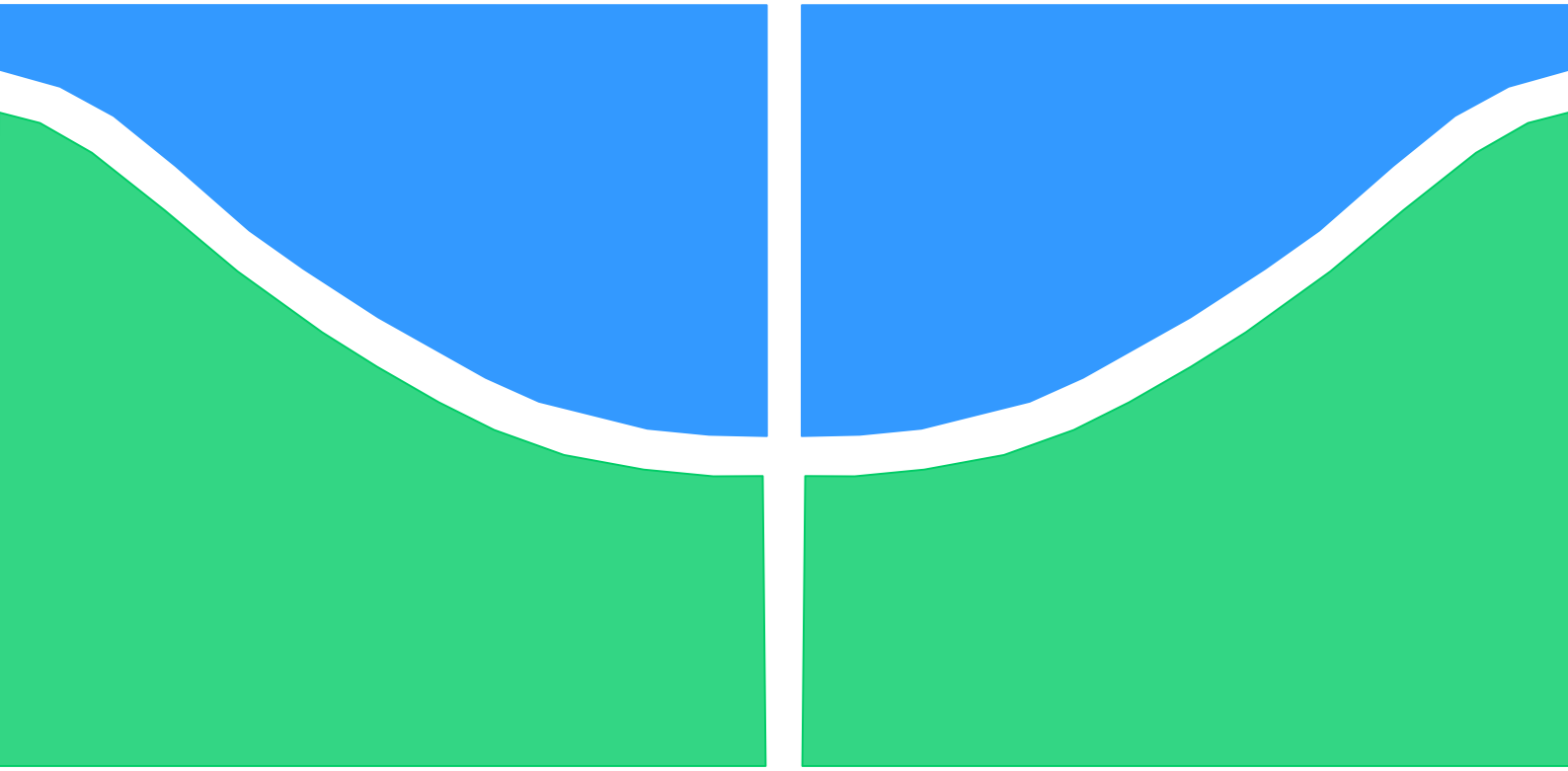

TRABALHO DE GRADUAÇÃO

PROTOCOLO AODV COM EFICIÊNCIA ENERGÉTICA PARA PLATAFORMA ANDROID

Hélder Paz Machado
Raphael Augusto Souza de Melo

Brasília, setembro de 2013

UNIVERSIDADE DE BRASILIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

PROTOCOLO AODV COM EFICIÊNCIA ENERGÉTICA PARA PLATAFORMA ANDROID

Hélder Paz Machado
Raphael Augusto Souza de Melo

Relatório submetido como requisito parcial para obtenção
do grau de Engenheiro de Redes de Comunicação

Banca Examinadora

Prof. Marcelo Menezes de Carvalho, ENE/UnB
(Orientador) _____

Prof. Renato Mariz de Moraes, ENE/UnB _____

Prof. Paulo Roberto de Lira Gondim, ENE/UnB _____

Dedicatórias

A Deus e minha família que estiveram comigo durante todo este curso e merecem esta singela lembrança.

Hélder Paz Machado

A todos que tiverem tempo e disposição para estudar este trabalho.

Raphael Augusto Souza de Melo

Agradecimentos

Agradeço a Deus pela vida e pelo cuidado que me trouxeram até este momento gratificante, aos meus pais, pois sempre estiveram comigo me dando suporte nesta maratona chamada graduação, à minha namorada, por saber entender e me incentivar em todos os desafios sempre que precisei, e a todos os professores por fazerem desta experiência um grande aprendizado para toda a vida. A todos vocês meu muito obrigado.

Hélder Paz Machado

A todos que me ajudaram desde o início da minha graduação. Aos meus pais, Carlos e Élida, que me deram todo o suporte para eu chegar até este momento tão esperado. A toda a minha família que sempre acreditou no meu potencial. A Camila que esteve ao meu lado durante estes 5 anos na alegria e na tristeza. Ao grande amigo Guilherme simplesmente por ser meu amigo. Ao meu orientador Marcelo por todo o apoio neste e em outros trabalhos. A todos que ajudaram com este projeto emprestando celulares para os nossos testes, especialmente Breno e Vitor. Aos colegas de curso que compartilharam os melhores e mesmo os momentos mais difíceis deste curso, em especial os companheiros do TNRG. E por fim a todos os professores os quais aprendi alguma coisa importante nesta Universidade.

