UNIVERSIDADE FEDERAL DO PAMPA

EDUARDO KLUWE VEIGA

ESTUDO E PROPOSTA DE FERRAMENTA PARA COMUNICAÇÃO EM AMBIENTES DE DESASTRES

Bagé
2016

EDUARDO KLUWE VEIGA

ESTUDO E PROPOSTA DE FERRAMENTA PARA COMUNICAÇÃO EM AMBIENTES DE DESASTRES

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Orientador: Érico Marcelo Hoff do Amaral

Bagé
2016

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais) .

V426e Veiga, Eduardo Kluwe
Estudo e proposta de ferramenta para comunicação em ambientes de desastres / Eduardo Kluwe Veiga.
103 p.

Trabalho de Conclusão de Curso(Graduação)-- Universidade Federal do Pampa, ENGENHARIA DE COMPUTAÇÃO, 2016.
"Orientação: Érico Marcelo Hoff do Amaral".

1. Redes de computadores. 2. Peer to peer. 3. Desastres. 4. Redes ad hoc. I. Título.

EDUARDO KLUWE VEIGA

ESTUDO E PROPOSTA DE FERRAMENTA PARA COMUNICAÇÃO EM AMBIENTES DE DESASTRES

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Trabalho de Conclusão de Curso defendido e aprovado em: 3 de dezembro de 2016

Banca examinadora:

Prof. Dr. Érico Marcelo Hoff do Amaral
Orientador
Engenharia de Computação/Campus Bagé - UNIPAMPA

Prof[a]. Dr[a]. Sandra Dutra Piovesan
Engenharia de Computação/Campus Bagé - UNIPAMPA

Prof. Dr Milton Roberto Heinen
Engenharia de Computação/Campus Bagé - UNIPAMPA

Bagé
2016

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACK	*Acknowledgement*
AODV	*Ad Hoc On-Demand Distance Vector*
AP	*Access Point*
BSS	*Basic Service Set*
CTS	*Clear to Send*
DCF	*Distributed Coordination Function*
DIFS	*Distributed inter-frame Space*
DS	*distribution system*
ESS	*Extended Service Set*
GO	*Group Owner*
ICS	*Incident Command System*
ISO	*International Standards Organization*
JVM	*Java Virtual Machine*
LAN	*Local Area Network*
MANET	*Mobile Ad Hoc Network*
MAN	*Metropolitan Area Network*
MPR	*Multipoint Relaying*
NAV	*Network allocation vector*
OHA	*Open Handset Aliance*
OLSR	*Optimized Link State Routing*
OSI	*Open Systems Interconnectation*
P2P	*Peer to Peer*
PAN	*Personal Area Network*
PCF	*Point Coordination Function*
QoS	*Quality of Service*
RREP	*route reply*
RREQ	*route request*
RRER	*route error*

RTS	*Request to Send*
SCO	Sistema de Comando de Operações
SIG	*Bluetooth Special Interest Group*
SSID	*Service Set Identifier*
TCP	*Tranmission Control Protocol*
TC	*Topology Control*
UDP	*User Datagram Protocol*
WAN	*Wide Area Network*
WLAN	*Wireless Local Area Network*
Wi-Fi	*Wireless-Fidelity*

SUMÁRIO

RESUMO

O gerenciamento de desastres é um processo sistemático que tem como objetivo reduzir os impactos negativos causados por esse tipo de situação. O mesmo descreve como as operações de resgate devem ser conduzidas, organizadas, e qual o papel de cada profissional dentro da operação. Considerando que durante um desastre os tradicionais meios de comunicação podem se encontrar danificados,este trabalho propõe o desenvolvimento de uma aplicação para dispositivos móveis que possa auxiliar a comunicação das equipes de resgate durante uma operação de resposta à algum tipo de desastre. Esta aplicação deve permitir a comunicação entre dispositivos por meio de uma rede não infraestuturada que conecte os dispositivos diretamente uns aos outros. Para isso, foram estudadas tecnologias de redes sem fio disponíveis em *smartphones* populares. O *Wi-fi Direct*, dentre as tecnologias estudadas, se mostrou a mais eficiente para criar redes ponto a ponto, permitindo a troca de informações entre aparelhos até 80 metros de distância um do outro e também a uma taxa de transferência de até 11Mbps. Usando essa tecnologia foi desenvolvida uma aplicação que é capaz de enviar mensagens de texto e mensagens padronizadas, baseadas no código internacional Q para uma rede composta por vários dispositivos conectados diretamente uns com os outros.

Palavras-chave: P2P. Desastre. *Wi-Fi Direct*. Operação de resgate. Redes ad hoc.

ABSTRACT

The disaster management is a systematic process that aims to reduce the negative impacts of disasters. It describes how rescue operations should be conducted, organized, and the each professional role on the operation. Whereas over a traditional media disaster can be found damaged, this paper proposes the development of an application for mobile devices that can aid communication of rescue teams during a response operation to a disaster incident. This application should allow a communication between devices over a infrastructure less network that connects the devices directly to each other. For that, Wireless network technologies available in popular smartphones were studied. The Wi-Fi Direct, among the technologies studied, was the most efficient to create peer to peer networks, allowing data exchange between devices up to 80 meters away from each other and a transfer up to 11Mbps. Using this technology was developed an aplication which is avaliable to send text messages and standarized messages, based on the international Q code to a network consisting several devices connected directly each others.

Keywords: P2P. Disaster. Wi-Fi Direct. Rescue operation. Ad hoc networks.

1 INTRODUÇÃO

Desastres são eventos que ocorrem quando uma adversidade física de origem natural, como terremotos, tsunamis e furacões, ou causada pela ação humana, como guerras, terrorismo e acidentes industriais, provoca direta ou indiretamente danos extensos à propriedade, faz um grande número de vítimas, ou ambos (LOUREIRO, 2012).

Com a finalidade de auxiliar e resgatar vítimas e minimizar danos ao patrimônio público e privado da região atingida, equipes de resgate são mobilizadas para o local. Essas equipes são formadas por vários profissionais, tais como, médicos, bombeiros, policiais, enfermeiros, motoristas, dentre outros. É evidente que para trabalharem em conjunto tais equipes necessitam de um modelo padronizado de ação, que damos o nome de gerência de desastres (CARRILLO *et al.*, 2012).

Um modelo de gerência de desastres serve para sistematizar a forma de atuação e organização de um centro de operações de emergência. Descreve como a operação deve ser conduzida, quem e como deve comandar a mesma e quais as funções cada profissional deve desempenhar (LOPES, 2009).

Outra necessidade em operações de emergência é a comunicação entre todas as partes envolvidas, o comando, os agentes, a população afetada e a mídia. Ferramentas tradicionais de comunicação demandam uma grande infraestrutura composta por fios, antenas, servidore e satélites que irão tornar possível o funcionamento dos mesmos (GOMES, 2009). Como a infraestrutura local pode estar danificada, a comunicação entre as partes deve se dar por meio de uma tecnologia que não necessite utilizar a infraestrutura local da região.

Dispositivos móveis como *smartphones* e *notebooks* possuem a capacidade de se interconectarem em redes sem fio por meio de uma topologia dinâmica e temporária sem a necessidade de utilizarem servidores e pontos de acesso fixos. Esse tipo de rede é chamado *Mobile Ad Hoc Network* (MANETs) (FERNANDES *et al.*, 2006). Nessas redes, cada dispositivo participante (também chamado de nó) funciona como um roteador, recebendo e enviando pacotes de dados de outros nós.

1.1 Problema de pesquisa

É possível e viável desenvolver uma aplicação de comunicação para dispositivos móveis que auxilie o trabalho de um centro de operações de emergência, provendo um meio confiável de troca de informações, mesmo em uma região desprovida de redes de telefonia e *internet*?

1.2 Objetivo geral

Desenvolver um estudo sobre centros de operações de emergência, seu funcionamento, organização e propor a implementação de uma aplicação de comunicação para dispositivos móveis que possa auxiliar a comunicação entre pessoas que se encontrem dentro da região de

desastre.

1.3 Objetivos específicos

- Compreender a organização de uma equipe de resgate, quais os profissionais envolvidos e quais suas demandas;
- Propor um modelo conceitual de aplicação;
- Realizar um estudo sobre tecnologias de comunicação sem fio;
- Estudar os protocolos de roteamento, realizar experimentos, analisar sua eficiência e propor melhorias;
- Realizar um estudo sobre dispositivos móveis e suas plataformas de programação;
- Estudar outros projetos já desenvolvidos na área;
- Analisar e desenvolver a aplicação;
- Realizar um estudo comparativo de outros projetos desenvolvidos com proposta similar a aqui apresentada;
- Realizar testes funcionais e comparativos da aplicação;
- Analisar os resultados dos testes e propor melhorias

1.4 Estrutura do texto

O primeiro capítulo deste trabalho, a introdução, é onde são abordados os conceitos iniciais do mesmo, seus objetivos e o problema de pesquisa. O segundo capítulo é o referencial teórico, ou seja, o capítulo que apresenta a pesquisa bibliográfica a respeito do tema. O terceiro capítulo descreve a metodologia utilizada para a realização do trabalho, desde a pesquisa até o desenvolvimento e testes. No quarto capítulo é abordado a implementação do aplicativo, detalhando seus requisitos, a modelagem e apresentando o primeiro protótipo desenvolvido. Também é apresentado os testes feitos com as tecnologias pesquisadas com os quais se escolheu qual delas era mais eficiente para o projeto.

O quinto capítulo descreve o desenvolvimento do aplicativo, onde cada funcionalidade implementada pelo programa tem sua implementação abordada.O sexto é o capítulo de testes, aqui é descrito todo o procedimento de testes, desde o planejamento, passando por sua execução e chegando aos resultados obtidos. Após isso, o sétimo capítulo apresenta as conclusões e uma descrição de possíveis trabalhos futuros.

2 REFERENCIAL TEÓRICO

Nesse capítulo serão discutidos conceitos teóricos que servirão de base para o projeto, e abordaremos todos os temas necessários para concluir os objetivos do estudo. Inicialmente será feito um estudo sobre gerenciamento de desastres e o foco será no Sistema de Comando de Operações, um modelo para gerenciamento de desastres aplicado por órgãos públicos brasileiros.

2.1 Gerenciamento de desastres

De acordo com Carrillo *et al.* (2012) desastre é um evento causado por forças naturais ou pela ação humana que resulta em danos a pessoas, bens, serviços, ou ao meio ambiente, de tal forma que excede a capacidade de recuperação da comunidade afetada. Portanto, uma situação de desastre demanda ajuda externa que, para proteger vidas e minimizar danos, deve ser eficientemente organizada. A gestão de desastres é um processo sistemático que tem como objetivo reduzir os impactos negativos causados por desastres. É uma atividade que envolve diversos profissionais, as comunidades atingidas, mecanismos de integração e comunicação entre as partes. Braga *et al.* (2011) define a gestão de desastres como um processo dividido em etapas denominadas:

1. Redução de riscos ou mitigação;
2. Reparação ou prontidão;
3. Gestão de emergências ou resposta;
4. Restauração e reconstrução, ou recuperação

A fase da mitigação trata da prevenção de emergências e da minimização de seus efeitos. A fase da prontidão cuida das atividades de preparação para tratar uma emergência. A fase de resposta envolve todas as atividades relacionadas com as reações a uma situação de emergência (BRAGA *et al.*, 2011). Evacuações e missões de busca e resgate são exemplos de operações de resposta (CARRILLO *et al.*, 2012). A recuperação envolve atividades de reconstrução para que o ambiente atingido retorne à normalidade (BRAGA *et al.*, 2011). Em uma operação de recuperação bem sucedida, a comunidade afetada deve retornar a uma situação igual ou melhor a que estava anteriormente ao desastre (CARRILLO *et al.*, 2012). O processo é cíclico como ilustrado na figura 1, após o fim da última etapa voltamos para a primeira.

Figura 1: Etapas da gestão de desastres

Fonte: Lopes (2009)

Existem muitos modelos de gerenciamento de desastres utilizados pelas diversas organizações em vários países. No Brasil, o Sistema de Comando de Operações é um modelo que se destaca, sendo utilizado pela Defesa Civil Nacional e por vários órgãos da segurança pública.

2.1.1 Sistema de Comando de Operações

Segundo Oliveira (2009),o Sistema de Comando de Operações(SCO) é um modelo consistente e padronizado de gerenciamento de desastres. E, de acordo com Lopes (2009), serve para comandar, controlar e coordenar as operações de resposta em situações críticas, fornecendo um meio de articular os esforços de agências individuais, com o objetivo de estabilizar uma situação crítica e proteger vidas, propriedades e meio ambiente.

O SCO teve sua origem com o *Incident Command System* (ICS), que nasceu durante os anos 70, nos Estados Unidos. Foi desenvolvido pelo Departamento Florestal(*U.S. Forestry*) em resposta a uma série de problemas operacionais surgidos após uma onda de incêndios ocorrida nessa época (GOMES JÚNIOR, 2006).

Oliveira (2009) afirma que o SCO possui uma estrutura modular, flexível e padronizada, com funções previamente definidas que são ativadas somente de acordo com a demanda da operação. Há sempre um comando unificado que, inicialmente, é acionado pelos primeiros que chegam ao local. Após a instauração da função de comando, as próximas funções serão disponibilizadas de acordo com a necessidade e disponibilidade de pessoal, podendo ser posteriormente transferida para um profissional mais capacitado.

A estrutura desse sistema organiza-se por meio de um organograma, no qual haverá, no topo, o comando, e este, por sua vez, terá seus subordinados diretos. Estes poderão ativar outras unidades se forem adequadas para a operação. Cada unidade necessita de objetivos claros para trabalhar, esses objetivos devem estabelecer que desempenho se espera de cada equipe ou

unidade (OLIVEIRA, 2009). Caso algum objetivo não possa ser alcançado, o comando deverá reajustá-lo, conforme destaca Losso (2012).

O comando, que é o responsável principal pela operação como um todo, incluindo o desenvolvimento e a implementação do plano de ação (LOPES, 2009), pode ser composto por uma única pessoa ou por um grupo de pessoas representando as várias organizações envolvidas na operação (OLIVEIRA, 2009).

O *staff* de comando, ou seja, esse grupo de pessoas envolvidas na ação em questão, serve de assessoria para a função de comando e é ativado somente quando é necessário. Ele é composto por um coordenador de segurança, um coordenador de ligações ,um porta-voz e um secretário. O coordenador de segurança irá monitorar e avaliar as condições de segurança nos trabalhos e podendo , inclusive, encerrar as operações caso presencie uma condição insegura (LOSSO, 2012). Sua função é desenvolver medidas que garantam a segurança de todos envolvidos na operação (LOPES, 2009). O coordenador de ligações tem a função de manter o contato entre todas as organizações que participam da operação e do comando. O coordenador de informações é responsável por manter a mídia informada e o coordenador da secretaria é responsável pelas atividades administrativas do comando (LOSSO, 2012).

Também há o *staff* principal, que de acordo com Losso (2012) é composto por uma seção de operações, que conduz as atividades de nível tático, executando o plano de ação estabelecido pelo comando da operação; a seção de planejamento, que prepara e documenta o plano de ação para alcançar os objetivos e prioridades estabelecidas pelo comando conforme destaca Oliveira (2009); a seção de logística, responsável pelo suporte às operações, garantindo instalações, serviços e suprimentos e, por fim, a seção de administração, que é responsável pela compras, locações, contratação de serviços e toda a área administrativa da operação.

A seção de operações pode ativar vários setores operacionais, dentre eles podemos destacar a seção de bombeiros, policial, saúde, defesa civil, operações aéreas. Cada Setor operacional será ativado somente se for necessário, dependendo da natureza do desastre e de qual for o plano de ação (OLIVEIRA, 2009).

A seção de planejamento também pode ativar algumas unidades como a de situação, que irá acompanhar a evolução da emergência; unidade de recursos, que irá registrar e monitorar os recursos operacionais; unidade de documentação, responsável por documentar o plano de ação e a unidade de especialistas, que será composta por profissionais com alguma determinada especialidade que auxiliam no planejamento específico (LOSSO, 2012). Assim como a seção de operações e de planejamento ativam e desativam unidades à medida que elas são necessárias, qualquer outra seção pode fazer o mesmo.

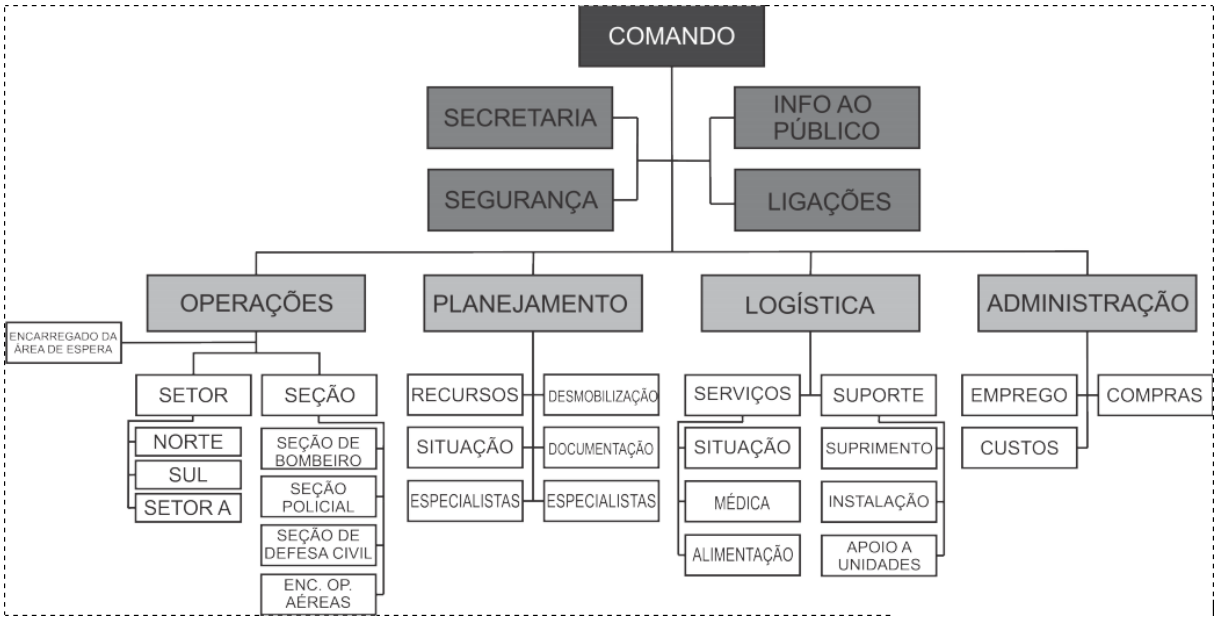
Outra característica do SCO é a presença de seis instalações padronizadas: posto de comando, base de apoio, acampamento, centro de informações ao público, helibases e helipontos. Essas instalações são determinadas pelo comando da operação em função das necessidades e complexidades da situação crítica. Com exceção do posto de comando, todas as outras instalações devem ser somente ativadas à medida que forem necessárias conforme descreve Oliveira

(2009).

O local onde são desenvolvidas as atividades de comando é chamado de posto de comando e é a primeira instalação a ser ativada; já as atividades logísticas são desenvolvidas em locais chamados de bases de apoio, dentre essas atividades, está incluso o abastecimento, manutenções de veículos, reparos etc. Acampamentos são instalações onde as pessoas envolvidas na operação conseguem alojamento, alimentação, atendimento médico e qualquer outro apoio necessário; os centros de informações são locais onde são desenvolvidas as atividades de atendimento à mídia; helibases e helipontos são onde se desenvolvem as atividades de suporte às operações aéreas; área de espera, também chamada de estacionamento, é onde serão vigiados os equipamentos, materiais e veículos dos membros da operação; e, por fim, as vítimas são reunidas, tratadas e recebem atendimento inicial na área de concentração, antes de serem encaminhadas para hospitais, se for necessário (OLIVEIRA, 2009).

A figura 2 mostra um organograma de exemplo para o SCO, onde é possível ver como toda a estrutura pode ser organizada.

Figura 2: Organograma do SCO



Fonte: Losso (2012)

Embora não exista uma sequência linear e obrigatória de etapas em todos os casos, no geral, conforme descrito por Oliveira (2009), segue-se as seguintes etapas:

1. Instalação do SCO;
2. Assunção do comando pela primeira equipe que chegou ao local do incidente;
3. Instalação do posto de comando em local apropriado;

4. Instalação da área de espera;
5. Coleta de informações sobre a situação crítica para formar um cenário mais completo da situação como um todo;
6. Elaboração de um plano de ação com base nas informações coletadas.

2.2 Comunicação em ambientes de desastre

O SCO não estabelece um modelo fixo de ferramenta de comunicação para ser usado durante uma operação, porém, de acordo com Araújo (2012) faz parte do plano de ação estabelecer que tecnologias serão necessárias para garantir a comunicação dentro do ambiente. A escolha é feita com base na análise das informações obtidas sobre o incidente, sua natureza e grau de destruição.

O coordenador de ligações e o coordenador de informações são responsáveis por garantir a comunicação dos órgãos envolvidos na operação e com a mídia, respectivamente. O Plano de comunicações é estabelecido para organizar uma forma integrada de comunicação, definindo quem se comunicará com quem e de que forma essa comunicação será feita. O Plano pode prever o estabelecimento de diferentes redes de comunicação, conforme listado por Gomes Júnior e Alves (2004):

- **Rede de comando**: integra o comando, o *staff* de comando e o *staff* principal;
- **Rede tática**: redes formadas para garantir a comunicação dentro de um mesmo setor (área geográfica) ou sessão (unidade operacional);
- **Rede administrativa**: formada para garantir a comunicação do setor logístico;
- **Rede terra-ar**: estabelecida para o controle de tráfego aéreo;
- **Rede ar-ar**: rede criada para garantir a comunicação entre aeronaves envolvidas na operação;
- **Rede de suporte médico**: utilizada para criar uma ponte de comunicação entre o SCO com hospitais e ou centros de referência para agilizar a remoção, transporte e transferência das vítimas para locais onde possam ser atendidas;
- **Rede estratégica**: é uma rede criada para estabelecer a comunicação entre o SCO e o nível de autoridade superior. Essa rede deve ser confiável, garantindo a privacidade dos interlocutores.

Conforme a necessidade, outras redes podem ser criadas uma vez que o sistema é dinâmico e modular. E, se não mais necessárias, redes de comunicação também podem ser desativadas. A separação da comunicação, através de redes independentes, garante que cada profissional

receba somente as informações necessárias para a realização do seu trabalho e evita a sobrecarga do mesmo com informações de pouca ou nenhuma utilidade. O SCO não estabelece nenhuma norma para a comunicação dos profissionais com os moradores da região onde a operação está sendo executada.

O rádio comunicador é um dispositivo mais utilizado para comunicação em operações da defesa civil em todo o mundo, por ser um meio seguro, resistente a falhas e relativamente barato. Embora o SCO não especifique nenhum padrão de comunicação organizado, o código internacional Q é largamente utilizado como uma forma de padronizar comunicações via rádio, provendo uma série de mensagens.

Criado pelo governo britânico no começo do século XX para a comunicação militar via rádio, o código internacional Q consiste em uma lista de siglas padronizadas representando mensagens que pudessem ser interpretadas em qualquer língua. O código hoje é adotado por vários órgãos e atividades que utilizam comunicação via rádio, desde fins militares ate para o rádio amadorismo. Ele é composto por 45 siglas porém várias organizações criaram novas expressões para fins específicos. (BRUSCATO; MORS, 2014).

A tabela 1 descreve o significado de algumas siglas do código Q.

Tabela 1: Lista resumida código Q

Código	**Pergunta**
QAP	Está na escuta?
QAM	Qual é a condição meteorológica?
QRA	Qual seu nome?
QRL	Esta ocupado?
QRM	Esta sofrendo interferência?
QRU	Tem algo para mim?
QRV	Está preparado?
QRX	Quando me chamará novamente?
QTW	Qual a situação dos sobreviventes?
QUA	Tem noticias de ...
QTZ	Esta continuando a busca?
QUS	Você avistou sobreviventes ou destroços?

Fonte: Instituto Defesa (2014)

Existem uma série de outras siglas que não estão sendo abordadas aqui pois a lista completa seria muito extensa. Essa seleção se baseia nas necessidades de equipes de resgate.

2.3 Redes de computadores

A comunicação via rádio é largamente utilizada em operações de emergência por ser um meio robusto, simples e bem conhecido de comunicação que não depende da infraestrutura

do local da operação. Isso garante seu funcionamento em praticamente qualquer local. Porém, a maioria da pessoas não dispõe rádios comunicadores para utilizarem quando for necessário. Redes de computadores, por outro lado, podem interconectar dispositivos utilizados pela maior parte da população, como *smartphones*, *desktops*, dentre outros.

2.3.1 Modelos de referência para redes de computadores

Para a comunicação entre quaisquer entidades ser bem sucedida é necessário que ambas entrem em acordo com relação a como a informação será representada e transmitida. Conforme explicado por Comer (2016), se dois dispositivos se comunicam por meio de uma rede com fios, ambos os lados devem definir os valores de tensão utilizados e de que forma esses sinais elétricos irão representar os dados, os procedimentos de comunicação e o formato das mensagens. Esse acordo entre os dispositivos é definido por meio de protocolos que especificam todas as etapas da comunicação

A fim de se organizarem melhor, as redes de computadores e seus protocolos utilizam modelos de referência para a organização dessas redes. Dentre esses modelos podemos destacar o modelo OSI e o protocolo TCP/IP.

2.3.2 Modelo OSI

No final dos anos 1970, foi proposto pela *International Standards Organization* (ISO) um modelo para organizar redes de computadores. Esse modelo, chamado de *Open Systems Interconnectation* (OSI), divide as redes de computadores em camadas onde cada uma responsável por uma tarefa. Segundo Tanenbaum (2003), alguns princípios foram aplicados para se chegar a essa divisão de 7 camadas, listados abaixo:

1. Uma camada deve ser criada onde houver necessidade de um grau de abstração adicional;
2. Cada camada deve executar uma função bem definida;
3. A função de cada camada deve ser escolhida tendo em vista a definição de protocolos padronizados internacionalmente;
4. Os limites de camadas devem ser escolhidos para minimizar o fluxo de informações pelas interfaces;
5. O número de camadas deve ser grande o bastante para que funções distintas não precisem ser desnecessariamente colocadas na mesma camada e, pequena o suficiente, para que a arquitetura não se torne difícil de controlar.

O modelo OSI, de acordo com Tanenbaum (2003), não é uma arquitetura propriamente dita, pois não especifica os serviços e os protocolos exatos que devem ser usados, apenas especifica a função de cada camada.

Kurose e Ross (2010) descrevem as 7 camadas do modelo OSI da seguinte forma: camada de aplicação, camada de apresentação, camada de sessão, camada de transporte, camada de rede, camada de enlace e camada física.

A camada de aplicação é onde residem as aplicações de rede e seus protocolos, estes definem os tipos de mensagens trocadas, como por exemplo, mensagens de requisição e de resposta,além de definirem a sintaxe e a semântica das mensagens e regras para determinar quando um processo envia e quando responde as mensagens. Quando se escreve um aplicativo de rede, utiliza-se as abstrações dessa camada. Desse modo, o desenvolvedor pouco precisa saber sobre como a mensagem sairá do dispositivo remetente e chegará ao destinatário, pois esses procedimentos estão definidos em outras camadas.

A camada de apresentação tem como função prover serviços que permitam que a camada de aplicação interprete o significado de dados trocados. De acordo com Farrel (2005), ela contém recursos como suporte ao idioma nacional, *buffer* de caracteres e recursos de vídeo.

