

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Patrick Godinho

PheroCast App

Uberlândia, Brasil

2014

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Patrick Godinho

PheroCast App

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Lasaro Camargos

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2014

Patrick Godinho

PheroCast App

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 24 de novembro de 2012:

Lasaro Camargos
Orientador

Professor

Professor

Uberlândia, Brasil
2014

*Dedico a meu pai Paulo Sergio de Jesus Oliveira que sempre foi um exemplo de
perseverança na minha vida!*

Agradecimentos

Agradeço a Deus, minha esposa e ao meu orientador Lasaro Camargos pelo apoio e dedicação para que eu conseguisse chegar ao final deste trabalho.

Resumo

Palavras-chave: Manet, Android, Redes Oportunísticas

Lista de ilustrações

Figura 1 – Arquitetura Proposta	16
Figura 2 – Processo de captura dos dados pelo aplicativo.	16
Figura 3 – Análise 1 - Poucas redes coletadas em percurso	21
Figura 4 – Análise 2 - Parte 1 - Muitas redes coletadas em percurso	22
Figura 5 – Análise 2 - Parte 2 - Muitas redes coletadas em percurso	23
Figura 6 – Análise 2 - Parte 3 - Muitas redes coletadas em percurso	24

Lista de tabelas

Sumário

1	INTRODUÇÃO	10
	Introdução	10
1.1	Contextualização	10
1.1.1	MANET	10
1.1.2	VANET	10
1.1.2.1	Conceito	10
1.1.2.2	Problema	11
1.1.3	Predição de localização	11
1.1.3.1	Motivo	11
1.1.3.2	PheroCast	11
1.1.3.2.1	Conceito	11
1.1.3.2.2	Resultados	12
1.1.3.2.3	Problema	12
1.2	Proposta	12
2	REFERENCIAL TEÓRICO	13
2.1	PheroCast	13
2.2	Android	14
2.2.1	Activities	14
2.2.2	Services	14
2.2.3	Content Providers	15
2.2.4	Broadcast receivers	15
2.2.5	Arquivo Manifest	15
3	PROPOSTA	16
3.1	Arquitetura	16
3.1.1	Cliente	16
3.1.2	Servidor	17
3.1.3	Limpeza dos dados	17
4	DESENVOLVIMENTO	18
4.1	Aplicativo	18
4.2	Google Docs	19
4.3	Limpeza dos dados	19
5	RESULTADOS	20

5.1	Dados Coletados	20
5.2	Análise	20
6	CONCLUSÃO	25
	Referências	26
	 ANEXOS	 27
	ANEXO A – CÓDIGO FONTE	28
A.1	MainActivity.java	28
A.2	HttpRequest.java	31
A.3	NetworkChangeReceiver.java	36
A.4	NetworkPoint.java	38
A.5	NetworkPointDAO.java	40
A.6	PersistenceHelper.java	44
A.7	UserEmailFetcher.java	45

1 Introdução

1.1 Contextualização

1.1.1 MANET

MANET é a abreviação de Mobile Ad-Hoc Networks, a qual se define pelas redes em que nós se movimentam livremente, comunicam entre si e entre o meio em que está, sem a necessidade de nenhuma infraestrutura de rede. Tais ações permitem que várias aplicações sejam desenvolvidas com objetivo por exemplo da troca de informações entre os nós ou até o compartilhamento de serviços, como por exemplo a funcionalidade que temos hoje em dia de rotear a internet do celular com dispositivos próximos.

Também definida por MANET pode ser conceituada como um sistema autônomo de servidores móveis (que também servem de roteadores), os quais unidos formam uma rede de comunicação representada por um grafo.

Os nós em uma MANET se movem arbitrariamente, e essa é a grande característica da rede, mas também o principal ofensor na qualidade da comunicação disponibilizada nesse ambiente, pois a arquitetura entre eles deve sempre estar se adaptando para os novos meios em que estão inseridos.

Desde que surgiu, a MANET tem sido vista como uma das mais desafiantes abordagens de redes móveis tanto pela promessa de crescimento de dispositivos móveis, quanto pela sua complexidade, o que impulsionou a pesquisa por esse paradigma. A partir das pesquisas intensas na MANET, surgiram outras redes móveis baseadas em sua proposta inicial. Como por exemplo a VANET que será introduzida na próxima seção.

1.1.2 VANET

1.1.2.1 Conceito

VANET são redes móveis MANET onde os nós são os veículos e até a estrada, onde há comunicação apenas entre os automóveis (IVC - Inter Vehicle Communication), ou entre os mesmos e a estrada (RVC - Road Vehicle Communication).

Assim como na MANET, a Vehicular Ad-Hoc Network também permite que várias aplicações com vários objetivos sejam desenvolvidos, bem como alertar obstáculos na estrada, broadcast de informações de segurança, compartilhar informações de tráfego ou até solicitar socorro em algum acidente.

As simulações e experimentos tem tido um papel importante no desenvolvimento

das soluções para redes veiculares. A atenção maior tem sido dada no desenvolvimento de modelos realistas de estrada e principalmente no estudo da mobilidade dos veículos. Por exemplo, modelos de como os carros se movem ao longo do trajeto, levando em consideração sua velocidade, sinais de trânsito, dentre outros fatores.

1.1.2.2 Problema

O grande desafio das Redes Ad-Hoc Veiculares é o roteamento entre os nós, devida a alta taxa de mobilidade dos veículos que são conectados entre si intermitentemente, dificultando a entrega das mensagens entre os nós ou estrada, pois frequentemente um nó muda seu endereço, ficando assim diferente de qual se identificou.

Além do problema citado acima, existe um agravante relacionado a velocidade de conexão entre os nós, os quais se comunicam muito rápido, levando em consideração que os mesmos estejam em uma via rápida.

1.1.3 Predição de localização

1.1.3.1 Motivo

A predição da localidade de um nó, o qual pode ser um automóvel, uma pessoa com um dispositivo móvel, é algo bastante motivador a se desenvolver pelo fato de trazer otimizações da comunicação das redes MANET e suas derivadas, através da gestão proativa. (??)

Por exemplo, caso um nó mande uma requisição para outro e logo após mude de localização, o responsável pela resposta será proativo o bastante para saber em qual posição o destinatário estará, resolvendo assim o problema da mobilidade, o qual é o principal desafio da VANET.

A predição de localização também é alvo de escopo de várias aplicações com o objetivo por exemplo de prever e otimizar tráfego em cidades inteligentes, ou até prever potenciais clientes para um passeio ecológico.

1.1.3.2 PheroCast

1.1.3.2.1 Conceito

PheroCast é o nome que se deu à abordagem de previsão da localização futura de um nó em uma rede ad-hoc móvel. Foi nomeado assim pela semelhança com o fenômeno que acontece na colônia das formigas, o qual para alertar problemas, ou marcar o caminho, por onde a formiga passar, ela deixará um rastro, formando assim um caminho.

Nessa abordagem, o algoritmo de previsão é desenvolvido para ser processado nos próprios nós móveis. Assim, a cada intervalo de tempo definido, o nó registra um rastro

de onde está. Daí vem a semelhança com o "Pheronomium" deixado no rastro por onde a formiga passa.

Assim, com o histórico dos "traces" de um determinado nó na forma de um grafo, é possível determinar uma probabilidade de onde ele estará em determinado horário a partir de um certo lugar.

1.1.3.2.2 Resultados

Segundo (??) foi feita uma avaliação do algoritmo PheroCast usando os caminhos dos onibus de Seattle Metro Transit, que vale a pena ressaltar, não foram extraídos para a abordagem nem com o intervalo de tempo ideal. Foi descoberto que a abordagem obteve bons resultados em relação à predição das futuras posições dos onibus. A avaliação obteve 77,8% de resultados positivos

1.1.3.2.3 Problema

O grande problema do PheroCast é a sua avaliação limitada, ou seja, é necessário, para uma abordagem com essas características, uma avaliação extensa com dados reais, que simulam de verdade a mobilidade humana.

Atualmente não existem "traces" reais, e sim módulos de mobilidade sintéticos, os quais não são realistas. Daí vem a proposta deste trabalho, o qual irá cooperar para avaliação confiável do PheroCast, coletando dados de mobilidade humana reais, através de seus dispositivos móveis.

1.2 Proposta

A proposta deste trabalho é desenvolver um sistema que rode em dispositivos móveis, para capturar "traces" reais de mobilidade humana. Para tal, o sistema irá coletar em um intervalo de tempo, todas as redes wireless, bem como suas informações, ao redor do dispositivo móvel do usuário e armazenar em um servidor para que os dados sejam compilados e utilizados para avaliação do PheroCast.

O objetivo é que o sistema seja utilizado por mais de um usuário para que assim tenhamos grande diversidade de histórico de traces.