Raphael Augusto Souza de Melo

RESUMO

Os *smartphones* vêm se tornando cada dia mais populares e acessíveis. Os mais variados sistemas operacionais, propiciam aos usuários diversas aplicações, dentre elas várias que utilizam o acesso a redes para a comunicação como jogos, correio eletrônico e redes sociais. Com o advento da plataforma Android, sistema operacional para dispositivos móveis disponível em grande parte dos aparelhos celulares fabricados atualmente, surgem diversos trabalhos que focam no desenvolvimento de melhorias para estes dispositivos inclusive também no âmbito das redes. As redes *ad hoc* podem facilitar esta comunicação entre dispositivos em alguns cenários dispensando o uso de pontos de acesso e possibilitando a conexão direta entre os aparelhos usando técnicas de roteamento específicas para estas redes. Neste sentido, este trabalho apresenta uma implementação do protocolo de roteamento AODV (do inglês *Ad hoc On-Demand Distance Vector*) para formação de redes *ad hoc* de comunicações sem fio em dispositivos habilitados com a tecnologia Android, tendo como foco a eficiência energética da rede. Para tanto, foram sugeridas mudanças no protocolo AODV a fim de torná-lo um protocolo mais eficiente em termos energéticos. Para isso, foram introduzidas informações da bateria remanescente de cada nó da rede dentro dos cabeçalhos dos pacotes de controle do AODV. Esta informação se propaga através da rede e cada nó passa a saber a informação de bateria de cada nó da rede. Dessa forma, propomos também introduzir custos nas rotas baseado nessas informações energéticas. Esta medida evita que rotas sejam formadas por dispositivos ou rotas que estão próximas de ficarem sem bateria. Outro recurso introduzido é a criação de limiares que impedem a criação de rotas por dispositivos que estão com pouca bateria remanescente. Isso permite que o restante da bateria daquele dispositivo seja economizada. Utilizamos uma biblioteca desenvolvida para Android que executa o roteamento em redes *ad hoc* com o uso do AODV. Fizemos ainda melhorias nessa biblioteca que agora é capaz de iniciar e configurar a rede *ad hoc* por si mesma. Anteriormente este procedimento deveria ser feito manualmente antes da inicialização do protocolo de roteamento. Uma série de experimentos foi realizada com o objetivo de verificar o correto funcionamento do protocolo e seu desempenho.

ABSTRACT

Smartphones are becoming increasingly more popular and less costly to the end user each day. Designed for working under various operating systems, a number of apps are available today, with many of them relying on communication networks for its operation, such as on-line games, e-mail, and social networks. With the advent of the Android platform---an operating system for mobile devices that is available in most cell phones today---a number of works have been done to improve the performance of these devices and their inter-netwoking as well. Ad hoc networks may facilitate communication among devices by avoiding the use of access points and by allowing direct communication among nodes through routing techniques that are specific to this kind of network. Based on that, this work presents an energy-efficient implementation of the Ad hoc On-Demand Distance Vector (AODV) routing protocol for Android smartphones. To accomplish that, information regarding remaining battery energy level at each node is embedded in the header of every AODV control message. As a result, information regarding each node's energy level is broadcast through the network, and each reachable node may receive energy information regarding every other node in the network. In addition, we propose the assignment of route costs based on energy information. This way, the routing protocol may avoid the selection of routes through nodes that are close to energy starvation. Another feature we introduce is the use of an energy threshold by which a node decides to not execute the routing functionality anymore and to inform, indirectly, the rest of the network about its decision. This allows fast path recovery and selection. Finally, an improvement to the basic AODV software is presented that allows self-initialization and self-configuration, as opposed to manual network setup. A set of experiments is performed and presented in order to evaluate correct protocol operation and performance.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Acrônimos

ADB	Android Debug Bridge
AM	Application Monitor
AODV	Ad hoc On-Demand Distance Vector
AP	Access Point
AVD	Android Virtual Device
AWERA	Application-Aware Energy Efficient Routing Algorithm
BER	Bit Error Rate
BSS	Basic Service Set
CMMBCR	Conditional Max-Min Battery Capacity Routing
CRC	Cyclic Redundancy Check
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CTS	Clear to Send
CW	Contention Window
DARPA	Defense Advanced Research Projects Agency
DBPSK	Differential Binary Phase-Shift Keying
DIFS	Distributed Interframe Space
DSDV	Destination-Sequenced Distance Vector
DSR	Dynamic Source Routing
DSSS	Direct Sequence Spread Spectrum
DVM	Dalvik Virtual Machine
ECCM	Energy Constraint Computation Module
ERP	Extended Rate PHY
FB-AODV	Flow-Based Ad hoc On-Demand Distance Vector
FHSS	Frequency-Hopping Spread Spectrum
HR-DSSS	High-Rate Direct Sequence Spread Spectrum
HT-PHY	High Throughput PHY
IBSS	Independent Basic Service Set
ICS	Ice Cream Sandwich
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IR	Infra-Red
JB	Jelly Bean
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LAN	Local Access Network
MAC	Media Access Control
MANET	Mobile Ad hoc Network
MBCR	Minimum Battery Cost Routing
MDR	Minimum Drain Rate
MIMO	Multiple-Input and Multiple-Output
MMBCR	Min-Max Battery Cost Routing
MTPR	Minimum (Total) Transmission Power Routing
NAV	Network Allocation Vector
NDK	Native Development Kit
NEDB	Network Energy Database
OFDM	Orthogonal Frequency-Division Multiplexing

OLSR	Optimized Link State Routing
OS	Operational System
OSI	Open Systems Interconnection
PN	Pseudo-Noise
QPSK	Quadrature Phase Shift Keying
REM	Remaining Energy Monitor
RREP	Route Reply
RREQ	Route Request
RERR	Route Error
RFC	Request for Comments
RT	Routing Table
RTM	Routing Table Maintenance Module
RTS	Request to Send
SDK	Software Development Kit
SIFS	Short Interframe Space
SSID	Service Set Identifier
TBRPF	Topology Dissemination Based on Reverse-Path Forwarding
TTL	Time to Live
UDP	User Datagram Protocol
USB	Universal Serial Bus
WCIM	Weighted Contention and Interference Routing Metric
WEP	Wired Equivalent Privacy
Wi-Fi	Wireless Fidelity
WPA	Wi-Fi Protected Access
WRP	Wireless Routing Protocol

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Nosso trabalho surge da grande proliferação de *smartphones* no mercado. A popularização destes aparelhos, que são dotados de sistema operacional próprio e comumente possuem interface Wi-Fi (protocolo IEEE 802.11), vem estimulando diversas pesquisas na área de comunicações entre estes dispositivos. Deriva também da necessidade de possibilitar aos usuários a comunicação entre si sem depender completamente da rede celular existente, tendo em vista a crescente nos valores dos planos de dados e a ainda ineficiente área de cobertura das operadoras de telecomunicações.