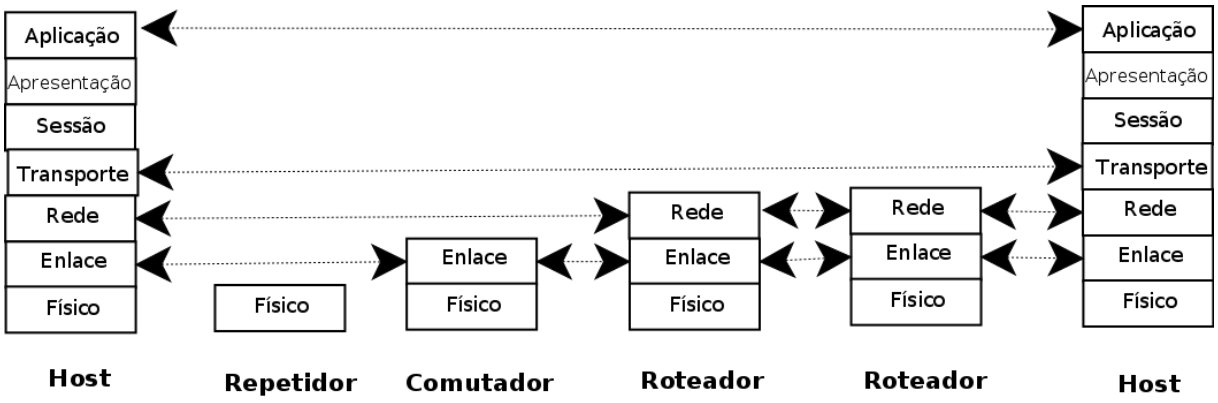
A camada de sessão delimita a sincronização da troca de dados e, abaixo dessas camadas, temos a camada de transporte que é responsável pelo transporte dos pacotes do emissor ao receptor; são providos, além disso, serviços de entrega garantida, que servem para certificar se uma mensagem enviada foi de fato recebida; serviços de controle de fluxo e tratamento de congestionamentos.

Em seguida, encontra-se a camada de rede, responsável pela entrega de dados fim a fim, lidando com o roteamento de pacotes que fará com que o a mensagem chegue seu destino, ou seja, a camada de rede trata da forma como o pacote será encaminhado entre vários nós até chegar ao seu destino final. Por outro lado, a camada de enlace é a responsável por transmitir esse pacote entre dois nós adjacentes por intermédio de um meio, chamado de enlace.

A última camada, de mais baixo nível, é a física que é responsável por transmitir não pacotes, mas *bits* individuais de um nó até outro.

A figura 3 ilustra, de acordo com Farrel (2005), a comunicação entre dois *hosts* em uma rede de computadores. Pode-se considerar que entidades vizinhas no mesmo nível de camada são adjacentes mesmo que existam saltos intermediários em níveis inferiores.

Figura 3: Comunicação entre 2 *hosts*, e a implementação de camadas em cada nó



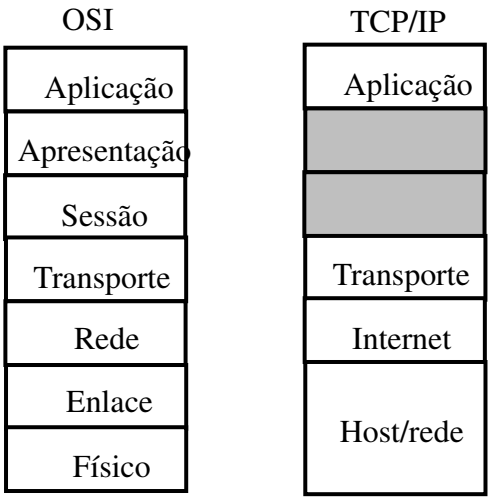
Fonte: Farrel (2005)

Toda aplicação final se situa na camada superior. Portanto, é nessa camada que o projeto aqui proposto irá se situar, utilizando, porém, os recursos de camadas inferiores. Em outras palavras, a aplicação não tem a necessidade de implementar funções que já existem nas camadas inferiores.

2.3.3 Protocolo TCP/IP

O TCP/IP é um protocolo para redes de computadores que também prevê um modelo baseado em camadas, porém organizadas de uma forma diferente, conforme ilustrado na figura 4.

Figura 4: comparação OSI e TCP/IP



Fonte: Tanenbaum (2003)

Segundo Tanenbaum (2003), os objetivo do projeto ARPANET era de prover um meio de comunicação que pudesse se manter funcionando, caso alguns dispositivos fossem destruídos ou desconectados, de forma que a troca de informações entre os dispositivos não fosse interrompida durante a queda de algum ou alguns nós intermediários.

Para que isso fosse possível, utilizou-se uma rede de comutação de pacotes, onde cada informação que precisa ser transferida pela rede é dividida em pacotes e cada pacote pode trafegar caminhos diferentes pela rede até chegar ao destino. A camada de inter-redes ou internet é a camada que organiza esse encaminhamento de pacotes entre os nós da rede. É análoga à camada de rede do modelo OSI. Esse modelo define o protocolo IP nessa camada, que será explicado adiante.

Acima da camada de *internet*, há a camada de transporte. Sua finalidade é a mesma da camada de mesmo nome do modelo OSI. Dois protocolos fim a fim foram desenvolvidos para essa camada: o *Tranmission Control Protocol* (TCP), que é um protocolo confiável,orientado à conexão e que permite a entrega sem erros de um fluxo de *bytes*, conforme Tanenbaum (2003). Esse protocolo fragmenta o fluxo de *bytes* em pacotes e envia esses pacotes para a camada de *internet* que irá encaminhar ao destinatário. No destino, os pacotes são reordenados e o fluxo de dados é recebido sem erros. O TCP pode verificar erros nos pacotes por meio de algoritmos de detecção e correção de erros (ECC). O TCP também cuida do controle de fluxo, impedindo que o transmissor sobrecarregue o receptor.

Outro protocolo definido nessa camada é o *User Datagram Protocol* (UDP), que possui funcionalidade similar ao TCP. Difere-se, porém, por não ser orientado à conexão e não confiável. Esse protocolo não prevê nenhum esquema para detecção e correção de erros e por isso tende a ser mais rápido e preferível em aplicações de tempo real que sejam tolerantes a perdas, como transmissão de vídeo e áudio *on-demand*.

A camada *host*/rede não é bem especificada no modelo. Ela é equivalente às camadas de enlace e física do modelo OSI. Segundo Tanenbaum (2003), somente é especificado que o *host* deve se conectar à rede utilizando algum protocolo para que seja possível enviar pacotes IP, embora esse protocolo não seja definido.

Acima da camada de transporte, há a camada de aplicação, onde se encontram todas as aplicações que funcionam sobre a rede. O Protocolo TCP/IP não prevê camadas de sessão e apresentação. Se uma aplicação necessitar das funcionalidades previstas nessas camadas, deverá ter a mesma implementada na aplicação.

2.4 Redes de computadores sem fio para dispositivos móveis

São considerados dispositivos móveis os aparelhos capazes de serem utilizados em movimento e que sejam facilmente portáveis. Alguns dispositivos móveis como celulares, *notebooks* e *smartphones* também são capazes de se conectarem com outros dispositivos por meio de redes sem fio (SACCOL; REINHARD, 2007).

Uma rede de computadores é formada por um conjunto de terminais, ou seja, dis-

positivos finais que podem trocar informações e por um sistema que interliga esses terminais. Podem ser classificadas de acordo com seu escopo segundo Grünewald (2005): *Personal Area Network*(PAN), redes com abrangência de no máximo alguns metros; *Local Area Network*(LAN), redes com abrangência maior, podendo interligar dispositivos por toda uma residência; *Wide Area Network* (WAN) que pode interligar dispositivos em ambientes maiores como prédios corporativos; *E Metropolitan Area Network* (MAN) que pode interconectar grandes ambientes urbanos.

Uma rede de computadores necessita de um enlace, ou canal de comunicação, que irá interconectar os dispositivos dessa rede. Redes sem fio utilizam enlaces sem fio, isto é, os pacotes de dados são propagados por meio de ondas eletromagnéticas pelo ar. Esse tipo de enlace possui uma série de características importantes que o diferenciam de enlaces cabeados. Dentre essas características, de acordo com Kurose e Ross (2010), podemos citar as seguintes :

- Redução da força do sinal: as radiações eletromagnéticas são atenuadas quando atravessam algum tipo de matéria, mesmo o ar, causando redução da sua força à medida que a distância entre o emissor e o receptor aumenta.
- Interferência de outras fontes: o sinal pode sofrer interferência de sinais vindos de outras fontes ou até mesmo de ruído eletromagnético presente no ambiente.
- Propagação multivias: ondas eletromagnéticas podem refletir em objetos e seguirem caminhos diferentes do emissor ao receptor. Isso resulta em um embaralhamento do sinal recebido pelo destinatário.

Todos essas características criam um desafio extra na manutenção de redes sem fio, uma vez que o sinal poderá chegar ao receptor com ruídos e de forma desordenada, cabendo ao receptor tratar essa situação e, ao emissor, amenizá-la. Embora tais dispositivos se comuniquem em redes sem fios, muitas vezes é necessária uma infraestrutura guiada que utiliza fios, servidores e pontos de acesso para que elas funcionem. Outros tipos de redes não necessitam dessa infraestrutura e permitem que os dispositivos se comuniquem entre si sem a necessidade de um meio físico; chamamos estas de redes de ad hoc (FIGUEIREDO; NAKAMURA, 2003).

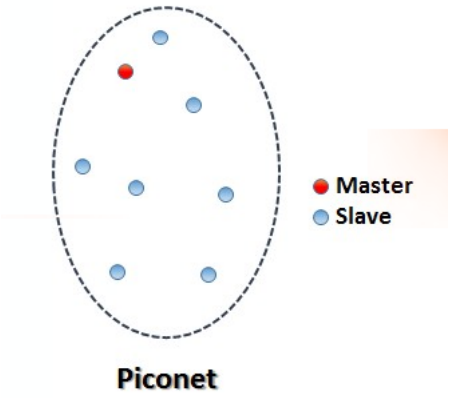
Dentre essas tecnologias, destacam-se o *Bluetooth*, o *Wi-Fi* e o *Wi-Fi Direct*. Todas essas disponíveis em *smartphones*, conforme descrito abaixo.

2.5 Bluetooth

Bluetooth é um padrão tecnológico para transferência de dados a curtas distâncias para dispositivos fixos e móveis. É o que se chama de *Personal Area Network* (PAN)(JOH; RYOO, 2014). É padronizado pela IEEE pela norma IEEE 802.15.1. O padrão é desenvolvido pela *Bluetooth Special Interest Group* (SIG). Um consórcio de mais de 19 mil empresas que detêm a marca e licencia o uso da especificação (JOH; RYOO, 2014).

O *Bluetooth* opera na faixa de frequência de 2400MHz até 2483.5MHz, dividida em 79 canais de 1MHz. A transferência de dados fica na faixa de 1Mb/s.O Protocolo *Bluetooth* utiliza uma estrutura mestre-escravo, onde cada mestre pode se conectar com até 7 escravos em uma topologia denominada *piconet*(JOH; RYOO, 2014), conforme ilustrado na figura 5.

Figura 5: Exemplo de uma piconet

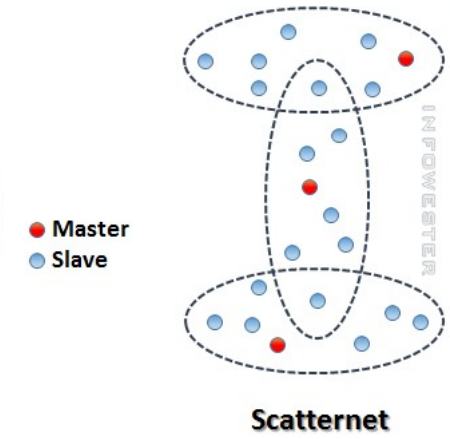


Fonte: Alecrim (2008)

Em uma *piconet* todos os nós operam no mesmo canal, utilizam a mesma *hopping sequence* e estão devidamente sincronizados com o *clock* do mestre (FERNANDES *et al.*, 2012).

Uma *scatternet* é um conjunto de várias *piconets* sobrepostas onde cada *piconet* utiliza diferentes *hopping sequences*, conforme ilustrado na figura 6. Caso um dispositivo deseje participar de mais de uma *piconet* ele tem que sincronizar com a *hopping sequence* da *piconet* que deseja participar. O Aumento da *scatternet* aumenta as possibilidades de colisões e causa a degradação da rede.(FERNANDES *et al.*, 2012).

Figura 6: Exemplo de uma scatternet

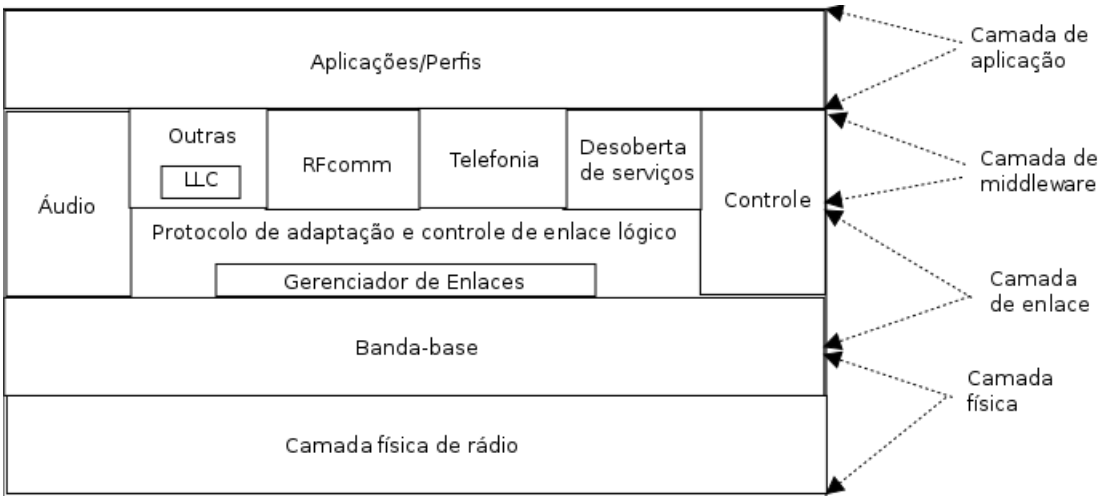


Fonte: Alecrim (2008)

Qualquer dispositivo *Bluetooth* pode ser mestre ou escravo, dependendo do cenário da aplicação. *Broadcast* é suportado ao se remover o endereço de destino das mensagens. Um escravo pode participar de várias *piconets* e um mestre pode abandonar seu posto e se tornar escravo de outra *piconet*. Um nó que conecta duas *piconets* é chamado de *bridge* (FERNANDES *et al.*, 2012).

Para atender aos mais variados tipos de dispositivos, o alcance máximo do *Bluetooth* foi dividido em três classes: Classe 1, potência máxima de 100 mW (miliwatt), alcance de até 100 metros; Classe 2, potência máxima de 2,5 mW, alcance de até 10 metros; Classe 3, potência máxima de 1 mW, alcance de até 1 metro (ALECRIM, 2008). O padrão *Bluetooth* tem muitos protocolos agrupados em uma estrutura de camadas que não segue nenhum modelo de referência conhecido. De acordo com Tanenbaum (2003), a figura 7 ilustra a arquitetura de protocolo *Bluetooth* e suas camadas.

Figura 7: Aquitetura de protocolos *Bluetooth*



Fonte: Tanenbaum (2003)

A camada inferior é a camada física de rádio, que lida com a transmissão e modulação de rádio. A segunda camada é a banda-base, que lida com o controle de *slots* de tempo pelo mestre e em como alocar esses *slots* de tempo em quadros de dados. A terceira camada contém uma série de protocolos inter-relacionados, como um gerenciador de enlaces, que cuida do estabelecimento dos canais lógicos entre dispositivos, o protocolo de adaptação de controle de enlace lógico, que isola as camadas superiores dos detalhes de transmissão e protocolos específicos para áudio e controle.

A camada seguinte, chamada de *middleware*, no qual encontram-se uma série de outros protocolos como o *RFcomm*, que simula conexões seriais, o protocolo de telefonia e o protocolo de descoberta de serviço. A camada superior é a camada de aplicações, que de forma análoga a camada de aplicações do TCP/IP e do modelo OSI, é onde ficam as aplicações que utilizam os protocolos das camadas inferiores.

2.6 Wi-Fi

O *Wireless-Fidelity* (Wi-Fi), proposto em 1990 pela organização *Wi-Fi Alliance*, é uma *Wireless Local Area Network* (WLAN) padronizada pela norma IEEE 802.11. Foi criado para padronizar as LANs sem fio, da mesma forma que as LANs com fio haviam sido padronizadas previamente. De acordo com Grünewald (2005), inicialmente o 802.11 podia transmitir somente a uma taxa máxima de 2mbps, por isso a tecnologia só foi ser popularizada posteriormente com o lançamento de sub-padrões visando a complementar e atualizar o protocolo.

O alcance de uma rede *Wi-Fi* pode ultrapassar 100m, variando de acordo com a potência dos dispositivos que está limitada a 1W. A faixa de frequência utilizada varia de acordo com o país, mas está na faixa de 2.4GHz e na faixa de 5GHz.

O padrão 802.11 foi, ao longo dos anos, sendo complementado por uma série de sub padrões que melhoram ou complementam o que já tinha sido previamente estabelecido.

2.6.1 802.11b

Primeiro padrão definido pelo comitê que padronizou o *Wi-Fi*. Tem uma taxa de transmissão máxima de 11Mbps e opera em 12 canais que vão de 2.412GHz até 2.484GHz e, um limite de 32 clientes conectados por *Access Point* (AP). Utiliza até 14 canais sobrepostos dos quais 3 não possuem sobreposição entre si. A tabela 2 mostra a faixa de frequência de cada canal.

Tabela 2: Canais do 802.11b e suas faixas de frequência

Canal	Frequência
1	2,412
2	2,417
3	2,422
4	2,427
5	2,432
6	2,437
7	2,442
8	2,447
9	2,452
10	2,457
11	2,462
12	2,467
13	2,472
14	2,484

Fonte: Grünewald (2005)

De acordo com Tanenbaum (2003), o padrão 802.11b, apesar de ser mais lento que o padrão 802.11a, possui um alcance sete vezes maior, o que é mais importante em muitas ocasiões.

2.6.2 802.11a

Finalizado após o 802.11b, embora tenha seu desenvolvimento inicial começado antes. A Velocidade pode chegar até 54 Mbps. Possui um limite máximo de 64 dispositivos conectados e opera na faixa de frequência de 5GHz divididos em 12 canais não sobrepostos. Por não operar na mesma faixa de frequência do 802.11b e do 802.11, não há compatibilidade entre esse padrão e os dois primeiros. O alto custo e a falta de compatibilidade fizeram o sub-padrão ter sido pouco utilizado.

2.6.3 802.11g

Pode transmitir a uma taxa de 54 Mbps e opera tanto na frequência de 2.4GHz quanto na frequência de 5GHz e é compatível com 802.11b, garantindo,assim, a interoperabilidade entre dispositivos que implementam cada padrão. Utiliza a autenticação WEP, porém suporta também WPA com criptografia dinâmica.

2.6.4 802.11n

Evolução do padrão 802.11g, que mantém a compatibilidade com os demais protocolos e utiliza a tecnologia *World Wide Spectrum Efficiency* , garantindo maior eficiência na propagação do sinal e fazendo com que a taxa de transferência chegue até 500mbps. Opera nas faixas de 2,4Ghz e 5Ghz e pode utilizar canais com largura de 40 MHz.

2.6.5 BSS e funções de coordenação

O *Basic Service Set* (BSS) é a base da arquitetura IEEE 802.11, sendo definida como um grupo de terminais sob o comando de uma função de coordenação, que pode ser a *Distributed Coordination Function* (DCF) ou *Point Coordination Function* (PCF).

A função DCF é obrigatória em todos os produtos compatíveis com o padrão e, de acordo com Duncan (2006) fornece um serviço *Best effort*, indicado para a transmissão de dados que não são sensíveis ao retardo da rede. Os terminais competem entre si para obter o acesso ao meio a cada transmissão de pacote (*Contention Mode*). Não há garantia de serviço (QoS) e, um alto número de estações transmitindo, ocasiona uma alta taxa de colisões de pacotes (JEAN, 2013). Kurose e Ross (2010) afirmam que antes de transmitir algum pacote, cada terminal verifica se o meio está ocioso, utilizando um mecanismo chamado de *Carrier Sense*. O meio é monitorado fisicamente, o terminal fica escutando o canal e só irá transmitir quando o canal estiver ocioso.

Se o canal estiver ocioso, o pacote será transmitido após um período de tempo conhecido como *Distributed inter-frame Space* (DIFS). Se não estiver, é atribuído um valor aleatório de *Backoff* que faŕa uma contagem regressiva sempre que perceber que o canal está ocioso. Quando o contador chegar a zero, o pacote será enviado. O Receptor enviará um ACK confirmando o recebimento do pacote após um período de tempo conhecido como *short inter-frame spacing*.

Também é possível utilizar mensagens de *Request to send* (RTS) e *clear to send* (CTS) para evitar colisões. Assim, quando um terminal desejar transmitir um pacote, ele irá enviar um RTS para o AP, indicando o tempo total requerido para a transmissão do pacote. O AP, por sua vez, transmite por *broadcast* uma mensagem, indicando ao terminal que deseja enviar um pacote e que ele tem acesso ao meio pelo tempo requerido. Dessa forma, os outros terminais não devem transmitir durante esse tempo. Os terminais da rede irão armazenar as informações do tempo total requerido para o envio do pacote em um vetor conhecido como *Network allocation*

vector (NAV) que utiliza esses dados para manter uma previsão de tráfego futuro no canal e estimar quando ele estará livre.

A Função PCF é opcional e indicada para transmissão de dados com alta sensibilidade ao retardo da rede, como, por exemplo, áudio e vídeo em tempo real. Nesse caso, o AP executa esse algoritmo centralizado, possui o controle do canal e repassa esse controle aos terminais sem fio no momento devido (*Contention-free Mode*)(DUNCAN, 2006). Segundo Rubinstein e Rezende (2002), o coordenador de ponto, que pertence ao AP, divide o período de acesso em períodos de superquadros. Cada superquadro compreende um período modo PCF e um período modo DCF. Durante o modo PCF, é função do coordenador enviar a cada terminal da rede uma mensagem solicitando o envio de um pacote e, caso esse terminal tenha algum pacote para transmitir, o mesmo será enviado e os outros terminais da rede não enviarão nada e, portanto, não haverá competição pelo meio. No próximo superquadro, caberá ao AP solicitar envio de pacote para outro terminal e assim por diante.

Um BSS pode ser classificado como uma rede com infraestrutura ou como uma rede ad hoc. Uma rede com infraestrutura é construída utilizando-se um ponto de acesso(AP). Kurose e Ross (2010) afirmam que o AP é normalmente o roteador e serve de ponto comum de acesso para todos os dispositivos da rede. o AP é responsável por transmitir e receber pacotes, encaminhando-os para seu destino correto. Dois terminais *Wi-Fi* não trocam informações diretamente, todos os dados são encaminhados pelo AP que estão conectados. Os APs conseguem aumentar a área de cobertura, pois são pontos de conexão entre vários BSSs, formando, então, um *Extended Service Set* (ESS). Esse serviço consiste na integração de múltiplos BSSs utilizando-se um *distribution system* (DS). O DS pode ser visualizado como um *Backbone* responsável pelo transporte de pacotes da sub-camada MAC entre diferentes BSSs.

Um ESS pode também fornecer acesso a uma rede com fio, como a Internet, através de um serviço chamado Portal. O Portal é uma entidade lógica que especifica o ponto de integração entre a rede IEEE 802.11 e um outro tipo de rede. (DUNCAN, 2006).

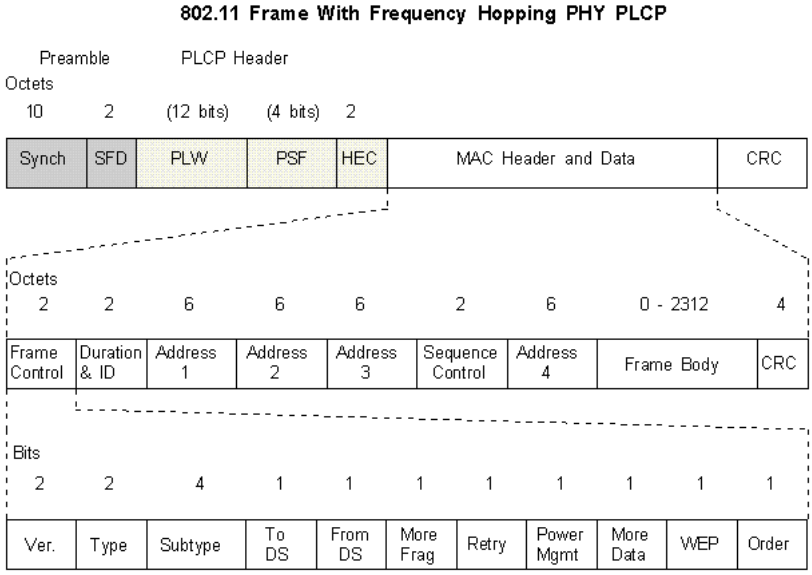
Por outro lado, uma rede *ad-hoc* é formada por um grupo de terminais *wireless*, dentro de um BSS, com comunicação direta entre si, sem a existência de um ponto centralizado de controle (AP). A falta de um AP obriga os próprios dispositivos a proverem serviços como roteamento e atribuição de endereços (KUROSE; ROSS, 2010).

Todas as estações em um BSS devem suportar a função DCF, porém, em uma rede *ad-hoc* somente é possível utilizar a função DFC, uma vez que não há um AP na rede(DUNCAN, 2006).

2.6.6 Frame 802.11

De acordo com Oliveira Júnior (2014), o *frame* do 802.11 é dividido em 9 campos, sendo o *frame control* dividido em 11 campos, totalizando 19 campos com tamanho máximo de 2346 *bytes*, conforme ilustrado na figura 8.

Figura 8: ilustração de um frame 802.11



Fonte: Haden (2000)

- O campo *Duration/ID*,de 16 *bits*, carrega a ID de associação da estação que transmitiu o quadro e o valor de duração definido para cada tipo de *frame*;
- Os 4 campos *address* servem para indicar o BSSID. Contêm 48 *bits* por endereço que são respectivamente, endereço de origem, de destino, estação transmissora e estação receptora;
- O campo *Sequence Control* é utilizado para manutenção dos *frames* em fluxo;
- *Frame Body* contém os dados propriamente ditos e seu comprimento é variável;
- O Campo *Frame Check Sequence* possui um CRC de 32 *bits* utilizado para identificar erros nos outros campos.

O campo *frame control* é dividido da seguinte forma:

- Protocol *Version* para identificar a versão do protocolo;
- *type* e *subtype* são utilizados em conjunto para identificar qual a função do *frame*, esta função pode ser de controle, de transmissão de dados ou de gerenciamento. A tabela 3 mostra algumas das combinações válidas e sua função;
- *To DS* e *from DS* utilizados para identificar se o *frame* está chegando ou sendo enviado e se a comunicação é entre dois *access points*;
- *More fragments*, utilizado para indicar a existência de mais fragmentos ou não;

- *Retry* serve para controle de retransmissões e *frames* duplicados;
- *Power Management*, se ativo, indica que o modo de gerenciamento econômico de energia da estação está ativado;
- *More Data*, usado para indicar a existência de unidades de dados em registro no *access point*;
- WEP, utilizado para indicar se o *frame* contém informações processadas pelo algoritmo criptográfico WEP;
- *Order* é utilizado para classificar se o fragmento utiliza a classe se serviço *strictilyOrdered* (DUARTE, 2003).

