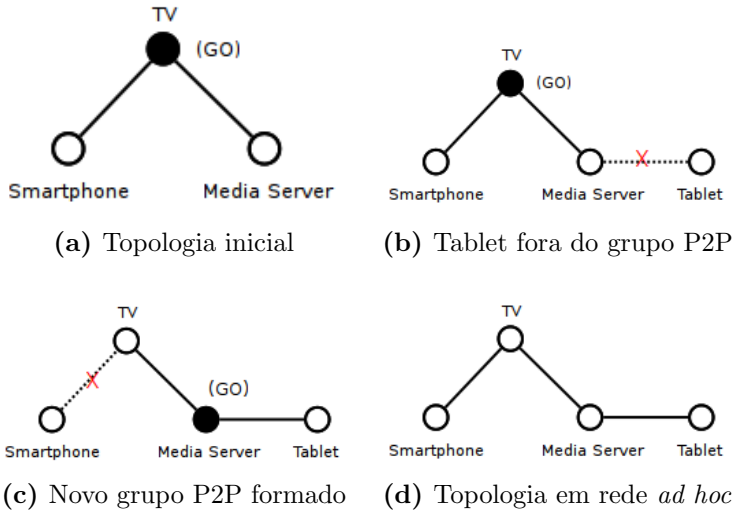
## 6.1 WI-FI DIRECT: SUBSTITUTO PARA REDES *AD HOC*?

No Capítulo 5 foram apresentadas várias limitações, impostas pela implementação do Wi-Fi Direct no Android. Aceitação de convites vindos de dispositivos desconhecidos, desmantelamento da rede no momento em que o *group owner* sai do grupo P2P e a não possibilidade de ter mais de um *group owner* são alguns dos obstáculos encontrados durante a utilização do Wi-Fi Direct no Android.

Acredita-se que o Wi-Fi Direct veio para substituir o tradicional modo de redes *ad hoc* nos dispositivos Android. Contudo, dadas as limitações encontradas durante o desenvolvimento deste projeto, será que a Google acertou em substituir o modo *ad hoc* pelo Wi-Fi Direct?

A Figura 12 apresenta uma topologia na qual é possível comparar os resultados quando o Wi-Fi Direct e o modo *ad hoc* são aplicados. Inicialmente, três dispositivos fazem parte de um grupo P2P, com a TV atuando como *group owner* (Figura 12(a)). O *smartphone* não consegue "enxergar" diretamente o *media server*, nem vice-versa, pois no Wi-Fi Direct apenas o *group owner* é visível para outros dispositivos (Thinktube Inc., 2015b).

Supõe-se que um quarto dispositivo – *tablet* – esteja próximo do *media server*,




**(a)** Topologia inicial **(b)** Tablet fora do grupo P2P

**(c)** Novo grupo P2P formado **(d)** Topologia em rede *ad hoc*

**Figura 12:** Cenário de utilização com Wi-Fi Direct e rede *ad hoc*.
Fonte: Adaptado de Thinktube Inc. (2015b).

mas distante do *group owner* (TV). O *tablet* requisita conexão com o *media server*, que rejeita-a, pois já faz parte de um grupo P2P. O *tablet*, então, não pode fazer parte deste grupo, pois não consegue contactar o *group owner* (Figura 12(b)) (Thinktube Inc., 2015b). A solução paliativa, nesse caso, seria o *media server* criar um novo grupo P2P (sendo o *group owner*) para que o *tablet* possa se conectar à ele (Figura 12(c)). Para fazer parte do grupo, a TV deveria se conectar ao novo *group owner*. Nesse processo, o *smartphone* ficaria de fora do grupo P2P, pois o *group owner* ao qual estava conectado – TV – faz parte de outro grupo (Thinktube Inc., 2015b).

Se o modo *ad hoc* fosse aplicado nesse mesmo cenário (Figura 12(d)), o *tablet* poderia se conectar ao *media server* independentemente dos outros dispositivos, sem a necessidade de recriar grupos e atribuir novos *group owners* (Thinktube Inc., 2015b).

A questão da rede desmantelar-se no momento em que o *group owner* deixa a mesma é um fator limitante quando o Wi-Fi Direct está sendo aplicado em situações que envolvem muita mobilidade, como é o caso de compartilhamento de informações sobre o transporte coletivo. Como o grupo P2P é extremamente dependente do *group owner* que, na implementação do Android, é único, se o mesmo deixar o grupo ou sair do alcance dos demais dispositivos da rede, é necessário a criação de um novo grupo, com outro *group owner* (Thinktube Inc., 2015b). No caso de redes *ad hoc*, não existem *group owners* e os dispositivos na rede não ficam "amarrados" uns com os outros. Negociações e convites, como no Wi-Fi Direct, não são necessários para que outros dispositivos ingressem na rede *ad hoc* (Thinktube Inc., 2015b).

Pelas razões apresentadas, o Wi-Fi Direct representa um protocolo mais de for-

mação de grupos (hierárquicos) e que funciona muito bem para poucos dispositivos, não muito distantes um do outro (Thinktube Inc., 2015b). O Wi-Fi Direct, ao contrário das redes *ad hoc*, não apresenta boa escalabilidade – os dispositivos ficam muito dependentes do *group owner*, que pode sair da rede a qualquer momento (Thinktube Inc., 2015b). Nas redes *ad hoc*, não há hierarquias, todos os dispositivos na rede possuem o mesmo papel e são independentes uns dos outros. O modo *ad hoc* é ideal para situações em que a rede pode crescer rapidamente, independente de infraestrutura e com possibilidades de ficar indisponível como, por exemplo, redes tolerantes a atrasos e redes *mesh* (Thinktube Inc., 2015b).

Wi-Fi Direct e redes *ad hoc* têm aplicações distintas, e um não pode substituir o outro. Por um lado, o Wi-Fi Direct funciona muito bem quando deseja-se conectar poucos dispositivos, por um período específico de tempo e de forma segura. Por outro, as redes *ad hoc* são escaláveis, permitindo a formação de redes que envolvem um grande número de dispositivos (Thinktube Inc., 2015b).

## 6.2 UTILIZAÇÃO DE UMA INFRAESTRUTURA (AP)

O Wi-Fi Direct tinha o propósito de possibilitar que o aplicativo móvel operasse sem a necessidade de uma infraestrutura, de um nó central. No entanto, conforme já discutido na seção 5.2, as limitações impostas pelo padrão Wi-Fi Direct, juntamente com a não compatibilidade de redes *ad hoc* pelo sistema operacional Android, restringiu, e muito, o escopo do projeto.

Assim, discutiu-se com as orientadoras a possibilidade de se utilizar um AP para criar a infraestrutura de comunicação dos dispositivos. Idealmente, esses APs estariam distribuídos nos ônibus, pontos de ônibus e terminais, de forma que as pessoas interessadas no serviço de transporte coletivo tivessem fácil acesso ao serviço provido pelo aplicativo. Os APs distribuídos pela cidade receberiam o mesmo SSID como, por exemplo, “URBS” ou “Cade meu onibus?”, para que a conexão fosse sempre transparente, uma vez que o usuário se conectasse à um dos pontos de acesso.

Apesar de se mostrar uma alternativa ao Wi-Fi Direct, ela necessitaria de intervenção por parte do governo para instalar os APs pela cidade e nos ônibus. Além disso, também dependeria do bom-senso da população para preservar os pontos de acesso. Ações como roubo e vandalismo prejudicaria o funcionamento do aplicativo em alguns pontos da cidade, uma vez que o mesmo seria extremamente dependente desses APs.

O suporte ao modo de infraestrutura com AP foi implementado no aplicativo e seus testes, embora restritos à apenas alguns pontos de acesso, foram satisfatórios.