O Android é atualmente o sistema operacional para dispositivos móveis mais popular. Baseado em Linux, seu código é aberto. Essa característica significa que seu código-fonte está disponível para qualquer um podendo ser alterado quase inteiramente. Por isso, é o mais utilizado por pesquisadores na área de *smartphones*. Com este sistema operacional em mãos é possível criar novos aplicativos de acordo com a necessidade do usuário, ou mesmo, alterar aspectos básicos do sistema, como por exemplo, o modo de operação do aparelho, modificando módulos de rede da interface para obter determinada funcionalidade não configurada originalmente pelo Android.

Por outro lado, o Android ainda possui pouco suporte à comunicação direta entre dispositivos. Em geral, a comunicação só se dá através de um dispositivo central que gerencia as conexões e o tráfego de dados. Este tipo de comunicação é conhecida como infraestruturada, pois depende de uma infraestrutura presente anteriormente à formação da rede. A necessidade de uma infraestrutura pré-estabelecida pode tornar-se um problema em ambientes onde um grupo de pessoas deseja se comunicar umas com as outras e não há rede disponível ou o acesso a ela é muito caro.

Nestes casos é mais conveniente criar uma rede local descentralizada. Estas redes, também conhecidas como *ad hoc*, devido a sua descentralização, não necessitam de infraestrutura prévia para gerenciar as comunicações. Há diversas aplicações que são mais vantajosas neste cenário: diversas aplicações militares, coleta de dados para ciência, troca de mensagens e arquivos de dados e até mesmo em jogos em rede.

Independentemente da aplicação, os transmissores sem fio dos dispositivos devem ser capazes de se comunicar com outros dispositivos na rede. Dada a limitação de potência de transmissão das interfaces atuais, a distância física entre dois indivíduos que desejam se comunicar pode ser bastante curta. Desta forma, as redes *ad hoc* devem possuir meios de estabelecer conexões através de quantos dispositivos intermediários forem necessários. Para isso, diversos protocolos de roteamento para redes *ad hoc* foram pensados para estabelecer estas comunicações da forma mais eficiente possível.

As redes *ad hoc* sem fio são em geral bem dinâmicas devido à mobilidade dos dispositivos e à descentralização de gerenciamento. Existem algumas limitações neste tipo de rede para celulares como, por exemplo, a capacidade de bateria, que nos dispositivos móveis é bastante limitada assim como seu poder de processamento e sua quantidade de memória. Assim, um protocolo de roteamento para estes dispositivos deve levar em consideração estas limitações.

1.2 OBJETIVOS DO TRABALHO

Neste trabalho focamos no problema da capacidade de bateria. Uma das dificuldades enfrentadas é que nem todos os nós da rede participam ativamente da transmissão de seu próprios dados. Assim, durante parte do tempo, eles utilizam seus recursos para repassar dados de terceiros. Com isso, se o consumo de energia não for baixo o bastante, não vale a pena para o usuário manter seu aparelho na rede, o que poderá causar o particionamento da rede e até mesmo inviabilizar sua operação.

Mesmo nos *smartphones* mais modernos, o tempo de vida da bateria é curto, pois diversas aplicações são executadas simultaneamente. Assim, uma nova funcionalidade deve prezar pela economia de energia para seu tempo de vida ser maximizado. Diversas pesquisas tem sido feitas para aumentar a capacidade de bateria destes dispositivos. Em termos de comunicações, estudos também têm sido feitos para diminuir o consumo de energia de transmissão das interfaces sem fio (focando nas camadas física e de enlace). O foco do nosso trabalho é na camada de roteamento, buscando um protocolo que consuma o mínimo de bateria possível para a formação de rotas.

Os protocolos existentes de roteamento para redes *ad hoc* podem ser classificados como proativos e reativos. Basicamente, o primeiro faz busca constante por rotas periodicamente, tentando mantê-las sempre atualizadas mesmo quando não há tráfego. O segundo tipo só envia a requisição por uma rota para um destino quando a fonte possui dados a serem entregues. Em termos do consumo de energia, os protocolos reativos são melhores, pois os proativos dispendem mais energia em ter de manter, constantemente, informação atualizada sobre rotas na rede mesmo quando não utilizadas. Assim, focamos nosso trabalho em protocolos reativos, também conhecidos como protocolos sob demanda.

Nos protocolos sob demanda tradicionais, como o AODV (do inglês *Ad hoc On-Demand Distance Vector*) [8] e o DSR (do inglês *Dynamic Source Routing*) [27], não há preocupação com parâmetros relacionados à economia de energia. O roteamento é todo baseado em quantidade de saltos. Assim, a formação de rota não leva em consideração que podem haver nós cuja bateria está próxima do descarregamento energético total ou que estão sobrecarregados de tráfego com um alto consumo de bateria. Naturalmente, as rotas não são rearranjadas caso um dispositivo esteja próximo de desconectar-se da rede por falta de bateria.

Com isso, nosso trabalho tem como objetivo buscar soluções para otimizar o consumo de energia e fazer a formação de rotas levando em consideração aspectos energéticos. Dentre os protocolos já conhecidos, escolhemos o AODV, por ser sob demanda e possuir implementação para Android, a qual utilizamos como base para testar as nossas propostas. A partir da implementação proposta, será possível testar diferentes técnicas de economia de energia para roteamento em redes *ad hoc* e então compará-las.

1.3 CONTRIBUIÇÕES DO TRABALHO

Como contribuição deste trabalho destacamos a inserção de informações de energia no cabeçalho dos pacotes de controle do protocolo AODV. Estas informações incluem a quantidade de bateria remanescente de cada nó pelo qual o pacote trafegou e o instante de tempo em que aquela informação foi atualizada. Com estas informações trafegando através da rede, cada nó deve conhecer o estado de bateria dos outros nós da rede e saber quão recente é esta informação. A partir disso, ele pode tomar decisões de escolha de rota a partir de medidas energéticas, evitando rotas em que nós estejam com baterias remanescentes muito baixas.