Tabela 3: Combinações de *type* e *subtype*

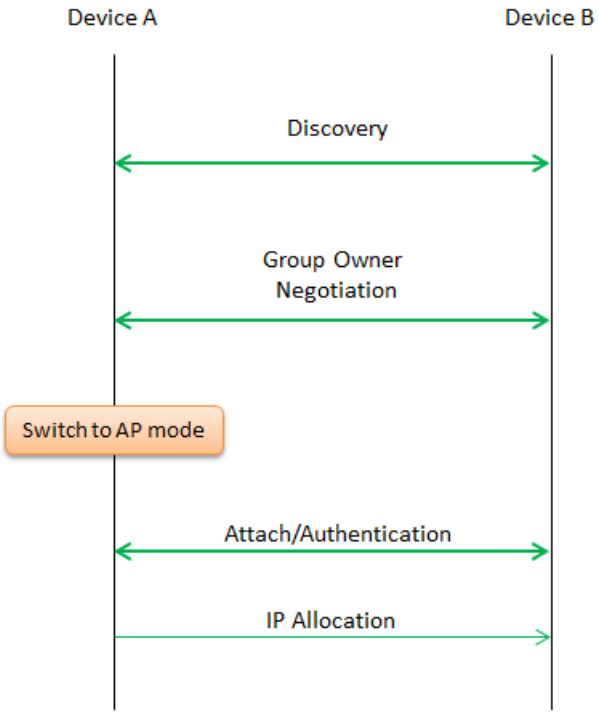
Type	**Descrição *type***	***Subtype***	**Descrição *subtype***
00	*Management*	0100	*Probe Request*
00	*Management*	0101	*Probe Respose*
00	*Management*	1000	*Beacon*
00	*Management*	1011	*Authentication*
01	*Control*	1011	*Request to Send* (RTS)
01	*Control*	1100	*Clear to Send* (CTS)
01	*Control*	1100	*Acknowledgement* (ACK)
10	*Data*	0000	*Data*

Fonte: Duarte (2003)

2.7 Wi-Fi Direct

O Padrão *Wi-Fi Direct* é segundo Machado *et al.* (2014) uma tecnologia que possibilita criar redes *ad-hoc* entre dispositivos *Wi-Fi* com a mesma facilidade encontrada em conexões *Bluetooth*. Os dispositivos que implementam *Wi-Fi Direct* comunicam-se estabelecendo grupos P2P, equivalentes às redes tradicionais *Wi-Fi* com infraestrutura pré-definida. Um dispositivo que se torna o proprietário de um grupo é chamado de *P2P Group Owner* (P2P GO), e o dispositivo que atua como cliente é chamado de *P2P Client*. Tudo sendo atribuído de forma dinâmica e automática (MACHADO *et al.*, 2014). Por outro lado, os dispositivos que não implementam o padrão ainda podem se comunicar na rede como *Legacy Clients*, eles enxergam o *P2P GO* como um AP comum (JOH; RYOO, 2014). Outros podem servir de *bridge*, conectando dois *P2P Groups*. Conforme ilustrado na figura 9, o processo de formação de um grupo pode ser dividido em duas etapas, denominadas *Discovery Phase* (fase de descoberta) e *Negotiation Phase* (fase de negociação).

Figura 9: Fases da criação de um *P2P Group*



Fonte: sharetechnote (2015)

2.7.1 Discovery phase

Nessa fase, o dispositivo irá alternar entre o estado de busca (*search state*) e o estado de escuta (*listen state*). Nesse último, o dispositivo irá escolher um dos canais de frequência disponíveis no protocolo e então escutá-lo, buscando por mensagens *probe request* para responder. Após algum tempo, o dispositivo muda para o estado de busca, nesse estado ele irá enviar um *probe request* em cada um dos canais e novamente voltar ao modo de escuta no canal definido previamente. (ZHANG *et al.*, 2014).

O *Wi-Fi Direct* trabalha utilizando os canais 1, 6 e 11 da banda de 2.4 GHz do *Wi-Fi*. Esses canais são os escolhidos pois não há sobreposição entre eles e, assim, evita-se interferências.

2.7.2 Negotiation phase

Nessa fase, o dispositivo entregará ao usuário uma lista de dispositivos que estão disponíveis e procurando um grupo. O Dispositivo, então, envia uma mensagem de *GO negotiation request* para um dos dispositivos com que deseja se conectar e, se este aceitar a conexão, responde com um *GO negociation response*;

Essa troca de mensagens é utilizada para definir quem será o *group owner* e qual será o canal utilizado na conexão. Para definir o *GO*, dois parâmetros são enviados nas mensagens de

GO negotiation. Um deles é chamado de *intent value*, um valor numérico entre 0 e 15 onde o dispositivo que enviar o maior *intent value* se tornará o *GO*. No caso de uma colisão de valores, o segundo parâmetro, *tiebreaker*, será utilizado. Esse parâmetro é um *bit* definido aleatoriamente pelo dispositivo que enviou o *GO negotiation request* e o dispositivo que responder com um *GO negotiation responde* definirá seu *tiebreaker* com o valor inverso e, quem obtiver o valor 1, será o *GO* (ZHANG *et al.*, 2014).

Segundo Camps-Mur *et al.* (2013), outra forma de se formar um *P2P Group* é pelo modo autônomo, no qual um dispositivo cria sozinho o grupo e se torna o *GO*. Então esse dispositivo manda mensagens para dispositivos próximos que estejam procurando um grupo para se conectar. Nesse caso, não será necessário realizar a negociação do *GO*.

A última forma de se realizar a criação de um *P2P Group* é utilizando o modo persistente, onde as credenciais da rede são salvas em cada dispositivo para que o grupo possa ser recriado futuramente sem a necessidade de renegociar o *GO*.

A segurança do *P2P Group* é provida pela tecnologia *Wi-Fi protected Setup* que é obrigatória em todos os dispositivos *Wi-Fi Direct*. O Procedimento, descrito por Camps-Mur *et al.* (2013) inicia-se com o *P2P GO*, gerando as credenciais da rede, ou seja, as chaves criptográficas de segurança e, após isso, o P2P *client* se conecta com suas credenciais de segurança geradas previamente. Em casos de redes persistentes, não é necessário gerar credenciais novamente.

Após criado o *P2P group* entre dois dispositivos, o próximo passo definido por Zhang *et al.* (2014) será enviar convites para que mais dispositivos se conectem ao grupo. Para isso, será necessário enviar uma mensagem de *P2P invitation request* para o dispositivo, e este irá responder com um *P2P invitation accept*, caso ele aceite o convite. O dispositivo convidado também precisará enviar um *Provision Discovery Request* para o *GO* que, ao ser respondido, concederá a permissão para a inclusão do novo dispositivo no *P2P group*. Os Convites podem ser enviados por qualquer membro do grupo, incluindo o *GO*. Se um dispositivo que não faz parte do grupo envia um *provision Discovery Request* para outro dispositivo do grupo que não seja o *GO* e o mesmo aceita, é criado um novo *P2P Group* e o dispositivo que aceitou o convite abandona o grupo antigo.

2.8 Roteamento em redes sem fio ad hoc

A camada de rede determina a rota ou caminho tomado pelos pacotes ao fluírem de um remetente ao destinatário. Os algoritmos que calculam essas rotas, segundo Kurose e Ross (2010), são chamados de algoritmos de roteamento.

Tanenbaum (2003) lista uma série de propriedades desejadas em um algoritmo de roteamento:

- Correção: o algoritmo deve escolher o caminho correto;
- Simplicidade: o algoritmo deve ser simples para evitar alto consumo de processamento;

- Robustez: o algoritmo deve ser capaz de lidar com falhas de hardware e software e seguir funcionando;
- Estabilidade: o algoritmo deve convergir a um equilíbrio;
- Equidade: o algoritmo deve distribuir a carga de modo justo entre os nós;
- Otimização: o algoritmo deve ser capaz de escolher o melhor caminho possível no menor tempo possível.

De acordo com Kurose e Ross (2010), podemos classificar os algoritmos de roteamento de acordo com certas características. Algoritmos de roteamento podem ser globais ou descentralizados. Os primeiros calculam as rotas com base em dados de todos os nós da rede, já os últimos fazem o cálculo de forma distribuída, onde cada roteador conhece apenas os seus vizinhos e não a rede inteira.

Outra forma de classificar os algoritmos é como estáticos ou dinâmicos. Os algoritmos estáticos não prevêm mudanças na topologia das redes e, portanto, o cálculo de suas rotas permanece fixo o tempo todo. Por outro lado, os dinâmicos mudam as rotas conforme variam o tráfego e a topologia da rede. Há também os algoritmos que podem considerar o congestionamento da rede para definirem os melhores caminhos, estes são chamados de algoritmos sensíveis à carga e, os que não consideram esse congestionamento, são algoritmos insensíveis à carga.

Conforme Tanenbaum (2003), o que torna as redes *ad hoc* diferentes é que a topologia das redes pode se alterar a qualquer momento e sem nenhum aviso, fazendo com que tais redes necessitem de algoritmos de roteamento próprios e diferenciados.

Protocolos de roteamento para redes *ad hoc* são classificados em 3 categorias de acordo com Farias (2008): Proativos, onde cada nó mantém uma tabela de roteamento para todos os nós da rede e essa tabela se mantém sempre atualizada para posterior utilização; Reativos, onde o cálculo das rotas é feito somente quando for necessário enviar um pacote; Híbridos, que combinam características dos dois primeiros.

Os algoritmos pró-ativos permitem baixas latências na transmissão de pacotes, pois cada nó tem tabelas de roteamento que são consultadas antes do envio dos pacotes, conforme destaca Dau (2013). Por outro lado, esses algoritmos levam a um maior consumo de energia e banda ao exigirem que as tabelas sejam continuamente atualizadas, mesmo em períodos em que a rede está ociosa. Algoritmos reativos, por outro lado, permitem uma economia de energia e banda ao realizarem a busca de uma nova rota somente no momento em que o emissor deseja enviar um pacote. Essa economia,porém, acarreta em uma maior latência no envio dos pacotes.

O protocolo pró-ativo mais difundido é o *Optimized Link State Routing* (OLSR), o qual utiliza uma técnica conhecida como *Multipoint Relaying* (MPR), que consiste em cada nó selecionar um grupo de nós para propagar as mensagens de controle. Tais mensagens são divididas em dois tipos básicos, segundo Almeida (2011):

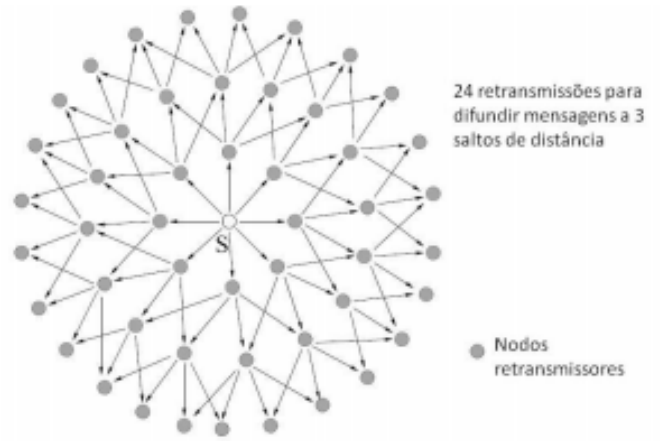
- *HELLO*: enviadas por cada nó apenas para os seus vizinhos e tem como função obter informações sobre o estado do enlace e, posteriormente, viabilizar que o nó construa seu MPR;

- *Topology Control* (TC): enviadas para todos os nós da rede, contendo informações que foram utilizadas para criar os MPR.

O Critério para se formar um MPR, de acordo com Almeida (2011), é de que os nós devem alcançar qualquer outro nó a dois saltos de distância do nó de origem. Escolhe-se um nó como integrante do MPR se ele possuir um enlace bidirecional para o nó de origem e alcançar o maior número de vizinhos a dois saltos deste nó de origem.

A figura 10 ilustra o processo de envio de mensagens por difusão sem MPR.

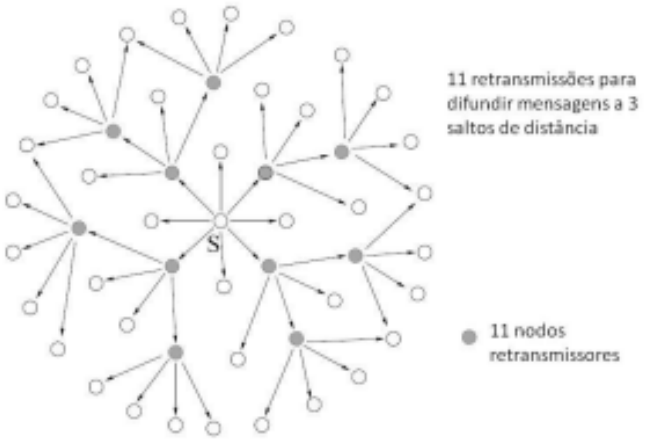
Figura 10: Difusão sem MPR



Fonte: Almeida (2011)

Em contrapartida, a figura 11 ilustra o mesmo processo com MPR.

Figura 11: Difusão com MPR

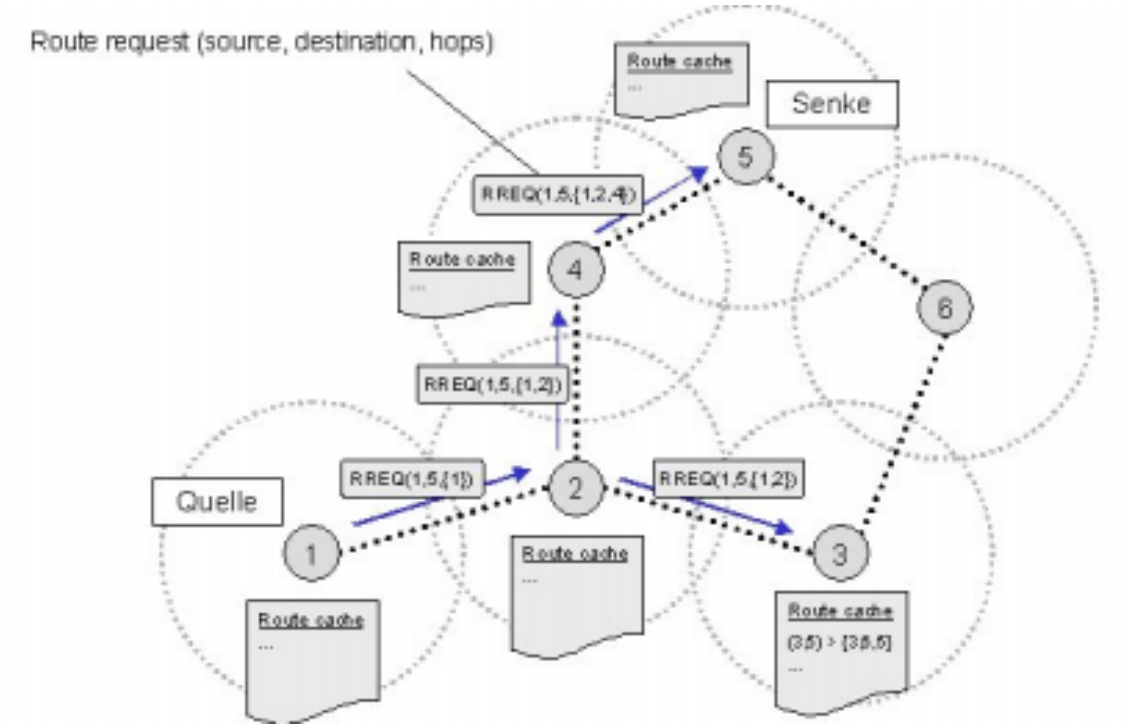


Fonte: Almeida (2011)

o *Dynamic Source Routing* é um protocolo reativo muito utilizado em redes *ad hoc*. Ele funciona com base em um *source routing* (roteamento pela fonte), ou seja, o nó transmissor conhece o caminho completo até o destino, e todas as mensagens de dados carregam essas informações.

Quando um nó deseja enviar uma mensagem para outro, de acordo com Bannack (2008), ocorre um processo para se descobrir a rota até o destino. O processo funciona enviando uma mensagem de *route request* (RREQ) em *flooding* para todos os outros nós. Cada nó que recebe um RREQ adiciona seu endereço e reencaminha a mensagem, caso esse nó seja o destinatário, ele envia um *route reply* (RREP), que retorna à origem utilizando a rota descoberta. A origem adiciona essa rota a uma tabela de roteamento e pode em seguida utilizar essa tabela para o encaminhamento dos pacotes. Mensagens de *route error* (RRER) podem ser encaminhadas indicando que algum nó foi desconectado da rede ou movido de localidade, nesse caso, quem tiver tabelas de roteamento utilizando aquele nó, irá descartá-la, solicitando nova rota. A figura 12 ilustra como uma mensagem RREQ se propaga pela rede.

Figura 12: Funcionamento da mensagem RREQ



Fonte: Farias (2008)

Outro protocolo reativo muito utilizado é o *Ad Hoc On-Demand Distance Vector* (AODV), que também utiliza mensagens RREQ,RREQ e RRER como o DSR para calcular as rotas, e envia essas mensagens somente no momento em que o emissor deseja enviar um pacote a um dado destino; porém, de acordo com Farias (2008), o que diferencia ambos os protocolos é que o roteamento não é baseado unicamente na fonte. No AODV cada nó em uma rota armazena somente informações sobre qual o próximo salto para o destino e não o caminho completo. O objetivo do protocolo é se adaptar de forma rápida e dinâmica às variações de condições dos *links* da rede e evitar desperdício de banda e consumo de memória e de processamento nos nós roteadores.

2.9 Trabalhos correlatos

Para o desenvolvimento do trabalho, foram consultados alguns projetos correlatos com o tema aqui proposto. Esses projetos podem ser divididos em aplicações desenvolvidas em um contexto não acadêmico e trabalhos acadêmicos propriamente ditos.

Os aplicativos pesquisados não foram desenvolvidos exclusivamente para a área de gerenciamento de desastres, sendo primariamente, aplicativos de troca de mensagens entre dispositivos. Utilizando tecnologias como *Bluetooth* e *Wi-Fi Direct*. Todos esses aplicativos são capazes de interconectar dispositivos diretamente. As tecnologias utilizadas são *Wi-Fi Direct* ou *Bluetooth*, ambas estudadas durante este trabalho

O aplicativo Fire Chat (GARDEN, 2014) é um *software open source* que permite a comunicação entre dispositivos por meio de redes não infraestuturadas utilizando *Bluetooth* e *Wi-Fi Direct*. O aplicativo também utiliza a comunicação com a internet quando a mesma está disponível. É uma ferramenta de *chat* que -permite troca de mensagens de texto e envio de imagens.

Os aplicativos Ensichat (ABLEITNER, 2014), Blue Chat (KANG, 2014) e Gilga (FREITAS, 2015) tem o mesmo propósito, porém, limitados ao uso do *Bluetooth*. Todos são *opesn source* e gratuitos. O aplicativo Echo (HENDERSON *et al.*, 2014), também é um *software* de *chat* que utiliza *Wi-fi Direct*, contudo, sua proposta é interligar os *P2P Groups* em uma rede mesh.

Além dos aplicativos, também foram pesquisados trabalhos correlatos na area de desastres que nao tinham como objetivo final a construção de algum *software*. Esses trabalhos abordam aspectos da comunicação em ambientes de desastres. Gielen (2012) apresenta um estudo sobre o uso da tecnologia *Wi-Fi Direct* para auxiliar a comunicação em ambientes de desastres. O estudo aborda as características da tecnologia como alcance,taxa de transferência, consumo energético, número de nós e suporte em *smartphones*. Essas características são comparadas com outras tecnologias de comunicação disponíveis.

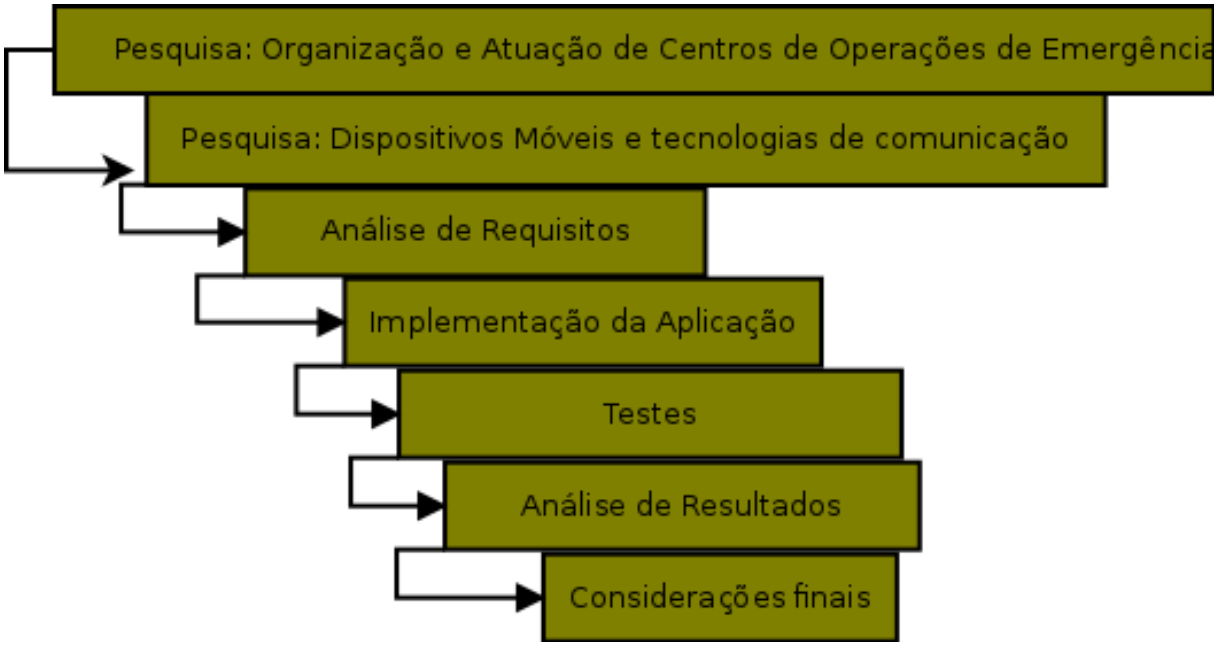
Channa e Ahmed (2010) abordam os desafios e necessidades em comunicações de emergência utilizando redes ad hoc. O estudo analisa uma série de *frameworks* para redes *ad hoc* com relação as necessidades de uma comunicação de emergência segura e eficiente. Dentre os desafios que a comunicação ad hoc possui pode se listar a privacidade, a autenticação de usuários, a integridade dos dados e o controle de acesso.

3 METODOLOGIA

O presente trabalho trata-se de uma pesquisa exploratória, que segundo Vergara (2009) é uma pesquisa que tem como objetivo proporcionar um maior conhecimento sobre um tema proposto, e bibliográfica, pois seu levantamento teórico se baseou em um uma análise da literatura disponível em artigos científicos, livros, manuais e documentação técnica. A pesquisa também pode ser classificada como uma pesquisa aplicada, de acordo com Gil (2002), pois é voltada para aplicação de conhecimentos já existentes com a finalidade de resolver um problema específico

Conforme a figura 13 ilustra, com um diagrama,o trabalho divide-se em etapas sequenciais que tem como finalidade atingir aos objetivos.

Figura 13: Diagrama da metodologia



Fonte: Próprio autor

A primeira tarefa consistiu em uma pesquisa sobre a organização e forma de atuação de centros de operações de emergência. Para isso, foi necessário consultar informações de organizações médicas, humanitárias ou governamentais que coordenam tais equipes ou participem de operações de auxílio humanitário.

A Segunda etapa consistiu em um estudo sobre dispositivos móveis, suas plataformas de programação, e tecnologias de comunicação sem fio utilizadas nestes dispositivos. Foi necessário escolher a plataforma alvo e qual tecnologia de comunicação seria utilizada no trabalho. baseado em questões técnicas e tecnológicas, estudadas e avaliadas. A avaliação da tecnologia envolveu testes práticos de comunicação devidamente documentados.

Com a pesquisa bibliográfica concluída, o próximo passo foi definir uma proposta de solução, identificar os requisitos para esta e elaborar a especificação da aplicação. Os requisitos funcionais basearam-se nas necessidades e desafios encontrados pelas centros de operações de emergência durante suas atividades.

Após definidas as especificações e tecnologias que deveriam ser utilizadas, o próximo passo caracterizou pela implementação da análise, projeto e desenvolvimento da aplicação que atenda os requisitos do modelo proposto. O aplicativo deveria ser implementado por etapas e dividido em módulos para que a validação funcional e os testes fossem executados à medida que a implementação estava sendo realizada. Para os testes, foi estabelecida uma métrica para a medição de desempenho.

A etapa final foi de testes e simulações, onde verificou-se se os requisitos propostos estavam corretamente implementados e avaliou-se o desempenho dos requisitos não funcionais. Ao final foi feita uma reflexão sobre os resultados dos testes comparativos e, assim, chegar a uma conclusão final a respeito do projeto desenvolvido. Com base nos resultados, foi possível propor melhorias para projetos futuros.

3.1 Projeto da ferramenta

Para o desenvolvimento do aplicativo inicialmente foi necessário definir os requisitos de *software* que de acordo com Sommerville (2007) são uma descrição de serviços fornecidos pelo sistema e suas restrições operacionais. São divididos em requisitos funcionais e não funcionais. Os primeiros descrevem o que o sistema deve fazer, detalhando entradas, saídas e exceções. Já os requisitos não funcionais são restrições sobre os serviços oferecidos pelo sistema.

3.1.1 Descrição da ferramenta

A aplicação consiste em uma ferramenta de *chat* utilizando o *Wi-Fi Direct* para realizar a conexão direta entre os dispositivos. O aplicativo deve possibilitar que usuários busquem por dispositivos no seu raio de alcance e estabeleçam uma conexão. Após a conexão ser estabelecida, o usuário poderá iniciar uma conversa com todos aqueles dispositivos conectados na rede ou somente com um grupo selecionado de usuários. É permitido que mensagens de texto sejam trocadas entre os dispositivos conectados.

Além de mensagens de texto o aplicativo poderá enviar uma série de mensagens padronizadas baseadas no código internacional Q. Para cada mensagem, há um botão diferente na tela de *chat* que ao ser pressionado permite a configuração da mensagem escolhida. As mensagens presentes no aplicativo são as seguintes:

1. QAP, utilizado para perguntar ou informar se o transmissor esta na escuta no momento;
2. QAM, utilizado para perguntar ou informar acerca das condições meterológicas;
3. QRA, utilizado para perguntar ou informar se o usuário esta ocupado ou não;

4. QRL, utlizado para perguntar ou informar o nome de quem esta operando o dispositivo;
5. QRM, utilizado para perguntar ou informar sobre as condiçoes da chamada, neste caso, o *status* da conexão.
6. QRU, utilizado para perguntar ou informar se há algo para enviar para o usuário;
7. QRV, utilizado para perguntar ou informar se o usuário esta preparado para algum evento;
8. QRX, utilizado para perguntar ou informar quando o usuário chamará novamente;
9. QTW, utilizado para perguntar ou informar acerca das condições dos sobreviventes;
10. QTZ, utilizado para perguntar ou informar se o usuário segue nas buscas;
11. QUS, utilizado para perguntar ou informar se o usuário achou os sobreviventes e qual a localização dos mesmos.

Todos os botões ao serem pressionados levam para telas diferentes, onde o usuário irá responder as informações solicitadas e assim retornar para a tela de chat com a mensagem formatada corretamente. Essa mensagem é transmitida para todos os destinatários que poderão responder quando desejarem.

O aplicativo também permite o compartilhamento de localização por GPS, e visualização da posição de todos os dispositivos por meio de um radar que pode ser acessado por um botão. Todas as mensagens recebidas e enviadas são salvas em um arquivo de texto contendo a data e hora de cada mensagem.

3.1.2 Requisitos funcionais

O objetivo do aplicativo será servir de ferramenta de comunicação para pessoas em um ambiente de desastre e para tal deverá prover as seguintes funcionalidades:

- Criar uma rede ponto a ponto entre dois ou mais dispositivos
- Enviar mensagens de texto para outro usuário, um grupo de usuários baseado em suas ocupações ou para todos os usuários.
- Compartilhar e visualizar a localização com base no GPS
- Enviar imagens e gravações de voz para outros usuários.
- Enviar mensagens para outros usuários utilizando o código Q como base para um conjunto de mensagens pré-definidas sem que seja preciso digitar um texto completo.
- Visualizar e atualizar o status da situação de desastre com informações relevantes como quantidade de pessoas resgatadas, quantidade de vítimas fatais, número de vítimas necessitando de auxílio médico, quantidade de equipamentos e mantimentos disponíveis, profissionais ocupados e ociosos.