## 6.3 IMPOSIÇÕES DA API DO GOOGLE MAPS ANDROID

Durante o desenvolvimento do aplicativo, discutiu-se a possibilidade de utilizar um mapa alternativo ao *Google Maps*. Pensou-se em até mapear a cidade de Curitiba para que os pontos e itinerários de ônibus fossem indicados de forma mais clara, deixando também o aplicativo mais independente e exigindo menos recursos do dispositivo, tais como conexão com a *Internet*, memória, processador, entre outros.

No entanto, após uma leitura minuciosa nos termos de uso da API do Google Maps Android contatou-se que, se o serviço de localização da *Google Play Services* está sendo utilizado, as coordenadas obtidas pelo mesmo só podem ser aplicadas em um *"Google Map"*. Essa condição é satisfeita pois, no desenvolvimento do aplicativo, foi utilizado a API *Google Play Services Location* para coletar as coordenadas do dispositivo móvel. O item (h) do Termo de Uso, subseção 10.1.1 *General Restrictions*, seção *10.1 Restrictions on How You May Use the Maps API(s)* aponta essa restrição.

> *(h) No Use of Content without a Google Map. You must not use or display the Content without a corresponding Google map, unless you are explicitly permitted to do so in the Maps APIs Documentation. [...] For example, you must not use geocodes obtained through the Service in conjunction with a non-Google map [...]* (Google Developers, 2015b).

Pensou-se em outra alternativa, que mostrasse as linhas de ônibus dispostas em "vetores", com os pontos de ônibus distribuídos nos mesmos. A ideia seria fornecer uma posição aproximada da localização do ônibus em relação à um determinado ponto, para que o usuário tivesse ao menos uma noção. Isso exigiria que as coordenadas de todos os pontos de ônibus fossem coletadas para serem distribuídas no vetor. No entanto, não foram encontradas informações referentes à latitude e longitude dos pontos de ônibus do transporte coletivo de Curitiba.

Portanto, manteve-se o uso da API do *Google Maps* para mostrar as informações sobre uma linha de ônibus. Para atenuar sutilmente as demandas de *hardware* da API, optou-se por utilizar o *"lite mode"* que, conforme explicado na seção 4.1.2, apresenta o mapa em uma figura, sem interação com o usuário. O *"lite mode"* reduz a exigência de recursos do dispositivo, como memória e processador. Contudo, a conexão com a *Internet* ainda se faz necessária para a exibição do mapa.

## 6.4 APLICATIVO MÓVEL NO MODO *OFFLINE*

Após reuniões com as orientadoras, optou-se por incluir um modo de funcionamento adicional ao aplicativo, para que o mesmo fique utilizável mesmo na ausência de conexão com a *Internet*.

Conforme mencionado na seção anterior, a API do *Google Maps*, utilizada para mostrar ao usuário a posição do ônibus, exige uma conexão ativa com a *Internet* para carregar o mapa no aplicativo. Assim, se o usuário receber informações sobre uma determinada linha e não estiver conectado à *Internet*, não poderá acessar essas informações.

Foi introduzida uma simples verificação de conexão com a *Internet* no momento em que o aplicativo é iniciado. Caso a mesma esteja disponível, a API do *Google Maps* será utilizada normalmente para carregar e mostrar o mapa ao usuário. Caso contrário, o aplicativo entrará em "Modo *Offline*" e, ao invés de tentar contactar os servidores da Google para mostrar o mapa, mostrará uma tela ao usuário com as informações sobre uma determinada linha de ônibus dispostas em formato texto.

# 7 CONCLUSÃO

Nos últimos tempos, os *smartphones* têm se tornado um artefato cada vez mais presente no dia a dia das pessoas. A contínua queda de preços desses dispositivos, aliada aos novos recursos e tecnologias que são implementadas nos mesmos periodicamente, contribuiu muito para essa popularização.

Pessoas utilizam seus *smartphones* praticamente em qualquer lugar da cidade. Ruas, parques e praças, shoppings, lanchonetes, restaurantes, dentro de ônibus (apesar do risco de furtos), em terminais, aeroportos, enfim, a lista de locais onde visualizam-se pessoas utilizando *smartphones* é interminável. Vários desses lugares oferecem conexão Wi-Fi grátis, ou seja, é possível manter-se conectado à Internet quase o tempo inteiro, mesmo quando as redes 3G/4G não estão disponíveis. Um dos resultados advindos disso é que as distâncias entre as pessoas encurtaram ainda mais: é possível manter contato com amigos e familiares estando longe de casa (e longe de um computador), com o auxílio de aplicativos móveis como o "WhatsApp" e redes sociais como o "Facebook".

E por que não se aproveitar do fato de que a maioria das pessoas se mantêm conectadas para, ao invés de conversar, trocar informações sobre fatos relevantes, como a situação do trânsito? Foi o que os desenvolvedores do aplicativo "Waze" – atualmente de propriedade da Google – pensaram. E a ideia deu certo. Segundo Cohan (2013), cerca de 50 milhões de pessoas utilizam o "Waze" (em vários países) para compartilhar informações sobre o trânsito. O aplicativo criou uma espécie de "comunidade" (ou, poderia-se dizer, rede social?) onde as pessoas podem ajudar umas as outras com informações importantes ("congestionamento na rua X", "*blitz* na rua Y", "acidente na rua Z", entre outras).

Ora, se informações sobre o trânsito são importantes, pode-se dizer que informações sobre o transporte público têm igual importância. As duas que, em um primeiro momento, parecem díspares, têm um ponto em comum: locomoção/meio de transporte. Segundo estatísticas da URBS, para o ano de 2014 mais de 2 milhões de pessoas foram transportadas por dia pela RIT (Rede Integrada de Transporte) – apenas na cidade de Curitiba (URBS, 2015).

Alguns aplicativos móveis que proveem informações sobre o transporte público em diversas cidades já existem – ou pelo menos já foram criados. Um bom exemplo aplicado à cidade de Curitiba foi o aplicativo "Busão Curitibano", que hoje encontra-se encerrado. Esses aplicativos geralmente fornecem informações como tabelas de horários ou estimam a melhor rota para que a pessoa se desloque de um ponto A até um ponto B. No entanto, um aplicativo com uma abordagem mais próxima do "Waze", mas voltado ao transporte público, não foi encontrado.

Neste projeto, buscou-se preencher esta lacuna com o desenvolvimento do aplicativo para Android chamado "Cade meu ônibus?". A ideia do projeto foi aplicar os conceitos de *crowdsourcing* e *smart cities*, de modo semelhante ao "Waze", e ir um pouco mais além: fazer com que o aplicativo funcionasse mesmo sem infraestrutura de rede disponível. Para que isso fosse possível, os dispositivos deveriam criar redes *ad hoc* para que outros pudessem se conectar e, então, trocar informações entre si.

Deparou-se, então, com o primeiro obstáculo durante o desenvolvimento. O sistema operacional Android não oferece suporte para redes *ad hoc*. Ou melhor, o sistema oferece, mas não oficialmente. É necessário utilizar um *kernel* customizado do S.O. através de um procedimento comumente chamado como *"root"*. Esse procedimento é por vezes arriscado e, obviamente, não suportado pela Google. Contudo, descobriu-se que a única forma de criar e se conectar às redes *ad hoc*, a partir de um dispositivo Android, é fazendo *"root"* no dispositivo.

O projeto pareceu, então, infactível, uma vez que a ideia original se apoiava inteiramente no conceito de redes *ad hoc* – redes sem infraestrutura. Até que, após pesquisas, descobriu-se que a Google recentemente vem suportando um modo "alternativo" ao modo *ad hoc* e apoiado inteiramente no Wi-Fi: o Wi-Fi Direct.