Diversas métricas de energia podem ser utilizadas para a escolha da melhor rota. Apesar de adotar uma única métrica para os nossos testes, toda a aplicação foi desenvolvida de modo que introduzimos o máximo de informação de energia trafegando através da rede, abrindo mão da eficiência, porém deixando uma aplicação onde diferentes métricas de energia podem ser facilmente analisadas no futuro. Dessa forma, este trabalho deixa uma plataforma de testes para *smartphones* Android para roteamento energeticamente eficiente em redes *ad hoc* baseado em AODV.

Ao levar em consideração parâmetros energéticos, o protocolo e o paradigma de redes *ad hoc* tornam-se mais amigáveis ao uso em *smartphones*. Também introduzimos limiares no AODV com o objetivo de que, a partir de níveis críticos de bateria, os dispositivos sejam impedidos de participar de qualquer rota e desperdiçar o restante de sua bateria.

Fizemos testes das medidas propostas a partir de uma biblioteca para Android que faz o roteamento do AODV desenvolvido por Jradi e Reedtz [11]. Também efetuamos contribuições nessa biblioteca fazendo com que a aplicação autoconfigure a rede *ad hoc* assim que iniciada, habilitando-o e configurando o endereço IP das interfaces sem fio do dispositivo. Anteriormente, este procedimento era feito manualmente.

1.4 ORGANIZAÇÃO DESTE TRABALHO

Este trabalho está estruturado na forma de capítulos para tratar de diferentes etapas do desenvolvimento. Iniciamos com o Capítulo 2 que traz todos os aspectos teóricos envolvidos neste projeto. Começamos com uma visão geral das redes *ad hoc*, apresentamos os elementos principais da norma IEEE 802.11 (Wi-Fi), a qual é utilizada nos *smartphones* que trabalhamos. Descrevemos o funcionamento do protocolo AODV em sua versão tradicional, apresentamos o sistema operacional Android e suas características principais para desenvolvimento nesta plataforma e finalizamos apresentando técnicas já conhecidas de economia de energia em redes *ad hoc*.

No Capítulo 3 discutimos trabalhos relacionados na área. Apresentamos trabalhos que buscam melhorar a eficiência energética de protocolos de roteamento e também trabalhos que buscam implementar redes *ad hoc* em *smartphones*. No Capítulo 4 apresentamos a nossa proposta de mudanças no AODV no sentido de que seu roteamento se dê levando em consideração aspectos energéticos e detalhamos o processo de desenvolvimento da mesma.

O Capítulo 5 traz a descrição dos testes que efetuamos juntamente com a análise dos resultados obtidos. Finalmente no Capítulo 6 faremos uma breve conclusão do trabalho e apresentamos propostas para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 APRESENTAÇÃO

Este capítulo tem como objetivo apresentar as tecnologias e o embasamento teórico utilizado em nosso trabalho. Iniciamos com a Seção 2.2 tratando o assunto das redes *ad hoc*, seu conceito e tipos de protocolos de roteamento para este tipo de rede. Na Seção 2.3 abordamos o padrão IEEE 802.11, conhecido também como Wi-Fi. Descrevemos seus modos de funcionamento, as principais características do protocolo de controle de acesso ao meio adotado na norma e a formatação de seus quadros. A Seção 2.4 descreve o protocolo de roteamento que utilizamos em nosso trabalho: o AODV. Apresentamos seu conceito, a sua forma de roteamento e as mensagens que ele utiliza. Finalizamos o capítulo com a Seção 2.5, que aborda o Android, sistema operacional para celulares que utilizamos em nosso trabalho. Discutimos um pouco do seu histórico, do mercado de *smartphones* e sobre o desenvolvimento para esta plataforma.

2.2 REDES *AD HOC*

Redes de comunicação de dados podem ser subdivididas em duas grandes classes conforme a organização e o modo de operação dos nós integrantes da rede: infraestruturada e *ad hoc*.

A ideia da criação de uma rede com o intuito de prover estrutura de comunicação entre elementos móveis em modo *ad hoc* é proveniente do DARPA (do inglês *Defense Advanced Research Projects Agency*) e tinha fins militares. Posteriormente, surgiu tal necessidade no meio popular tendo em vista o crescimento no número de equipamentos dotados de interfaces sem fio comercializados e a necessidade de comunicação independente de pontos de acesso dando força ao tema no meio acadêmico novamente. Tinha-se em vista a formação de uma rede criada tão somente pelos próprios dispositivos e muito mais resistente a falhas devido a sua topologia descentralizadora da comunicação.

As redes *ad hoc* permitem a intercomunicação de dispositivos móveis ligados entre si para fazer a comunicação ao longo da rede independentemente de um ponto centralizador que coordena o compartilhamento do canal entre os usuários, como acontece no modo infraestrutura. Neste tipo de rede cada nó atua como um roteador, executando funções relacionadas à descoberta e escolha das melhores rotas para diferentes destinos e participando no repasse de pacotes de dados em trânsito pela rede. Na rede *ad hoc*, cada dispositivo deve ser capaz de tomar decisões de roteamento para alcançar o equipamento de destino caso o dispositivo não esteja no seu raio de alcance. Um exemplo de topologia de uma rede *ad hoc* pode ser visto na Figura 1.

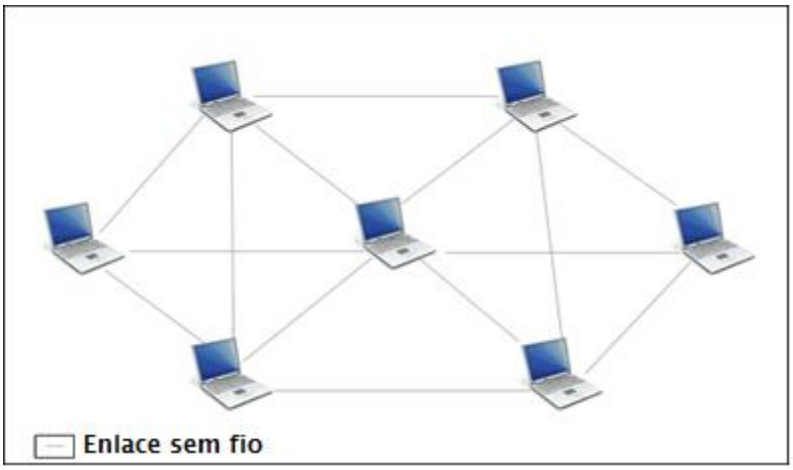

Figura 1. Exemplo de uma rede *ad hoc* na qual nota-se a ausência de estações de controle.