3.1.3 Requisitos não funcionais

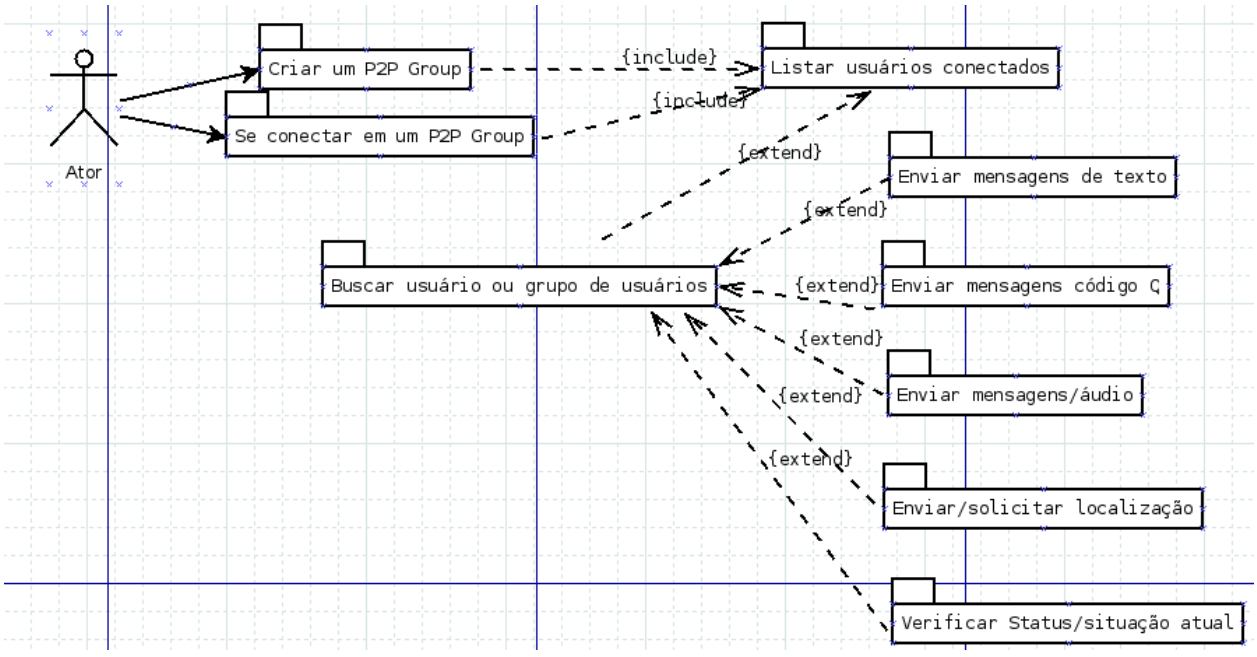
Para atender aos requisitos funcionais do sistema, pode-se elencar alguns requisitos não funcionais:

- Minimizar consumo de energia: o aplicativo deverá considerar a capacidade limitada de bateria dos dispositivos móveis e com isso evitar o desperdício de energia;
- Minimizar atrasos: o aplicativo deverá transmitir as mensagens rapidamente pela rede, pois em um ambiente de desastre;
- Interface simplificada: o aplicativo deverá possuir uma interface intuitiva e simples, que seja de fácil compreensão;
- Sistema operacional popular: o aplicativo deve funcionar em uma plataforma popular de dispositivos móveis, permitindo assim que o mesmo aplicativo possa ser executado em vários dispositivos diferentes sem a necessidade de adaptação de código.

3.1.4 Modelagem da aplicação

Com base nos requisitos da aplicação pode-se iniciar a modelagem da aplicação com um diagrama de casos de uso que apresenta uma visão simplificada da interação com o aplicativo, conforme figura 14

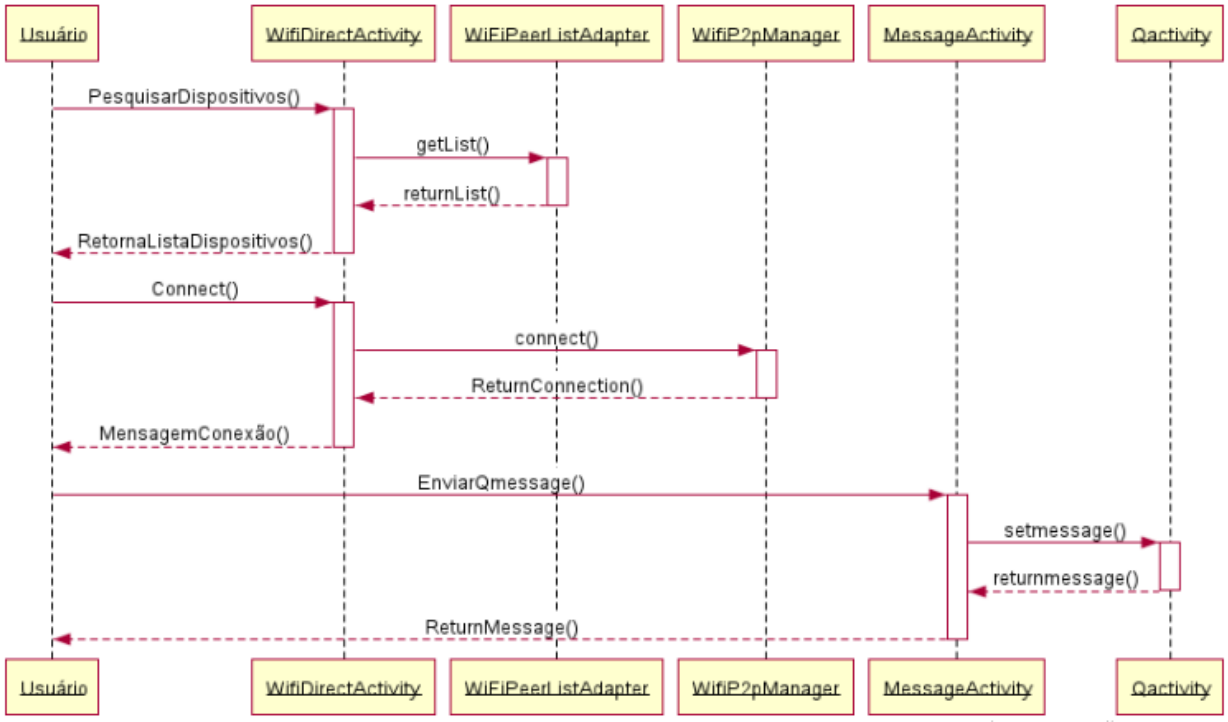
Figura 14: Casos de uso do aplicativo



Fonte: Próprio autor

Para ilustrar o processo de conexão e envio de mensagens temos o diagrama de sequência da figura 15.

Figura 15: Diagrama de sequencia do aplicativo



Fonte: Próprio autor

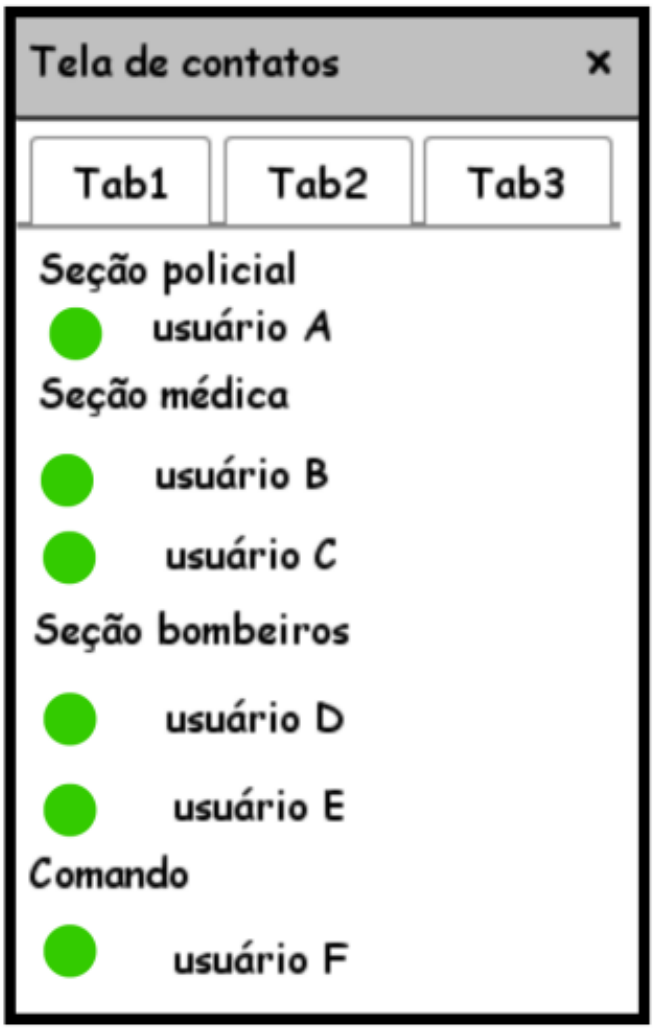
Um protótipo de como ficaria o aplicativo em sua versão final é ilustrado na figura 16. Nesta ilustração pode se ver uma pequena caixa de texto onde é possível digitar mensagens para o destinatário. O resto da tela é composto por uma série de botões com mensagens pré estabelecidas extraídas do código Q utilizado na comunicação por rádio. Este conjunto de botões serve para evitar que seja necessário escrever longas mensagens de texto.

Figura 16: Tela principal do apicativo

Fonte: Próprio autor

Outro protótipo da tela ilustrando como seria o funcionamento do aplicativo é apresentada na figura 17.

Figura 17: Lista de contatos do apicativo



Fonte: Próprio autor

Nesta tela, pode-se visualizar todos os usuários conectados na rede naquele momento classificados em grupos de acordo com suas funções.

3.1.5 Plataforma de desenvolvimento

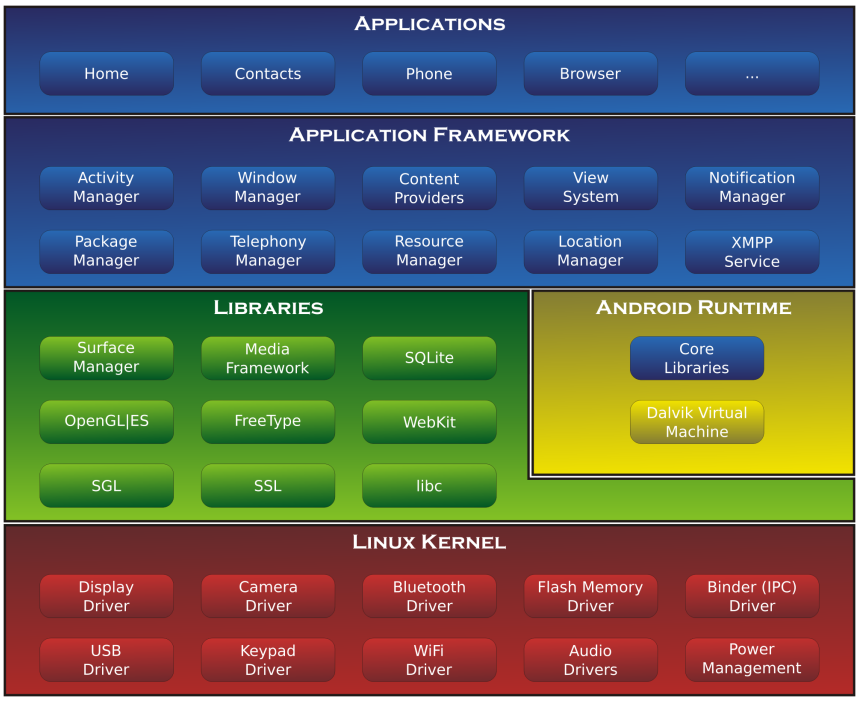
Dentre as várias platformas para dispositivos móveis disponíveis no mercado, esta pesquisa adotou o Android como solução, visto sua popularidade, que domina 68,60% do mercado de dispositivos móveis atualmente, segundo estatísticas do Netmarketshare (2016).

De acordo com PEREIRA e SILVA (2009), o Android é uma plataforma para tecnologia móvel completa que inclui um sistema operacional baseado no *kernel* Linux, *middleware*, aplicativos e interface de usuário. Desenvolvida pela *Open Handset Aliance* (OHA), um consórcio de mais de 40 empresas do ramo de telefonia móvel como por exemplo: fabricantes de aparelhos celulares, operadoras de telefonia, fabricantes de semicondutores, empresas de software e

de comercialização (LECHETA, 2013).

A Plataforma Android, ilustrada na figura 18 , é composta por uma série de camadas com diferentes níveis de abstração. Cada camada utiliza ferramentas da camada mais abaixo, sem a necessidade de que o desenvolvedor conheça a implementação interna da ferramenta que está sendo utilizada.

Figura 18: Plataforma Android



Fonte: PEREIRA e SILVA (2009)

A camada que está no topo é a Camada de Aplicações e todos os aplicativos são executados sobre ela pelo usuário. Esses aplicativos executam sobre uma máquina virtual, denominada *Dalvik*. Embora a programação para Android seja feita em JAVA, a máquina virtual *Dalvik* não pode ser considerada uma máquina virtual JAVA, pois ela não interpreta os *Bytecodes* JAVA,mas sim executa aquivos .dex, que utilizam *Bytecodes* diferentes da Java Virtual Machine (JVM) (BORDIN, 2010).

Aplicações normalmente são compostas por *activities*, que é uma ação específica que um usuário pode realizar dentro de um aplicativo. De acordo com Bordin (2010), essas ações podem incluir inicialização de outras atividades através de *intents*, criando o que se chama de tarefa.

Na camada *Framework*, encontram-se APIs e recursos usados pelos aplicativos. O Ciclo de vida das *activities* é controlado pelo *Activity Manager*, podendo iniciar, encerrar e organizar as mesmas por meio de pilhas e tarefas. O *Package Manager* mantém o registro de todos os aplicativos instalados no dispositivo (BORDIN, 2010), e pode dizer ao sistema quais aplicati-

vos estão sendo utilizados no momento e quais suas capacidades. *ManagerManager* é quem gerencia as apresentações de janelas e determina qual janela está ativa e qual não está, junto ao *View* System que disponibiliza todo o tratamento gráfico para a aplicação como botões, *layouts* e *frames* (LECHETA, 2013).

O *Content provider*, que é um dos componentes dessa camada *Framework*, é responsável por fazer o compartilhamento de dados entre diferentes aplicações. *Services*, que também compõe a camada em questão, é responsável pela execução de operações em segundo plano e , diferentemente das *activities*, não possuem interface de usuário. *Broadcast Recievers* são componentes que permitem que aplicações possam reagir a determinados eventos por meio de *intents*. Nessa mesma camada, também ficam as APIs que controlam *Wi-Fi*, GPS, *Bluetooth*, acelerômetro, dentre outros (LECHETA, 2013)

Outra camada da arquitetura *Android* é a Biblioteca, que contém um conjunto de bibliotecas escritas em C utilizadas pelo sistema. Elas controlam funções de mais baixo nível, o acesso ao banco de dados, renderização 3D, aceleração de hardware, renderização de fontes,etc. Essas bibliotecas estão disponíveis para a camada *Framework* que, por sua vez, é utilizada nos aplicativos.

A última camada é o *kernel* Linux, que atua como uma camada de abstração entre o hardware e o resto da pilha de software. É responsável pela gestão de memória e de processos, pela comunicação com componentes por meio de *drivers*, dentre outras finalidades. (LECHETA, 2013)

Para o desenvovimento da aplicação no Android adotou-se o *Android SDK* como plataforma de desenvolvimento. Esse ambiente contém um emulador, ferramentas utilitárias e uma API completa com todas as classes necessárias para desenvolver as aplicações. Cada versão do *Android* possui uma versão de API correspondente, então, ao se programar para *Android*, deve-se ter em mente quais os dispositivos que irão executar aquela aplicação. Uma API de nível superior tende a ser compatível com uma API de nível inferior, salvo algumas mudanças que podem descontinuar certas funcionalidades. O inverso não ocorre, então, o mais habitual é programar utilizando a última API disponível para que o aplicativo seja compatível com todas as versões *Android*.

No contexto desta solução é conveniente salentar que, desde a versão 4.0, com a API nível 14, o protocolo *Wi-Fi Direct* é suportado pelo *Android*. Desta forma, para a utilização do *Wi-Fi Direct* é necessário utilizar versões de API iguais ou superiores.

Dois pacotes presentes na API são necessários para a construção de aplicações *Wi-Fi Direct*. O Pacote android.net.wifi fornece classes para gerenciar a funcionalidade *Wi-Fi*. De acordo com Rezende (2013), esse pacote contém o necessário para adicionar, salvar, encerrar e iniciar conexões *Wi-Fi* e a capacidade de obter informações sobre a rede, como endereçamento IP, velocidade de conexão e estado, bem como, informações sobre outras redes disponíveis. O pacote android.net.wifi.p2p fornece também classes para criar conexões *Wi-Fi Direct*, procurar outras redes existentes e solicitar a participação nas mesmas.

A tabela 4, descreve algumas das funcionalidades encontradas no pacote *android.net.wifi*,

Tabela 4: Funcionalidades do pacote *android.net.wifi*

Método	Descrição
ScanResults	Provê informações sobre os pontos de acesso detectados
WifiConfiguration	Configuração da rede *Wi-Fi*, incluindo segurança
WifiConfiguration.AuthAlgorithm	Autenticação 802.11
WifiConfiguration.KeyMgmt	Autenticação de chaves
WifiConfiguration.PairwiseCipher	Autenticação de cifras pares para WPA
WifiConfiguration.*Protocol*	Autenticação de protocolo de rede
WifiConfiguration.Status	Status de conexão da rede
WifiEnterpriseConfig	Detalhes de configuração
WifiEnterpriseConfig.Eap	Método de autenticação usado
WifiEnterpriseConfig.Phase2	Método de autenticação interno
WifiInfo	Descreve o estado de uma conexão *Wi-Fi*
WifiManager.MulticastLock	Fornece API primária para gerenciar todos os aspectos da conectividade *Wi-Fi*
WifiManager.WifiLock	Permite que a aplicação mantenha o *Wi-Fi* ligado
WpsInfo	Classe que representa o WPS

Fonte: Rezende (2013)

A classe *WifiP2pManager* provê métodos para gerenciar redes *Wi-Fi Direct*, conforme listado a seguir de acordo com a tabela 5.

Tabela 5: Métodos da classe *WifiP2pManager*

Método	Descrição
initialize()	Registra a aplicação com o *Framework Wi-Fi*.
connect()	Inicia uma conexão P2P com um dispositivo usando uma configuração especificada
cancelConnect()	Cancela a conexão em andamento
requestConnectInfo()	Retorna informações de conexão do dispositivo
createGroup()	Criar um novo P2P group e se tornar o GO
removeGroup()	Remove o atual *P2P Group*
requestGroupInfo()	Retorna informações a respeito do *P2P Group*
discoverPeers()	Inicia a descoberta de dispositivos *Wi-Fi Direct*
requestPeers()	Retorna a lista de dispositivos *Wi-Fi Direct*

Fonte: ANDROID DEVELOPERS (2011)

Visto a opção pela utiliação do *Wi-Fi Direct* é preciso definir as permissões necessárias para sua utilização que são descritas no arquivo MANIFEST. CHANGE_WIFI_STATE e ACCESS_WIFI_STATE darão permissão para que a aplicação possa acessar o *Wi-Fi* e modificar seu estado. A Permissão de acesso à internet também é requerida, pois, embora o *Wi-Fi*

Direct não precise de acesso à internet, a classe *socket* necessita dessa permissão para funcionar. Conforme o código abaixo, podemos ver como o arquivo MANIFEST deve declarar as permissões que serão solicitadas pelo aplicativo, conforme ilustrado na figura 19.

Figura 19: Arquivo MANIFEST do aplicativo

```
manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.nsdchat"
    ...

    <uses-permission
        android:required="true"
        android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission
        android:required="true"
        android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission
        android:required="true"
        android:name="android.permission.INTERNET"/>
    ...
```

Fonte: adaptado de ANDROID DEVELOPERS (2011)

Para o aplicativo lidar com certas mudanças de estado assíncronas é importante criar um *Broadcast Reciever* para o mesmo. Conforme ilustrado na figura 20, o *Broadcast Reciever* irá monitorar mudanças de estado relevantes para o aplicativo.

Figura 20: Broadcast Reciever do aplicativo

```
if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
    //  Verificar se o Wi-Fi Direct esta habilitado
    //e notificar a atividade apropriada
    } else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {
            //Chamar WifiP2pManager.requestPeers() paara obter
            //a lista de de dispositivos disponíveis
    } else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {
            // Responder para novas conexões ou descobertas
    } else if (WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action)) {
            //Respondr a uma mudança de status Wi-Fi
}
```

Fonte: adaptado de ANDROID DEVELOPERS (2011)

E essas mudanças estão descritas na tabela 6.

Tabela 6: Descrição dos *intents* da classe WifiP2pManager

Intent	**Descrição**
WIFI_P2P_CONNECTION_CHANGED_ACTION	Mudança de status de conexão
WIFI_P2P_PEERS_CHANGED_ACTION	Função DiscoverPeers() é chamada
WIFI_P2P_STATE_CHANGED_ACTION	Mudança no status do *Wi-Fi*
WIFI_P2P_THIS_DEVICE_CHANGED_ACTION	Alguma mudança de status do dispositivo

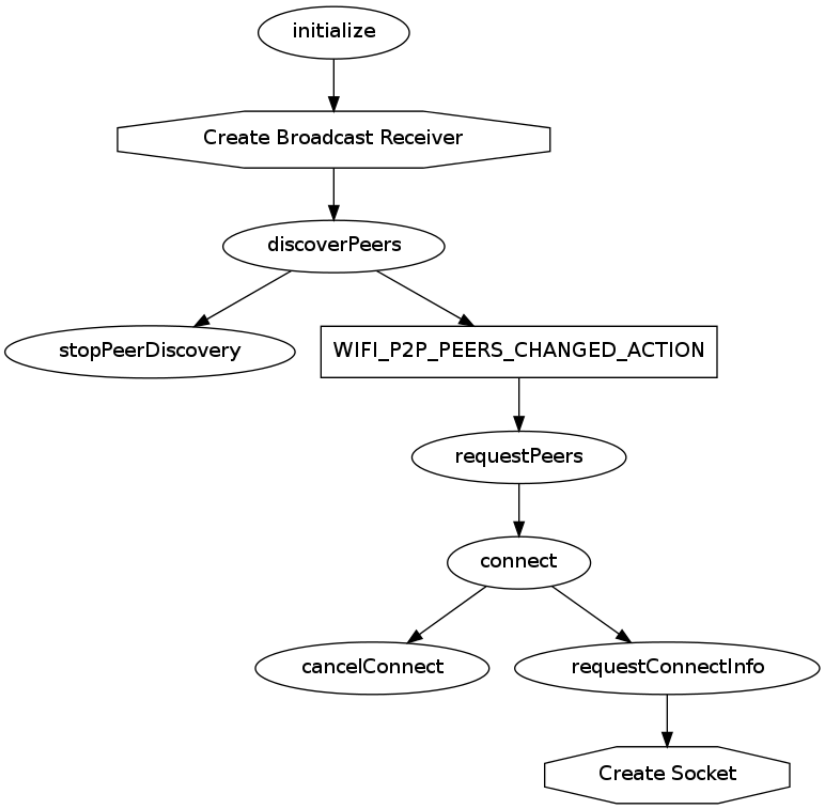
Fonte: ANDROID DEVELOPERS (2011)

A constante WifiP2pManager.WIFI_P2P_STATE_ENABLED é utilizada para verificar se o *Wi-Fi Direct* está disponível ou não. Caso seu valor seja positivo, significa que o mesmo está disponível. Após iniciada a conexão *Wi-Fi Direct*, por meio do método initialize(), existem duas opções possíveis de conexão. Pode-se chamar o método CreateGroup() para se criar um *P2P Group* somente com o dispositivo criador como membro e GO, ou iniciar a descoberta de dispositivos para se descobrir grupos já existentes ou criar um em conjunto com outro dispositivo.

Os métodos DiscoverPeers() e stopPeerDiscovery() iniciam e param a busca por dispositivos. A Função DiscoverPeers() é assíncrona e, portando, trabalha em background. Os métodos onSuccess() e onFailure() são executados para informar o sucesso ou falha da operação de descoberta de dispositivos. O Método DiscoverPeers não retorna um resultado.Para receber a lista de dispositivos encontrados, é necessário chamar o método requestPeers().

O Método Conect() é responsável por iniciar a conexão e, na execução desse método, os dois dispositivos criam um *P2P Group*. A Criação do grupo, o canal utilizado, a escolha do GO e a distribuição dos Ips por DHCP é feita de forma automática e aleatória conforme explicado no algoritmo do *Wi-Fi Direct*. Após feita a conexão, cria-se um *socket*, usando as interfaces padrão como se fosse uma conexão *Wi-Fi* habitual. A figura 21 ilustra cada etapa em um diagrama/fluxograma. O método *Connect* também é assíncrono e utiliza-se *onSuccess()* e *onFailure()* para informar se a conexão foi efetivada e executar o procedimento adequado em cada situação. O Método *requestConnectInfo()* informa os detalhes daquela.

Figura 21: Processo de conexão *Wi-Fi Direct*



Fonte: Khatko (2014)

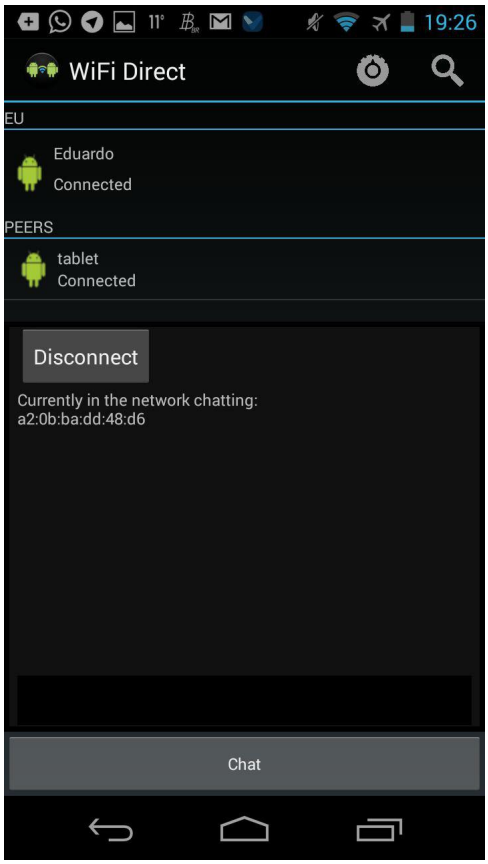
Com a conexão da rede criada, o aplicativo pode utilizar os métodos do pacote *Wi-Fi* do Android e trocar informações como se estivesse em uma rede comum.

3.2 Projeto Piloto

O Desenvolvimento do aplicativo proposto no trabalho depende da escolha de uma tecnologia adequada para a sua finalidade. O *Bluetooth* e o *Wi-Fi Direct* foram as tecnologias estudadas e inicialmente elencadas para este trabalho. Contudo, a fim de definir qual delas apresenta melhor desempenho com base nos requisitos do SCO, propôs-se a realização de um projeto piloto com o intuito de identificar a viabilidade de adoção dessas tecnologias e apontar a que melhor de adequa.

A figura 24 apresenta a tela inicial do aplicativo.

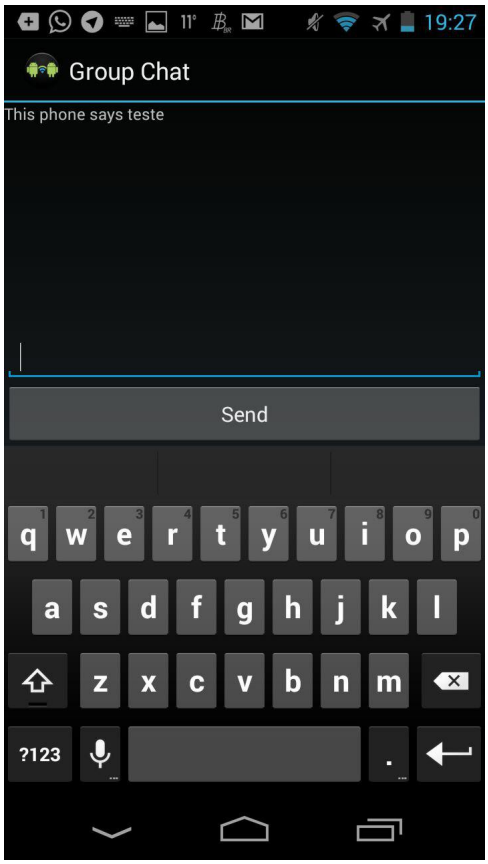
Figura 22: Tela inicial do aplicativo



Fonte: Próprio autor

Na figura 25 é apresentada a tela onde são trocadas as mensagens do aplicativo. O usuário nesta tela pode enviar e receber mensagens de texto para o outro dispositivo conectado na rede.