2 Referencial Teórico

2.1 PheroCast

O PheroCast, conjunto de algoritmos para prever a localização de um nó em alguma rede móvel, foi desenvolvido por um grupo de pesquisa da Universidade Federal de Uberlândia, o The Distributed Systems and Networks Research Group, criado em 2013, constituído de doutores, mestres e alunos da Faculdade de Computação. citar grupo.

Nosso objetivo nessa seção não é de descrever o algoritmo bem como seus passos, cálculos e funções, e sim, explicar o conceito do algoritmo e seus principais componentes. Tais assuntos foram utilizados para o desenvolvimento deste trabalho.

Para entendermos a abordagem do Pherocast, que consiste na previsão da posição dos nós de uma rede móvel baseada no conceitos dos feromônios, vamos conceituar e exemplificar alguns componentes utilizados pelo algoritmo.

Como as formigas passam a vida em contato com o solo, em suas caminhadas, elas deixam um rastro de feromônio que pode ser seguida por outras formigas. Trazendo para o nosso contexto, um nó em uma rede móvel pode sinalizar os pontos em que passou, e assim construir um caminho que percorreu de um ponto a outro. Podemos também afirmar com esse conceito que um nó está em uma localização X se X é o sinal mais perto de sua posição.

Cada marca, ou log deixado pelo nó contém a informação da localização e a hora em que passou por ele, e a direção tornando assim análises como a velocidade do nó, ou a grandeza da localização. Por exemplo, caso um nó sinalize que está em uma localização em vários intervalos de tempo consecutivos, podemos concluir que essa localização, ou área de alcance é grande, ou que o nó está em baixa velocidade. Por outro lado, caso uma localização é pouco marcada, o nó pode estar em uma alta velocidade ou a área é pequena, onde pouco "feromônio" foi liberado.

No PheroCast esses sinais que contém a localização e a hora, são convertidos em Phero Trails, um dígrafo que contém como vértices os locais visitados pelos nós, e a direção em que estão (norte, sul, leste, oeste) ligados por arestas de forma em que represente a movimentação do nó, e assim seu caminho o qual possui uma localização inicial e uma final.

Com os Phero Trails definidos, temos a capacidade de utilizar um dos algoritmos do PheroCast e traçar o que chamamos de Phero Maps, dígrafos que contém além da localização e a direção, a quantidade de vezes que o nó passou por ali, ligados por arestas

que indicam a próxima localização que o nó passou, e assim por diante. Assim conseguimos concluir que para chegar em uma localização final, o nó passou por diversos caminhos diferentes, também definimos as possíveis rotas entre um determinado ponto e outro.

Dado um Phero Map definido, temos todas as variáveis que o algoritmo de previsão de uma futura localização do nó, descrita no Phero Cast necessita. Tais elas são: localização, direção, quantidade de vezes que passou pela localização.

O produto final desse trabalho, tem como objetivo produzir dados reais de localização, tratando os feromônios como registros de rede wifi, registrados pelo dispositivo móvel em um determinado intervalo de tempo, e armazenando em um local para ser reproduzidas em mapas e diagramas, como aborda o Phero Cast.

2.2 Android

Para o desenvolvimento desse trabalho, utilizamos o sistema operacional Android, baseado em Linux que opera em celulares, netbooks, tablets, dentre outros dispositivos. O Android nos fornece um robusto framework de aplicação que nos permite implementar aplicações para dispositivos móveis utilizando Java.

Tal framework disponibiliza alguns componentes, os quais alguns deles foram utilizados na construção desse trabalho. Iremos detalhar cada componente a seguir:

<http://developer.android.com/guide/components/fundamentals.html>

2.2.1 Activities

O componente Activity representa uma única tela com interface para o usuário. Por exemplo, imaginemos um aplicativo de lista de compras, iremos ter uma Activity para exibir a lista de compras, outra activity para adicionar um novo produto na lista de compras, e outra para exibir um relatório de quanto gastamos no mês. Uma Activity é implementada no código do aplicativo estendendo a classe Activity do Android.

2.2.2 Services

Service é o componente que roda em background no Android, podendo processar longas operações ou processar algum processo remoto. Um service não é responsável por fornecer interface para o usuário. Para entendermos melhor o funcionamento do Service, podemos exemplificar com o processo de tocar música enquanto se navega em outro aplicativo. Um Service é implementado no código do aplicativo estendendo a classe Service do Android.

2.2.3 Content Providers

Os Content Providers são parte importantíssima da arquitetura de um sistema android. É responsabilidade deles prover às aplicações o conteúdo que elas precisam para funcionar, ou seja, os dados.

As aplicações poderiam muito bem acessar diretamente um banco de dados, por exemplo. Porém, é uma boa prática tornar o modo como os dados são gravados transparente à aplicação.

Além disso, essa técnica permite a criação de Shared Content Providers, que são providers “públicos” que podem ser acessados por várias aplicações. Por exemplo, existe o content provider de SMS/MMS que permite a qualquer aplicação ler as mensagens recebidas por um telefone celular.

2.2.4 Broadcast receivers

O Broadcast Receiver é o componente responsável por receber as mensagens broadcast do Sistema Operacional, por exemplo anúncios de que a tela foi desligada, ou a bateria está com pouca carga, ou uma foto foi tirada, ou algo na rede foi alterado. Esse componente, não fornece telas para o usuário, mas é possível criar uma barra de notificação através dele.

2.2.5 Arquivo Manifest

Para que o Android consiga iniciar um componente da aplicação, o sistema deve saber que esse componente existe através do arquivo `AndroidManifest.xml`. A aplicação desenvolvida deve conter as declarações de todos os componentes nesse arquivo, o qual está sempre localizado na pasta raiz do projeto.

3 Proposta

3.1 Arquitetura

A arquitetura proposta consiste em duas partes, cliente e servidor. O cliente é responsável pela coleta dos dados, como posição atual, lista de SSID visíveis, e quaisquer outros dados que se deseje capturar. Oportunisticamente o cliente enviará os dados ao servidor. O servidor deverá consumir os dados enviados pelo dispositivo através de uma interface REST implementada e armazenar em um banco de dados, como mostra na Figura 1. A seguir detalhamos cada um das partes.



Figura 1 – Arquitetura Proposta

3.1.1 Cliente

Do lado do cliente desenvolveremos um sistema para rodar em dispositivos com o sistema operacional Android, utilizando a linguagem Java.

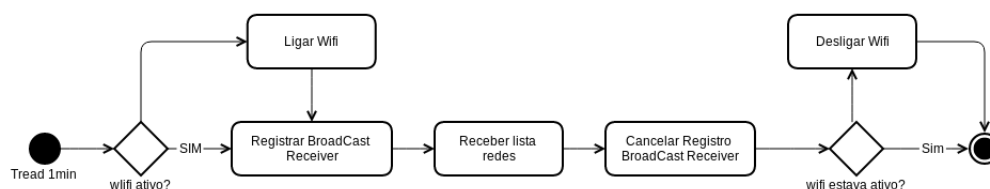


Figura 2 – Processo de captura dos dados pelo aplicativo.

O aplicativo irá utilizar os serviços de Wifi do aparelho para realizar a coleta dos pontos de rede que estão ao redor naquele determinado momento. Após coleta, o sistema irá armazenar em base de dados local do dispositivo, para isso utilizaremos o banco de dados SQLite. Tal procedimento é ilustrado na Figura 2.

Para realizar o envio ao servidor, o aplicativo será alertado quando o dispositivo se conectar a uma rede WiFi com rede Internet, assim irá enviar os dados armazenados em sua base local para o servidor de aplicação na nuvem e apagar os mesmos.

Caso o serviço de WiFi esteja desligado, o aplicativo deverá ser responsável por após ligar o Wifi e coletar os dados, desligar o mesmo, voltando assim para o estado inicial antes da coleta. Fazendo com que assim, não haja alteração no funcionamento do aparelho.

3.1.2 Servidor

Para o servidor, propomos utilizar algum servidor na nuvem de alta escalabilidade e alta disponibilidade, por se tratar de inserções contínuas todo o tempo.

O servidor deverá armazenar as informações virão do cliente, como:

- SSID
- BSSID
- Recursos
- Frequencia
- Level
- Horário
- Identificador do Usuário do Dispositivo

3.1.3 Limpeza dos dados

O sistema deverá tratar os dados do cliente com anonimidade, ou seja deverá haver algoritmos de segurança que tornem os dados do cliente anonimos, para preservar a identidade do usuário.

4 Desenvolvimento

4.1 Aplicativo

Como já mencionado, o produto final deste trabalho foi um aplicativo para dispositivos Android, o qual armazena periodicamente as redes wireless disponíveis em seu alcance, e disponibiliza em um certo momento essas informações na nuvem.