Na teoria, o Wi-Fi Direct parecia ser um bom substituto para as redes *ad hoc*. Na prática, contudo, foi observado exatamente o contrário. Primeiramente porque o Wi-Fi Direct necessita que uma das partes envolvidas no compartilhamento – o *group owner* – aceite cada uma das solicitações vindas de dispositivos tidos como "desconhecidos", ou seja, que ainda não fazem parte do grupo P2P. Segundo, porque o Wi-Fi Direct não funciona totalmente sem infraestrutura. O chamado *group owner*, no qual outros dispositivos se conectam ou estão conectados, funciona como um AP. Se o *group owner* – AP – sair do grupo por qualquer razão, seja ela intencional ou não, o grupo se desfaz e os dispositivos que faziam parte desse grupo precisam formar um novo grupo.

O processo de recriação de grupos P2P não é automático, tampouco transparente

aos usuários. Seria necessário que requisições fossem enviadas por algum (ou alguns) dos dispositivos pertencentes ao grupo e que algum dos dispositivos receptores aceitasse as requisições para formação de novo grupo. Aí surge novamente o problema de aceitar solicitações porque, apesar de os dispositivos já terem previamente pertencido à um mesmo grupo, eles não se "conheciam" – apenas o *group owner* tinha conhecimento dos dispositivos que estavam conectados nele. Esse processo acaba se tornando exaustivo e se repete caso o novo *group owner* deixe o grupo. Por consequência, a usabilidade do aplicativo móvel ficaria prejudicada. Foi observado, também, que o aplicativo móvel não funcionava corretamente com os chamados grupos P2P persistentes que poderiam, em teoria, eliminar o processo de aceitação de solicitações caso um dispositivo tentasse entrar em um grupo ao qual já fez parte.

O segundo obstáculo encontrado para o desenvolvimento do projeto acabou se tornando o próprio Wi-Fi Direct (que, por ironia, foi a – única – solução encontrada para o primeiro obstáculo do projeto). Neste ponto, pensou-se que o projeto era realmente infactível, até que surgiu a ideia de se utilizar um AP para centralizar – gerenciar – as conexões. Uma espécie de AP virtual foi criado com o recurso de redes hospedadas sem fio do sistema operacional Microsoft Windows, e os resultados foram satisfatórios. Se fosse pensado que, em larga escala, vários APs com o mesmo SSID e a mesma forma de autenticação fossem espalhados pela cidade – e nos ônibus também – as informações sobre o transporte público poderiam ser trocadas entre os usuários sem maiores problemas.

Foi por essa razão que ambos os modos – Wi-Fi Direct e AP – foram implementados e são suportados pelo aplicativo móvel. Na ausência de um AP (de SSID simbólico "Cade meu onibus?") o aplicativo funciona no modo Wi-Fi Direct – com suas limitações.

Espera-se que num futuro próximo o padrão Wi-Fi Direct seja aperfeiçoado e possa substituir completamente o modo *ad hoc* nos dispositivos Android pois, no presente momento, um não pode substituir o outro. Conforme exposto neste projeto, cada um dos modos funciona para um propósito diferente. Aceitação automática de solicitações, vários *group owners* em um mesmo grupo P2P ou recriação automática de grupos são apenas alguns exemplos de aperfeiçoamento do Wi-Fi Direct. E, se não for possível melhorá-lo, por que não incorporar o tradicional modo *ad hoc* ao Android?

Em relação ao desenvolvimento do projeto, a equipe trabalhou em conjunto quase que na totalidade do tempo, o que facilitou a tomada de decisões e resolução de conflitos. Felizmente, nenhum dos riscos previstos ocorreu. Não houveram atrasos durante o desenvolvimento do projeto, pois as tarefas foram cumpridas dentro dos prazos estabe-

lecidos. Inicialmente, o tempo total de trabalho estimado pela equipe, para a conclusão de todas as tarefas, era de 912 horas. Todavia, conforme o projeto se desenvolvia algumas incompatibilidades, entre o que fora proposto e o que estava sendo executado, foram observadas.

Desse modo, realocando cada tarefa ao seu tempo correto, estimou-se que o tempo total de execução deste exposto foi de 870 horas. Sobreleva-se que uma das tarefas foi retirada do cronograma, pois não apresentavam utilidade prática, ao passo que outras demandaram um tempo maior ou menor para serem concebidas. São esses os motivos que levaram a uma mudança no total de horas planejadas com relação as horas realizadas.

No que tange projetos futuros, caso um provável aperfeiçoamento do Wi-Fi Direct realmente ocorra ou o S.O. Android passe a suportar redes *ad hoc*, o aplicativo móvel poderia ser alterado para contemplar essas mudanças. Com toda certeza, o presente projeto se tornará ainda mais interessante se a Google flexibilizar o padrão Wi-Fi Direct ou oferecer suporte à criação/conexão de redes *ad hoc*. Além disso, pode-se inserir todas as linhas de ônibus de Curitiba no aplicativo, de forma que haja cobertura total de informações em toda a cidade.

# REFERÊNCIAS

ALVES, D.; MARTINEZ, L. M.; VIEGAS, J. M. Retrieving real-time information to users in public transport networks: An application to the lisbon bus system. *Procedia - Social and Behavioral Sciences*, v. 54, n. 0, p. 470 – 482, 2012. ISSN 1877-0428. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877042812042279>.

Android Developers. *Dashboards*. 2014. Disponível em: <https://developer.android.com/about/dashboards/index.html>. Acesso em: 09 de novembro de 2014.

Android Developers. *Activity*. 2015. Disponível em: <http://developer.android.com/reference/android/app/Activity.html>. Acesso em: 01 de junho de 2015.

Android Developers. *App Manifest*. 2015. Disponível em: <http://developer.android.com/guide/topics/manifest/manifest-intro.html>. Acesso em: 03 de junho de 2015.

Android Developers. *AsyncTask*. 2015. Disponível em: <http://developer.android.com/reference/android/os/AsyncTask.html>. Acesso em: 01 de junho de 2015.

Android Developers. *BroadcastReceiver*. 2015. Disponível em: <http://developer.android.com/reference/android/content/BroadcastReceiver.html>. Acesso em: 01 de junho de 2015.

Android Developers. *Fragments*. 2015. Disponível em: <http://developer.android.com/guide/components/fragments.html>. Acesso em: 01 de junho de 2015.

Android Developers. *Getting the Last Known Location*. 2015. Disponível em: <https://developer.android.com/training/location/retrieve-current.html>. Acesso em: 05 de junho de 2015.

Android Developers. *Intents*. 2015. Disponível em: <http://developer.android.com/reference/android/content/Intent.html>. Acesso em: 01 de junho de 2015.

Android Developers. *Manifest permission*. 2015. Disponível em: <http://developer.android.com/reference/android/Manifest.permission.html>. Acesso em: 16 de junho de 2015.

Android Developers. *Services*. 2015. Disponível em: <http://developer.android.com/guide/components/services.html>. Acesso em: 01 de junho de 2015.

Android Developers. *Wi-Fi Peer-to-Peer*. 2015. Disponível em: <http://developer.android.com/guide/topics/connectivity/wifip2p.html>. Acesso em: 02 de junho de 2015.

CACCIAPUOTI, A. S.; CALEFFI, M.; PAURA, L. Mobile p2p: peer-to-peer systems over delay tolerant networks. *Delay Tolerant Networks: Protocols and Applications*, p. 1–35, 2011.

CAMPS-MUR, D.; GARCIA-SAAVEDRA, A.; SERRANO, P. Device-to-device communications with wi-fi direct: overview and experimentation. *Wireless Communications, IEEE*, v. 20, n. 3, p. 96–104, June 2013. ISSN 1536-1284.

Cisco Systems Inc. *The significance of beacon frames and how to configure the beacon interval on Access Points*. 2015. Disponível em: <https://supportforums.cisco.com/document/24886/significance-beacon-frames-and-how-configure-beacon-interval-access-points>. Acesso em: 24 de junho de 2015.