Figura 23: Tela de *chat* do aplicativo



Fonte: Próprio autor

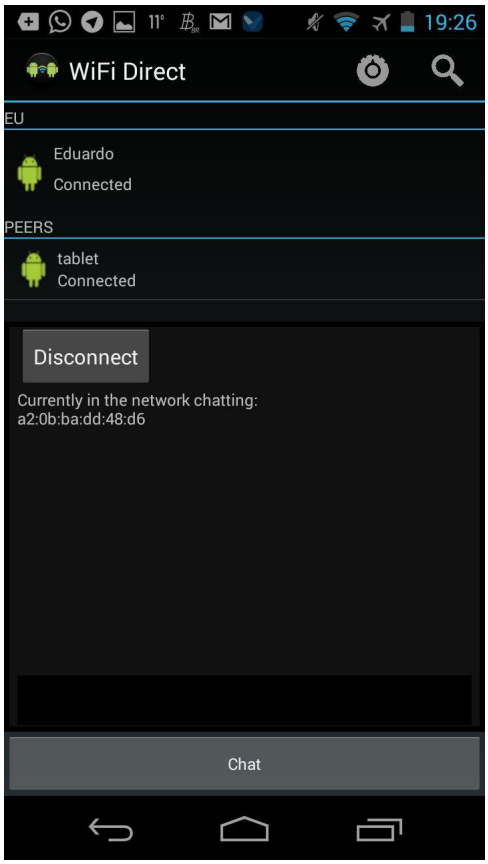
O aplicativo piloto desenvolvido é capaz enviar mensagens de texto entre dispositivos e serve como exemplo para testar a viabilidade da ferramenta. Ele não é capaz de interconectar *P2P Groups* diferentes e nem possui uma interface de usuário intuitiva que traga recursos úteis aos usuários que a utilizam em uma situação de desastre.

Os dispositivos utilizados no teste foram dois smartphones *Android*: um *Galaxy Nexus* da Samsung com *Android* 4.3 e um *Prime Plus* da LG, com *Android* 5.0.2. A escolha da plataforma *Android* para a realização do projeto se deu pelo fato do mesmo ser o sistema operacional mais utilizado para *smartphones*.

Para a comunicação *Bluetooth* foram utilizados os seguintes aplicativos: *Bluetooth Messenger*, *Ensichat*, *Gilga* e *Fire Chat*. Com exceção do primeiro, todos opensource. Para a comunicação por meio de *Wi-Fi Direct*, foi utilizado um protótipo próprio baseado nos exemplos já presentes na documentação do *Android*. Também foi utilizado o aplicativo *Hostpot Chat* que envia mensagens utilizando o SSID do ponto de acesso do *Android*.

O aplicativo piloto desenvolvido é capaz enviar mensagens de texto entre dispositivos e serve como exemplo para testar a viabilidade da ferramenta. A figura 24 apresenta a tela inicial do aplicativo.

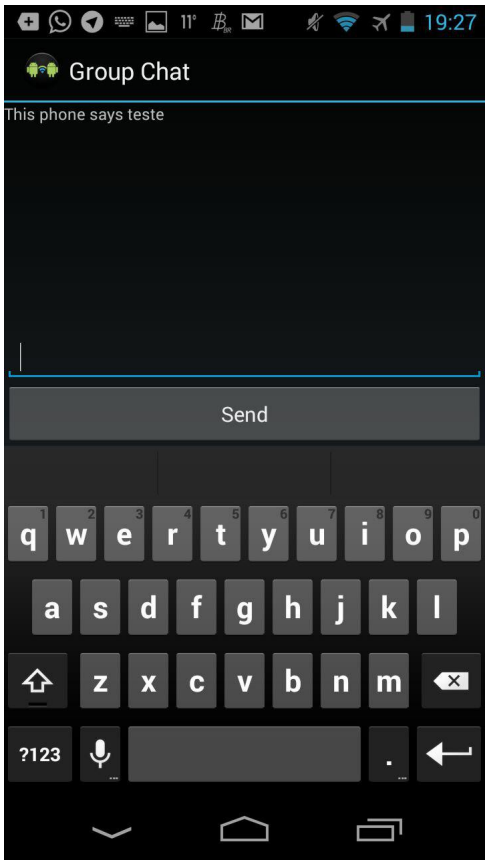
Figura 24: Tela inicial do aplicativo



Fonte: Próprio autor

Na figura 25 é apresentada a tela onde são trocadas as mensagens do aplicativo. O usuário nesta tela pode enviar e receber mensagens de texto para o outro dispositivo conectado na rede.

Figura 25: Tela de *chat* do aplicativo



Fonte: Próprio autor

Este protótipo não era capaz de interconectar *P2P Groups* diferentes e nem possuia uma interface de usuário intuitiva que trouxesse recursos úteis aos usuários que a utilizam em uma situação de desastre.

O procedimento consistiu em conectar os *smartphones*, utilizando-se um dos aplicativos disponíveis para cada tecnologia para iniciar uma troca de mensagens entre os mesmos. Com o auxílio de uma fita métrica, mediu-se a distância entre cada aparelho e, se ainda fosse possível trocar mensagens, aumentava-se a distância e repetia-se o procedimento. Também foi medida a variação da taxa de transferência de dados de cada tecnologia, utilizando o mesmo procedimento. Com o *Wi-Fi Direct* ainda foi possível medir a variação da latência e da potência do sinal com a utilização de aplicativos específicos.

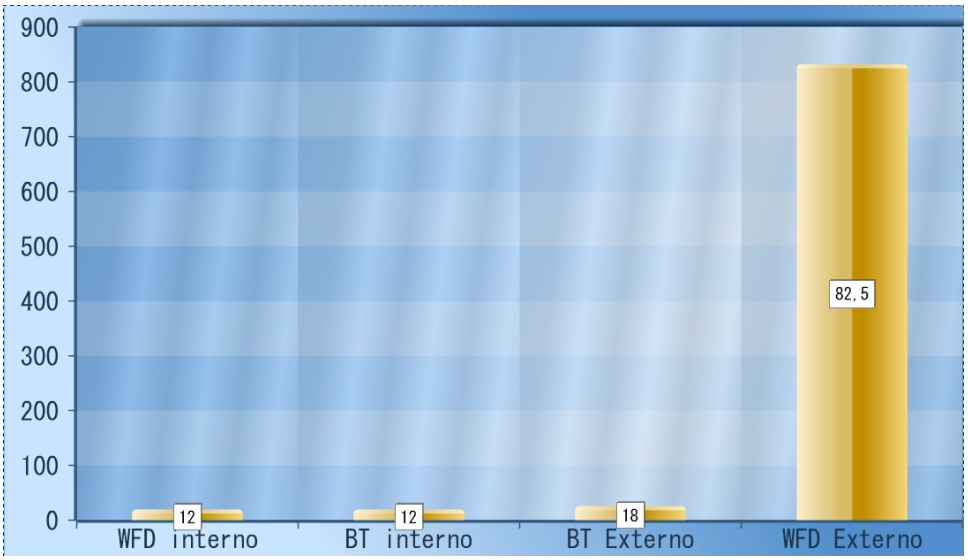
Testou-se a conexão entre os dispositivos e a troca de mensagens avaliando a distância entre os mesmos, inicialmente em um ambiente fechado, com um roteador a uma distância de aproximadamente 2 metros do emissor.

O *Bluetooth* conseguiu enviar mensagens a uma distância de 12 metros entre cada aparelho; já o *Wi-Fi Direct* foi capaz de enviá-las de até no máximo 7,5 metros. Em outro momento, o teste com o *Wi-Fi Direct* foi reproduzido novamente e chegou-se igualmente a 12 metros de

distância. Entre os motivos possíveis para essa variação no alcance da conexão está o fato de que o *Wi-Fi Direct* escolhe um canal aleatório para criar a rede e, dependendo de qual for o canal, pode ser que algumas vezes sofra interferência de outras redes próximas.

Os testes foram repetidos novamente, porém em uma área externa. Nesse caso, o *Bluetooth* obteve êxito na troca de mensagens de até 18 metros de distância entre os dispositivos; por outro lado, o *Wi-Fi Direct* conseguiu bem mais, chegando até 82,5 metros. A troca de mensagens por meio do SSID também foi testada e chegou a mesma distância do *Wi-Fi Direct*. A figura 26 ilustra essa comparação entre as distâncias máximas obtidas.

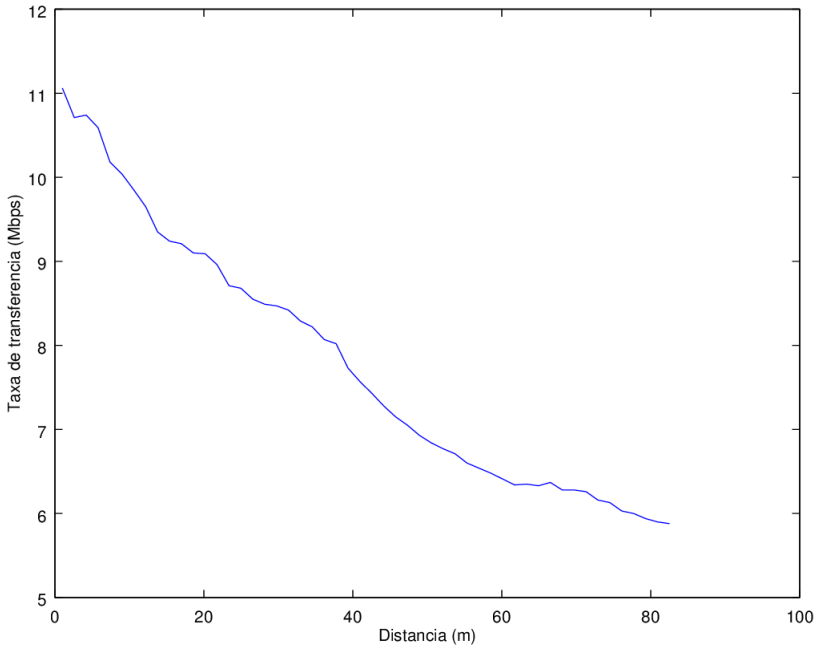
Figura 26: Gráfico ilustrando o alcance máximo de cada tecnologia em ambientes externos e internos



Fonte: Próprio autor

O Próximo passo foi medir as velocidades e, para isso, utilizou-se o mesmo procedimento já descrito anteriormente. Os Gráficos das figuras 27 e 28 ilustram a queda da velocidade de conexão ao longo do aumento da distância.

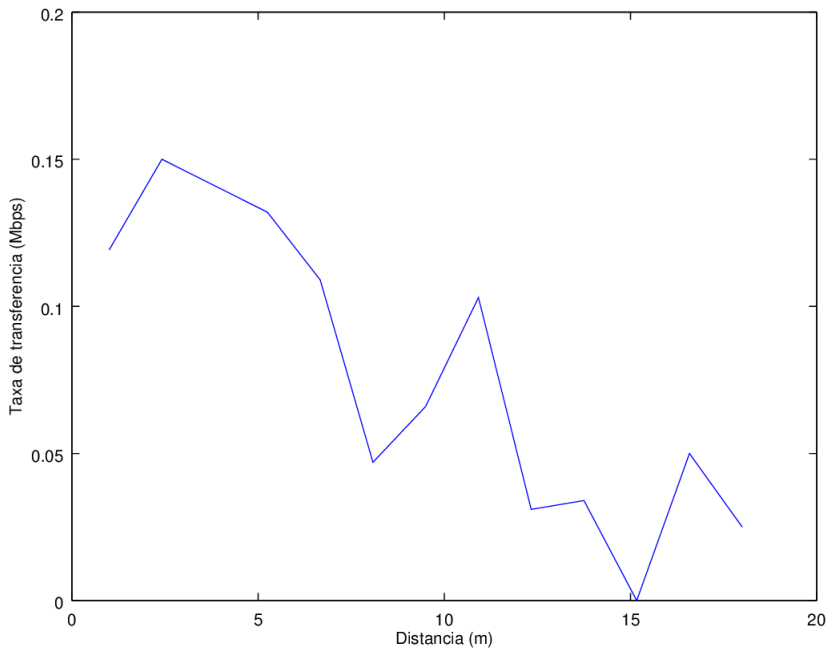
Figura 27: Gráfico da taxa de transferência de dados do *Wi-Fi Direct* pela distância



Fonte: Próprio autor

O *Wi-Fi Direct* atingiu uma velocidade máxima de 11Mbps e uma velocidade média de 7,8 Mbps, Em contrapartida, o *Bluetooth* atingiu a velocidade máxima de 1,5Mbps e a velocidade média de 0,77Mbps.

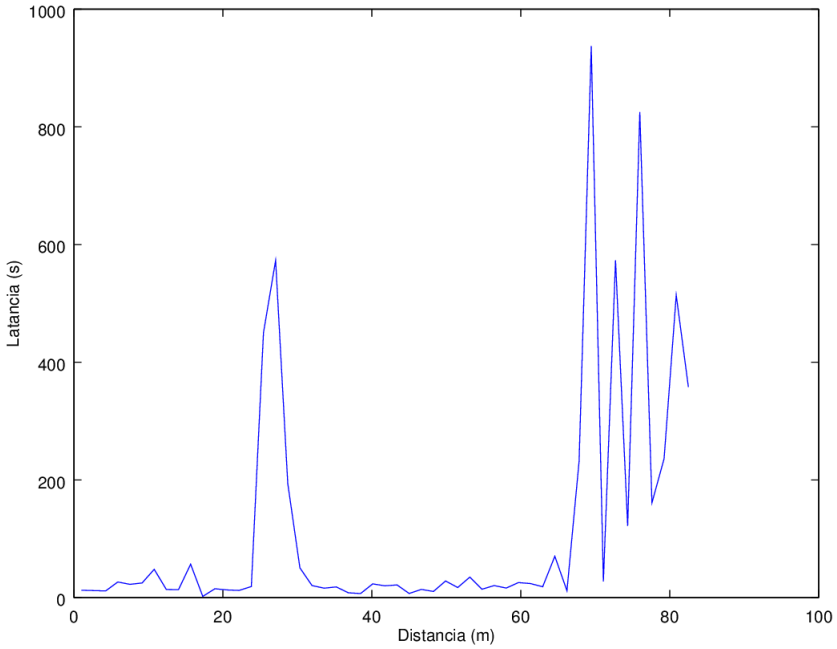
Figura 28: Gráfico da taxa de transferência de dados do *Wi-Fi* Bluetooth pela distância



Fonte: Próprio autor

Para medir a latência do *Wi-Fi*, utilizou-se a ferramenta *ping* que já está inclusa no *Android* e, para medir o nível de sinal, utilizou-se o aplicativo *Wi-Fi Analyzer*. A figura 29 mostra a variação da latência ao longo da distância.

Figura 29: Gráfico da latência do *Wi-Fi Direct* pela distância



Fonte: Próprio autor

Não foi encontrado, por outro lado, nenhum outro aplicativo capaz de fazer essa mesma medição em uma conexão *Bluetooth*, porém pode-se observar, por meio desta, que há uma variação da latência pequena e, portando, desprezível. No *Wi-Fi Direct* a latência só começa a aumentar em distâncias longas mantendo-se na média de 117ms e chegando ao pico de 937ms no final do percurso. A figura 30 mostra a variação do sinal conforme a distância.

Figura 30: Gráfico ilustrando a variação da potência do sinal da rede conforme o aumento da distância

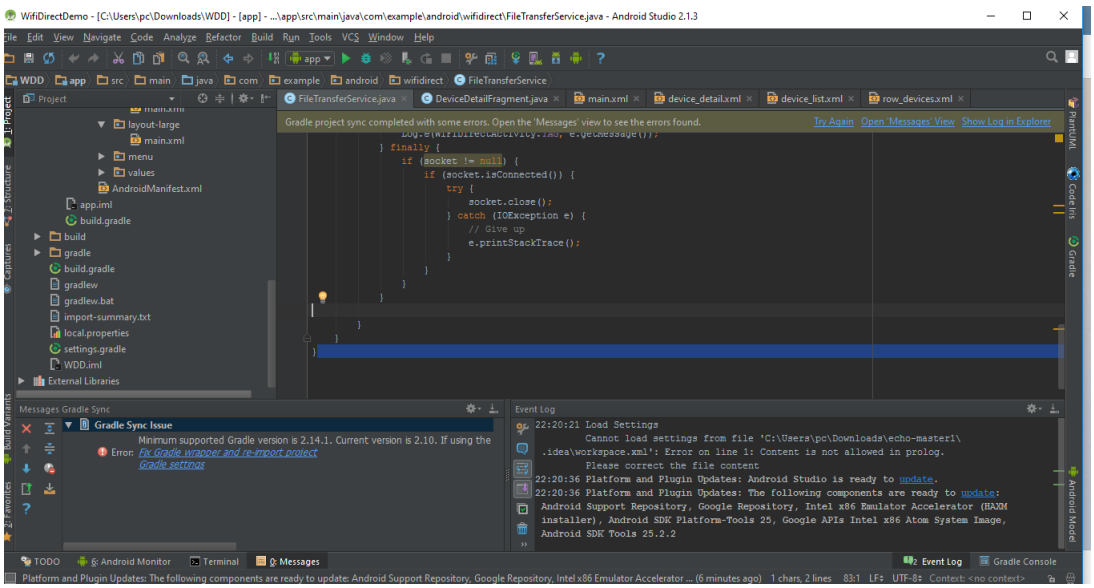
Fonte: Próprio autor

Pelos resultados obtidos, concluiu-se que a tecnologia *Wi-Fi Direct* é a que melhor atende aos requisitos do trabalho proposto pois possui um alcance consideravelmente maior que o *Bluetooth* e também uma alta taxa de transferência de dados. Para que a aplicação tenha sucesso no auxílio de equipes de resgate durante uma operação, é essencial que o alcance da ferramenta não esteja limitado a poucos metros.

4 DESENVOLVIMENTO DO APLICATIVO

Para o desenvolvimento do aplicativo, foi utilizada a IDE que provê um ambiente completo para programação *Android*, sendo a ferramenta oficial do *Google*. Com o *Android Studio*, é possível programar aplicacões completas, testar no emulador embutido, transferir aplicações em desenvolvimento para um dispositivo físico e depurar pela USB. O ambiente vem com um SDK completo, permite baixar automaticamente dependências e novas versões do SDK, se for necessário. A imagem 31 mostra a tela principal do *software* em execução.

Figura 31: Android Studio em execução



Fonte: Próprio autor

Além disso, a IDE permite escrever os códigos *Java* e XML do *Android*, possuindo diversos recursos de autocomplementação de comandos, insere *imports* automaticamente e lista métodos que podem ser executados sem que o usuário precise consultar toda a documentação de cada classe. Todos esses recursos facilitam o desenvolvimento ao agilizar o processo de escrita do código. O emulador permite que, mesmo sem um dispositivo, seja possível executar as aplicações que estão sendo desenvolvidas. É possível acompanhar o ambiente de depuração para encontrar erros e informações relevantes de *log*.

Inicialmente a aplicação começou a ser desenvolvida e testada no emulador, porém, no notebook utilizado, o emulador consumia muita memória e muito processamento, deixando o ambiente lento de forma a prejudicar a produtividade. Decidiu-se, então, que seria desejável utilizar um dispositivo real para testar a aplicação. O dispositivo escolhido foi um *Galaxy Nexus*, da Samsung. Um dispositivo *Android* que vem com um sistema *Android* puro, isto é, sem modificações de fabricantes e operadoras, tornando-se, com isso, o melhor ambiente para

se testar o desenvolvimento de uma aplicação. Outros aparelhos foram utilizados, durante os testes, que serão melhor descritos posteriormente.

A aplicação consiste em um software de comunicação que utiliza a tecnologia *Wi-Fi Direct* para prover comunicação entre dispositivos sem a necessidade de uma conexão com a internet. Para isso, a aplicação faz uso dos pacotes *adroid.net.wifi.p2p* e *android.net.wifi* para gerenciar as conexões, conforme foi explicado no detalhamento do protótipo.

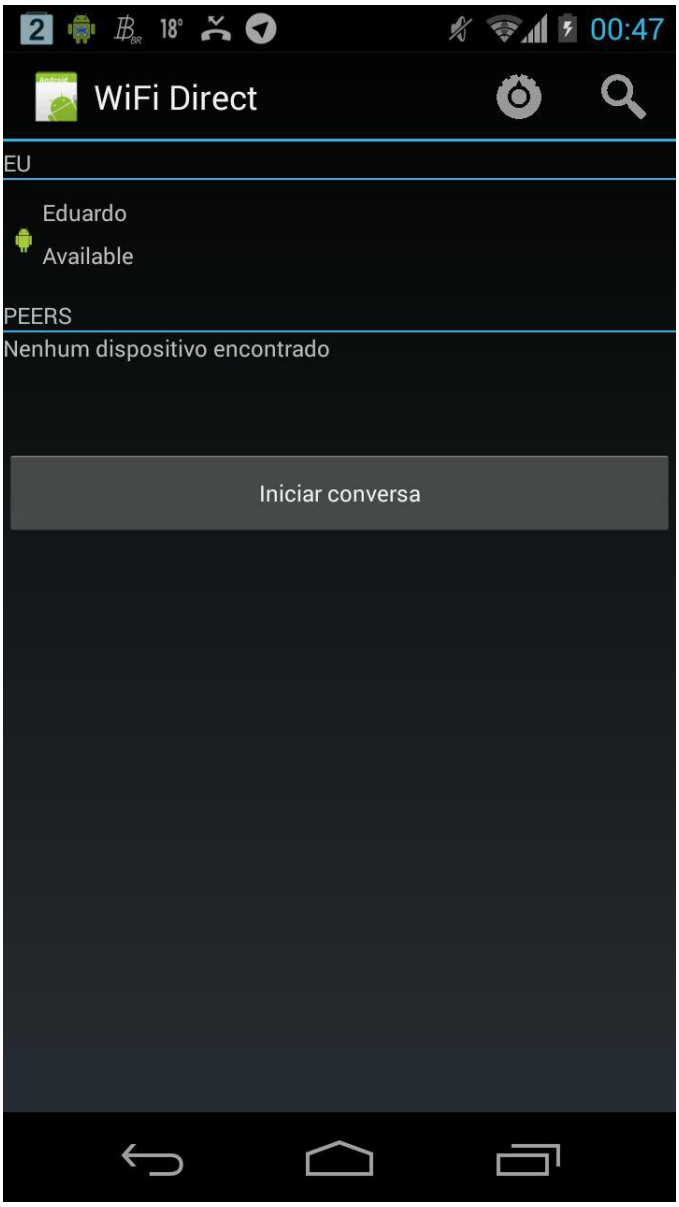
O desenvolvimento ocorreu de forma modular com as funcionalidades sendo implementadas uma a uma. Inicialmente, para o projeto piloto, foi desenvolvidas as funcionalidades de conexão e troca de mensagens, que se encontram nas *activities WifiDirectActivity* e *MessageActivity* respectivamente.

Após, ocorreu a implementação de cada uma das *activities* que implementam as mensagens Código Q. Durante a implementação da *QRLActity* se inseriu o banco de dados SQL que seria posteriormente trocado por uma estrutura de *hashmaps*, para armazenamento de informações do usuário e pela função *writetofile* para o armazenamento de *logs* de conversa.

As ultimas funcionalidades desenvolvidas foram as *activities* que lidam com o radar GPS e com o encerramento de aplicações de terceiros. Ambas, utilizando classes já desenvolvidas e disponibilizadas em código aberto.

A *Activity* WifiDirectActivity, utilizando os métodos das classes WifiP2pConfig, WifiP2pDevice e WifiP2pManager, é responsável por descobrir e conectar dispositvos *Wi-Fi Direct* com a finalidade de criar a rede *ad hoc* pela qual a troca de informação entre os dispositivos irá ocorrer. Essa *Activity* utiliza outras classes para seu funcionamento, dentre elas, a classe *DeviceListFragment*, responsável por listar os dispositivos *Wi-Fi Direct* que estejam dentro do raio de alcance e exibi-los para o usuário. Outra classe responsável por comandar as interações entre dispositivos *Wi-Fi Direct* é a classe *DeviceDetailFragment*, que é capaz de configurar as conexões de rede, distribuindo endereços e estabelecendo a segurança da conexão por meio da criptografia WPA2 e elegendo o *Group Owner*. A Figura 32 demonstra uma tela do software , onde essa *activity* esta ativa

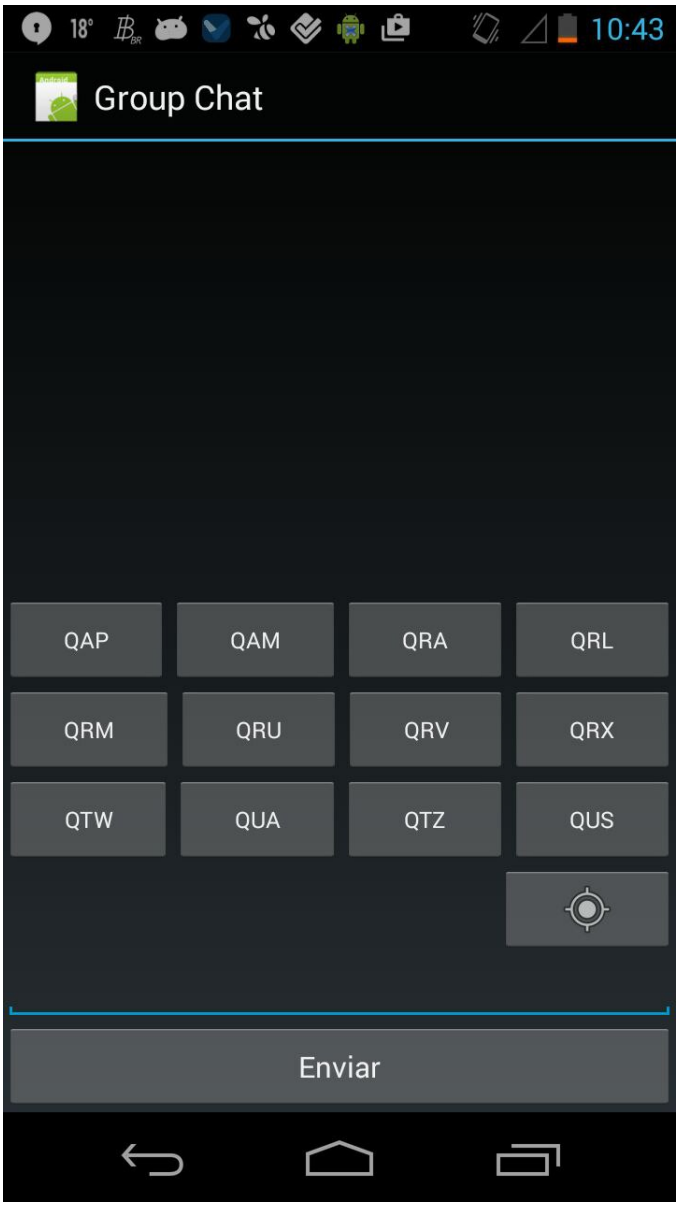
Figura 32: WifiDirectActivity em execução



Fonte: Próprio autor

Após a conclusão da conexão entre os dispositivos, o próximo passo é iniciar a troca de mensagens entre os usuários do aplicativo. A *activity Message Activity* possui a responsabilidade de prover a interface que permite essa troca de mensagens. Essa interface é composta por uma *Textview*, onde serão exibidas as mensagens, uma caixa de texto para inserir texto e uma série de botões baseados no código internacional Q, conforme podemos ver ilustrado na figura 33.