Para tal, utilizamos a API do Android para operações que envolveu o funcionamento do dispositivo, como por exemplo, a coleta das redes em alcance, o sinal de que foi encontrado uma rede com internet, a ação de desligar e ligar o receptor wifi do aparelho, dentre outras operações. Além disso, implementamos o código orientado a objetos, onde cada rede coletada foi representada por um objeto.

Foram utilizadas também classes que auxiliam nas funcionalidades do aplicativo, como por exemplo, o armazenamento em uma base de dados do dispositivo, o envio dos dados para a nuvem, a identificação do usuário pelo e-mail cadastrado no dispositivo.

Abaixo detalharemos cada classe do código, com o objetivo de entendermos a estrutura do desenvolvimento do aplicativo.

Começaremos pela MainActivity, classe estendida do componente Activity do Android, o qual foi detalhado no nosso referencial teórico, que é a nossa principal classe, ou seja código responsável pela inicialização e processamento do aplicativo, através do método onCreate(). Esse método cria uma tarefa agendada que roda em intervalos de um minuto. Tal tarefa, é responsável por verificar a situação do Wifi do aparelho, e, caso este esteja desligado, é ligado. Toda essa verificação do Wifi, é feita no método getWifiState().

Após tratar a situação do sinal wireless do aparelho, a tarefa registra através de um Intent, a classe WifiScanReceiver estendida do componente BroadcastReceiver, responsável por receber o resultado da varredura das redes wireless ao alcance do aparelho realizada.

A responsabilidade da coleta das redes sem fio é do objeto instanciado da classe WifiManager da API do Android, o qual é executado através do método startScan(), após a tarefa registrar o receptor das redes encontradas.

Ao receber a lista de redes através do método getScanResults da classe WifiManager, a classe WifiScanReceiver converte a lista recebida em um array de objetos da classe NetworkPoint, classe criada para representar a rede e seus atributos que são BSSID, SSID, Capabilities, Frequency, level e a hora que foi capturada. O array de NetworkPoint é salvo na base de dados, utilizando a classe NetworkPointDAO, criada em conjunto com a Per-

sistenceHelper, para auxiliar nas operações com o banco de dados do aplicativo, o qual foi utilizado SQLite.

Para armazenar na nuvem as redes coletadas, foi utilizada a classe NetworkChangeReceiver, estendida do componente BroadcastReceiver, que tem como objetivo principal enviar um POST para o serviço de armazenamento que utilizamos através da classe HttpRequest, caso o dispositivo conecte em alguma nova rede sem fio e que a mesma tenha acesso a internet. Para cada rede armazenada no dispositivo, uma requisição POST é realizada com a informação contendo o e-mail do usuário dono do dispositivo. Tal informação é adquirida através da classe userEmailFetcher. Após o envio das informações da rede, a mesma é excluída da base de dados do aplicativo no dispositivo.

4.2 Google Docs

Com o objetivo de simular o serviço de armazenamento na nuvem, utilizamos o Google Docs, o qual nos fornece um serviço de criação, edição e utilização de formulários de pesquisa, onde as respostas são convertidas em planilhas compartilhadas. Assim, na nossa requisição POST, com a mesma URL e parâmetros que o botão de responder o formulário que criamos com as informações esperadas das redes. Com isso, obtivemos várias linhas em planilha contendo informações das redes sem fio, a hora em que foram capturadas e o usuário responsável pela coleta.

4.3 Limpeza dos dados

Como tópico de trabalhos futuros, pretendemos utilizar um codificador do dado que carrega as informações do usuário, tornando assim os dados limpos e possíveis de serem colaborados com a comunidade em trabalhos que necessitem de dados reais, como no caso do PheroCast.

5 Resultados

5.1 Dados Coletados

Após divulgação do trabalho para amigos e comunidade, obtivemos mais de 45 mil registros de redes coletadas em um período de aproximadamente um mês. Compilamos os dados com apenas 5 usuários, os quais foram mais colaborativos em questão de quantidade e consideramos o suficiente para analisar os dados. Com os dados coletados e separados por colaborador compilados em uma planilha, conseguimos o suficiente para realizar algumas análises sobre as informações adquiridas. Tais análises, serão exibidas na próxima seção.

5.2 Análise

Por se tratar de uma quantidade grande de dados coletados, poderíamos ter realizado diversas análises com propósitos diferentes, mas escolhemos algumas que mais relacionam com o propósito deste trabalho. Por exemplo, na Análise 1, consideramos que a rede "ALGARTELECOMVISITANTES" seja a do local de trabalho, e a rede "Lar Doce Lar" seja a de sua residência, temos um cenário onde o usuário se locomoveu no dia de um local para outro no dia 30/07/2014, coletando e armazenando as redes sem fio ao seu alcance.

Provavelmente, o usuário do dispositivo utiliza de um caminho afastado da zona residencial, onde poucas redes foram capturadas, ou a velocidade que estava percorrendo o caminho era alta.

Já na Análise 2, com redes coletadas no dia 30/07, consideramos que a rede capturada em seu local de trabalho seja a UFU-Portal e em sua residência a "nao e sua", e tivemos um percurso com mais redes coletadas em um intervalo de tempo semelhante com a análise apresentada na Análise 1. Podemos afirmar então que o usuário da Análise 2 percorreu por um caminho com mais redes sem fio, ou que estava em baixa velocidade permitindo maior frequência de varredura das redes ao redor.

Para considerarmos que um local é o trabalho do usuário e outro é a residência, analisamos a repetição da rede coletada em um determinado intervalo de horário, por exemplo, se no horário comercial um conjunto de redes X é frequentemente capturado, pode-se afirmar que essa localização provavelmente é onde o dono do dispositivo trabalha.

SSID	BSSID	FREQUENCY	LEVEL	Capturada em	ID
ALGARTELECOM_VISITANTE	00:1f:ca:5f:12:73	2412	-93	30/07/2014 06:04:41	godinhopatrck@gmail.com
Invicta	28:10:7b:3c:31:84	2437	-75	30/07/2014 06:04:41	godinhopatrck@gmail.com
Maurilio	10:fe:ed:c8:ab:e6	2412	-90	30/07/2014 06:04:41	godinhopatrck@gmail.com
Tudo vidros	c8:d7:19:eb:f6:c7	2462	-93	30/07/2014 06:04:41	godinhopatrck@gmail.com
b0c76a	54:be:f7:70:75:4b	2462	-93	30/07/2014 06:04:41	godinhopatrck@gmail.com
PREMIUMHFC	bc:f6:85:41:b3:b0	2462	-93	30/07/2014 06:04:41	godinhopatrck@gmail.com
CTBC	9c:d6:43:7a:fd:e3	2432	-93	30/07/2014 06:04:41	godinhopatrck@gmail.com
Tudo vidros	c8:d7:19:eb:f6:c7	2462	-93	30/07/2014 06:04:41	godinhopatrck@gmail.com
HP-Print-4C-LaserJet 1102	f4:b7:e2:3c:67:4c	2437	-81	30/07/2014 06:04:41	godinhopatrck@gmail.com
Tudo vidros	c8:d7:19:eb:f6:c7	2462	-93	30/07/2014 06:04:41	godinhopatrck@gmail.com
CTBC	9c:d6:43:7a:fd:e3	2432	-93	30/07/2014 06:04:41	godinhopatrck@gmail.com
Invicta	28:10:7b:3c:31:84	2437	-75	30/07/2014 06:04:41	godinhopatrck@gmail.com
ALGARTELECOM_VISITANTE	00:1f:ca:5f:12:73	2412	-93	30/07/2014 06:04:41	godinhopatrck@gmail.com
SPEED_STM2	00:0c:42:26:73:39	2417	-90	30/07/2014 06:16:03	godinhopatrck@gmail.com
SPEED_STM2	00:0c:42:26:73:39	2417	-90	30/07/2014 06:16:03	godinhopatrck@gmail.com
HM2 - FONE: 3084-9493	00:80:48:73:fb:db	2432	-93	30/07/2014 06:23:01	godinhopatrck@gmail.com
HM2 - FONE: 3084-9493	00:80:48:73:fb:db	2432	-93	30/07/2014 06:23:01	godinhopatrck@gmail.com
CTBC	9c:d6:43:7a:bc:0b	2447	-93	30/07/2014 06:25:45	godinhopatrck@gmail.com
Amanda	c0:a0:bb:0f:68:de	2447	-93	30/07/2014 06:25:45	godinhopatrck@gmail.com
CTBC	9c:d6:43:7a:bc:0b	2447	-93	30/07/2014 06:25:45	godinhopatrck@gmail.com
Amanda	c0:a0:bb:0f:68:de	2447	-93	30/07/2014 06:25:45	godinhopatrck@gmail.com
Lar Doce Lar	c0:a0:bb:0e:fa:1e	2447	-66	30/07/2014 06:28:28	godinhopatrck@gmail.com