CLEMMONS, R. K. *Project Estimation with Use Case Points*. 2006. Disponível em: <http://www.crosstalkonline.org/storage/issue-archives/2006/200602/200602-Clemmons.pdf>. Acesso em: 03 de março de 2015.

COHAN, P. *Four reasons Google Bought Waze*. 2013. Disponível em: <http://www.forbes.com/sites/petercohan/2013/06/11/four-reasons-for-google-to-buy-waze/>. Acesso em: 28 de junho de 2015.

CONTI, M. et al. Experimenting opportunistic networks with wifi direct. In: *Wireless Days (WD), 2013 IFIP*. [S.l.: s.n.], 2013. p. 1–6. ISSN 2156-9711.

Google Developers. *Getting Started*. 2015. Disponível em: <https://developers.google.com/maps/documentation/android/start>. Acesso em: 04 de junho de 2015.

Google Developers. *Google Maps/Google Earth APIs Terms of Service*. 2015. Disponível em: <https://developers.google.com/maps/terms>. Acesso em: 06 de junho de 2015.

Google Developers. *Map Objects*. 2015. Disponível em: <https://developers.google.com/maps/documentation/android/map>. Acesso em: 04 de junho de 2015.

HOWE, J. The Rise of Crowdsourcing. *Wired Magazine*, v. 14, n. 6, 06 2006. Disponível em: <http://www.wired.com/wired/archive/14.06/crowds.html>.

KANHERE, S. Participatory sensing: Crowdsourcing data from mobile smartphones in urban spaces. In: *Mobile Data Management (MDM), 2011 12th IEEE International Conference on*. [S.l.: s.n.], 2011. v. 2, p. 3–6.

KUROSE, J. F.; ROSS, K. W. *Redes de Computadores e a Internet: Uma abordagem top-down*. Trad. 5 ed. São Paulo: Pearson, 2010.

Microsoft. *Using Wireless Hosted Network and Internet Connection Sharing*. 2015. Disponível em: <https://msdn.microsoft.com/en-us/library/windows/desktop/dd815252(v=vs.85).aspx>. Acesso em: 22 de junho de 2015.

PAN, G. et al. Trace analysis and mining for smart cities: issues, methods, and applications. *Communications Magazine, IEEE*, v. 51, n. 6, p. 120–126, 2013. ISSN 0163-6804.

PARADELLS, J. et al. Infrastructureless smart cities. use cases and performance. In: *Smart Communications in Network Technologies (SaCoNeT), 2014 International Conference on*. [S.l.: s.n.], 2014. p. 1–6.

PERKINS, C. E. *Ad Hoc Networking*. 1. ed. [S.l.]: Addison-Wesley Professional, 2008. ISBN 0321579070, 9780321579072.

SCHUURMAN, D. et al. Smart Ideas for Smart Cities: Investigating Crowdsourcing for Generating and Selecting Ideas for ICT Innovation in a City Context. *JTAER*, v. 7, n. 3, 2012. Disponível em: <http://dblp.uni-trier.de/db/journals/jtaer/jtaer7.html\#SchuurmanBMM12>.

SIMS, G. *I want to develop Android Apps - What language should I learn?* 2014. Disponível em: <http://www.androidauthority.com/want-develop-android-apps-languages-learn-391008/>. Acesso em: 21 de novembro de 2014.

SUJATHA, K. et al. Design and Development of Android Mobile based Bus Tracking System. *First International Conference on Networks & Soft Computing (ICNSC)*, p. 231–235, 2014.

TANENBAUM, A. *Computer Networks*. 4th. ed. [S.l.]: Prentice Hall Professional Technical Reference, 2002. ISBN 0130661023.

Thinktube Inc. *Ad-Hoc (IBSS) mode support for Android 4.2.2 / 4.3 / 4.4 / 5.0*. 2015. Disponível em: <http://www.thinktube.com/android-tech/46-android-wifi-ibss>. Acesso em: 22 de junho de 2015.

Thinktube Inc. *WiFi Direct*. 2015. Disponível em: <http://www.thinktube.com/tech/android/wifi-direct>. Acesso em: 22 de junho de 2015.

URBS. *Estatísticas do transporte*. 2015. Disponível em: <http://www.urbs.curitiba.pr.gov.br/transporte/estatisticas>. Acesso em: 28 de junho de 2015.

WALRAVENS, N. Mobile Business and the Smart City: Developing a Business Model Framework to Include Public Design Parameters for Mobile City Services. *Journal of theoretical and applied electronic commerce research*, scielocl, v. 7, p. 121 – 135, 12 2012. ISSN 0718-1876. Disponível em: <http://www.scielo.cl/scielo.php?script=sci_arttext&pid=S0718-18762012000300011&nrm=iso>.

Wi-Fi Alliance. *Wi-Fi Direct*. 2015. Disponível em: <http://www.wi-fi.org/discover-wi-fi/wi-fi-direct>. Acesso em: 08 de junho de 2015.

# APÊNDICE A – PROJETO DE SOFTWARE

## A.1 REQUISITOS FUNCIONAIS

**RF1** O aplicativo móvel deverá ser executado em um *smartphone* ou *tablet* com sistema operacional Android.

**RF2** O aplicativo móvel deverá coletar as coordenadas do dispositivo no qual está sendo executado.

**RF3** O aplicativo móvel deverá apresentar ao usuário uma tela com as opções para acompanhamento de ônibus.

**RF4** O aplicativo móvel deverá oferecer um recurso para que o usuário submeta *feedbacks* sobre uma determinada linha de ônibus.

RF4.1 Os *feedbacks* deverão estar pré-definidos no aplicativo.

RF4.2 Os *feedbacks* devem conter informações relevantes sobre um determinado ônibus, como por exemplo, se o mesmo está lotado, se atrasou, ou se houve algum acidente no seu percurso.

**RF5** O aplicativo móvel deverá armazenar as informações sobre uma determinada linha de ônibus.

RF5.1 As informações deverão ficar armazenadas por 10 minutos pois, além desse tempo, as mesmas se tornam obsoletas e perdem seu caráter informativo.

RF5.2 As informações podem ser provenientes do próprio usuário/dispositivo móvel ou de outros usuários/dispositivos móveis.

**RF6** O aplicativo móvel deverá procurar por usuários/dispositivos próximos que estejam utilizando o aplicativo.

RF6.1 Os dispositivos móveis devem estar com Wi-Fi ligado e executando o aplicativo móvel.

**RF7** O aplicativo móvel deverá encaminhar suas informações armazenadas para quaisquer outros dispositivos móveis (*smartphones* ou *tablets*) encontrados.

RF7.1 Os dispositivos móveis devem estar com Wi-Fi ligado e executando o aplicativo móvel.