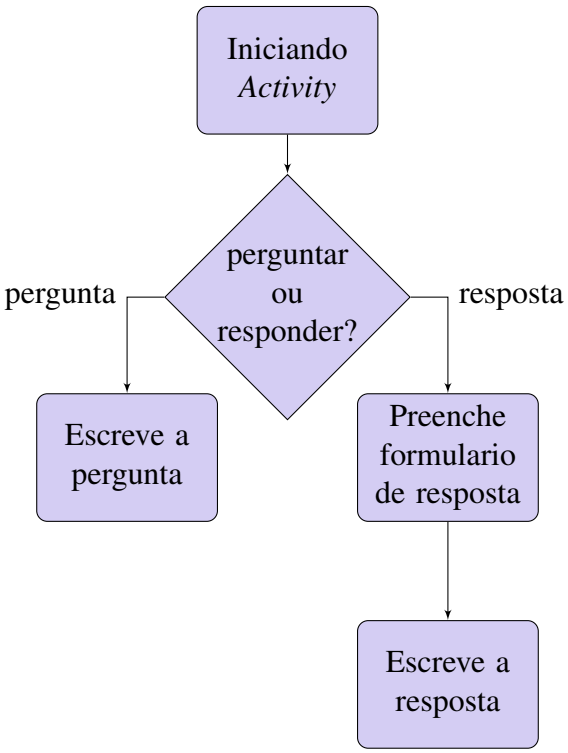
Figura 33: interface da *MessageActivity*



Fonte: Próprio autor

Cada um desses botões inicia uma nova *activity* que será responsável por gerar uma mensagem. Cada mensagem no código Q pode ser interpretada como uma pergunta ou uma resposta e, portanto, é responsabilidade da *activity* específica de cada botão coletar dados a respeito da intenção do usuário e formatar a mesma em uma sucinta mensagem de texto, que será enviada para a conversa e exibida na *Textview* já mencionada. O diagrama da imagem 34 detalha, em termos gerais, como funciona o processo de envio de uma mensagem código Q pelo aplicativo.

Figura 34: Fluxograma do envio da mensagem Q



Fonte: Próprio autor

Todas as atividades Q são chamadas e tratadas pelo *MessengerActivity*, para isso, elas são chamadas com o *startActivityForResult*, o que significa que se espera que elas retornem algum valor para quem as chamou. Na figura 35, é demonstrado algumas chamadas de *Activities*.

Figura 35: Chamadas das *Activities*

```
final Button buttonQRL = (Button) findViewById(R.id.button4);

buttonQRL.setOnClickListener((v) -> {
        Intent i = new Intent(getApplicationContext(), QrlActivity.class);
        startActivityForResult(i, 1);

});

final Button buttonQRX = (Button) findViewById(R.id.button8);

buttonQRX.setOnClickListener((v) -> {
        Intent i = new Intent(getApplicationContext(), QrxActivity.class);
        startActivityForResult(i, 1);

});

final Button buttonQAM = (Button) findViewById(R.id.button2);

buttonQAM.setOnClickListener((v) -> {
        Intent i = new Intent(getApplicationContext(), QamActivity.class);
        startActivityForResult(i, 1);
```

Fonte: Próprio autor

Quando a *Activity* que fora chamada encerra sua execução, ela retorna valores que precisam se tratados pelo método *onActivityResult*. Esse método é responsável por extrair esses valores retornados e processá-los. A figura 36 ilustra parte desse método.

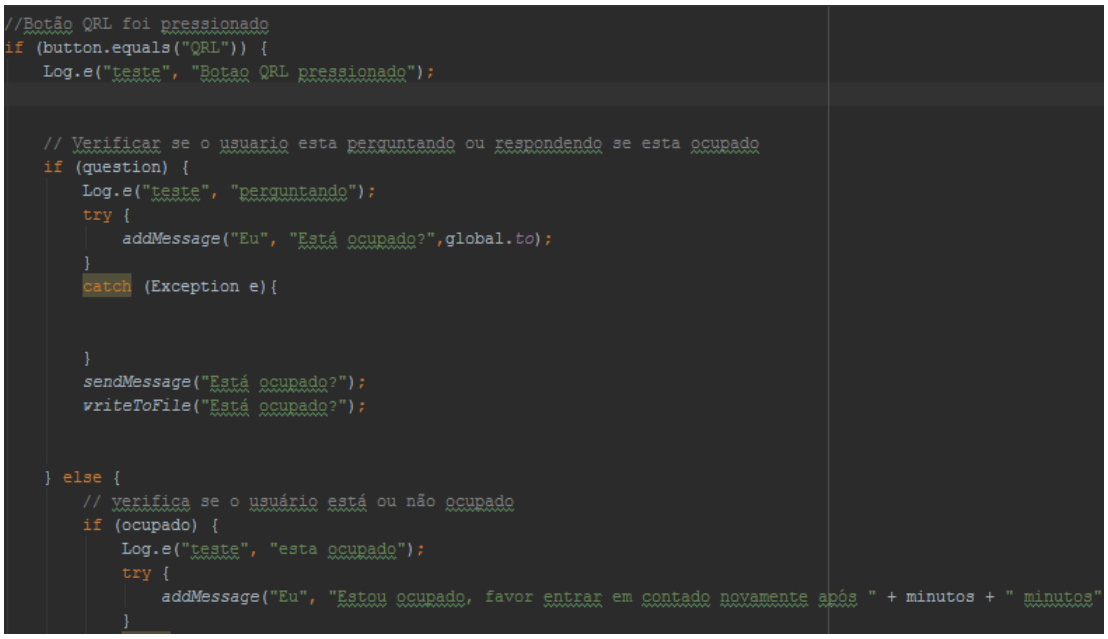
Figura 36: Código do métdo *onActivityResult*

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent i) {

    super.onActivityResult(requestCode, resultCode, i);

    if (requestCode == 1) {
        if (resultCode == RESULT_OK) {

            Intent intent = i;


            String button = intent.getStringExtra("buttonpressed");
            String texto = intent.getStringExtra("message");
            Boolean question = intent.getBooleanExtra("?", true);
            Boolean ocupado = intent.getBooleanExtra("statestring", false);
            int minutos = intent.getIntExtra("minutes", 0);
            int horas = intent.getIntExtra("hour", 0);
            String setmessage = intent.getStringExtra("setmessage");


            String setprofile = intent.getStringExtra("profiles");
```

Fonte: Próprio autor

Inicialmente,o método extrai os valores que retornaram da *activity* encerrada com o *getStringExtra* e, assim, ele avalia os valores extraídos para concluir qual texto deve ser processado. Para isso, ele analisará qual o botão que foi pressionado,indicando qual o tipo de mensagem que deseja formular que está armazenado na variável *button*. E, com o intuito de conhecer a intenção do usuário, ou seja, se era de seu interesse formular uma pergunta ou se ele estava respondendo, ele avalia o valor armazenado na variável *question*, dentre os outros valores. A figura 37 ilustra o trecho de código onde o botão QRL foi pressionado e precisa ser tratado.

Figura 37: *onActivityResult* tratando comando QRL ao botão ser pressionado



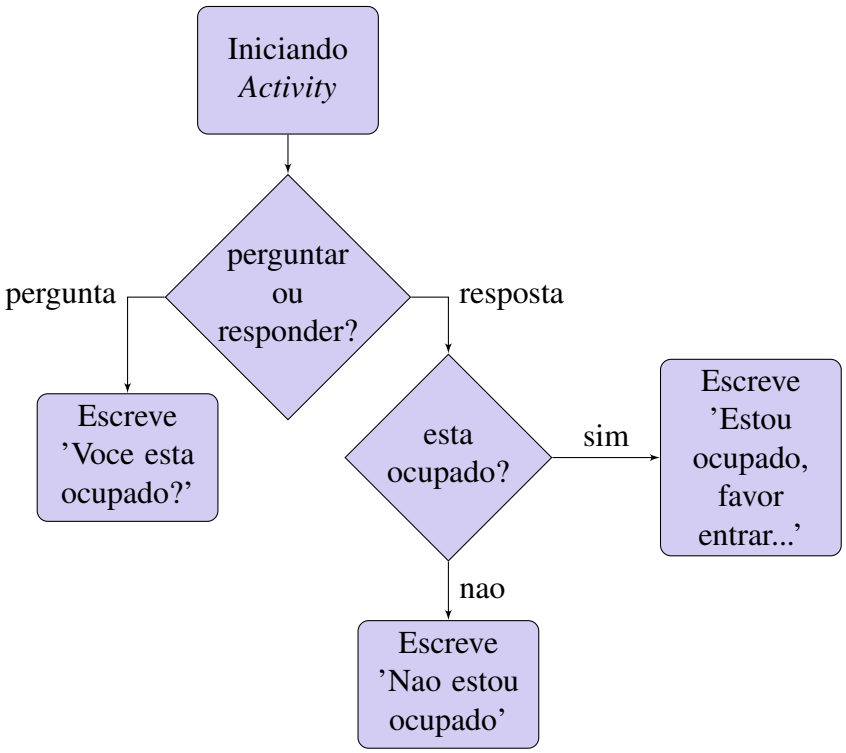
```
//Botão QRL foi pressionado
if (button.equals("QRL")) {
    Log.e("teste", "Botao QRL pressionado");


    // Verificar se o usuario esta perguntando ou respondendo se esta ocupado
    if (question) {
        Log.e("teste", "perguntando");
        try {
            addMessage("Eu", "Está ocupado?",global.to);
        }
        catch (Exception e){


        }
        sendMessage("Está ocupado?");
        vriteToFile("Está ocupado?");


    } else {
        // verifica se o usuário está ou não ocupado
        if (ocupado) {
            Log.e("teste", "esta ocupado");
            try {
                addMessage("Eu", "Estou ocupado, favor entrar em contado novamente após " + minutos + " minutos"
            }
```

Fonte: Próprio autor

A *QRLActivity* é responsável por tratar do código QRL. Esse código é utilizado para perguntar e informar se um determinado usuário está ocupado e se pode atender no momento. Para implementar essa funcionalidade, a interface desenvolvida conta com dois botões, um chamado *pergunta* e outro chamado *resposta*, um *Switch* e um *NumberPicker*.

O usuário que deseja descobrir se a outra parte está disponível ou ocupada deve pressionar o botão de pergunta. Fazendo isso, a *Qrlactivity* retorna para a *MessageActivity* a informação e, então, formata a pergunta "Você está ocupado?"e envia para o destinatário. No caso de o usuário desejar informar se está ocupado ou não, ele o fará alterando o valor do *Switch* para ocupado ou desocupado e selecionando em quanto tempo deve ser contactado novamente por meio do *NumberPicker*. Da mesma forma, esses valores retornam ao *MessageActivity*, que irá formatar em texto as mensagens "Não estou ocupado"e "Estou ocupado,respectivamente, "Favor entrar em contado novamente após x minutos". O diagrama de atividades da figura 38 ilustra esse procedimento.

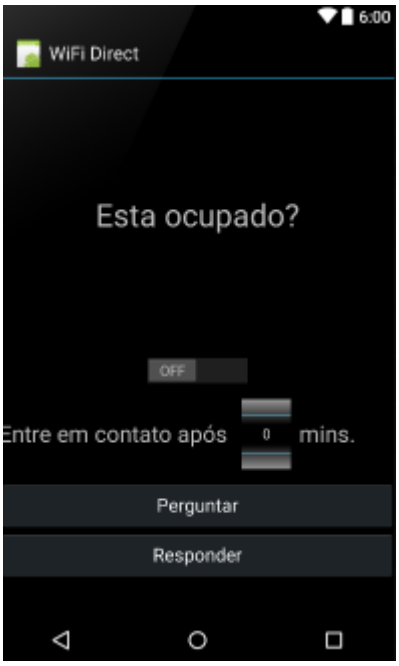
Figura 38: Fluxograma do envio da mensagem QRL



Fonte: Próprio autor

A imagem 39 ilustra a interface dessa *activity*, onde se pode observar as estruturas utilizadas para se obter as respostas que serão base da mensagem.

Figura 39: Interface da *QRLactivity*



Fonte: Próprio autor

O código dessa *activity* é extremamente simples, limitando-se a extrair os valores inseridos pelo usuário no momento que um dos botões for pressionado e, assim, encerrar suas atividades, retornando os valores para a função que a chamou. A figura 40 apresenta parte desse código.

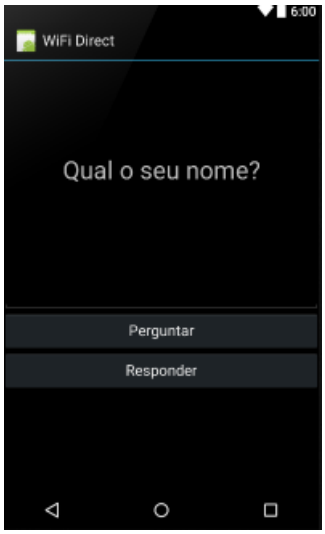
Figura 40: Trecho de código da *QRLActivity*

```
buttonans.setOnClickListener((v) → {
        boolean statestring = onoff.isChecked();
        int minutes = num.getValue();

        Intent i = new Intent(getApplicationContext(), MessageActivity.class);
        i.putExtra("minutes", minutes);
        i.putExtra("statestring", statestring);
        i.putExtra("buttonpressed", "QRL");
        i.putExtra("?", false);

        setResult(Activity.RESULT_OK,i);
        finish();


});

buttonask.setOnClickListener((v) → {


        Intent i = new Intent(getApplicationContext(), MessageActivity.class);

        i.putExtra("?", true);
        i.putExtra("buttonpressed", "QRL");

        setResult(Activity.RESULT_OK,i);
        finish();
```

Fonte: Próprio autor

A *QRAActivity* é responsável por tratar do código QRA, que é utilizado para perguntar e informar o nome do usuário interlocutor. Essa funcionalidade é implementada por meio de uma interface onde o usuário pode digitar seu nome e clicar em um dos dois botões disponíveis, um de pergunta e outro de resposta. Ao formular uma pergunta, é enviada a mensagem de "Qual seu nome?"para a outra parte que, por sua vez, pode responder inserindo seu nome na caixa de texto e pressionando o botão de resposta. Ao responder, o remetente envia a mensagem "Meu nome é", seguido do nome digitado para a *Activity* de conversa. A figura 41 ílustra a interface dessa *activity*.

Figura 41: Interface da *QRAactivity*



Fonte: Próprio autor

A imagem 42 apresenta parte do código dessa *activity*. A estrutura de todas as *activities* é muito similar. Nesse caso, também, há dois botões, o de perguntar e o de responder e, para cada botão pressionado, há o retorno de dados.

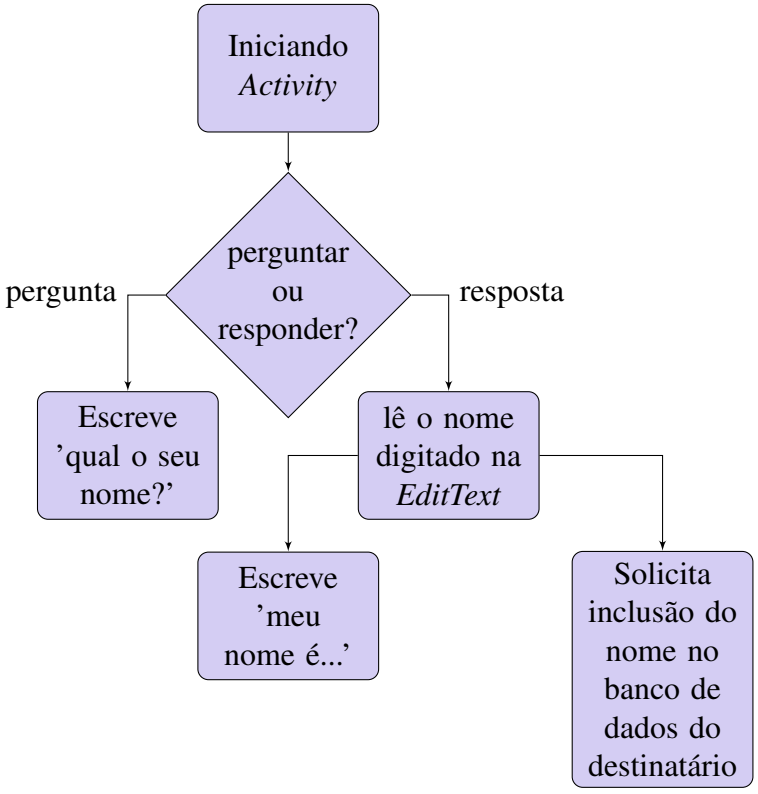
Figura 42: Trecho de código da *QRAactivity*

```
buttonq.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        sendreturn("Qual o seu nome?");

    }


});

buttonans.setOnClickListener((v) -> {
        String namestring = name.getText().toString();
        String message = "Meu nome é "+namestring;
        String setmessage = "/setname "+namestring;


        //resultado = crud.insert();


        sendreturn(message,setmessage);

});
```

Fonte: Próprio autor

A fim de armazenar os dados dos usuário adotou-se uma estrutura de banco de dados, contendo a relação de endereços *mac* dos dispositivos da rede e seus nomes de usuário, caso estes tenham sido informados. Para implementar o banco de dados, foram utilizadas as classes *SQLiteOpenHelper* e *SQLiteDatabase* que permitem a criação e manipulação de bancos da dados *SQlite*, respectivamente. Foi criada a classe *CreateDB* que,utilizando métodos das duas anteriores, é responsável por realizar a criação da tabela que conterá os dados armazenados. A inserção, atualização e remoção de dados do banco é responsabilidade da classe *DBcontroller*. Essa classe formata as *queries* SQL a partir de valores de entradas recebidos por parâmetro. Sempre que um usuário informa seu nome na conversa, o *DBcontroller* é chamado para inserir essa identificação no banco. Para isso, ele recebe o nome escolhido e o endereço *mac* do remetente e os relaciona. O fluxograma 43 descreve com detalhes esse procedimento.

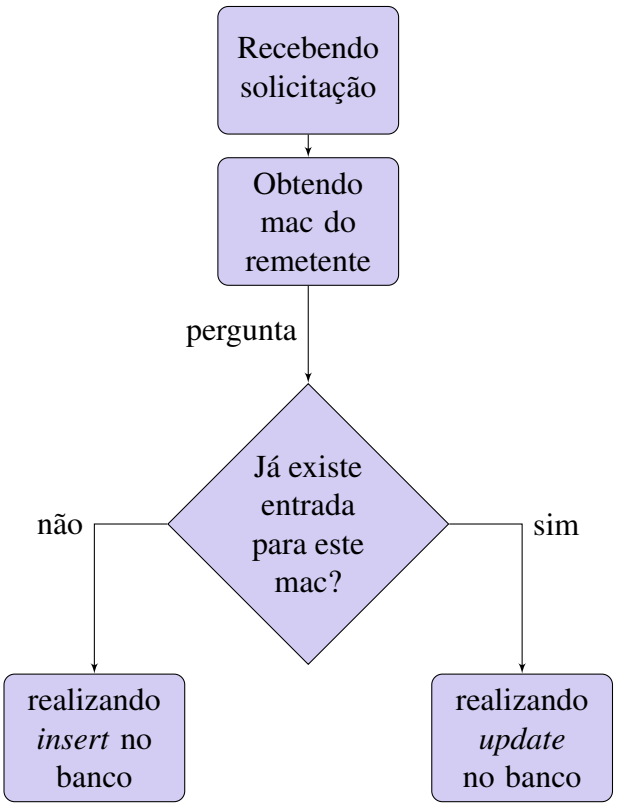
Figura 43: Fluxograma do envio da mensagem QRA



Fonte: Próprio autor

O processo de inclusão do nome do usuário no banco de dados é ilustrado no fluxograma 44.

Figura 44: Fluxograma de inserção do nome de um usuário no banco de dados



Fonte: Próprio autor

Porém, nos testes realizados, concluiu-se que a utilização de um banco de dados deixava o aplicativo mais lento. No momento que este iria fazer uma gravação no banco, o mesmo levava em torno de dois segundos para concluir a operação. Portanto, decidiu-se, com base nesses dados, fazer uma alteração no código.Para a gravação de *logs*, utilizou-se simples arquivos de texto que não produzem *lags* perceptíveis durante o uso. Para armazenar os nomes de usuários e posições do GPS, uma segunda análise concluiu que não era necessário armazenar esses valores após a utilização do aplicativo, então, esses dados foram armazenados em estruturas *hashmaps*.

Hashmaps funcionam mapeando valores de qualquer tipo para valores de chaves. O objetivo do uso dessa estrutura era armazenar vários atributos de cada dispositivo conectado na rede, portando decidiu-se considerar que a informação ideal para considerar como chave, deveria ser o endereço *mac* do dispositivo. Assim, utilizando o endereço *mac* de cada usuário, conseguimos obter as informações sobre ele da mesma forma que era feito no banco de dados, conforme ilustrado na figura 45

```
public static Map<String, Location> LocationMap = new HashMap<~>();
public static Map<String, String> UsersMap = new HashMap<~>();
public static Map<String, List> Usersprofile = new HashMap<~>();


public  static void setLocation(String id ,float x, float y){

    Location me = new Location("");
    me.setLatitude(x);
    me.setLongitude(y);
    LocationMap.put(id,me);
```

Figura 45: Trecho de código apresentando *Hashmaps* do aplicativo

Fonte: Próprio autor

A *QAMActivity* é responsável por tratar o código QAM, utilizado para perguntar ou informar quais as condições meterológicas da região. Ela é composta por uma série de botões, sendo um utilizado para questionar a outra parte, e os outros, para formular a resposta. Cada botão possui um ícone com um desenho de uma condição diferente e, ao ser pressionado, ele irá formatar uma mensagem de texto e retornar para a *MessengerActivity*. Por utilizar ícones, é utilizado a *widget ImageButton* no lugar de *Button*, como podemos ver na figura 46, que mostra um trecho do código XML de layout da interface.

Figura 46: Código da interface da *QAMActivity*

```
<ImageButton
    android:layout_width="98dp"
    android:layout_height="84dp"
    android:id="@+id/imageButton2"
    android:src="@drawable/cloud">
</ImageButton>

 <ImageButton
    android:layout_width="98dp"
    android:layout_height="84dp"
    android:id="@+id/imageButton3"
    android:src="@drawable/cloud_1">
 </ImageButton>

<ImageButton
    android:layout_width="98dp"
    android:layout_height="84dp"
    android:id="@+id/imageButton4"
    android:src="@drawable/haze">
</ImageButton>
<ImageButton
    android:layout_width="98dp"
    android:layout_height="84dp"
    android:id="@+id/imageButton5"
    android:src="@drawable/moon_phase_outline">
</ImageButton>
<ImageButton
    android:layout_width="98dp"
```

Fonte: Próprio autor

O código retorna a condição meteorológica do botão pressionado em forma de texto, conforme ilustrado na figura 47.

Figura 47: Trecho de código da *QAMActivity*

```
button3.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        sendreturn("neve");

    }
});
button4.setOnClickListener((v) -> { sendreturn("neblina"); });
button5.setOnClickListener((v) -> { sendreturn("noite sem nuvens"); });
button6.setOnClickListener((v) -> { sendreturn("noite com nuvens"); });
button7.setOnClickListener((v) -> { sendreturn("chuva"); });
button8.setOnClickListener((v) -> { sendreturn("sol com nuvens"); });
button9.setOnClickListener((v) -> { sendreturn("nuvens carregadas"); });
button10.setOnClickListener((v) -> { sendreturn("trovoadas"); });
button11.setOnClickListener((v) -> { sendreturn("chuva com trovoadas"); })
button12.setOnClickListener((v) -> { sendreturn("muito calor"); });
button13.setOnClickListener((v) -> { sendreturn("sol sem nuvens"); });
button14.setOnClickListener((v) -> { sendreturn("ventos fortes"); });
button15.setOnClickListener((v) -> { sendreturn("neve e trovoadas"); });
button16.setOnClickListener((v) -> { sendreturn("tornado"); });
button17.setOnClickListener((v) -> { sendreturn("vento e chuva"); });
```

Fonte: Próprio autor

O método SendReturn, que pode ser observado na figura 48, é utilizado nessa e em outras *activities* e é responsável por fazer o retorno dos dados e finalizar a execução.

Figura 48: Trecho de código da *Activity* QAM

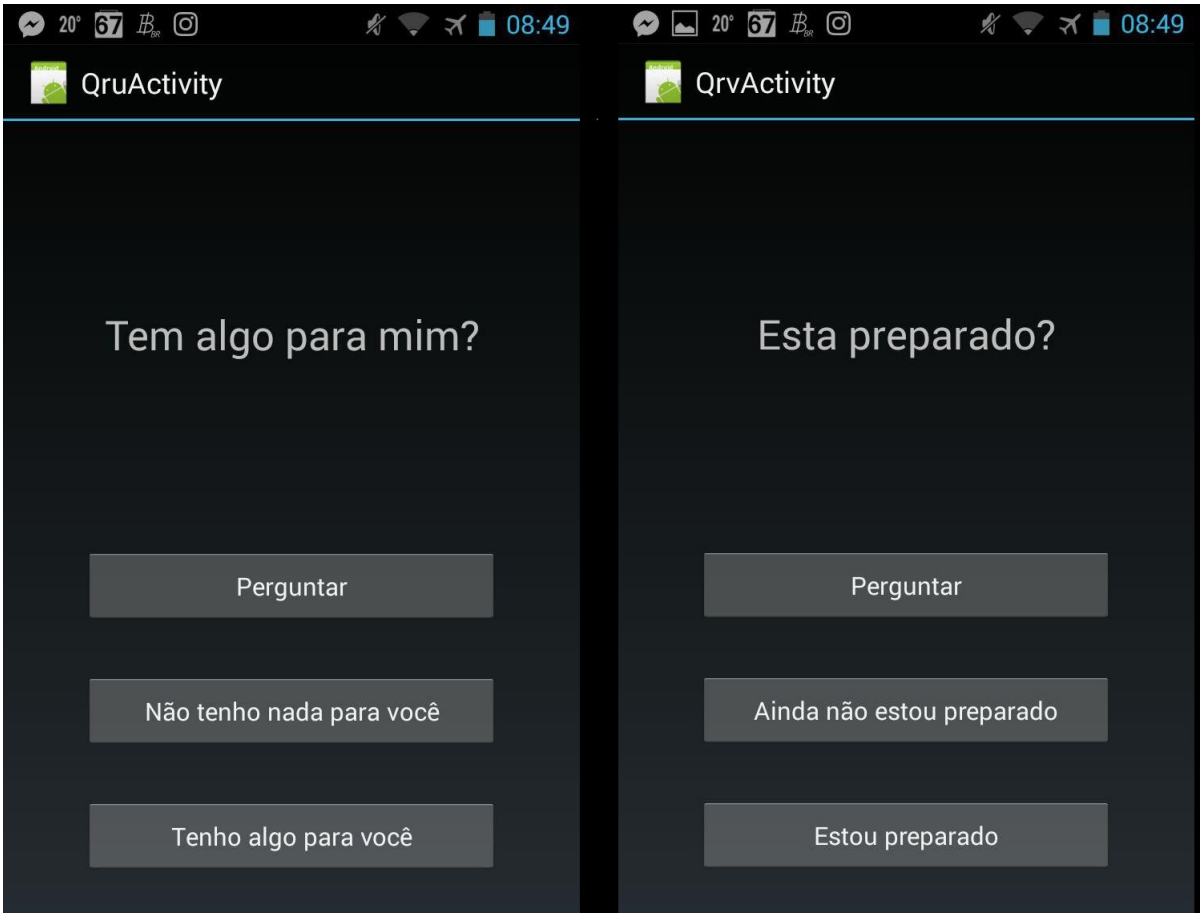
```
public void sendreturn(String wheater){