Figura 3 – Análise 1 - Poucas redes coletadas em percurso

SSID	BSSID	FREQUENCY	LEVEL	Capturada em	ID
UFU-Portal	d8:c7:c8:d5:79:41	2437	-71	04/08/2014 05:36:4	lasaro@gmail.com
UFU-Portal	d8:c7:c8:d5:79:41	2437	-71	04/08/2014 05:36:4	lasaro@gmail.com
DaComp	f8:1a:67:13:79:90	2437	-91	04/08/2014 05:36:4	lasaro@gmail.com
dlink	00:21:91:6f:18:00	2437	-93	04/08/2014 05:36:4	lasaro@gmail.com
UFU-Portal	d8:c7:c8:d5:78:a1	2412	-76	04/08/2014 05:36:4	lasaro@gmail.com
UFU-Portal	d8:c7:c8:d5:78:a1	2412	-76	04/08/2014 05:36:4	lasaro@gmail.com
OpenWrt	90:f6:52:3e:ca:58	2447	-77	04/08/2014 05:36:4	lasaro@gmail.com
OpenWrt	90:f6:52:3e:ca:58	2447	-77	04/08/2014 05:36:4	lasaro@gmail.com
BENARENCE	00:1d:1a:0a:91:ea	2462	-90	04/08/2014 05:45:0	lasaro@gmail.com
MM GAMA	20:aa:4b:a9:b7:57	2462	-93	04/08/2014 05:45:0	lasaro@gmail.com
b7b380	54:be:f7:87:b0:a7	2462	-82	04/08/2014 05:45:0	lasaro@gmail.com
ultrabl_1149_20m	20:aa:4b:4f:a9:ca	2437	-84	04/08/2014 05:45:0	lasaro@gmail.com
b13c34	54:be:f7:71:bc:bd	2462	-86	04/08/2014 05:45:0	lasaro@gmail.com
22ks	90:f6:52:68:7c:c2	2437	-90	04/08/2014 05:45:0	lasaro@gmail.com
VERTICAL_FILMES	20:aa:4b:d0:62:c3	2462	-91	04/08/2014 05:45:0	lasaro@gmail.com
WJS	2a:a4:3c:7d:48:52	2437	-84	04/08/2014 05:45:0	lasaro@gmail.com
CLIENTE-DGUSTO	90:f6:52:a6:36:32	2437	-90	04/08/2014 05:45:0	lasaro@gmail.com
Serafim	d8:5d:4c:d6:40:2c	2442	-93	04/08/2014 05:45:0	lasaro@gmail.com
b0ed44	54:be:f7:6c:51:7b	2412	-90	04/08/2014 05:45:0	lasaro@gmail.com
enio	c8:be:19:01:b8:a8	2457	-92	04/08/2014 05:45:0	lasaro@gmail.com
NetVirtua_582	90:0d:cb:f4:97:00	2462	-92	04/08/2014 05:45:0	lasaro@gmail.com
c7917c	70:54:d2:c7:03:04	2462	-92	04/08/2014 05:45:0	lasaro@gmail.com
abacaxi	70:54:d2:c3:4c:43	2462	-94	04/08/2014 05:45:0	lasaro@gmail.com
Marcio D-Link	28:10:7b:3e:23:fb	2437	-88	04/08/2014 05:45:0	lasaro@gmail.com
c7184a	70:54:d2:c6:16:bf	2412	-88	04/08/2014 05:45:0	lasaro@gmail.com
CAETES02	98:fc:11:ce:e6:ae	2452	-93	04/08/2014 05:45:0	lasaro@gmail.com
Viveiro Rosalia	9c:d6:43:7b:51:d3	2447	-85	04/08/2014 05:49:3	lasaro@gmail.com
EDC	20:aa:4b:d8:84:f3	2412	-85	04/08/2014 05:49:3	lasaro@gmail.com
Bom Jesus	1c:7e:e5:99:68:10	2442	-87	04/08/2014 05:49:3	lasaro@gmail.com
fransergio de camargo	84:c9:b2:cc:ce:3d	2437	-91	04/08/2014 05:49:3	lasaro@gmail.com
c6eb7a	70:54:d2:c4:25:27	2462	-92	04/08/2014 05:49:3	lasaro@gmail.com
versii_ap2	94:0c:6d:aa:1e:e6	2437	-88	04/08/2014 05:49:3	lasaro@gmail.com
HarryPotter	64:66:b3:da:c7:95	2422	-93	04/08/2014 05:49:3	lasaro@gmail.com
Advocacia	00:1d:0f:d5:e2:de	2437	-93	04/08/2014 05:49:3	lasaro@gmail.com
Nasa Hi-Fi	b0:48:7a:f2:2a:5a	2452	-91	04/08/2014 05:49:3	lasaro@gmail.com
HP-Print-96-LaserJet 1102	1c:3e:84:f5:ef:96	2437	-93	04/08/2014 05:49:3	lasaro@gmail.com
vargas	20:aa:4b:d0:63:32	2437	-90	04/08/2014 05:49:3	lasaro@gmail.com
Francine	c8:d7:19:60:95:90	2437	-94	04/08/2014 05:49:3	lasaro@gmail.com
ALINEMAMEDE-PC_Networ	54:e6:fc:a4:d8:48	2427	-91	04/08/2014 05:49:3	lasaro@gmail.com
Consultorio	78:54:2e:ee:28:80	2437	-89	04/08/2014 05:54:1	lasaro@gmail.com
DIRECT-KJSCX-3400 Series	02:15:99:d0:07:09	2437	-92	04/08/2014 05:54:1	lasaro@gmail.com
ATENDI	b8:62:1f:52:8f:36	2462	-81	04/08/2014 05:54:1	lasaro@gmail.com
Exclusiviagem	20:aa:4b:55:5b:bc	2462	-86	04/08/2014 05:54:1	lasaro@gmail.com
c6f1fe	70:54:d2:c1:63:1d	2412	-70	04/08/2014 05:54:1	lasaro@gmail.com
CRECI_UDI	f4:ec:38:b5:66:36	2452	-85	04/08/2014 05:54:1	lasaro@gmail.com
Bambola Pizzaria	64:66:b3:82:1c:c6	2457	-86	04/08/2014 05:54:1	lasaro@gmail.com
wifi-castro	00:1d:7e:f8:6d:09	2437	-88	04/08/2014 05:54:1	lasaro@gmail.com