**RF8** O aplicativo móvel deverá oferecer ao usuário um recurso para mostrar um mapa informando a posição de um ônibus.

## A.2 REQUISITOS NÃO FUNCIONAIS

**RNF1** O aplicativo móvel deverá executar sobre o sistema operacional Android versão 4.1 (Jelly Bean), no mínimo.

**RNF2** O aplicativo móvel deverá utilizar a tecnologia Wi-Fi para seu correto funcionamento.

**RNF3** O aplicativo móvel deverá utilizar uma arquitetura de rede descentralizada, integrando conceitos de DTN e P2P.

**RNF4** O aplicativo móvel deverá ser desenvolvido na linguagem Java.

**RNF5** O aplicativo móvel deverá ser desenvolvido utilizando o conjunto de ferramentas do Android SDK.

**RNF6** O aplicativo móvel deverá ser desenvolvido utilizando o ambiente de desenvolvimento Android Studio.

## A.3 DIAGRAMAS DE CASOS DE USO

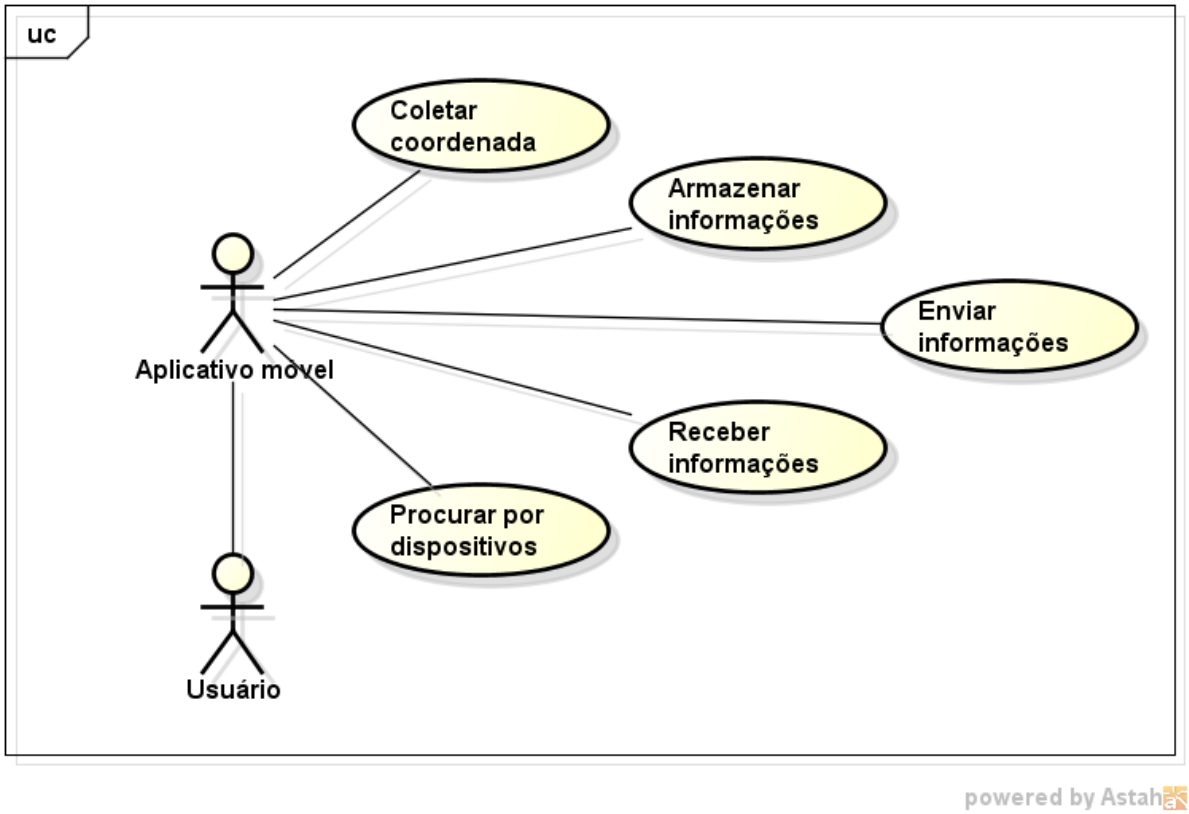



**Figura 13:** Diagrama de Casos de Uso.

**Tabela 6:** Caso de uso: coletar coordenada.

| | |
|---|---|
| **Ator principal** | Aplicativo móvel. |
| **Ator de bastidor** | N/A. |
| **Descrição** | Quando se desejar conhecer a localização em que se encontra o dispositivo móvel, o GPS embutido no *smartphone* será acionado para que as coordenadas sejam coletadas. |
| **Pré-condição** | O dispositivo móvel possui um sistema de GPS. |
| **Pós-condição** | O GPS retorna a localização em que se encontra o dispositivo. |
| **Fluxo básico** | 1. O Aplicativo móvel coleta a localização atual do dispositivo móvel, através do GPS integrado.<br>2. As coordenadas obtidas pelo GPS são armazenadas na memória do *smartphone*. |
| **Fluxo alternativo** | 1. O Aplicativo móvel não consegue acionar o dispositivo de GPS: uma mensagem de erro é apresentada na tela do aparelho.<br>2. O GPS é acionado, mas não é possível obter as coordenadas em que se encontra o dispositivo: uma mensagem de erro é apresentada na tela do aparelho.<br>3. Não há espaço suficiente para armazenar as novas coordenadas obtidas do GPS: uma mensagem de erro é apresentada na tela do aparelho. |
| **Regras de Negócio** | Nenhuma. |

**Tabela 7:** Caso de uso: armazenar informações.

| | |
|---|---|
| **Ator principal** | Aplicativo móvel. |
| **Ator de bastidor** | Usuário. |
| **Descrição** | Ocorre quando o usuário submete alguma informação referente à algum ônibus, quando as coordenadas são coletadas ou quando informações são recebidas de outros dispositivos. |
| **Pré-condição** | Necessário haver informações para armazenamento, sejam elas inseridas pelo próprio usuário, coletadas pelo GPS integrado ou recebidas de outros dispositivos. |
| **Pós-condição** | Informação é armazenada na memória do dispositivo, por 10 minutos. |
| **Fluxo básico** | 1. Após o Usuário submeter alguma informação, o aplicativo móvel armazena-a em memória por 10 minutos. |
| **Fluxo alternativo I** | 1. Informações são recebidas de outros dispositivos.<br>2. Essas informações são armazenadas em memória por 10 minutos. |
| **Fluxo alternativo II** | 1. A coordenada do dispositivo foi coletada.<br>2. Armazena a coordenada em memória, por 10 minutos. |
| **Regras de Negócio** | Nenhuma. |

**Tabela 8:** Caso de uso: enviar informações.

| | |
|---|---|
| **Ator principal** | Aplicativo móvel. |
| **Ator de bastidor** | Usuário. |
| **Descrição** | Ocorre quando há informações armazenadas no dispositivo, prontas para serem enviadas. |
| **Pré-condições** | 1. Necessário haver informações para envio, sejam elas inseridas pelo próprio usuário, coletadas pelo GPS integrado ou recebidas de outros dispositivos.<br>2. Outros dispositivos móveis nas proximidades devem estar conectados e executando a aplicação. |
| **Pós-condição** | Informações são enviadas para outros dispositivos móveis próximos que estejam conectados. |
| **Fluxo básico** | 1. O Usuário decide notificar demais usuários sobre alguma informação relevante.<br>2. O Usuário seleciona a informação no menu com opções pré-definidas.<br>3. O Usuário confirma determinada informação para o ônibus em específico.<br>4. O Aplicativo móvel envia a informação para quaisquer outros dispositivos móveis próximos que estejam conectados.<br>5. A informação fica armazenada em memória por 10 minutos. |
| **Fluxo alternativo I** | 1. Informações são recebidas de outros dispositivos móveis.<br>2. As informações são repassadas para demais dispositivos móveis próximos que estejam conectados.<br>3. Essas informações ficam armazenadas em memória por 10 minutos. |
| **Fluxo alternativo II** | 1. A coordenada do dispositivo foi coletada.<br>2. A coordenada é repassada para demais dispositivos móveis próximos que estejam conectados.<br>3. A coordenada fica armazenada em memória por 10 minutos. |
| **Regras de Negócio** | Nenhuma. |