    Intent i = new Intent(getApplicationContext(), MessageActivity.class);
    i.putExtra("message",wheater);
    i.putExtra("buttonpressed","QAM");
    Log.e("teste","saindo QAM");

    setResult(Activity.RESULT_OK, i);
    finish();
```

Fonte: Próprio autor

As mensagens QRU e QRV servem para o usuário perguntar e responder a outro se tem algo para enviar, ou se estão preparados para uma atividade qualquer. Como essas mensagens

são bem simples, a construção das respectivas interfaces se baseou em simples botões, como podemos ver na figura 49.

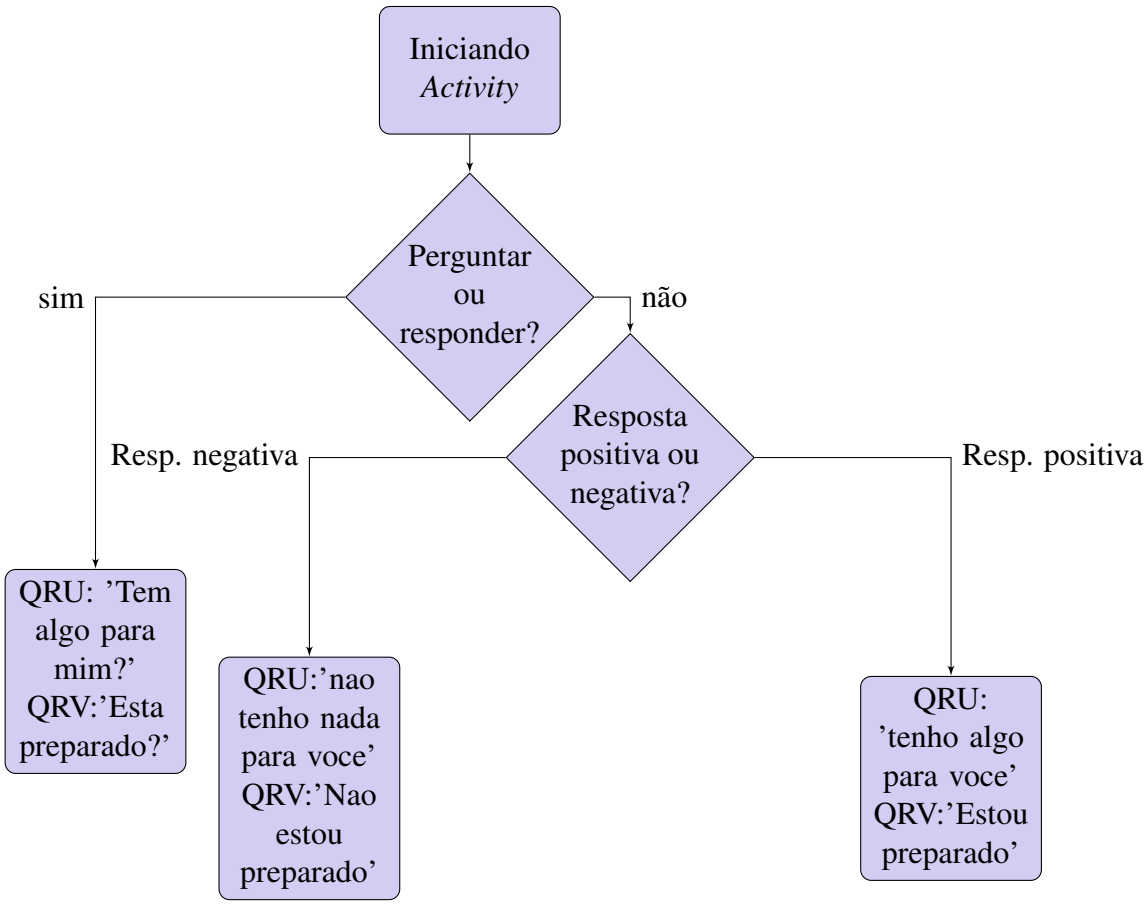
Figura 49: Tela principal das mensagens QRU e QRV



Fonte: Próprio autor

Se a intenção do usuário for a de formular uma pergunta, terá uma opção de botão para pressionar, se a intenção do usuário for a de formular uma resposta terá duas opções de resposta disponíveis. O fluxograma 50 ilustra bem a situação das duas *activities*.

Figura 50: Fluxograma das *activities* QRU e QRV



Fonte: Próprio autor

Já as mensagens QTW, QTZ e QUS têm como objetivo perguntar e responder, respectivamente, acerca das condições de saúde dos sobreviventes, se o usuário segue as buscas e se ele avistou sobreviventes. Para isso, as *activities* utilizam *CheckBox* e *Spinners* para marcar e selecionar as opções desejadas. Para se formular a mensagem de resposta, pressiona-se o botão respectivo e sua mensagem será anexada ao texto. Cada opção marcada nas *CheckBox* anexará uma frase diferente no texto final, conforme ilustrado no código da figura 51.

Figura 51: Trecho de código da *QUSActivity*

```
buttonq.setOnClickListener((v) → {

        message = "Você avistou sobreviventes?";
        sendreturn();

});

buttonn.setOnClickListener((v) → {

        message = "Não avistei sobreviventes";
        sendreturn();
});

buttony.setOnClickListener((v) → {

        message = "Avistei:\n";

        if (check1.isChecked()) message = message + "\tSobreviventes na água\n";
        if (check2.isChecked()) message = message + "\tSobreviventes em balsas\n";
        if (check3.isChecked()) message = message + "\tDestrocos\n";
        sendreturn();
```

Fonte: Próprio autor

Essas *activities* lidam com o envio de mensagens para outros usuários e também com a exibição das mesmas na tela. Os métodos *addMessage*,*sendMessage* e *writeToFile* são responsáveis por enviar essas mensagens para as saídas previstas pelo aplicativo: *addMessage* escreve a mensagem na tela do dispositivo, mais precisamente na *TextView* da *MessageActivity*; *sendMessage* envia essa mensagem pela rede, para que seja recebida pelos outros dispositivos participantes da conversa; *writeToFile* escreve a mensagem em um arquivo, para posterior análise.

O método *addMessage*, como podemos ver no código da figura 52, possui, como entrada, 3 variáveis: *from*,*to* e *text*. A primeira contém o nome do remetente da mensagem que, quando enviada pelo próprio dispositivo, será sempre "eu". A segunda variável é To, que é o nome de usuário do destinatário, ou seu perfil, que será falado mais adiante. E a terceira, é text, que é a mensagem de fato. Quando esse método é chamado, ele escreve a mensagem na tela do usuário, porém não a envia pela rede.

Figura 52: Código do método *addmessage*

```
public static void addMessage(String from, String text, String to) throws Exception {


    messageView.append(from + " (para " + to + ")" + ": " + text + "\n");
    final int scrollAmount = messageView.getLayout().getLineTop(messageView.getLineCount())
            - messageView.getHeight();
    // if there is no need to scroll, scrollAmount will be <=0
    if (scrollAmount > 0)
        messageView.scrollTo(0, scrollAmount);
    else
        messageView.scrollTo(0, 0);
```

Fonte: Próprio autor

O método *sendMessage*,mostrado na figura 53, é responsável pela tarefa de enviar as mensagem pela rede. Ele possui como entrada o texto a ser enviado e utiliza a classe *sender*, que faz uso de um *socket tcp* para enviar as mensagens.

Figura 53: Código do método *sendmessage*

```
public static void sendMessage(String text)    {

    String msgStr = "(para " + global.to + ")" + ": " + text + "\n";

    // Send to other clients as a group chat message
    for (AllEncompasingP2PClient c : MeshNetworkManager.routingTable.values()) {
        if (c.getMac().equals(MeshNetworkManager.getSelf().getMac()))
            continue;
        Sender.queuePacket(new Packet(Packet.TYPE.MESSAGE, msgStr.getBytes(), c.getMac(),
                WiFiDirectBroadcastReceiver.MAC));
    }

}
```

Fonte: Próprio autor

Por fim, *writeToFile* escreve essas mensagens em um arquivo, armazenando a hora para fins de relatar tudo. Na figura 54 podemos observar esse método.

Figura 54: Código do método *writeToFile*

```
public static void writeToFile(String message) {

    try {
        Date date = new Date();
        out.write(String.valueOf(date.getHours() + ":" + date.getMinutes() + ":" + date.getSeconds() + " "));
        out.write("Eu" + " (para " + global.to + ")" + ": " +message + "\n");
        out.flush();
```

Fonte: Próprio autor

Assim como a classe *Sender* é responsável por configurar um *socket tcp* para envio das mensagens, o mesmo ocorre com o recebimento, que é gerenciado pela classe *Receiver*, utilizando *sockets* TCP para o recebimento das mesmas, como é possível ver na figura 55.

Figura 55: Trecho do código da classe *Receiver*

```
try {
    this.serverSocket = new ServerSocket(port);
} catch (IOException e) {
    System.err.println("Server socket on port " + port + " could not be created. ");
    e.printStackTrace();
}
this.packetQueue = queue;
```

Fonte: Próprio autor

O *Receiver* também é responsável por tratar as mensagens recebidas, exibindo-as na tela através do *addMessage*, além de tratar mensagens especiais, chamadas de *setmessages*, que são responsáveis por salvar dados relevantes para o aplicativo. Como exemplo prático, podemos citar a *acitvity* QRA, onde o usuário informa seu nome para um outro usuário. Quando esse nome é propagado pela rede, é do interesse dos usuários que essas informações sejam salvas. Para isso, existe no *Receiver* uma função para tratar as chamadas *setmessages*, comandos que, ao serem recebidos, irão executar alguma ação ou salvar alguma informação no dispositivo receptor. Os comandos suportados são os seguintes:

- *setname*: serve para salvar um nome de usuário no *hashmap*. A função utiliza o endereço *mac* do remetente passa associar ao nome fornecido;
- *setposition*: salva uma determinada posição GPS no *hashmap*;
- *setprofile*: salva o perfil de um usuário, guardando a sua ocupação, que pode ser de policial, médico, bombeiro, vítima, dentre outros;

- *ping*: quando uma mensagem de ping é recebida, o dispositivo responde com uma mensagem de *pong*. Esse comando é utilizado para verificar a latência da rede e confirmar se as mensagens estão de fato chegando no outro dispositivo.

Na figura 56, podemos ver uma parte da implementação dessa função que trata as *setmessages*. Inicialmente a mensagem recebida é desmembrada para que cada palavra possa ser analisada de forma independente. Dependendo de qual for o comando recebido, a função irá realizar o processamento adequado.

Figura 56: Trecho do método que trata das *setmessages*

```
if(splitedsetmessage[0].equals("/setname")){
    global.UsersMap.put(mac,splitedsetmessage[1].replace("\n",""));
    Log.e("username salvo",splitedsetmessage[1].replace("\n",""));
}

if(splitedsetmessage[0].equals("/setposition"))
    {
        Log.e("teste","setposition");
        global.setLocation(mac,Float.valueOf(splitedsetmessage[1]),Float.valueOf(splitedsetmessage[2]));
    }

if(splitedsetmessage[0].equals("/setprofile")){
    List<String> profile = new ArrayList<~>();
    for(int i=1;i<splitedsetmessage.length;i++){
        profile.add(splitedsetmessage[i]);
    }
    global.Usersprofile.put(mac,profile);
```

Fonte: Próprio autor

Os perfis de usuário, presentes na função *addmessage* servem para que as mensagens sejam exibidas somente para o grupo de usuários desejado. Ao iniciar a execução do aplicativo, é solicitado ao usuário que escolha um ou mais perfis de usuário. Esses perfis servem para definir a atuação do profissional conforme listado a seguir:

- Policial
- Médico
- Bombeiro
- Operador Aéreo
- Operador Naval
- Vítima

A escolha do perfil do usuário é gerenciado pela *WhoActivity*. Ao selecionar um dos perfis possíveis essa *Activity* salva o resultado na variável *profile* para posterior utilização. A imagem da figura 57 ilustra parte desse código.

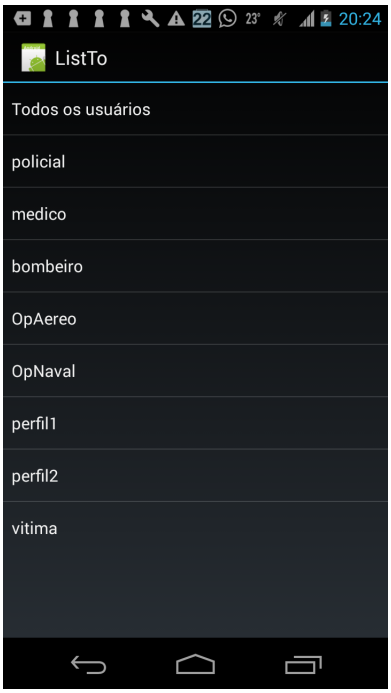
Figura 57: Trecho do código da *WhoActivity*

```
if(policial){
    setprofile=setprofile+" policial";
    myprofile.add("policial");
}

if(medico){
    setprofile=setprofile+" medico";
    myprofile.add("medico");
}

if(bombeiro){
    setprofile=setprofile+" bombeiro";
    myprofile.add("bombeiro");