Figura 4 – Análise 2 - Parte 1 - Muitas redes coletadas em percurso

navegantes	08:10:74:32:8f:ba	2412	-84	04/08/2014 05:54:1	lasaro@gmail.com
c6fcd2	70:54:d2:c3:44:ab	2437	-94	04/08/2014 05:54:1	lasaro@gmail.com
Exclusiviagem	20:aa:4b:55:5b:bc	2462	-91	04/08/2014 05:58:0	lasaro@gmail.com
c6f1fe	70:54:d2:c1:63:1d	2412	-63	04/08/2014 05:58:0	lasaro@gmail.com
navegantes	08:10:74:32:8f:ba	2412	-92	04/08/2014 05:58:0	lasaro@gmail.com
c6fcd2	70:54:d2:c3:44:ab	2437	-88	04/08/2014 05:58:0	lasaro@gmail.com
TELEMACO2013	20:aa:4b:55:5c:67	2462	-92	04/08/2014 05:58:0	lasaro@gmail.com
wifi-castro	00:1d:7e:f8:6d:09	2437	-92	04/08/2014 05:58:0	lasaro@gmail.com
CRECI_UDI	f4:ec:38:b5:66:36	2452	-94	04/08/2014 05:58:0	lasaro@gmail.com
ATENDI	b8:62:1f:52:8f:36	2462	-92	04/08/2014 05:58:0	lasaro@gmail.com
wifi-castro	00:1d:7e:f8:6d:09	2437	-92	04/08/2014 05:58:0	lasaro@gmail.com
SalaTv	78:54:2e:ee:28:8c	2462	-93	04/08/2014 05:58:0	lasaro@gmail.com
Consultorio	78:54:2e:ee:28:80	2437	-87	04/08/2014 06:04:1	lasaro@gmail.com
Abgail	20:aa:4b:b4:86:17	2412	-84	04/08/2014 06:04:1	lasaro@gmail.com
CRECI_UDI	f4:ec:38:b5:66:36	2452	-78	04/08/2014 06:04:1	lasaro@gmail.com
guarita unifi	24:a4:3c:7a:ce:2b	2472	-85	04/08/2014 06:04:1	lasaro@gmail.com
residencia unifi	2a:a4:3c:0d:23:1e	2437	-87	04/08/2014 06:04:1	lasaro@gmail.com
b71b76	54:be:f7:6f:f8:bb	2412	-92	04/08/2014 06:04:1	lasaro@gmail.com
rachel_sala	c0:4a:00:6b:d6:ec	2462	-93	04/08/2014 06:04:1	lasaro@gmail.com
DIRECT-CHSCX-3400 Serie	32:cd:a7:5d:1d:79	2462	-81	04/08/2014 06:04:1	lasaro@gmail.com
RAZZAO	a0:f3:c1:a3:c7:ea	2437	-90	04/08/2014 06:04:1	lasaro@gmail.com
wifi-castro	00:1d:7e:f8:6d:09	2437	-92	04/08/2014 06:04:1	lasaro@gmail.com
b113f6	54:be:f7:71:78:d1	2462	-71	04/08/2014 06:09:4	lasaro@gmail.com
estancia santa rosalia	20:aa:4b:55:60:cf	2412	-78	04/08/2014 06:09:4	lasaro@gmail.com
NADSON	c8:3a:35:11:2f:78	2412	-83	04/08/2014 06:09:4	lasaro@gmail.com
Advocacia	00:1d:0f:d5:e2:de	2437	-85	04/08/2014 06:09:4	lasaro@gmail.com
Danger_zone	14:d6:4d:83:01:a4	2437	-86	04/08/2014 06:09:4	lasaro@gmail.com
ULTRABL_20MB_1315	c8:d7:19:eb:ec:44	2412	-87	04/08/2014 06:09:4	lasaro@gmail.com
cris e rique	10:fe:ed:32:ef:7a	2437	-85	04/08/2014 06:09:4	lasaro@gmail.com
advogados_	70:54:d2:c4:06:67	2462	-92	04/08/2014 06:09:4	lasaro@gmail.com
GVT-F26F	28:10:7b:96:f2:70	2462	-87	04/08/2014 06:09:4	lasaro@gmail.com
HP-Print-96-LaserJet 1102	1c:3e:84:f5:ef:96	2437	-81	04/08/2014 06:09:4	lasaro@gmail.com
inove	bc:f6:85:de:d0:a1	2437	-78	04/08/2014 06:09:4	lasaro@gmail.com
	00:24:82:22:f6:ba	2462	-81	04/08/2014 06:09:4	lasaro@gmail.com
familhareis	20:aa:4b:de:5d:cb	2462	-92	04/08/2014 06:09:4	lasaro@gmail.com
WIFI-UNIUBE 2	00:24:82:62:f6:b8	2462	-81	04/08/2014 06:09:4	lasaro@gmail.com
WIFI-UNIUBE	00:24:82:22:f6:b8	2462	-81	04/08/2014 06:09:4	lasaro@gmail.com
belkin54g	00:1c:df:98:3d:e4	2412	-89	04/08/2014 06:09:4	lasaro@gmail.com
Rone	a0:f3:c1:46:da:7f	2412	-78	04/08/2014 06:09:4	lasaro@gmail.com
Ultra Telecom	00:01:e3:e2:79:f6	2422	-81	04/08/2014 06:09:4	lasaro@gmail.com
Play music bar	f8:1a:67:c4:fb:70	2462	-76	04/08/2014 06:09:4	lasaro@gmail.com
Loranne	9c:d6:43:7a:dc:13	2462	-84	04/08/2014 06:09:4	lasaro@gmail.com
Kamel_Home	00:4f:67:02:d9:e3	2412	-79	04/08/2014 06:09:4	lasaro@gmail.com
MEROLA	00:1c:10:57:94:90	2437	-88	04/08/2014 06:09:4	lasaro@gmail.com
PaisagemBrasil	00:25:86:c7:d7:02	2437	-90	04/08/2014 06:09:4	lasaro@gmail.com
Rede	78:52:1a:7c:d9:b9	2462	-84	04/08/2014 06:09:4	lasaro@gmail.com
MMCV38	90:f6:52:ef:4c:a8	2427	-89	04/08/2014 06:09:4	lasaro@gmail.com
Amanda	20:aa:4b:55:5c:61	2412	-90	04/08/2014 06:09:4	lasaro@gmail.com

Figura 5 – Análise 2 - Parte 2 - Muitas redes coletadas em percurso

MURILO	c8:be:19:8a:86:96	2422	-93	04/08/2014 06:09:4	lasaro@gmail.com
alexandre Freitas	78:44:76:07:5b:c1	2462	-79	04/08/2014 06:09:4	lasaro@gmail.com
Ana_Carolina	10:fe:ed:33:17:7a	2437	-90	04/08/2014 06:09:4	lasaro@gmail.com
c6f1ec	70:54:d2:c1:a0:31	2412	-82	04/08/2014 06:12:5	lasaro@gmail.com
MARIO 3.0	70:54:d2:b4:a5:d2	2462	-85	04/08/2014 06:12:5	lasaro@gmail.com
Yuji	74:ea:3a:fc:3b:58	2427	-86	04/08/2014 06:12:5	lasaro@gmail.com
VELOSO	00:4f:81:03:ce:b0	2437	-83	04/08/2014 06:12:5	lasaro@gmail.com
paulocesar	70:54:d2:c1:c1:dd	2462	-90	04/08/2014 06:12:5	lasaro@gmail.com
Proservice piso2	34:08:04:c0:5f:1c	2437	-90	04/08/2014 06:12:5	lasaro@gmail.com
c78666	70:54:d2:c3:bd:cf	2412	-88	04/08/2014 06:12:5	lasaro@gmail.com
viera fa	a0:f3:c1:43:2e:88	2427	-79	04/08/2014 06:12:5	lasaro@gmail.com
Fred	10:bf:48:8f:b3:5c	2462	-87	04/08/2014 06:12:5	lasaro@gmail.com
b76766	54:be:f7:6f:c1:87	2412	-81	04/08/2014 06:12:5	lasaro@gmail.com
Cavalo de Troia	70:54:d2:c1:95:fd	2412	-81	04/08/2014 06:12:5	lasaro@gmail.com
c7832a	70:54:d2:c1:60:81	2437	-82	04/08/2014 06:12:5	lasaro@gmail.com
VELOSO	54:be:f7:6d:0a:cf	2437	-79	04/08/2014 06:12:5	lasaro@gmail.com
Netvirtua_1366	e0:ce:c3:d6:ff:ce	2412	-87	04/08/2014 06:12:5	lasaro@gmail.com
c6f156	70:54:d2:c3:5f:1f	2462	-88	04/08/2014 06:12:5	lasaro@gmail.com
nao e a sua	54:be:f7:87:b7:ef	2462	-54	04/08/2014 06:12:5	lasaro@gmail.com
fmm	70:54:d2:c3:60:1b	2462	-78	04/08/2014 06:12:5	lasaro@gmail.com
c6e38e	70:54:d2:c2:fa:cf	2437	-90	04/08/2014 06:12:5	lasaro@gmail.com
spoof net	ec:1a:59:cf:ab:f8	2462	-53	04/08/2014 06:12:5	lasaro@gmail.com
LULUZINHAS	70:54:d2:c1:cd:99	2437	-87	04/08/2014 06:12:5	lasaro@gmail.com
b7692e	54:be:f7:6f:c5:53	2437	-76	04/08/2014 06:12:5	lasaro@gmail.com
c74646	70:54:d2:c3:bc:b7	2437	-85	04/08/2014 06:12:5	lasaro@gmail.com
c6ea7e	70:54:d2:c2:f2:af	2437	-90	04/08/2014 06:12:5	lasaro@gmail.com
VELOSO	00:4f:81:03:ce:b0	2437	-92	04/08/2014 06:17:0	lasaro@gmail.com
spoof net	ec:1a:59:cf:ab:f8	2462	-82	04/08/2014 06:17:0	lasaro@gmail.com
nao e a sua	54:be:f7:87:b7:ef	2462	-70	04/08/2014 06:17:0	lasaro@gmail.com
b7692e	54:be:f7:6f:c5:53	2437	-85	04/08/2014 06:17:0	lasaro@gmail.com
c74646	70:54:d2:c3:bc:b7	2437	-93	04/08/2014 06:17:0	lasaro@gmail.com
c6ea7e	70:54:d2:c2:f2:af	2437	-89	04/08/2014 06:17:0	lasaro@gmail.com
c6e118	70:54:d2:c3:57:3f	2412	-92	04/08/2014 06:17:0	lasaro@gmail.com
c70416	70:54:d2:ad:d2:3b	2437	-88	04/08/2014 06:17:0	lasaro@gmail.com
VELOSO	00:4f:81:03:ce:b0	2437	-89	04/08/2014 06:18:0	lasaro@gmail.com
spoof net	ec:1a:59:cf:ab:f8	2462	-77	04/08/2014 06:18:0	lasaro@gmail.com
nao e a sua	54:be:f7:87:b7:ef	2462	-83	04/08/2014 06:18:0	lasaro@gmail.com

Figura 6 – Análise 2 - Parte 3 - Muitas redes coletadas em percurso

6 Conclusão

Com o desenvolvimento desse trabalho, concluímos que existe um potencial enorme em pesquisas no campo de redes móveis, principalmente no trabalho base seguido, o PheroCast. Ficamos satisfeitos com o resultado e em poder contribuir com a pesquisa de predição da localização de um nó em uma rede móvel com os dados capturados através do produto final deste trabalho, o aplicativo PheroCast App.