**Tabela 9:** Caso de uso: receber informações.

| | |
|---|---|
| **Ator principal** | Aplicativo móvel. |
| **Ator de bastidor** | N/A. |
| **Descrição** | Ocorre quando o aplicativo móvel recebe informações/mensagens de outros dispositivos. |
| **Pré-condição** | As mensagens recebidas devem estar construídas de acordo com um protocolo pré-configurado no dispositivo. |
| **Pós-condição** | As informações/mensagens são armazenadas na memória por 10 minutos. |
| **Fluxo básico** | 1. O aplicativo móvel recebe uma informação/mensagem de outro dispositivo móvel.<br>2. O aplicativo móvel verifica se reconhece o padrão da mensagem recebida.<br>3. O aplicativo móvel armazena a nova mensagem na memória de mensagens recebidas (para consulta local).<br>4. O aplicativo móvel armazena a nova mensagem na memória de mensagens a enviar (para propagar a mensagem a outros dispositivos). |
| **Fluxo alternativo** | 1. A mensagem recebida não possui os padrões esperados e é automaticamente descartada.<br>2. A memória destinada a alocação de mensagens recebidas está cheia: uma mensagem de notificação é mostrada na tela do dispositivo. |
| **Regras de Negócio** | Nenhuma. |

**Tabela 10:** Caso de uso: procurar por dispositivos.

| | |
|---|---|
| **Ator principal** | Aplicativo móvel. |
| **Ator de bastidor** | Usuário. |
| **Descrição** | Ocorre periodicamente durante a execução do aplicativo móvel. |
| **Pré-condições** | 1. O aplicativo móvel deve estar executando.<br>2. O Wi-Fi no dispositivo móvel deve estar ligado. |
| **Pós-condição** | Uma conexão é estabelecida com outros dispositivos móveis que estejam próximos e executado o aplicativo. |
| **Fluxo básico** | 1. O Usuário liga a conexão Wi-Fi no dispositivo móvel.<br>2. O Usuário inicia o Aplicativo móvel.<br>3. O Aplicativo móvel procura por outros dispositivos móveis nas proximidades.<br>4. Se um dispositivo é encontrado, inclui o mesmo na lista de dispositivos encontrados e estabelece conexão.<br>5. O Aplicativo móvel aguarda dois minutos.<br>6. Realiza nova busca, retornando ao passo 3. |
| **Fluxo alternativo** | 1. O usuário pressiona o botão para procurar por dispositivos imediatamente.<br>2. Vai para o passo 3 do Fluxo Básico. |
| **Regras de Negócio** | Nenhuma. |

## A.4 PONTOS DE CASO DE USO

A estimativa por *Use Case Points* consiste em alguns passos. Clemmons (2006) exemplificou os passos para o cálculo do *Use Case Points*, os quais serão seguidos neste capítulo.

O primeiro passo é calcular o UAW (*Unadjusted Actor Weight*), o peso total dos atores do sistema. Os atores foram classificados como: ator complexo (Usuário), com peso 3 e ator médio (Aplicativo móvel), com peso 2.

$$\text{UAW} = 3\text{x}1 + 2\text{x}1 = 5$$

Em seguida, calcula-se o peso não-ajustado de cada Caso de Uso do sistema. Os casos de uso foram classificados da seguinte forma:

- Coletar coordenada: caso de uso médio (peso 10).
- Armazenar informações: caso de uso médio (peso 10).

- Enviar informações: caso de uso complexo (peso 15).
- Receber informações: caso de uso médio (peso 10).
- Procurar por dispositivos: caso de uso médio (peso 10).

O UUCW (*Unadjusted Use Case Weight*) é então calculado:

$$\text{UUCW} = 10\text{x}3 + 15\text{x}2 = 60$$

No terceiro passo, calcula-se o UUCP (*Unadjusted Use Case Point*), que consiste na soma de UAW com UUCW:

$$\text{UUCP} = \text{UAW} + \text{UUCW} = 65$$

Prossegue-se então para o cálculo de fatores técnicos, que cobre requisitos funcionais do sistema, e fatores de ambiente, que cobre requisitos não-funcionais associados ao processo de desenvolvimento.

A variável de fatores técnicos, TFactor, é obtida através do somatório dos níves F1 a F13, multiplicados pelo seu peso:

**Tabela 11:** Fatores técnicos.

| **Fator** | **Fatores que contribuem para a complexidade** | **Peso** | **Valor** | **Total** |
|---|---|---|---|---|
| F1 | Sistemas distribuídos | 2 | 5 | 10 |
| F2 | Tempo de resposta | 1 | 5 | 5 |
| F3 | Eficiência para o usuário final (on-line) | 1 | 5 | 5 |
| F4 | Processamento interno complexo | 1 | 4 | 4 |
| F5 | Código reusável | 1 | 2 | 2 |
| F6 | Facilidade de instalação | 0,5 | 1 | 0,5 |
| F7 | Facilidade de uso (facilidade operacional) | 0,5 | 4 | 2 |
| F8 | Portabilidade | 2 | 3 | 6 |
| F9 | Facilidade de mudança | 1 | 3 | 3 |
| F10 | Concorrência (acesso simultâneo à aplicação) | 1 | 4 | 4 |
| F11 | Recursos de segurança | 1 | 3 | 3 |
| F12 | Fornece acesso direto para terceiros | 1 | 0 | 0 |
| F13 | Requer treinamento especial para o usuário | 1 | 0 | 0 |
| TFactor | | | | 44,5 |

O TCF (*Technical Complexity Factor*) é então calculado:

$$\text{TCF} = 0{,}6 + (0{,}01 \text{ x } \text{TFactor}) = 0{,}6 + (0{,}01 \text{ x } 44{,}5) = 1{,}045$$

A variável de fatores de ambiente, EFactor, é calculada através do somatório dos níveis F1 a F8, multiplicados pelo seu peso:

**Tabela 12:** Fatores de ambiente.

| **Fator** | **Fatores que contribuem para a complexidade** | **Peso** | **Valor** | **Total** |
|---|---|---|---|---|
| F1 | Familiaridade da equipe com o processo formal de desenvolvimento adotado | 1,5 | 4 | 6 |
| F2 | Colaboradores de meio período | -1 | 2 | -2 |
| F3 | Capacidade do líder de projeto em análise de requisitos e modelagem | 0,5 | 4 | 2 |
| F4 | Experiência da equipe em desenvolvimento de aplicações do gênero em questão | 0,5 | 5 | 2,5 |
| F5 | Experiência em orientação a objetos | 1 | 5 | 5 |
| F6 | Motivação da equipe | 1 | 5 | 5 |
| F7 | Dificuldades com a linguagem de programação | -1 | 3 | -3 |
| F8 | Requisitos estáveis | 2 | 3 | 6 |
| EFactor | | | | 21,5 |

O EF (*Environmental Factor*) é então calculado:

$$EF = 1{,}4 + (-0{,}03 \text{ x } EFactor) = 1{,}4 + (-0{,}03 \text{ x } 21{,}5) = 0{,}755$$

Com os resultados obtidos nos passos anteriores, O valor de UCP (*Use Case Points*) pode ser então calculado:

$$UCP = UUCP \text{ x } TCF \text{ x } EF = 51{,}28$$

O tempo de trabalho estimado pode ser obtido multiplicando-se UCP por 20. Portanto, estimou-se 1025,6 horas de trabalho.

## A.5 GERENCIAMENTO DE TEMPO

Para a elaboração da Rede PERT-CPM foi necessário, primeiramente, levantar as tarefas do projeto em questão. As tarefas levantadas, bem como a duração de cada uma delas podem ser observadas na Tabela 13:

Utilizando-se dos resultados expostos, a Rede PERT-CPM para o projeto é mostrada na Figura 14.