2.2.1 TIPOS DE PROTOCOLO DE ROTEAMENTO *AD HOC*

Os protocolos de roteamento em redes *ad hoc* podem ser classificados de duas formas: proativos e reativos. Os protocolos proativos se caracterizam pela descoberta e manutenção constante de rotas mesmo não havendo necessidade de sua utilização. Já os protocolos classificados como reativos, somente fazem a busca por rotas a um determinado destino quando desejam comunicar-se com ele.

Nas redes roteadas sob demanda (protocolos reativos), para que haja descoberta de nós e formação de rotas é necessário primeiro existir uma solicitação por parte do nó originador da conexão, para em seguida iniciar-se o processo de descoberta de caminho até o destino. Uma desvantagem neste modo de operação é o fato de haver uma latência maior no envio de dados, pois pode não haver rota para o destino no momento da requisição de envio de dados pela camada de aplicação. Por outro lado, uma grande vantagem destes protocolos está no uso eficiente da banda de transmissão disponível, pois em se tratando de redes sem fio, seja infraestrutura ou *ad hoc*, deve-se levar em conta a alta mobilidade dos nós com deslocamentos constantes e o fato de limitados espectros de transmissão não permitir grandes taxas de envio de informação. Além disso, os nós não mais precisam carregar informações a respeito de toda a rede sendo muito mais simples a manutenção das rotas pelos nós nos protocolos reativos. Nesta categoria encontram-se os protocolos AODV e o DSR.

Os protocolos de roteamento podem ainda ser classificados em duas outras categorias tendo em vista a forma de operação e o processo de formação e difusão de informações de rotas pela rede, os chamados protocolos estado de enlace e vetor de distâncias. Nos protocolos do tipo estado de enlace, os nós mantêm rotas para todos os outros nós da rede com o custo associado a cada enlace [1]. Todos os nós da rede fazem uma difusão sempre que ocorre uma atualização em um enlace ao qual está diretamente conectado. Desta forma, é viável calcular o melhor caminho baseando-se nas rotas com os melhores enlaces. São exemplos de protocolos de roteamento estado de enlace: OLSR (do inglês *Optimized Link State Routing*) [2] e TBRPF (do inglês *Topology Broadcast based on Reverse-Path forwarding*) [3]. Já nos protocolos baseados no método vetor de distâncias, cada nó mantêm uma tabela de roteamento com as melhores rotas e realiza uma difusão na rede afim de difundir estas informações de rota aos seus nós vizinhos que atualizam as suas rotas conforme sua necessidade de rota ou se melhores rotas tiverem sido recebidas. Cada rota é dada por um vetor contendo o destino, o custo de acordo com a métrica usada, e o próximo nó no roteamento de pacotes até o destino [3]. Exemplos de protocolos vetor de distância são: DSDV (do inglês *Destination-Sequenced Distance Vector*) [4] e WRP (do inglês *Wireless Routing Protocol*) [5]

2.3 WI-FI (IEEE 802.11)

Deste a criação dos *notebooks*, *palmtops* e outros dispositivos móveis na década de 1990 tornou-se mais comum a necessidade de acesso às redes já existentes, principalmente à Internet, de forma sem fio. Dessa maneira, diversas soluções proprietárias foram propostas para a comunicação entre estes dispositivos móveis e a rede. Entretanto, a proliferação desta tecnologia esbarrou em problemas de velocidade, compatibilidade entre dispositivos e até mesmo de custos destas soluções. Dado esse cenário, era imprescindível o estabelecimento de um padrão para mitigar estes problemas. Com isso, o IEEE criou o padrão 802.11 [6] que estabelece as normas de comunicação para as LAN sem fio atualmente conhecidas como redes Wi-Fi (do inglês *Wireless Fidelity*).

Atualmente, estas redes são bastante populares e sua tecnologia está presente nos mais diversos dispositivos, como *smartphones* e *tablets*. Da mesma forma, também está presente em grande parte das residências, prédios e área públicas, como cafés, aeroportos, *shoppings*, etc. Existem diversas extensões da norma 802.11 que representam a evolução do próprio padrão. Alguns exemplos dessas melhorias no padrão são o 802.11b, o 802.11g e o 802.11n, por exemplo.

As diferenças nessas versões representam mudanças na faixa de frequência, na banda disponível e também na taxa de transmissão, dentre outras características. No entanto, todas estas extensões utilizam o CSMA/CA (do inglês *Carrier Sense Multiple Access with Collision Avoidance*) como protocolo de controle de acesso ao meio, e permitem a operação em ambos os modos infraestrutura e *ad hoc*.

Para o desenvolvimento deste padrão, os desafios da comunicação sem fio tiveram que ser levados em consideração. Dentre eles podem ser citados a sua maior suscetibilidade à interferência, a possiblidade de um mesmo sinal percorrer múltiplos percursos e as perdas de conectividade causadas

pela redução de potência do sinal. Assim pode-se dizer que o canal sem fio é menos confiável que o meio cabeado. Dessa forma, o estabelecimento do padrão deve levar em conta essas características de forma a amenizar estes problemas.

Outro grande problema da comunicação sem fio é a segurança. Como os dados trafegam livremente pelo ar, torna-se mais fácil a interceptação de pacotes de forma que as informações nele contida sejam reveladas. Com isso, os padrões adotam protocolos para criptografar os pacotes transmitidos. Os mais comumente utilizados são o WEP (do inglês *Wired Equivalent Privacy*), o WPA (do inglês *Wi-Fi Protected Access*) e o WPA2, que veio como uma evolução do seu anterior. Atualmente não é mais recomendado utilizar o WEP, pois já existem diversas técnicas para quebrar a sua chave de segurança.

2.3.1 REDES WI-FI NO MODO INFRAESTRUTURADO

As redes Wi-Fi baseadas no modelo com infraestrutura apresentam componentes básicos, sendo que o fundamental deles é o BSS (do inglês *Basic Service Set*). Este componente representa a zona de cobertura da rede sem fio. Ele é composto por uma ou mais estações sem fio sendo que uma delas é a estação-base central, conhecida como ponto de acesso (AP, do inglês *Access Point*). Os APs em geral estão conectados, ou interconectados, a outros componentes de rede (por exemplo, *switches*, *hubs*, etc) que farão a interconexão com o restante da rede, sem fio e/ou cabeado. A Figura 2 mostra uma estrutura com diversas BSSs com seus APs e dispositivos conectados à infraestrutura de rede.