Para trabalhos futuros, pretendemos implementar também a coleta e armazenamento das localizações por GPS, capturadas também pelo dispositivo móvel. Também fica como sugestão, outras análises mais criteriosas que possam contribuir mais ainda para a comunidade.

Para melhorar mais ainda o trabalho, indicamos também um serviço de armazenamento na nuvem com uma base de dados mais robusta, a fim de gerar relatórios mais analíticos, pois o Google Docs, serviço utilizado nesse trabalho, não atende uma grande demanda de dados, visto que com os dados coletados neste trabalho a performance na operação das planilhas é bem pequena.

Concluímos também que a arquitetura, implementação e desenvolvimento do aplicativo bem como a produção das análises com os dados produzidos pela utilização do software por algumas pessoas da comunidade, foi consequência da base de conhecimento adquirida durante o curso de Sistemas de Informação da Universidade Federal de Uberlândia, bem como o tempo de pesquisa aplicado nesse trabalho. Tais ações e resultados implicam no sucesso deste trabalho de conclusão de curso.

Referências

Anexos

ANEXO A – Código fonte

A.1 MainActivity.java

```
1 package ufu.tcc.patrick.pherocast;
2 import android.annotation.SuppressLint;
3 import android.content.BroadcastReceiver;
4 import android.content.Context;
5 import android.content.Intent;
6 import android.content.IntentFilter;
7 import android.net.wifi.ScanResult;
8 import android.net.wifi.WifiManager;
9 import android.os.Bundle;
10 import android.os.Handler;
11 import android.support.v7.app.ActionBarActivity;
12 import android.widget.TextView;
13 import android.widget.Toast;
14
15 import java.io.PrintStream;
16 import java.text.SimpleDateFormat;
17 import java.util.ArrayList;
18 import java.util.Date;
19 import java.util.List;
20 import java.util.Timer;
21 import java.util.TimerTask;
22
23 public class MainActivity extends ActionBarActivity {
24     final Handler handler = new Handler();
25     boolean scanInitiated;
26     TimerTask scanTask;
27     Timer t = new Timer();
28     TextView text;
29     WifiManager wifi;
30     boolean wifiOn;
31     WifiScanReceiver wifiReciever;
32     String[] wifis;
33     List<ScanResult> localList = new ArrayList<ScanResult>();
34
35     private String pegarHoraAtual() {
```

```
36         return new SimpleDateFormat("dd/MM/yyyy hh:mm:ss").format
           (new Date());
37     }
38
39     public void getWifiState() {
40         System.out.println("passou aqui " + wifi.isWifiEnabled())
           ;
41         if (!wifi.isWifiEnabled()) {
42             wifiOn = false;
43             Toast.makeText(getApplicationContext(),
44                 "Wifi desativado, estamos ativando...", 1).
               show();
45             wifi.setWifiEnabled(true);
46             return;
47         }
48         wifiOn = true;
49     }
50
51     public void onBackPressed() {
52         moveTaskToBack(true);
53     }
54
55     protected void onCreate(Bundle paramBundle) {
56         super.onCreate(paramBundle);
57         setContentView(2130903064);
58         wifi = ((WifiManager) getSystemService("wifi"));
59         wifiReciever = new WifiScanReceiver();
60         scanTask = new TimerTask() {
61             public void run() {
62                 handler.post(new Runnable() {
63                     public void run() {
64                         getWifiState();
65                         registerReceiver(wifiReciever, new
66                             IntentFilter(
67                                 "android.net.wifi.SCAN_RESULTS"))
68                             ;
69                         scanInitiated = true;
70                         wifi.startScan();
71                     }
72                 });
73             }
74         };
75     }
76 }
```

```
73         t.schedule(scanTask, 300L, 60000L);
74     }
75
76     class WifiScanReceiver extends BroadcastReceiver {
77         WifiScanReceiver() {
78         }
79
80         @SuppressWarnings({ "UseValueOf", "NewApi" })
81         public void onReceive(Context paramContext, Intent
            paramIntent) {
82
83             NetworkPointDAO localNetworkPointDAO =
            NetworkPointDAO
84                 .getInstance(getBaseContext());
85             if (scanInitiated) {
86                 localList = wifi.getScanResults();
87                 wifis = new String[localList.size()];
88
89             }
90             for (int i = 0;; i++) {
91                 if (i >= localList.size()) {
92                     scanInitiated = false;
93                     System.out.println(wifiOn);
94                     if (!wifiOn)
95                         wifi.setWifiEnabled(false);
96                     unregisterReceiver(this);
97                     return;
98                 }
99                 wifis[i] = (((ScanResult) localList.get(i)).SSID
            + ", "
100                     + ((ScanResult) localList.get(i)).
            frequency + ", "
101                     + ((ScanResult) localList.get(i)).level +
            ", " + ((ScanResult) localList
102                     .get(i)).BSSID);
103                 localNetworkPointDAO
104                     .salvar(new NetworkPoint(
105                     ((ScanResult) localList.get(i)).
            BSSID,
106                     ((ScanResult) localList.get(i)).
            SSID,
107                     ((ScanResult) localList.get(i)).
```

```
108         capabilities ,
109         ((ScanResult) localList.get(i)).
            frequency ,
110         ((ScanResult) localList.get(i)).
            level ,
            pegarHoraAtual()));
111     }
112 }
113 }
114 }
```