**Tabela 13:** Tarefas do projeto.

| Número | Tarefa | Duração |
|---|---|---|
| 1 | Estudo de APIs e do Android SDK | 100 |
| 2 | Projeto de software (modelagem UML) | 100 |
| 3 | Implementação do projeto de software | 280 |
| 4 | Testes do protótipo utilizando um dispositivo móvel (emulado) com sistema operacional Android | 60 |
| 5 | Testes do protótipo utilizando um dispositivo móvel (real) com sistema operacional Android | 100 |
| 6 | Finalização do artefato | 192 |
| 7 | Testes e ajustes finais no software | 104 |
| 8 | Finalização da documentação | 80 |
| 9 | Preparação da apresentação para banca | 8 |
| Total | | 1024 |

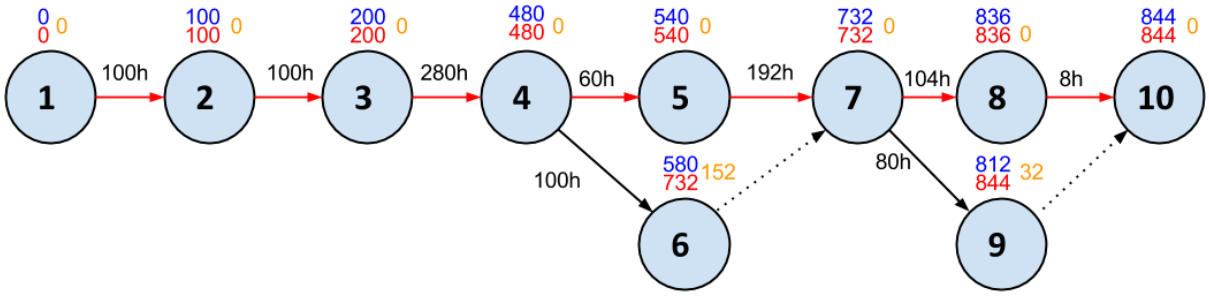



**Figura 14:** Rede PERT-CPM.

Na figura, o "cedo"de cada evento é evidenciado na cor azul, ao passo que o "tarde"de cada evento é evidenciado na cor vermelha. A folga, diferença entre tarde e cedo, é apresentada na cor laranja/amarelo.

O caminho crítico fica destacado pelas setas em vermelho, e perpassa pelos eventos 1, 2, 3, 4, 5, 7, 8 e 10 e, por consequência, pelas tarefas 1, 2, 3, 4, 6, 7 e 9.

## A.6 ANÁLISE DE RISCOS

A análise consiste, basicamente, em seis riscos, que são:

1. Escolha de tecnologias que não atendam às necessidades do projeto.
2. Dificuldades no desenvolvimento do software.

3. Descumprimento do prazo estabelecido para a realização de alguma etapa prevista no cronograma do projeto.

4. Definição imprecisa do escopo do projeto.

5. Desistência de um membro da equipe.

6. Inexistência de tempo para a conclusão do projeto.

Na sequência serão apresentadas a descrição, avaliação do risco, impacto, probabilidade e estratégia de ação correspondentes a cada um dos riscos levantados. A avaliação do risco e a probabilidade são classificadas na seguinte escala: baixa, baixa-média, média, média-alta e alta.

**Tabela 14:** Análise do risco 1.

| | |
|---|---|
| **Risco** | Escolha de tecnologias que não atendam às necessidades do projeto. |
| **Descrição** | O projeto envolve o uso de várias tecnologias, portanto, há a possibilidade de escolha equivocada de alguma delas o que pode vir a não satisfazer os requisitos mínimos necessários. |
| **Avaliação do risco** | Médio-Alto. |
| **Impacto** | Caso o emprego de uma tecnologia seja incapaz de suprir as necessidades mínimas requeridas, a etapa que faz uso deste não poderá ser finalizada. |
| **Probabilidade** | Média. |
| **Estratégia de ação** | As tecnologias que serão utilizados terão seus requisitos detalhadamente avaliados para que o grupo possa eliminar escolhas equivocadas. |

**Tabela 15:** Análise do risco 2.

| | |
|---|---|
| **Risco** | Dificuldades no desenvolvimento do software. |
| **Descrição** | Apesar de um médio conhecimento das tecnologias, corre-se o risco de empecilhos aparecem mesmo com esse conhecimento prévio. Deve-se observar ainda que o trabalho propõem-se a desenvolver uma nova arquitetura de rede que poderá se mostrar impraticável de implementação. |
| **Avaliação do risco** | Alto. |
| **Impacto** | Sendo o desenvolvimento do aplicativo o cerne do projeto, problemas com o mesmo terão que ser cuidadosamente analisados e solucionados para não inviabilizarem o projeto. |
| **Probabilidade** | Média-Alta. |
| **Estratégia de ação** | Antes de se iniciar a codificação, a equipe fará a modelagem do software para que se tenha claramente definido o comportamento do mesmo, facilitando testes e depuração de código. |

**Tabela 16:** Análise do risco 3.

| | |
|---|---|
| **Risco** | Descumprimento do prazo estabelecido para a realização de alguma etapa prevista no cronograma do projeto. |
| **Descrição** | O projeto será dividido em etapas e estas serão temporizadas de acordo com o seu grau de dificuldade para então designar-se um membro da equipe para realizá-la. Caso este membro não a conclua no prazo estipulado, etapas dependentes desta terão que ser adiadas, prejudicando o projeto como um todo. |
| **Avaliação do risco** | Alto. |
| **Impacto** | Embora algumas tarefas possam ser realizadas paralelamente, existem aquelas que são dependentes de outras e, caso ocorra atraso em uma, a tarefa dependente terá que esperar até o cumprimento da outra, o que por sua vez pode atrasar outras tarefas, ocasionando um efeito em cadeia que pode prejudicar fortemente o desenvolvimento do projeto. |
| **Probabilidade** | Média. |
| **Estratégia de ação** | Os membros da equipe terão reuniões semanais para verificar o progresso de cada etapa definida, pretende-se realizar as tarefas em grupo e dessa forma, facilitar o entendimento de uma tarefa que não esteja com um progresso adequado. |

**Tabela 17:** Análise do risco 4.

| | |
|---|---|
| **Risco** | Definição imprecisa do escopo do projeto. |
| **Descrição** | Uma definição equivocada do escopo da proposta pode trazer grandes dificuldades no desenvolvimento do projeto, culminando na entrega incompleta do mesmo. |
| **Avaliação do risco** | Médio-Alto. |
| **Impacto** | Como em todo planejamento de projeto, procura-se deixar o escopo do mesmo o mais claro possível para evitar o não cumprimento dos objetivos. Dessa forma, o cumprimento das metas é de fundamental importância seja em ambiente acadêmico ou empresarial. Logo, a descoberta de uma indefinição nesse aspecto colocaria em risco todo o planejamento e execução das tarefas. |
| **Probabilidade** | Média. |
| **Estratégia de ação** | Durante o planejamento do projeto, buscou-se definir seu escopo de tal forma que os membros da equipe já identificassem as principais áreas técnicas para o seu desenvolvimento. Assim, evita-se dispêndio de tempo para adquirir conhecimentos não pertinentes ao projeto. |

**Tabela 18:** Análise do risco 5.

| | |
|---|---|
| **Risco** | Desistência de um membro da equipe. |
| **Descrição** | Ao longo do projeto, há a possibilidade de que algum membro da equipe não se identifique mais com o projeto e queira abandoná-lo ou que seja obrigado, por motivos de força maior, a se afastar do projeto. |
| **Avaliação do risco** | Alto. |
| **Impacto** | Com a redução de integrantes da equipe, haveria uma sobrecarga sobre o membro remanescente. |
| **Probabilidade** | Baixa-Média. |
| **Estratégia de ação** | Na fase inicial do projeto todos os membros foram instigados à expor suas ideias de tal forma que o projeto como um todo tivesse a participação da dupla que o compõe, desta forma, aumentando a motivação individual dos integrantes para a completa realização do projeto. Todavia, se de fato essa situação ocorrer haverá a necessidade de uma reestruturação do plano de projeto. |

**Tabela 19:** Análise do risco 6.

| | |
|---|---|
| **Risco** | Inexistência de tempo para a conclusão do projeto. |
| **Descrição** | Como os integrantes da equipe tem que fazer outas atividades em paralelo, pode faltar tempo para terminar o projeto em sua plenitude. |
| **Avaliação do risco** | Alto. |
| **Impacto** | Caso haja escassez de tempo será inviável o término do projeto dentro do prazo estipulado, ocasionado atraso no dia de sua apresentação. |
| **Probabilidade** | Baixa-Média. |
| **Estratégia de ação** | Estimar prazos para cada tarefa, e se necessário, reduzir os requisitos finais, diminuindo assim o tempo do projeto. |

## A.7 CRONOGRAMA

O cronograma de desenvolvimento e viabilidade apresentado na Tabela 20, foi definido no início do projeto e contém informações acerca das tarefas desenvolvidas com suas respectivas datas de início e fim, bem como a duração total que possuiriam a priori.