Este tipo de rede possui a denominação com infraestrutura devido à presença de equipamentos como os APs e outros elementos de rede que possuem o papel de interconectar a rede. Por outro lado, as redes *ad hoc* não apresentam nenhum controle central e nenhuma conexão com o "mundo externo". Neste modo, as redes são formadas somente pelos equipamentos finais devido à sua necessidade de comunicar-se diretamente uns com os outros ou utilizando outros nós para alcançar o seu destino.

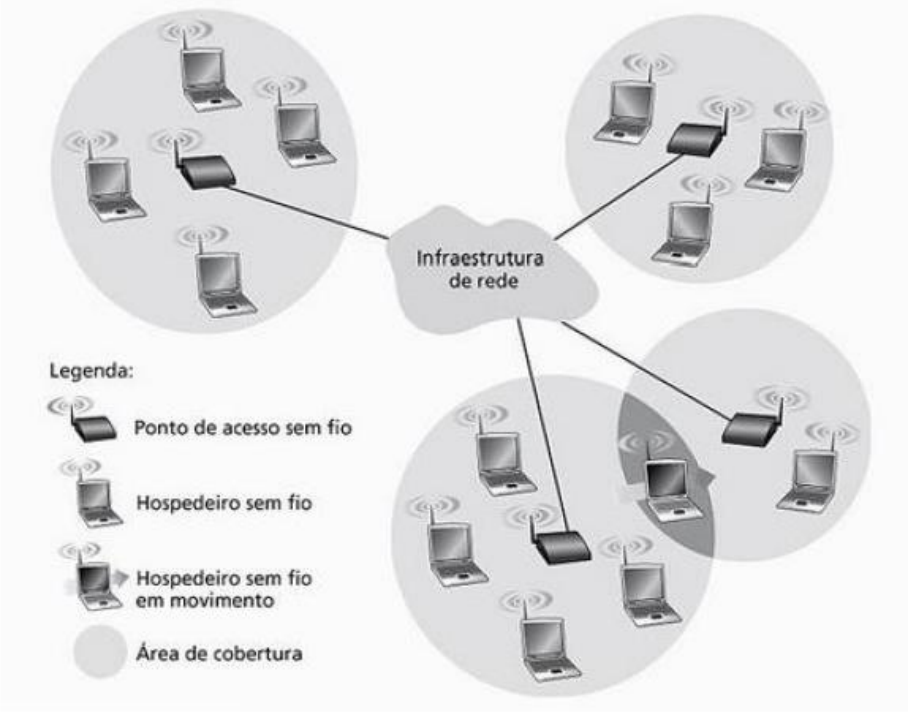

Figura 2. Rede Wi-Fi no modo infraestruturado com os dispositivos interconectados através de APs. Figura retirada de [7].

No modo com estrutura, os nós precisam se associar a um AP para passar a fazer parte da rede e poder comunicar-se com os outros equipamentos que fazem parte desta rede. Para associar-se a um AP, em geral, deve ocorrer a autenticação anteriormente, comumente baseado em uma senha compartilhada com aqueles que devem acessar a rede ou baseado em filtro pelo MAC daqueles que tem permissão de acesso à rede. Após ocorrer a autenticação com sucesso, diz-se que o equipamento

associou-se ao AP. Ao associar-se a um AP, o dispositivo passa a fazer parte da mesma sub-rede IP que o AP faz parte.

No momento da instalação de um AP, é necessário fornecer a ele um nome denominado de SSID (do inglês *Service Set Identifier*), um número de canal e o método de autenticação. O SSID serve como um nome que identificará aquele AP que ajudará aos equipamentos que desejam associar-se a ele para que o encontre com mais facilidade. O número do canal determina em qual faixa de frequência. dentre aquelas disponíveis na extensão do IEEE 802.11 utilizada o AP irá operar. A existência dessas faixas é importante para que APs muito próximos uns aos outros não operem na mesma frequência causando interferência um ao outro.

Na extensão IEEE 802.11g, por exemplo, existem 11 canais disponíveis. Entretanto, as faixas destes canais se sobrepõem entre si. De forma que, mesmo em BSSs operando em canais diferentes, ainda há a chance de haver interferência. Nestes 11 canais, pode-se configurar três canais que não interferem entre si, que são o 1, 6 e 11. Com isso, numa "selva de Wi-Fis", como denominado por Kurose [7], onde há a existência de sinal forte de pelo menos dois APs, possivelmente a taxa efetiva de transmissão será afetada devido à interferência que um sinal causará no outro. Na Figura 3, podemos visualizar uma representação da disposição destes canais.

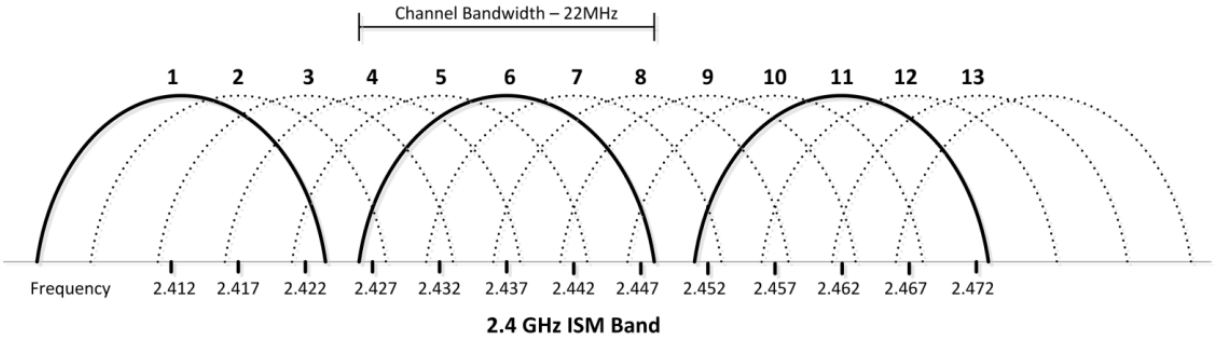

Figura 3. Canais na faixa de frequência de 2,4GHz baseado nas frequências da extensão IEEE 802.11g.