A.2 HttpRequest.java

```
1 package ufu.tcc.patrick.pherocast;
2
3
4 import java.io.IOException;
5 import java.io.InputStream;
6 import java.io.UnsupportedEncodingException;
7 import java.net.HttpURLConnection;
8 import java.net.URL;
9 import java.net.URLConnection;
10 import java.net.URLEncoder;
11
12 import org.apache.http.HttpResponse;
13 import org.apache.http.client.methods.HttpGet;
14 import org.apache.http.client.methods.HttpPost;
15 import org.apache.http.client.params.ClientPNames;
16 import org.apache.http.client.params.CookiePolicy;
17 import org.apache.http.entity.StringEntity;
18 import org.apache.http.impl.client.DefaultHttpClient;
19 import org.apache.http.params.BasicHttpParams;
20 import org.apache.http.params.HttpConnectionParams;
21 import org.apache.http.params.HttpParams;
22 import org.apache.http.protocol.BasicHttpContext;
23 import org.apache.http.protocol.HttpContext;
24 import org.apache.http.util.EntityUtils;
25 import org.json.JSONObject;
26
27 import android.util.Log;
28
29 /*
```



```
30  * This helper class was created by StackOverflow user: MattC
    http://stackoverflow.com/users/21126/mattc
31  * IT was posted as an Answer to this question: http://
    stackoverflow.com/questions/2253061/secure-http-post-in-
    android
32  */
33
34  public class HttpRequest{
35
36      DefaultHttpClient httpClient;
37      HttpContext localContext;
38      private String ret;
39
40      HttpResponse response = null;
41      HttpPost httpPost = null;
42      HttpGet httpGet = null;
43
44      public HttpRequest(){
45          HttpParams myParams = new BasicHttpParams();
46
47          HttpConnectionParams.setConnectionTimeout(myParams,
              10000);
48          HttpConnectionParams.setSoTimeout(myParams, 10000);
49          httpClient = new DefaultHttpClient(myParams);
50          localContext = new BasicHttpContext();
51      }
52
53      public void clearCookies() {
54          httpClient.getCookieStore().clear();
55      }
56
57      public void abort() {
58          try {
59              if (httpClient != null) {
60                  System.out.println("Abort.");
61                  httpPost.abort();
62              }
63          } catch (Exception e) {
64              System.out.println("Your App Name Here" + e);
65          }
66      }
67  }
```

```
68     public String sendPost(String url, String data) {
69         return sendPost(url, data, null);
70     }
71
72     public String sendJSONPost(String url, JSONObject data) {
73         return sendPost(url, data.toString(), "application/json");
74         ;
75     }
76
77     public String sendPost(String url, String data, String
78         contentType) {
79         ret = null;
80
81         httpClient.getParams().setParameter(ClientPNames.
82             COOKIE_POLICY, CookiePolicy.RFC_2109);
83
84         httpPost = new HttpPost(url);
85         response = null;
86
87         StringEntity tmp = null;
88
89         Log.d("Your App Name Here", "Setting httpPost headers");
90
91         httpPost.setHeader("User-Agent", "SET YOUR USER AGENT
92             STRING HERE");
93         httpPost.setHeader("Accept", "text/html,application/xml,
94             application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,
95             image/png,*;q=0.5");
96
97         if (contentType != null) {
98             httpPost.setHeader("Content-Type", contentType);
99         } else {
100             httpPost.setHeader("Content-Type", "application/x-www
101                 -form-urlencoded");
102         }
103
104         try {
105             tmp = new StringEntity(data,"UTF-8");
106         } catch (UnsupportedEncodingException e) {
107             Log.e("Your App Name Here", "HttpUtils :
108                 UnsupportedEncodingException : "+e);
109         }
110     }
```

```
102
103     httpPost.setEntity(tmp);
104
105     Log.d("Your App Name Here", url + "?" + data);
106
107     try {
108         response = httpClient.execute(httpPost,localContext);
109
110         if (response != null) {
111             ret = EntityUtils.toString(response.getEntity());
112         }
113     } catch (Exception e) {
114         Log.e("Your App Name Here", "HttpUtils: " + e);
115     }
116
117     Log.d("Your App Name Here", "Returning value:" + ret);
118
119     return ret;
120 }
121
122 public String sendGet(String url) {
123     httpGet = new HttpGet(url);
124
125     try {
126         response = httpClient.execute(httpGet);
127     } catch (Exception e) {
128         Log.e("Your App Name Here", e.getMessage());
129     }
130
131     //int status = response.getStatusLine().getStatusCode();
132
133     // we assume that the response body contains the error
134     // message
135     try {
136         ret = EntityUtils.toString(response.getEntity());
137     } catch (IOException e) {
138         Log.e("Your App Name Here", e.getMessage());
139     }
140
141     return ret;
142 }
```

```
143     public InputStream getHttpStream(String urlString) throws
        IOException {
144         InputStream in = null;
145         int response = -1;
146
147         URL url = new URL(urlString);
148         URLConnection conn = url.openConnection();
149
150         if (!(conn instanceof HttpURLConnection))
151             throw new IOException("Not an HTTP connection");
152
153         try{
154             HttpURLConnection httpConn = (HttpURLConnection) conn
                ;
155             httpConn.setAllowUserInteraction(false);
156             httpConn.setInstanceFollowRedirects(true);
157             httpConn.setRequestMethod("GET");
158             httpConn.connect();
159
160             response = httpConn.getResponseCode();
161
162             if (response == HttpURLConnection.HTTP_OK) {
163                 in = httpConn.getInputStream();
164             }
165         } catch (Exception e) {
166             throw new IOException("Error connecting");
167         } // end try-catch
168
169         return in;
170     }
171
172
173     public void postData() {
174         String fullUrl = "https://docs.google.com/forms/d/1
            AYvVOgFgB1hBuoRKnMsXy1LyF8-Ce8VAshAtho6Z08s/
            formResponse";
175         HttpRequest mReq = new HttpRequest();
176         String col1 = "Hello";
177         String col2 = "World";
178
179         String data = "entry.1680144410=" + URLEncoder.encode(
            col1) + "&" +
```

```
180         "entry.1558298396=" + URLEncoder.encode(
181             col2);
182     String response = mReq.sendPost(fullUrl, data);
183     Log.i("DocsUpload", response);
184 }
```

A.3 NetworkChangeReceiver.java

```
1 package ufu.tcc.patrick.pherocast;
2
3 import android.content.BroadcastReceiver;
4 import android.content.Context;
5 import android.content.Intent;
6 import android.net.ConnectivityManager;
7 import android.net.NetworkInfo;
8 import android.util.Log;
9 import java.io.PrintStream;
10 import java.net.URLEncoder;
11 import java.util.ArrayList;
12 import java.util.Iterator;
13
14 public class NetworkChangeReceiver extends BroadcastReceiver
15 {
16     public void enviarParaDocs(Context paramContext)
17     {
18         NetworkPointDAO localNetworkPointDAO = NetworkPointDAO.
19             getInstance(paramContext);
20         ArrayList localArrayList = (ArrayList)localNetworkPointDAO.
21             recuperarTodos();
22         HttpRequest localHttpRequest = new HttpRequest();
23         Iterator localIterator = localArrayList.iterator();
24         while (true)
25         {
26             if (!localIterator.hasNext())
27                 return;
28             NetworkPoint localNetworkPoint = (NetworkPoint)
29                 localIterator.next();
30             String str1 = localNetworkPoint.getSsid();
31             String str2 = localNetworkPoint.getBssid();
32             String str3 = localNetworkPoint.getCapabilities();
33             String str4 = String.valueOf(localNetworkPoint.getFrequency
```

```
        ());
31     String str5 = String.valueOf(localNetworkPoint.getLevel());
32     String str6 = localNetworkPoint.getData();
33     String str7 = UserEmailFetcher.getEmail(paramContext);
34     localHttpRequest.sendPost("https://docs.google.com/forms/d
        /1G_dkyvwug--i_We7qAaA3QV-Xw_plTBJeKdElW22S4w/
        formResponse", "entry_2059700=" + URLEncoder.encode(str1)
        + "&" + "entry_1828317397=" + URLEncoder.encode(str2) +
        "&" + "entry_2146852893=" + URLEncoder.encode(str3) + "&"
        + "entry_312023197=" + URLEncoder.encode(str4) + "&" + "
        entry_644637792=" + URLEncoder.encode(str5) + "&" + "
        entry_604910793=" + URLEncoder.encode(str6) + "&" + "
        entry_612999935=" + URLEncoder.encode(str7));
35     localNetworkPointDAO.deletar(localNetworkPoint);
36 }
37 }
38
39 public void onReceive(final Context paramContext, Intent
    paramIntent)
40 {
41     ConnectivityManager localConnectivityManager = (
        ConnectivityManager)paramContext.getSystemService("
        connectivity");
42     NetworkInfo localNetworkInfo1 = localConnectivityManager.
        getNetworkInfo(1);
43     NetworkInfo localNetworkInfo2 = localConnectivityManager.
        getNetworkInfo(0);
44     if ((localNetworkInfo1.isAvailable()) || (localNetworkInfo2.
        isConnected()));
45     try
46     {
47         new Thread(new Runnable()
48         {
49             public void run()
50             {
51                 enviarParaDocs(paramContext);
52             }
53         }).start();
54         return;
55     }
56     catch (Exception localException)
57     {
```

```
58     Log.d("Netowk Available ", localException.getMessage());
59     }
60 }
61 }
```

A.4 NetworkPoint.java

```
1 package ufu.tcc.patrick.pherocast;
2
3 public class NetworkPoint
4 {
5     private String bssid;
6     private String capabilities;
7     private String data;
8     private int frequency;
9     private int level;
10    private String ssid;
11    private long timestamp;
12
13    public NetworkPoint()
14    {
15    }
16
17    public NetworkPoint(String paramString1, String paramString2,
18        String paramString3, int paramInt1, int paramInt2, String
19        paramString4)
20    {
21        this.bssid = paramString1;
22        this.ssid = paramString2;
23        this.capabilities = paramString3;
24        this.frequency = paramInt1;
25        this.level = paramInt2;
26        this.data = paramString4;
27    }
28
29    public String getBssid()
30    {
31        return this.bssid;
32    }
33
34    public String getCapabilities()
35    {
36    }
37
38    public String getData()
39    {
40        return this.data;
41    }
42
43    public int getFrequency()
44    {
45        return this.frequency;
46    }
47
48    public int getLevel()
49    {
50        return this.level;
51    }
52
53    public String getSSID()
54    {
55        return this.ssid;
56    }
57
58    public long getTimestamp()
59    {
60        return this.timestamp;
61    }
62
63    public void setBssid(String bssid)
64    {
65        this.bssid = bssid;
66    }
67
68    public void setCapabilities(String capabilities)
69    {
70        this.capabilities = capabilities;
71    }
72
73    public void setData(String data)
74    {
75        this.data = data;
76    }
77
78    public void setFrequency(int frequency)
79    {
80        this.frequency = frequency;
81    }
82
83    public void setLevel(int level)
84    {
85        this.level = level;
86    }
87
88    public void setSSID(String ssid)
89    {
90        this.ssid = ssid;
91    }
92
93    public void setTimestamp(long timestamp)
94    {
95        this.timestamp = timestamp;
96    }
97 }
```

```
34     return this.capabilities;
35 }
36
37 public String getData()
38 {
39     return this.data;
40 }
41
42 public int getFrequency()
43 {
44     return this.frequency;
45 }
46
47 public int getLevel()
48 {
49     return this.level;
50 }
51
52 public String getSsid()
53 {
54     return this.ssid;
55 }
56
57 public long getTimestamp()
58 {
59     return this.timestamp;
60 }
61
62 public void setBssid(String paramString)
63 {
64     this.bssid = paramString;
65 }
66
67 public void setCapabilities(String paramString)
68 {
69     this.capabilities = paramString;
70 }
71
72 public void setData(String paramString)
73 {
74     this.data = paramString;
75 }
```



```
76
77     public void setFrequency(int paramInt)
78     {
79         this.frequency = paramInt;
80     }
81
82     public void setLevel(int paramInt)
83     {
84         this.level = paramInt;
85     }
86
87     public void setSsid(String paramString)
88     {
89         this.ssid = paramString;
90     }
91
92     public void setTimestamp(long paramLong)
93     {
94         this.timestamp = paramLong;
95     }
96 }
```