**Tabela 20:** Cronograma Planejado.

| N° | Tarefa | Inicio | Fim | Duração |
|---|---|---|---|---|
| 1 | Estudo de APIs e do Android SDK | 18/03/2015 | 27/03/2015 | 100 h |
| 2 | Projeto de software (modelagem UML) | 28/03/2015 | 06/04/2015 | 100 h |
| 3 | Implementação do projeto de software | 07/04/2015 | 05/05/2015 | 280 h |
| 4 | Testes do protótipo utilizando um dispositivo móvel (emulado) com sistema operacional Android | 06/05/2015 | 15/05/2015 | 60 h |
| 5 | Testes do protótipo utilizando um dispositivo móvel (real) com sistema operacional Android | 16/05/2015 | 25/05/2015 | 100 h |
| 6 | Finalização do artefato | 26/05/2015 | 19/06/2015 | 192 h |
| 7 | Finalização da documentação | 22/06/2015 | 04/07/2015 | 80 h |
| Total | | | | 912 h |

Conforme o projeto se desenvolvia, assinalou-se o tempo total (de forma estimada) de execução de cada tarefa, tal como as datas de duração, a fim de comparar o cronograma real com o que fora anteriormente planejado. Os resultados dessas medições podem ser vistos na Tabela 21.

**Tabela 21:** Cronograma Executado.

| N° | Tarefa | Inicio | Fim | Duração |
|---|---|---|---|---|
| 1 | Estudo de APIs e do Android SDK | 18/03/2015 | 27/05/2015 | 130 h |
| 2 | Projeto de software (modelagem UML) | 28/03/2015 | 06/04/2015 | 80 h |
| 3 | Implementação do projeto de software | 07/04/2015 | 30/05/2015 | 280 h |
| 4 | Testes do protótipo utilizando um dispositivo móvel | 07/04/2015 | 25/06/2015 | 180 h |
| 5 | Finalização do artefato | 02/06/2015 | 21/06/2015 | 80 h |
| 6 | Finalização da documentação | 20/05/2015 | 04/07/2015 | 120 h |
| Total | | | | 870 h |

Isto posto, percebe-se, vistoriando as duas tabelas, que existem discrepâncias entre o que foi proposto inicialmente e o que foi de fato realizado. O aspecto mais evi-

dente tem relação com os períodos de realização de cada tarefa, já que de forma inócua acreditava-se que as tarefas não se sobrepunham, o que na prática mostrou-se uma inverdade.

No que tange a duração de cada afazer, os tempos variaram para mais ou para menos. Com atenção especial, fala-se das tarefas 1 e 4, as quais demandaram muito mais tempo do que se imaginava, justamente pelos problemas encontrados e já apresentados no Capítulo 6. Outro aspecto interessante e passível de análise está na redução da quantidade de horas para a realização do projeto. O motivo principal está na constatação de que não havia a necessidade de se testar o aplicativo de forma emulada, ligadamente aos novos tempos de consumação de cada tarefa.

## A.8 DIAGRAMA DE CLASSES

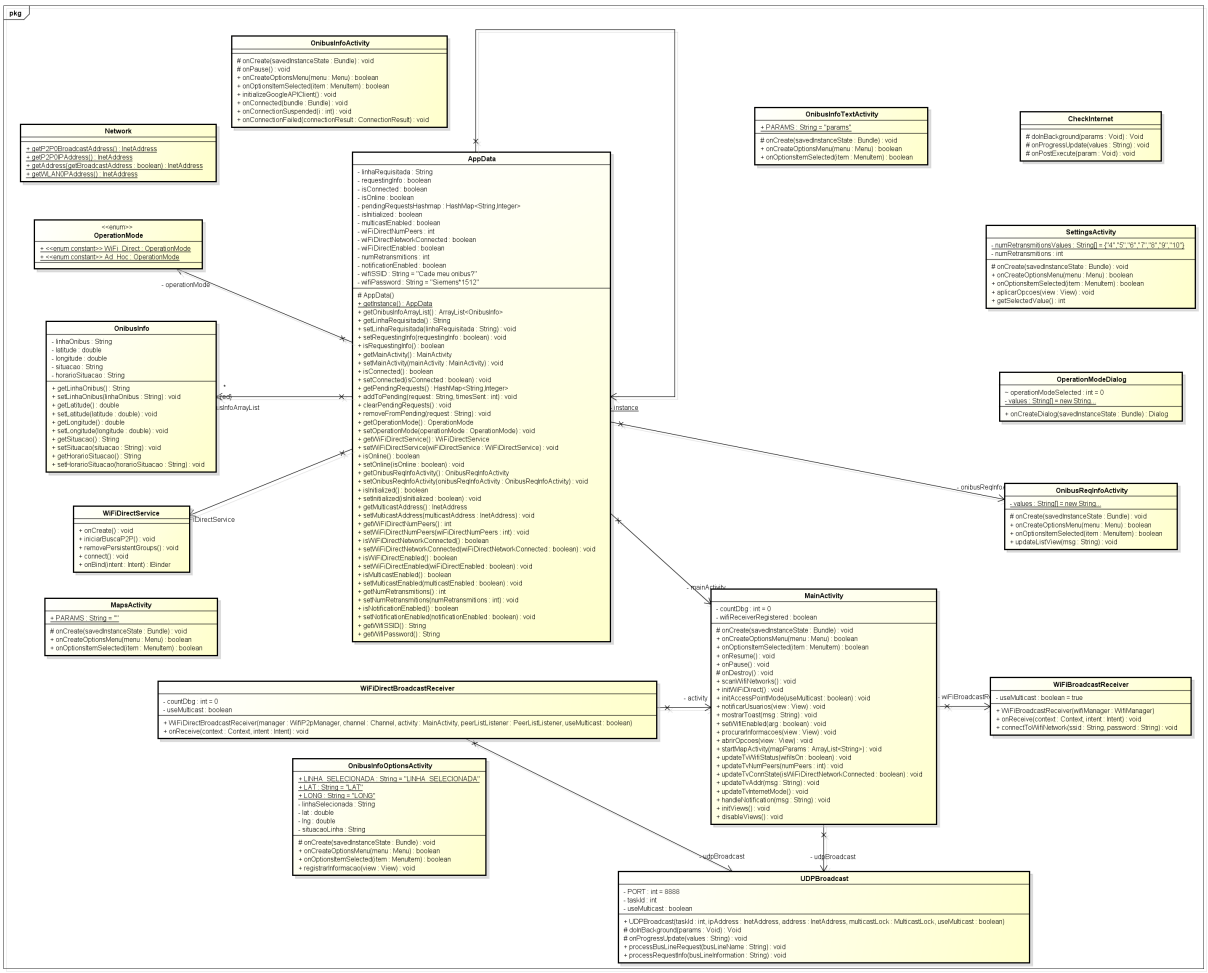



**Figura 15:** Diagrama de classes do aplicativo móvel.