Para que possam ser descobertos, os APs enviam quadros de sinalização periodicamente os quais contém o seu SSID e o seu endereço MAC. Dessa forma, as estações sem fio próximas ao AP farão uma varredura pelos 11 canais sem fio disponíveis e, ao identificarem estes pacotes, saberão que determinado AP está ao seu alcance, sendo possível associar-se a ele, caso seja autorizado.

2.3.2 REDES WI-FI NO MODO *AD HOC*

As redes *ad hoc* são formadas por agrupamentos de equipamentos que compõem BSSs independentes (IBSS, do inglês *Independent Basic Service Set*). Os equipamentos dentro de uma IBSS podem estabelecer comunicação diretamente com dispositivos dentro da mesma IBSS sem a necessidade de algum equipamento intermediário central de controle como no modo com infraestrutura. Dessa forma, para que haja comunicação para fora da IBSS, um dos dispositivos terá que atuar como gateway com alguma interface para outra rede. Da mesma forma como no modo com infraestrutura, uma IBSS atuará dentro de algum dos canais disponíveis na faixa de frequência do IEEE 802.11.

A Figura 4 mostra exemplos da aplicação de redes *ad hoc* com algumas IBSSs e seus dispositivos envolvidos. Nota-se a ausência de equipamento central de controle e os dispositivos comunicam-se diretamente entre si. Neste modo, a área de alcance da rede é determinada pelo alcance que o sinal dos dispositivos atinge. Caso nenhum nó esteja comunicável, a rede é desfeita.

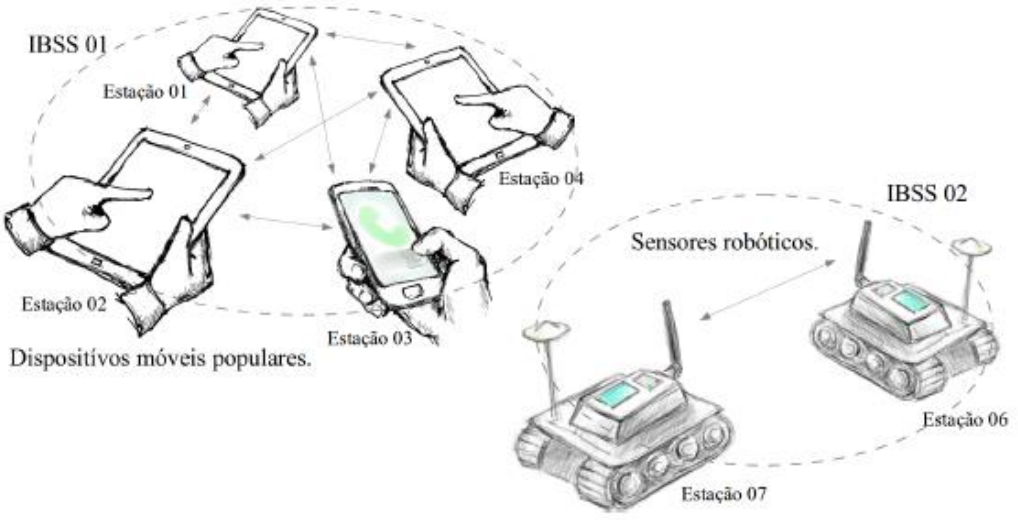

Figura 4. Rede Wi-Fi no modo *ad hoc* e suas IBSS.

2.3.3 CONTROLE DE ACESSO AO MEIO NO IEEE 802.11

Como o meio físico da transmissão sem fio é compartilhado entre vários usuários, mesmo que por dispositivos de redes diferentes quando elas utilizam o mesmo canal, podem ocorrer colisões de pacotes enviados ao mesmo tempo dentro da mesma faixa de frequência. Para que a comunicação seja efetiva, é preciso evitar a colisão de pacotes assim como é feito no meio cabeado. Caso dois pacotes sejam enviados ao mesmo tempo na mesma frequência, eles sofrerão interferência e o pacote poderá ser decodificado com erros no seu destino. Para evitar estas colisões, o IEEE 802.11 utiliza o protocolo de controle de acesso ao meio conhecido como CSMA/CA.

O princípio deste protocolo é o de monitorar o canal antes de enviar uma mensagem. No caso de encontrar, incialmente, o canal disponível, o nó aguardará um tempo chamado de DIFS (do inglês *Ditributed Inter-Frame Space*) antes de iniciar a sua transmissão. Caso o canal esteja ocupado, o nó não faz a sua transmissão e estabelece um tempo de recuo. Enquanto perceber que o canal está ocioso um temporizador é iniciado e, ao atingir o tempo de recuo determinado, a transmissão é efetuada. No período em que ele percebe o canal ocupado, este temporizador permanece parado.

O estabelecimento deste tempo de recuo, tem por objetivo minimizar a probabilidade de ocorrerem colisões espalhando o momento das transmissões de diferentes dispositivos. Para isso, ele é escolhido através de uma função exponencial binária aleatória. O tempo deste recuo é dado, então, pela seguinte equação:

$$\text{Backoff Time} = \text{Random}() * \text{aSlotTime} \quad (1)$$

Onde Random() é um número pseudoaleatório gerado por uma distribuição uniforme no intervalo [0, CW], CW (do inglês *Contention window*) é um inteiro entre aCWmin e aCWmax. Este valor CW inicia-se com o valor aCWmin e deve ser aumentado de forma exponencial binária para o próximo valor da série cada vez que uma transmissão falhar. Quando CW atinge o valor aCWmax, ele é mantido neste valor até que ele seja reiniciado. O valor de CW é reiniciado sempre que ocorrer uma transmissão sem falhas. Como o valor CW cresce de forma exponencial binária, a chance de ocorrerem erros sucessivos é diminuída a cada estágio. Já o parâmetro aSlotTime é um valor determinado pelas características do meio físico. Dessa forma, o tempo de recuo (*backoff*) será um múltiplo desse tamanho de *slot*.

Ao iniciar a transmissão, o nó não permanece monitorando o canal para detectar colisões. Dessa forma, uma vez iniciada, a transmissão de um quadro ocorrerá até o final (seu último *bit*), mesmo que ocorram colisões durante a transmissão. Caso ocorra uma colisão, o destino identificará este problema

com a verificação de que o pacote chegou com erros. Assim, todo o pacote deverá ser retransmitido pela fonte. Com isso, a prevenção da colisão é um assunto importante no 802.11 para que pacotes muito grandes não sejam perdidos fazendo com que a taxa de transmissão efetiva da rede não seja comprometida.