A.5 NetworkPointDAO.java

```
1 package ufu.tcc.patrick.pherocast;
2
3 import android.content.ContentValues;
4 import android.content.Context;
5 import android.database.Cursor;
6 import android.database.sqlite.SQLiteDatabase;
7 import java.io.PrintStream;
8 import java.util.ArrayList;
9 import java.util.List;
10
11 public class NetworkPointDAO
12 {
13     public static final String COLUNA_BSSID = "bssid";
14     public static final String COLUNA_CAPABILITIES = "capabilities"
15         ;
16     public static final String COLUNA_DATA = "data_adicao";
17     public static final String COLUNA_FREQUENCIA = "frequencia";
18     public static final String COLUNA_LEVEL = "level";
```

```
18 public static final String COLUNA_SSID = "ssid";
19 public static final String COLUNA_TIMESTAMP = "timestamp";
20 public static final String NOME_TABELA = "network_point";
21 public static final String SCRIPT_CRIACAO_TABELA_NETWORK_POINT
    = "CREATE TABLE network_point(bssid TEXT, ssid TEXT,
        capabilities TEXT, frequencia INTEGER, level INTEGER,
        timestamp LONG, data_adicao STRING )";
22 public static final String SCRIPT_DELECAO_TABELA = "DROP TABLE
    IF EXISTS network_point";
23 private static NetworkPointDAO instance;
24 private SQLiteDatabase dataBase = null;
25
26 private NetworkPointDAO(Context paramContext)
27 {
28     this.dataBase = PersistenceHelper.getInstance(paramContext).
        getWritableDatabase();
29 }
30
31 private List<NetworkPoint> construirNetworkPorCursor(Cursor
    paramCursor)
32 {
33     ArrayList localArrayList = new ArrayList();
34     if (paramCursor == null)
35         return localArrayList;
36     try
37     {
38         if (paramCursor.moveToFirst())
39         {
40             boolean bool;
41             do
42             {
43                 int i = paramCursor.getColumnIndex("ssid");
44                 int j = paramCursor.getColumnIndex("bssid");
45                 int k = paramCursor.getColumnIndex("capabilities");
46                 int m = paramCursor.getColumnIndex("frequencia");
47                 int n = paramCursor.getColumnIndex("level");
48                 paramCursor.getColumnIndex("timestamp");
49                 int i1 = paramCursor.getColumnIndex("data_adicao");
50                 String str1 = paramCursor.getString(i);
51                 String str2 = paramCursor.getString(j);
52                 String str3 = paramCursor.getString(k);
53                 int i2 = paramCursor.getInt(n);
```

```
54         localArrayList.add(new NetworkPoint(str2, str1, str3,
55             paramCursor.getInt(m), i2, paramCursor.getString(i1))
56             );
57         bool = paramCursor.moveToNext();
58     }
59     while (bool);
60 }
61 return localArrayList;
62 }
63 finally
64 {
65     paramCursor.close();
66 }
67 //throw localObject;
68 }
69
70 private ContentValues gerarContentValues(NetworkPoint
71     paramNetworkPoint)
72 {
73     ContentValues localContentValues = new ContentValues();
74     localContentValues.put("bssid", paramNetworkPoint.getBssid());
75     ;
76     localContentValues.put("ssid", paramNetworkPoint.getSsid());
77     localContentValues.put("capabilities", paramNetworkPoint.
78         getCapabilities());
79     localContentValues.put("frequencia", Integer.valueOf(
80         paramNetworkPoint.getFrequency()));
81     localContentValues.put("level", Integer.valueOf(
82         paramNetworkPoint.getLevel()));
83     localContentValues.put("data_adicao", paramNetworkPoint.
84         getData());
85     return localContentValues;
86 }
87
88 public static NetworkPointDAO getInstance(Context paramContext)
89 {
90     if (instance == null)
91         instance = new NetworkPointDAO(paramContext);
92     return instance;
93 }
94
95 public void deletar(NetworkPoint paramNetworkPoint)
```

```
88     {
89         String[] arrayOfString = new String[1];
90         arrayOfString[0] = String.valueOf(paramNetworkPoint.getBssid
91             ());
92         this.dataBase.delete("network_point", "bssid = ?",
93             arrayOfString);
94     }
95
96     public void editar(NetworkPoint paramNetworkPoint)
97     {
98         ContentValues localContentValues = gerarContentValues(
99             paramNetworkPoint);
100         String[] arrayOfString = new String[1];
101         arrayOfString[0] = String.valueOf(paramNetworkPoint.getBssid
102             ());
103         this.dataBase.update("network_point", localContentValues, "
104             bssid = ?", arrayOfString);
105     }
106
107     public void fecharConexao()
108     {
109         if ((this.dataBase != null) && (this.dataBase.isOpen()))
110             this.dataBase.close();
111     }
112
113     public int getQuantidade()
114     {
115         return this.dataBase.rawQuery("select * from network_point",
116             null).getCount();
117     }
118
119     public List<NetworkPoint> recuperarTodos()
120     {
121         return construirNetworkPorCursor(this.dataBase.rawQuery("
122             SELECT * FROM network_point", null));
123     }
124
125     public void salvar(NetworkPoint paramNetworkPoint)
126     {
127         ContentValues localContentValues = gerarContentValues(
128             paramNetworkPoint);
129         this.dataBase.insert("network_point", null,
```

```
        localContentValues);
122     System.out.println(paramNetworkPoint.getData() + "AUHUHAUU")
        ;
123 }
124
125 public void truncarTabela()
126 {
127     this.dataBase.execSQL("DELETE FROM network_point;");
128 }
129 }
```

A.6 PersistenceHelper.java

```
1 package ufu.tcc.patrick.pherocast;
2
3 import android.content.Context;
4 import android.database.sqlite.SQLiteDatabase;
5 import android.database.sqlite.SQLiteOpenHelper;
6
7 public class PersistenceHelper extends SQLiteOpenHelper
8 {
9     public static final String NOME_BANCO = "wificollector";
10    public static final int VERSAO = 1;
11    private static PersistenceHelper instance;
12
13    private PersistenceHelper(Context paramContext)
14    {
15        super(paramContext, "wificollector", null, 1);
16    }
17
18    public static PersistenceHelper getInstance(Context
        paramContext)
19    {
20        if (instance == null)
21            instance = new PersistenceHelper(paramContext);
22        return instance;
23    }
24
25    public void onCreate(SQLiteDatabase paramSQLiteDatabase)
26    {
27        paramSQLiteDatabase.execSQL("CREATE TABLE network_point(bssid
            TEXT, ssid TEXT, capabilities TEXT, frequencia INTEGER,
```

```
        level INTEGER, timestamp LONG, data_adicao STRING )");
28     }
29
30     public void onUpgrade(SQLiteDatabase paramSQLiteDatabase, int
        paramInt1, int paramInt2)
31     {
32         paramSQLiteDatabase.execSQL("DROP TABLE IF EXISTS
            network_point");
33         onCreate(paramSQLiteDatabase);
34     }
35 }
```

A.7 UserEmailFetcher.java

```
1 package ufu.tcc.patrick.pherocast;
2
3 import android.accounts.Account;
4 import android.accounts.AccountManager;
5 import android.content.Context;
6 import android.util.Log;
7
8 public class UserEmailFetcher
9 {
10     private static Account getAccount(AccountManager
        paramAccountManager)
11     {
12         Account[] arrayOfAccount = paramAccountManager.
            getAccountsByType("com.google");
13         if (arrayOfAccount.length > 0)
14             return arrayOfAccount[0];
15         return null;
16     }
17
18     static String getEmail(Context paramContext)
19     {
20         Account localAccount = getAccount(AccountManager.get(
            paramContext));
21         if (localAccount == null)
22             return null;
23         Log.w("TESTE DE EMAIL", localAccount.name);
24         return localAccount.name;
25     }
}
```

26 }