}
```

Fonte: Próprio autor

Por outro lado, a *ListToActivty* serve para que o usuário possa escolher qual o destinatário de suas mensagens. Nessa *Activity* é apresentada uma lista de todos os perfis e usuários disponíveis, e assim, permite ao usuário escolher com quem ele deseja se comunicar. O destinatário desejado é salvo na variável global *from* e utilizado na função *addmessage*, que exibirá a mensagem na tela somente se o dispositivo atual for um possível destinatário. Na figura 58, é ilustrada uma tela do aplicativo com a *activity* em exibição.

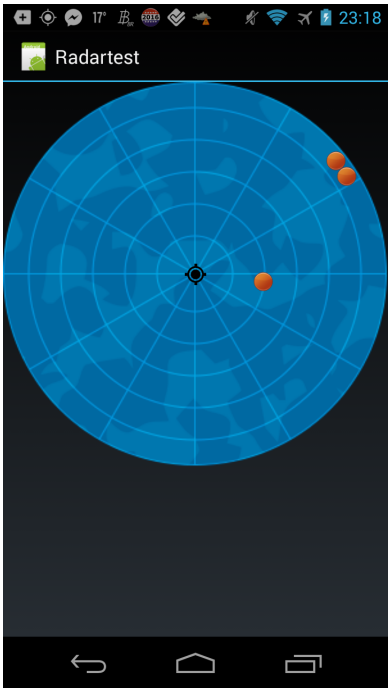
Figura 58: Interface da *ListToActivity*



Fonte: Próprio autor

A *Activity Radartest* é utilizada para criar um radar, mostrando a posição de outros usuários como pontos. A figura 59 mostra a tela dessa atividade.

Figura 59: Trecho do código da *RadartestActivity*



Fonte: Próprio autor

A classe Radar é a responsável por desenhar o radar. Essa classe não foi desenvolvida para o projeto e sim importada para o mesmo, pois seu código encontra-se disponível e é *open-source*. A classe precisou de algumas alterações, visto que não mostrava o posicionamento correto dos pontos no radar.

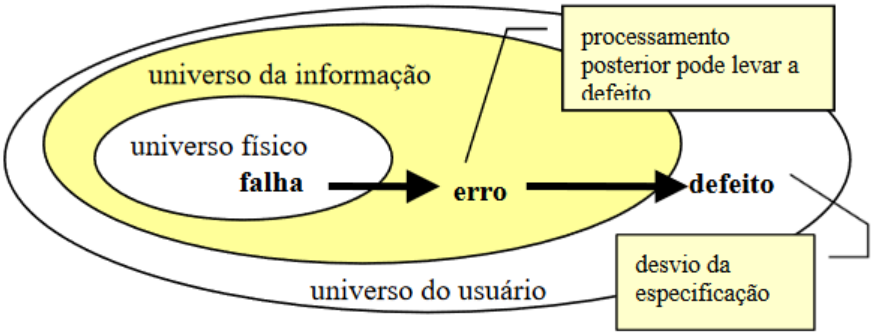
A última classe implementada foi a classe *SendPicture*, responsável pelo envio de arquivos de imagem. Essa função foi baseada nas demonstrações do próprio Android. A função permite a escolha de uma imagem utilizando o aplicativo *Gallery* do Android. Essa imagem será enviada para o destinatário da conversa atual.

5 TESTES DO APLICATIVO

De acordo com Myers *et al.* (2011), teste de software é o processo de execução de um programa com a intenção de encontrar erros, com o intuito de agregar valor ao programa, elevando sua qualidade e confiabilidade. Rios e Moreira (2006) afirmam que testar é verificar se o software é executado de forma controlada e se está fazendo o que deveria fazer, de acordo com seus requisitos, e se não está fazendo o que não deveria fazer.

Conforme afirma SILVA (2014), o objetivo de um teste é encontrar uma falha, ou erro no sistema que está sendo testado. Segundo Sandhof e Filgueiras (2006), um defeito é definido como um desvio da especificação, e diz-se que há um erro no sistema, quando o processamento posterior pode levar a um defeito e a falha é a causa física ou algorítmica desse erro, conforme ilustrado na figura 60.

Figura 60: Modelo de 3 universos: falha, erro e defeito

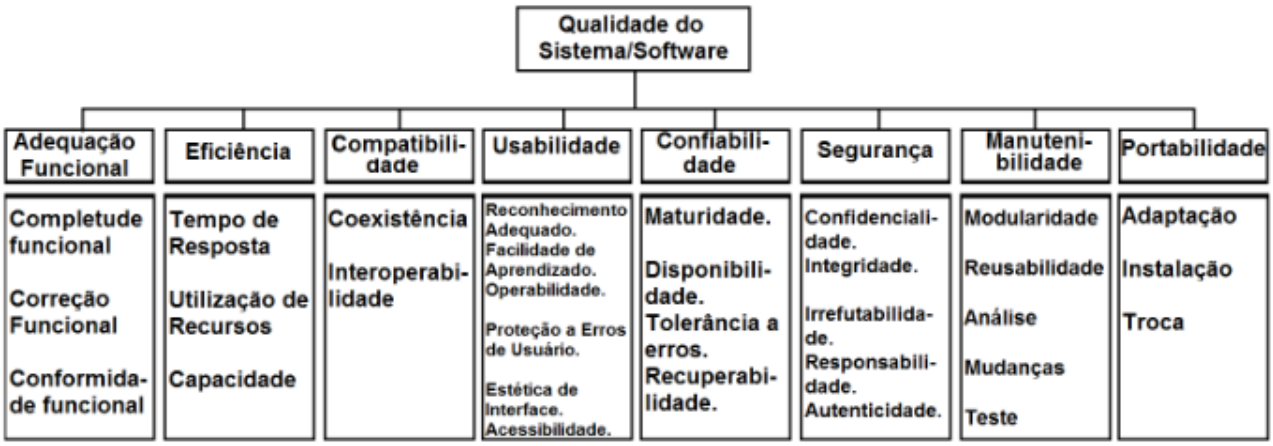


Fonte: Sandhof e Filgueiras (2006)

Os testes podem ser divididos em dois grupos, testes funcionais e não funcionais. Testes funcionais servem para concluir se as especificações definidas nos requisitos estão implementadas corretamente. De acordo com (BARBOSA *et al.*, 2000), esse tipo de teste trata o software como uma caixa preta, onde os detalhes da implementação não são considerados e a avaliação se dá pela perspetiva do usuário, a partir das funcionalidades definidas. Para isso, são fornecidas uma série de entradas e saídas esperadas a partir da especificação, e as mesmas confrontadas com as saídas obtidas na execução.

Testes não funcionais verificam características do software que não se relacionam diretamente com a funcionalidade, de acordo com Ferreira (2016). Algumas dessas características não funcionais podem ser observadas na figura 61.

Figura 61: Características não funcionais de software.



Fonte: Ferreira (2016)

Com base nisso, organizou-se uma estratégia de testes para o aplicativo desenvolvido neste trabalho. Testes funcionais foram realizados para testar cada uma das funções desenvolvidas no aplicativo; testes não funcionais foram utilizados para medir o alcance da ferramenta, a latência da comunicação, o consumo de bateria e a usabilidade.

5.1 Materiais e Métodos

A fim de atender uma série de requisitos funcionais e não funcionais do aplicativo desevolvido, decidiu-se organizar os testes nas seguintes categorias:

1. testes funcionais, onde buscou-se testar se as funcionalidades previstas na especificação foram atendidas corretamente;

2. testes de alcance e latência, para testar até que distância ainda era possível manter a comunicação entre os dispositivos e qual o custo no tempo de envio das mensagens a medida que a distância aumenta;

3. testes de usabilidade, onde procurou-se estabelecer a facilidade de se executar uma ação baseado no numero de interações com o aplicativo que são necessárias para sua conclusão;

4. testes de consumo, para verificar o impacto no consumo de energia da aplicação.

Para realização dos testes planejados, foi necessária a utilização de um conjunto de dispositivos *Android* com a aplicação instalada e um conjunto de outros softwares para realizar outras medições. Os dispositivos utilizados nos testes estão descritos em detalhes na tabela 7.

Tabela 7: Dispositivos utilizados durante os testes

Dispositivo	**Marca**	**Processador**	**Memória**	**Android**
Galaxy Nexus	*Samsung*	Dual-core 1.2 GHz Cortex-A9	1GB	4.3 Jelly Bean
Nexus 10	*Samsung*	Dual-core 1.7 GHz Cortex-A15	2GB	5.1 Lollipop
Prime Plus	*LG*	1.3 GHz Quad Core	1 GB	5.0.1 Lollipop
L20	*LG*	Dual-core 1.0 GHz Cortex-A7	512MB	4.4 Kit Kat
S3 Mini	*Samsung*	1.0 GHz dual-core Cortex-A9	1GB	5.0.2 Lollipop

Fonte: Próprio autor

A figura 62 mostra o conjunto de alguns dos dispositivos usados durante os testes.

Figura 62: Foto dos dispositivos utilizados durante os testes.

Fonte: Próprio autor

Além dos dispositivos também foram utilizados outros materiais durante os testes:

1. o aplicativo *CPU Monitor Advance*, utilizado para medir o consumo de processamento e memória do aplicativo desenvolvido;
2. o aplicativo *GSam Battery Monitor*, para medir o consumo de bateria;
3. uma fita métrica, para medir distâncias no teste de alcance;
4. um notebook *Acer Aspire E 15*, para acompanhar os *logs* de depuração nos testes funcionais.

5.2 Testes funcionais

Os primeiros testes realizados foram os testes funcionais, o objetivo era comprovar o funcionamento de cada função da aplicação de acordo com o esperado pelo requisitos. A apli-

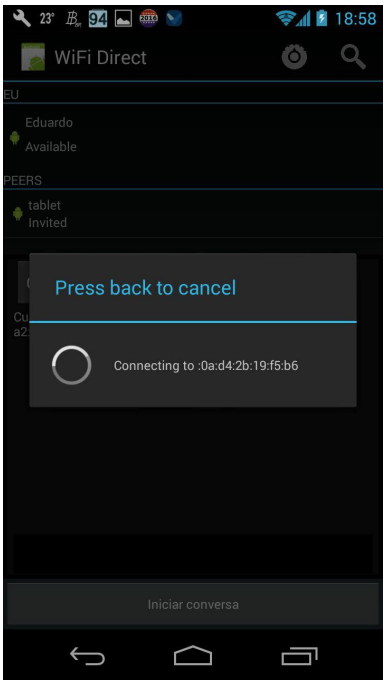
cação é composta por um conjunto de *Activities*, cada uma possui uma funcionalidade, conforme descrito no capítulo anterior. Essas *Activities* são compostas por botões,seletores, caixas de entrada de texto e outros elementos que formatam as mensagens que devem ser exibidas na tela e enviadas para um ou mais destinatários. Então, para se testar a aplicação, foi necessário iniciar todas as *Activities* e selecionar todas as opções possíveis e verificar se as mesmas executam as funções esperadas e se não possuem erros de programação que poderiam causar falhas no aplicativo.

Esses testes começaram a ser executados durante o desenvolvimento do aplicativo. A cada nova *Activity* implementada, era realizado um teste funcional. Iniciava-se o aplicativo no dispositivo e, na tela de *chat*, executava-se a *activity* desejada. Todas as possibilidades eram selecionadas a fim de atender todos os caminhos possíveis para o usuário. Se a mensagem correta fosse formatada na tela do dispositivo, o teste era considerado bem sucedido. Caso ela não fosse enviada, formatada de forma incorreta, ou mesmo, se o aplicativo fechasse inesperadamente, utilizava-se as informações de depuração do *Android Studio* para procurar o problema e corrigir.

Inicialmente o teste ocorria somente em um dispositivo, o *Galaxy Nexus*, e, logo, testava-se apenas a capacidade de formatação correta da mensagem e exibição da mesma na tela. Não se testava se efetivamente as mensagens estavam sendo transmitidas pela rede. Mais tarde, utilizou-se um segundo dispositivo, o *Galaxy S3 mini*, para poder testar o envio e recebimento de fato das mensagens. Nesse caso, apenas o *Nexus* se mantinha conectado ao *notebook* para depuração via USB, e o segundo dispositivo somente recebia e enviava mensagens.

A *Activity WifiDirectActivity* é a responsável por gerenciar as conexões *Wifi*, utilizando-a, os dispositivos podem encontram uns aos outros e iniciar a conexão. Para realizar o teste, os dois dispositivos foram colocados para pesquisar outros dispositivos. A busca retornava uma lista de todos os que estavam utilizando a aplicação na região. Então, com essa lista, é possível escolher a qual dispositivo se deseja conectar e, assim, é criada a rede. Essa *activity* funcionou durante os testes, porém, em algumas situações, o estabelecimento de conexão falhava e era necessário começar de novo. Foram registradas em torno de 15% de tentativas de estabelecimento de conexão fracassadas durante os testes dessa função. A figura 63 ilustta uma dessas tentativas fracassadas de conexão, onde o dispositivo fica sem resposta e não conclui o procedimento.

Figura 63: Aplicativo não respondendo a tentativa de conexão



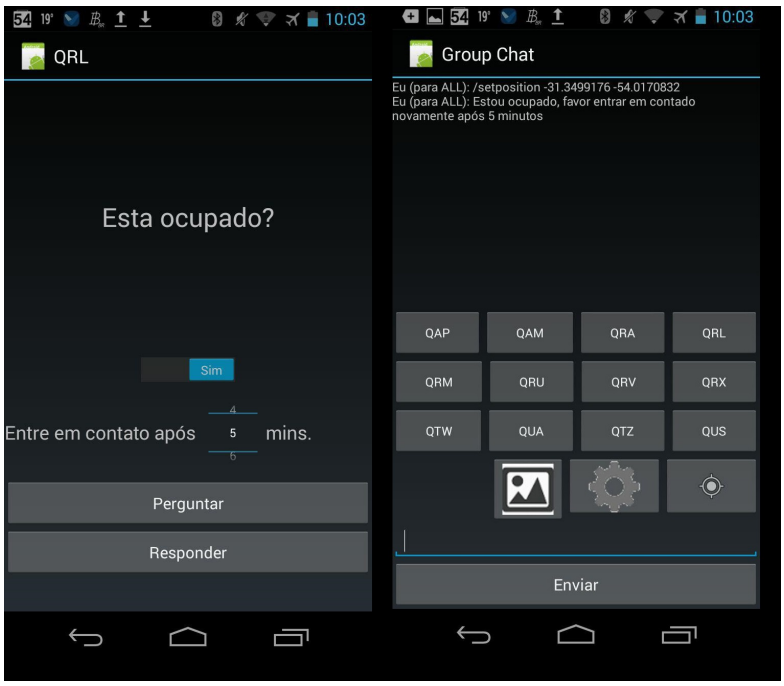
Fonte: Próprio autor

Também, ocorreu problemas na tentativa de comunicação com múltiplos dispositivos. O *LG Prime Plus* e o *LG L20* não se conectaram em redes com múltiplos dispositivos, no momento em que um terceiro dispositivo tentava estabelecer uma conexão, o aplicativo encerrava a conexão antiga.

A *MessengerActivity* é chamada diretamente pela *WifiDirectActivity* e é responsável por prover o ambiente de *chat* entre os dispositivos, contendo um campo para digitar texto, uma caixa para exibir as conversas e botões para acionar outras *activities*. O procedimento de teste consistiu no envio de mensagens para um outro dispositivo. As mensagens eram digitadas no campo de entrada e o botão de envio era pressionado, caso as mensagens fossem recebidas no outro dispositivo, considerava-se o teste como bem sucedido. Não houve problemas para testar essa função que funcionou sem nenhum imprevisto.

As demais *Activities* são lançadas ao pressionar os outros botões presentes nessa tela. As *activities* QAP, QAM, QRA, QRL, QRU, QRV, QRX, QTW, QUA, QTZ e QUS são responsáveis por formatar mensagens que irão retornar para a *MessengerActivity* e, então, serão enviadas aos usuários. Cada uma delas foi testada de forma individual, com um dos dispositivos lançando a *activity*, preenchendo as entradas e enviando a mensagem corretamente ao usuário. Como exemplo, na *activity* QRL, a mensagem "Está ocupado"deve ser exibida na tela e enviada ao destinatário, quando o usuário clica no botão "pergunta". Quando o usuário deseja informar se está ou não ocupado, e por quanto tempo, deve selecionar no seletor e escolher o número de minutos no qual estará ocupado e, assim, a mensagem será enviada. Na imagem 64, podemos observar a *activity* sendo utilizada e o resultado que retornou para a tela de *chat*.

Figura 64: Tela do aplicativo da *QRLActivity* e o resultado produzido na tela de *chat* após sua utilização.



Fonte: Próprio autor

Desse modo, ocorreu a validação de todas as funções programadas e correção de problemas que não haviam sido previstos. O primeiro problema encontrado estava relacionado com o retorno dos dados para a *activity* anterior. No momento em que os dados retornavam, todas as informações que estavam sendo exibidas na *TextView* da *MessengerActivity* eram perdidas, uma vez que não tinha nenhuma função responsável por salvar os dados durante a mudança de contexto. Essa função foi implementada e, assim, a conversa passou a ser mantida em sua integridade.

Outro problema que levou à reimplementação de parte do aplicativo estava relacionado com o banco de dados. Como foi explicado no capítulo anterior, foi necessário substituir o banco de dados *SQLite* por uma estrutura de *Hashmaps* por conta da lentidão observada. A conclusão foi obtida durante os testes funcionais

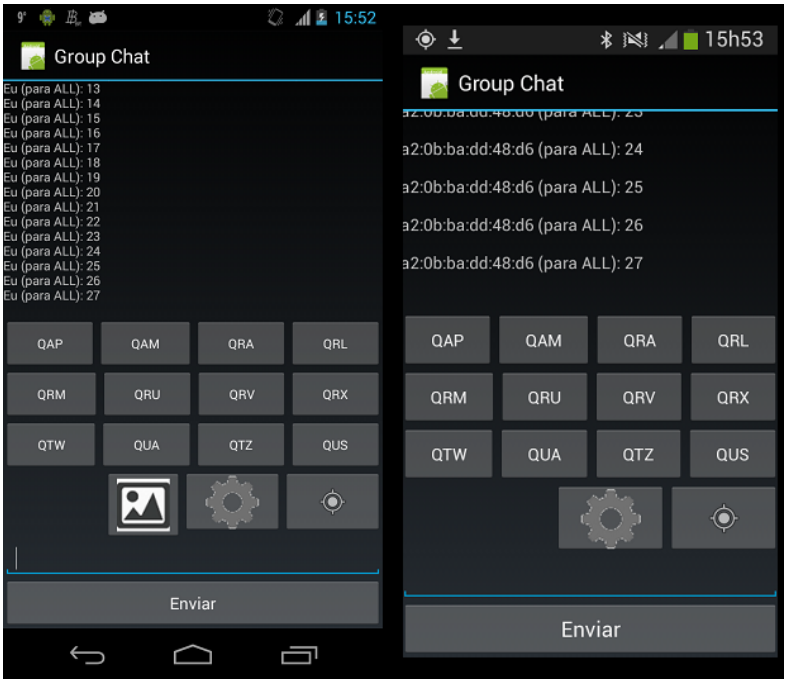
Outra mudança executada, em decorrência dos resultados dos testes, foi na *activity*, responsável por gerar o radar GPS. Ela não era capaz de posicionar os pontos do GPS no local correto. Portando foi necessário reescrever a função a fim de mudar a forma como era calculado o ângulo do ponto no radar.

5.3 Testes de alcance e latência

O próximo passo foi testar o alcance de envio das mensagens, da mesma forma que havia sido testado no protótipo. Para isso, foi necessário a utilização de 2 *smartphones* e uma fita

métrica. O teste consistiu em conectar os dois *Smartphones* e colocá-los para trocar mensagens, lado a lado, em um ambiente externo. Após o envio e recebimento da mensagem, utilizando a fita métrica, aumentava-se a distância entre dos dispositivos em 1 metro e repetia-se o envio de mensagem. Nesse teste, chegou-se a um alcance de 82 metros. A figura 65 mostra os dois dispositivos trocando mensagens durante o teste.

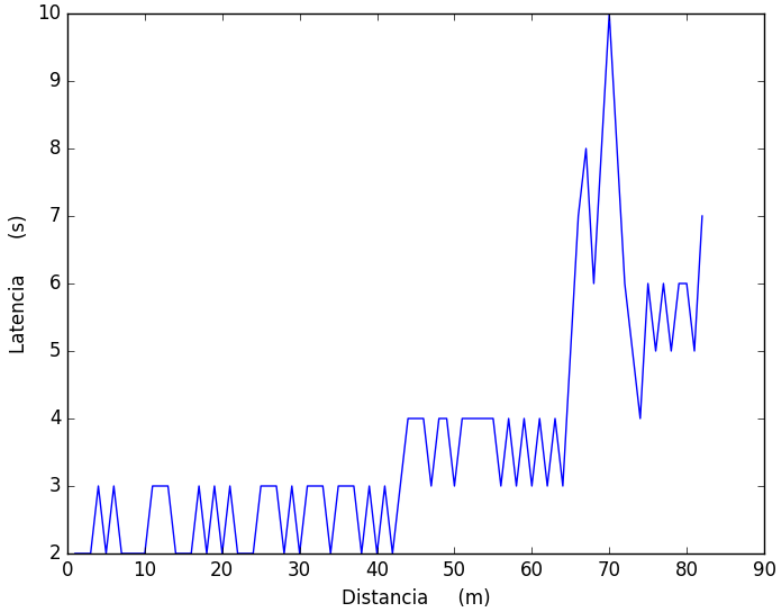
Figura 65: Tela do aplicativo nos dois dispositivos android com o *Galaxy Nexus*, na esquerda, enviando mensagens para o *Galaxy S3 Mini*, na direita.



Fonte: Próprio autor

Para se testar a latência no envio de mensagens, foi utilizada a função de ping, implementada no próprio aplicativo. Com essa função, é possivel calcular o tempo que uma mensagem leva para sair do remetente e chegar ao destinatário. Na figura 66, podemos observar o gráfico da latência do aplicativo ao longo do distanciamento dos dispositivos.

Figura 66: Gráfico da latência ao longo da distância no envio de mensagens pelo aplicativo.



Fonte: Próprio autor

Inicialmente foi utilizado o *Galaxy Nexus* e o *Galaxy s3 Mini*,logo após, também inseriu-se o dispositivo *Nexus 10* na conexão para testar se ele também conseguia atingir o mesmo alcance. O resultado foi positivo, o *Nexus 10* também enviou mensagens na distância de 82 metros.

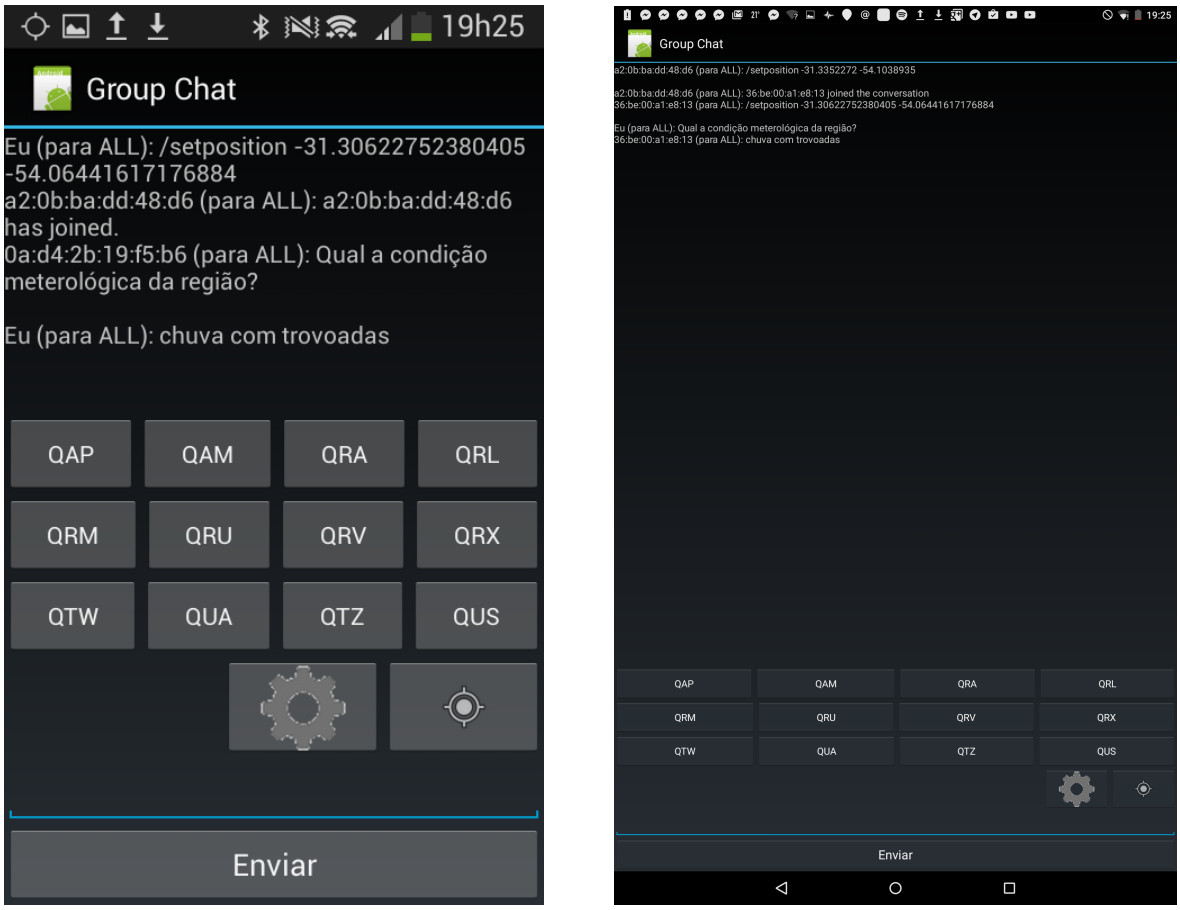
Para se testar a capacidade de envio de mensagens roteadas entre dispositivos, isto é, enviar mensagem de um dispositivo para outro passando por um intermediário, foram conectados 3 dispositivos e os mesmos foram posicionados em uma determinada distância um do outro, de modo que o dispositivo em uma extremidade não tivesse capacidade de enviar mensagens diretamente para o dispositivo que se encontrava na outra extremidade. No mapa da figura 67, pode-se ver a posição em que cada dispositivo foi posicionado.

Figura 67: Mapa extraído do *Google Maps* mostrando a localização dos 3 dispositivos durante o teste de envio de mensagens, utilizando um dispositivo como roteador.

Fonte: Próprio autor

Com os dispositivos posicionados, iniciou-se a transferência de mensagens. O dispositivo transmissor enviou mensagens tendo como destinatário o dispositivo mais distante do mesmo. As mensagens foram recebidas e exibidas na tela do destinatário, como podemos ver na figura 68, porém com uma latência perceptivelmente maior.

Figura 68: Tela do aplicativo nos dois dispositivos enquanto enviam e recebem mensagens



Fonte: Próprio autor

A latência na troca de mensagens entre os dispositivos ficou na média de 8 segundos. Sendo metade desse tempo para ir do dispositivo transmissor para o intermediário, e a outra metade do intermediário para o final.

5.4 Testes de usabilidade

Para realização dos testes de usabilidade, foi considerada uma única variável para mensurar a simplicidade no uso, que foi o número de cliques ou interações com o aplicativo para finalizar uma tarefa. Para essa medição, dividimos o fluxo de utilização do aplicativo em duas etapas. A etapa inicial, chamada de configuração, vai desde a inicialização do mesmo até o momento em que é aberta a tela de chat. A fase de comunicação é a seguinte, onde o usuário inicia sua interação com os outros usuários do aplicativo.

Na fase inicial, que é igual para qualquer usuário, o dispositivo deve iniciar a conexão clicando no botão de pesquisar dispositivos e, em seguida, no botão conectar. Após esses dois cliques, estará estabelecida a conexão. Em seguida, clica-se em "Iniciar conversa", para direcionar-se à outra *activity*.

Na primeira conexão será necessário escolher com quais usuários se deseja conectar e, em seguida, preencher o nome do usuário e seu perfil. Essas tarefas encerram a fase de configuração com o redirecionamento do usuário para a tela de *chat* da *MessengerActivity*.

Nesse momento, cada usuário poderá optar por uma das alternativas existentes de comunicação, e o número de interações para cada ação pode variar. Para enviar uma mensagem de texto simples, é necessário apenas preencher a *Textbox* e pressionar o botão Enviar. Para enviar uma mensagem QAM, isto é, perguntar ou responder acerca das condições metereológicas, é necessário clicar no botão QAM e no botão escolhido que indica a condição metereológica atual, ou, também, para questionar a outra parte a respeito do mesmo assunto. Para enviar uma mensagem QRA, ou seja, perguntar ou informar o nome do usuário, é necessário pressionar o botão QRA e, em seguida, o botão indicando a pergunta, caso ela seja do interesse do usuário, e preencher a *textbox* e pressionar o botão de resposta, caso seja do interesse do usuário responder. A tabela 8 resume o número de interações mínima e máxima para cada uma das *activities*.

Tabela 8: Número de interações por *activity*

Activity	**Formular uma pergunta**	**Formular uma resposta**
QRA	2 iterações	3 iterações
QRL	2 iterações	de 2 até 4 iterações
QRM	1 iteração	nenhuma interação
QAM	2 iterações	2 iterações
QRU	2 iterações	2 iterações
QRV	2 iterações	2 iterações
QRX	2 iterações	4 iterações
QTW	2 iterações	de 3 até 6 iterações
QTZ	2 iterações	de 2 até 5 iterações
QUS	2 iterações	de 2 até 5 iterações

Fonte: Próprio autor

Dentre essas iterações, somente duas envolvem digitar informações em texto, o que evita que se gaste muito tempo na entrada de dados.

5.5 Testes de consumo

Os testes para medir o consumo de bateria foram realizados como última etapa. Dois dispositivos foram colocados para trocar mensagens pelo aplicativo. Para automatizar o processo, foi criada uma função que envia mensagens aleatórias, sem a necessidade de interação com o usuário. O software foi colocado para funcionar e medido o consumo da bateria.

Há uma função no software que tem como objetivo fechar aplicativos desnecessários para economizar bateria. Com essa função desativada, o consumo em 17 minutos de execução,

enviando e recebendo mensagens, foi de 4.1%. com a função desativada, esse consumo atingiu 3.9%. A diferença foi de pouca relevância para a redução no consumo de bateria.

5.6 Resultados e discussões

Nos testes realizados o aplicativo apresentou algumas instabilidades para iniciar uma conexão. O tempo para estabelecer a conexão costumava ser curto, em torno de 2 segundos, porém, às vezes, o aplicativo parava de responder durante o estabelecimento de conexão e ela nunca se efetuava. Ocasionalmente ocorreram travamentos, durante essa fase de estabelecimento de conexão, que não ocorriam novamente quando o aplicativo era executado uma segunda vez. Alguns *smartphones* não permitiam conexão entre mais de 2 dispositivos, tornando o aplicativo de pouca utilidade para a função a qual foi projetado.

Porém, o aplicativo cumpriu todas as funcionalidades que haviam sido projetadas, enviando e recebendo, com sucesso, cada mensagem enviada e permitindo a conexão entre dois *smartphones,* que não estavam no raio de alcance direto um do outro, por meio do roteamento de um terceiro. O ponto negativo dessa funcionalidade é o aumento considerável da latência, que duplica a cada novo dispositivo necessário para reencaminhar a mensagem. Contudo esta latência não desabona a aplicação, visto que o intuito é atender uma área maior em um ambiente de desastre.

Com relação a usabilidade, pode-se observar que, para qualquer caso de uso da aplicação, são necessários poucos cliques e interações do usuário, evitando tempo desperdiçado e, assim, agilizando o envio de cada mensagem.

O teste comparativo do consumo de bateria não retornou diferenças significativas. A causa mais provável para isso encontra-se no fato de que dispositivos sem o privilégio de *root* possuem uma capacidade limitada para lidar com outros processos. Mesmo que a aplicação encerre um outro processo, ela não pode impedir que o processo volte a ser executado novamente, de forma automática na maioria dos casos, e com isso precisa encerrar aquele processo mais uma vez. O encerramento consecutivo de processos pode fazer com que o dispositivo consuma mais bateria do que consumiria se não houvesse nenhuma função para encerrar aplicativos.

Em comparação com outras similares estudadas, a aplicação aqui desenvolvida possui ganhos para a utilização como ferramenta para o auxilio de equipes de resgate. Um dos seus benefícios é que não é somente uma ferramenta de *chat*, possuindo a capacidade de enviar uma série de mensagens padronizadas, baseadas em necessidades de comunicação destas situações de emergência. Outro ponto positivo é a utilização de uma tecnologia capaz de manter a comunicação por distâncias mais longas que as permitidas pelo *bluetooth*.

No futuro, é possível implementar novas funcionalidades, que atendam melhor aos objetivos da ferramenta e corrigir os problemas das instabilidades ocasionais na aplicação.

6 CONSIDERAÇÕES FINAIS

O *Wi-Fi Direct* é um padrão *Wi-Fi* que permite que dispositivos se conectem diretamente uns com os outros sem a necessidade de uma infra-estrutura previa no ambiente. Seu funcionamento se baseia em utilizar os próprios dispositivos como roteadores, criando então, uma rede dinâmica de comunicação.

Com as facilidades do *Bluetooth* e velocidade do *Wi-Fi tradicional* a tecnologia pode ser de grande ajuda em ambientes onde o acesso a internet é restrito e a necessidade de comunicação é vital. Foi proposto nesse trabalho a implementação de uma ferramenta de comunicação para ambientes de desastres utilizando *Wi-Fi Direct* como principal tecnologia.

O *Wi-Fi Direct* se mostrou viável neste projeto, o aplicativo desenvolvido é capaz de criar uma rede dinâmica onde mensagens podem ser trocadas entre os usuários mesmo sem uma conexão com a internet, redes de telefonia ou similares. A aplicaçao desenvolvida é capaz de trocar mensagens de texto, e formatar mensagens com base no código internacional Q. Além disso, também é possível compartilhar a localização por GPS, e transferir fotos.

Os testes funcionais demonstraram que a ferramenta cumpre os requisitos propostos no trabalho, uma vez que todas as funcionalidades previstas funcionam corretamente. Por outro lado, instabilidades na conexão costumam ocorrer, ocasionando eventuais perdas de informação e a necessidade de tempo para reestabelecer a conexão novamente. Tudo isso indica que a tecnologia ainda esta imatura, uma vez que existe há pouco tempo e não é utilizada em massa.

Como estudos futuros, é possivel implementar melhorias na ferramenta, inserindo mais funcionalidades, como por exemplo, mapas offline no lugar do radar, criptografia para proteção das comunicações, conversas por áudio e vídeo e contas de usuário. Também é possivel procurar meios de tornar a ferramenta mais estável e acessível ao usuário.

REFERÊNCIAS

ABLEITNER, F. **ensichat**. 2014. (). Disponível em: <https://github.com/Nutomic/ensichat>.

ALECRIM, E. **Tecnologia Bluetooth: o que é e como funciona?** 2008. (acessado em 2 de setembro de 2015). Disponível em: <http://www.infowester.com/bluetooth.php>.

ALMEIDA, V. D. D. Análise de desempenho de protocolos de roteamento ad hoc dtn em redes de emergência. UFMG, 2011.

ANDROID DEVELOPERS. **android.net.wifi.p2p**. 2011. (acessado em 30 de maio de 2016). Disponível em: <https://developer.android.com/reference/android/net/wifi/p2p/package-summary.html>.

ARAÚJO, B. S,"administração de desastres: conceitos & tecnologias". **Rio de Janeiro**, 2012.

BANNACK, A. Aplicando gestão de energia ao protocolo de roteamento para redes ad hoc móveis vrp. 2008.

BARBOSA, E. F. *et al.* Introdução ao teste de software. **Minicurso apresentado no XIV Simpósio Brasileiro de Engenharia de Software (SBES 2000)**, 2000.

BORDIN, M. V. Introdução a arquitetura android. 2010.

BRAGA, M. de M. *et al.* Aplicação das técnicas de gestão do conhecimento no gerenciamento de desastres naturais. In: **SIMPOSIO SOBRE LA SOCIEDAD DE LA INFORMACION**. La Plata: [s.n.], 2011.

BRUSCATO, G. C.; MORS, P. M. Ensinando física através do radioamadorismo. **Revista brasileira de ensino de física. Vol. 36, n. 1 (mar. 2014), 1506, 8 p.**, 2014.

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

CARRILLO, G. *et al.* **Introduction to Disaster Management**. Vancouver:Virtual University for Small States of the Commonwealth, 2012.

CHANNA, M. I.; AHMED, K. M. Emergency response communications and associated security challenges. **arXiv preprint arXiv:1010.4887**, 2010.

COMER, D. E. **Redes de Computadores e Internet-6ª Edição**. [S.l.]: Bookman Editora, 2016.

DAU, M. L. Comunicação entre manets com protocolo de roteamento baseado na fonte. 2013.

DUARTE, L. O. Análise de vulnerabilidades e ataques inerentes a redes sem fio 802.11 x. **UNESP–IBILCE–São José do Rio Preto**, 2003.

DUNCAN, I. B. Modelagem e análise do protocolo ieee 802.11. **COPPE/UFRJ, M. Sc., Engenharia de Sistemas e Computação**, 2006.

FARIAS, M. M. **Protocolo de roteamento para redes wireless mesh**. Tese (Doutorado) — Pontifícia Universidade Católica do Rio Grande do Sul, 2008.

FARREL, A. **A Internet e seus Protocolos**. [S.l.]: Elsevier Brasil, 2005. v. 1.

FERNANDES, A. *et al.* Wirelessteams: Comparação de tecnologias sem fios em equipas de robôs móveis. **Relatório técnico, Coimbra**, 2012.

FERNANDES, N. C. *et al.* Ataques e mecanismos de segurança em redes ad hoc. In: A (Ed.). **Minicursos do Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg'2006)**. Santos: [s.n.], 2006. v. 2006, p. 49–102.

FERREIRA, A. P. M. Um estudo sobre ferramentas para o teste de aplicações android no contexto do laboratório leds. 2016.

FIGUEIREDO, C. M.; NAKAMURA, E. Computação móvel: Novas oportunidades e novos desafios. **T&C Amazônia**, p. 16–28, 2003.

FREITAS, N. **Gilga**. 2015. (). Disponível em: <https://github.com/n8fr8/gilgamesh/>.

GARDEN, O. **Fire Chat**. 2014. (). Disponível em: <https://www.opengarden.com/firechat.html>.

GIELEN, M. Ad hoc networking using wi-fi during natural disasters: Overview and improvements. In: **17th Twente Student Conference on IT**. [S.l.: s.n.], 2012. v. 17.

GIL, A. C. Como elaborar projetos de pesquisa. **São Paulo**, v. 5, p. 61, 2002.

GOMES, F. d. C. Infraestrutura de comunicação para a governança e o desenvolvimento: o cinturão digital do ceará. **Congresso Consad de Gestão Pública**, Brasília, 2009.

GOMES JÚNIOR, C. A. d. A. O uso do incident command system em operações de preservação da ordem pública. Florianópolis, 2006. Monografia(especialização em administração pública) — Universidade do Sul de Santa Catarina.

GOMES JÚNIOR, C. A. de A.; ALVES, M. L. **Capacitação em Defesa Civil - Sistema de Comando em operações**. Florianópolis:Governo do Estado de Santa Catarina, 2004.

GRÜNEWALD, M. A. Redes sem fio – tecnologia, segurança e usabilidade. **Faculdade de Informática e Administração Paulista**, São Paulo, 2005. Monografia(pós-graduação Lato Sensu em Gestão de Tecnologia da Informação) — Faculdade de Informática e Administração Paulista.

HADEN, R. **TWireless LAN**. 2000. (acessado em 20 de outubro de 2015). Disponível em: <http://www.rhyshaden.com/wireless.htm>.

HENDERSON, P. *et al.* Echo - a mesh chat application for android. 2014.

Instituto Defesa. **Código Q – Muito mais que QSL e QAP**. 2014. (acessado em 17 de julho de 2016). Disponível em: <http://www.defesa.org/codigo-q-muito-mais-que-qsl-e-qap/>.

JEAN, A. Gerenciamento wi-fi. Florianópolis, 2013. Monografia(Barachel em ciência da computação) — Universidade Federal de Santa Catarina.

JOH, H.; RYOO, I. A hybrid wi-fi p2p with bluetooth low energy for optimizing smart device's communication property. **Peer-to-Peer Networking and Applications**, Springer, p. 1–11, 2014.

KANG, A. **Blue Chat**. 2014. (). Disponível em: <https://github.com/AlexKang/blue-chat>.

KHATKO, R. **Peer-to-Peer Applications on Intel Android Developer Guide**. 2014. (acessado em 30 de maio de 2016). Disponível em: <https://software.intel.com/en-us/android/articles/peer-to-peer-applications-on-intel-android-developer-guide>.

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

LECHETA, R. R. **Google Android-3ª Edição: Aprenda a criar aplicações para dispositivos móveis com o Android SDK**. [S.l.]: Novatec Editora, 2013.

LOPES, D. da C. **Construindo Comunidades Mais Seguras: preparando para a ação cidadã em defesa civil**. [S.l.]: UFSC/CEPED, 2009.

LOSSO, D. B. Gerenciamento do erro e da ameaça: Uma ferramenta de gestão para operações de resposta em eventos críticos. Florianópolis, 2012. Monografia(especialização em gestão de eventos críticos) — Universidade do Sul de Santa Catarina.

LOUREIRO, R. dos S. **Resgate em Estruturas Colapsadas**. Rio De Janeiro:Corpo de Bombeiros Militar do Estado do Rio de Janeiro, 2012.

MACHADO, F. A. O.; PINTO, A. V.; TEIXEIRA, M. M. Uma rede de compartilhamento de conteúdo multimídia em dispositivos móveis baseados na plataforma android. In: **Anais do II Workshop de Comunicação em Sistemas Embarcados Críticos - WoCCES 2014**. Florianópolis: [s.n.], 2014.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The art of software testing**. [S.l.]: John Wiley & Sons, 2011.

Netmarketshare. **Mobile/Tablet Operating System Market Share**. 2016. (acessado em 21 de novembro de 2016). Disponível em: <https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=1>.

OLIVEIRA JÚNIOR, E. da C. **Atribuição Dinâmica de Endereços em Redes Ad Hoc sem Fio em Plataforma Android**. 2014. Monografia (Engenheiro em Redes de computação), Universidade de Brasília, Brasília.

OLIVEIRA, M. de. **Livro Texto do Projeto Gerenciamento de Desastres - Sistema de Comando de Operaçcões**. Florianópolis:Ministério da Integração Nacional, 2009.

PEREIRA, L. C. O.; SILVA, M. L. D. **Android para desenvolvedores**. [S.l.]: Brasport, 2009.

REZENDE, H. **Android e WiFi**. 2013. (acessado em 30 de maio de 2016). Disponível em: <https://prezi.com/dyro5wfzzmio/android-e-wifi>.

RIOS, E.; MOREIRA, T. **Teste de software**. [S.l.]: Alta Books Editora, 2006.

RUBINSTEIN, M. G.; REZENDE, J. F. Qualidade de serviço em redes 802.11. **XX Simpósio Brasileiro de Redes de Computadores (SBRC2002)**, 2002.

SACCOL, A. Z.; REINHARD, N. Tecnologias de informação móveis, sem fio e ubíquas: definições, estado-da-arte e oportunidades de pesquisa. **Revista de administração contemporânea**, SciELO Brasil, v. 11, n. 4, p. 175–198, 2007.

SANDHOF, K.; FILGUEIRAS, L. Defeitos de software como erros humanos. In: **II Workshop Um olhar sociotécnico sobre a Engenharia de Software (WOSES 2006)**. [S.l.: s.n.], 2006.

SHARETECHNOTE. **WLAN - WiFi Direct**. 2015. Disponível em: <http://www.sharetechnote.com/html/WLAN_WiFi_Direct.html>.

SILVA, B. E. d. S. Processo de testes de aplicações mobile: uma abordagem prática com base na norma isso/iec/ieee 29119. 2014.

SOMMERVILLE, I. Engenharia de software, 8ª edição. **São Paulo: Pearson Addison-Wesley**, v. 22, p. 103, 2007.

TANENBAUM, A. **Redes de Computadores**. [S.l.]: Campus, 2003.

VERGARA, S. C. Projetos e relatórios de pesquisa em administração. são paulo: Atlas, 2000. **Métodos de pesquisa em administração**, v. 3, 2009.

ZHANG, H.; WANG, Y.; TAN, C. C. Wd2: an improved wifi-direct group formation protocol. In: ACM. **Proceedings of the 9th ACM MobiCom workshop on Challenged networks**. [S.l.], 2014. p. 55–60.