Para fazer a verificação de erros, o receptor da mensagem faz a verificação CRC (do inglês *Cyclic Redundancy Check*). No caso de receber um pacote sem erros, o receptor enviará a mensagem ACK ao transmissor após um tempo chamado de SIFS (do inglês *Short Inter-Frame Spacing*). Assim o transmissor terá certeza de que seu quadro foi entregue com sucesso. Caso o transmissor tenha mais pacotes a enviar, ele escolherá novamente um tempo aleatório de recuo após receber a mensagem de reconhecimento e então, após este tempo, enviará a mensagem seguinte.

Ao receber uma mensagem de confirmação ACK a fim de evitar colisões e transmissões dispendiosas (lembrando novamente que o CSMA/CA não faz a detecção de colisões e, com isso, não aborta transmissões), o transmissor espera um tempo antes de transmitir a sua mensagem seguinte. Isso é feito para que todas as estações escolham tempos aleatórios de recuo e, considerando que estes tempo sejam diferentes como é esperado, a primeira estação que perceber o canal ocioso iniciará a transmissão e as outras aguardarão até o canal ficar ocioso novamente para reiniciarem seus temporizadores. Parte deste processo de transmissão pode ser visualizado no diagrama de tempo da Figura 5 em um cenário onde não ocorreram erros.

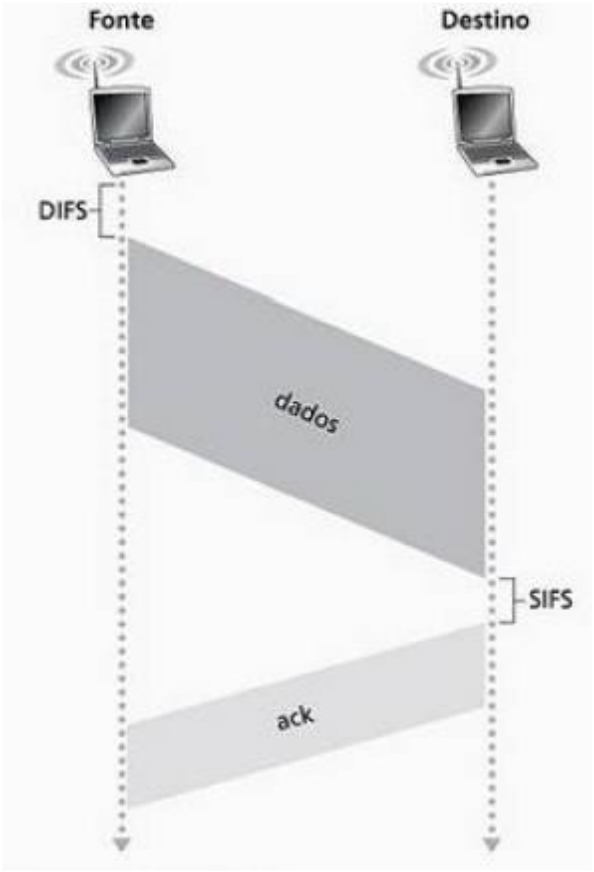

Figura 5. Diagrama de tempo de transmissão de quadro de dados no Wi-Fi com confirmação do receptor. Figura retirada de [7].

Há, entretanto, um problema conhecido como terminal escondido que surge nas redes sem fio. Duas estações conectadas a um mesmo AP podem estar longe o suficiente uma da outra de modo que o sinal transmitido por uma delas não alcance a outra. Dessa forma, ao realizar o procedimento de escuta do canal, mesmo que a outra delas esteja transmitindo, a estação identificará incorretamente que o canal está ocioso e iniciará sua transmissão. Com isso, ocorrerá uma colisão no receptor e os quadros sofrerão

erros. Um exemplo de situação em que ocorre o problema do terminal escondido por ser visualizado na Figura 6.

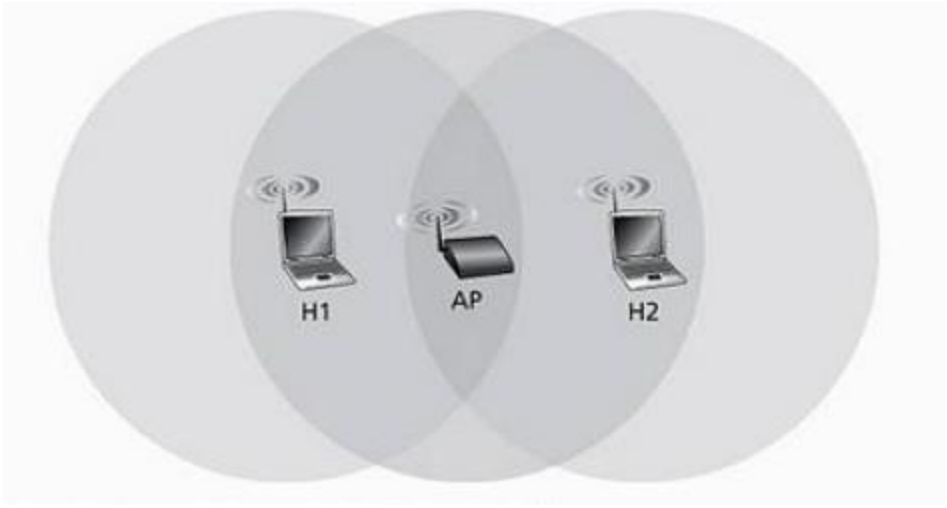

Figura 6. Exemplo de topologia com infraestrutura com o problema do terminal escondido. Figura retirada de [7].

Para solucionar este problema, o 802.11 implementa, de forma opcional, uma reserva inteligente do canal. Após esperar o tempo DIFS, o transmissor que está pronto enviará um pacote denominado RTS (do inglês *Request to send*), solicitando permissão para iniciar o envio de seus dados especificando o tempo necessário para a transmissão. Caso o AP esteja efetivamente ocioso, após aguardar o tempo SIFS, enviará um pacote de CTS (do inglês *Clear to send*) em forma de broadcast autorizando aquele nó a enviar seus dados durante o período solicitado. Ao receber o CTS, o nó requisitante iniciará a transmissão de seus dados após aguardar novamente o tempo SIFS. Por outro lado, os outros nós aguardarão aquele tempo em que o canal está reservado antes de recomeçarem as suas transmissões.

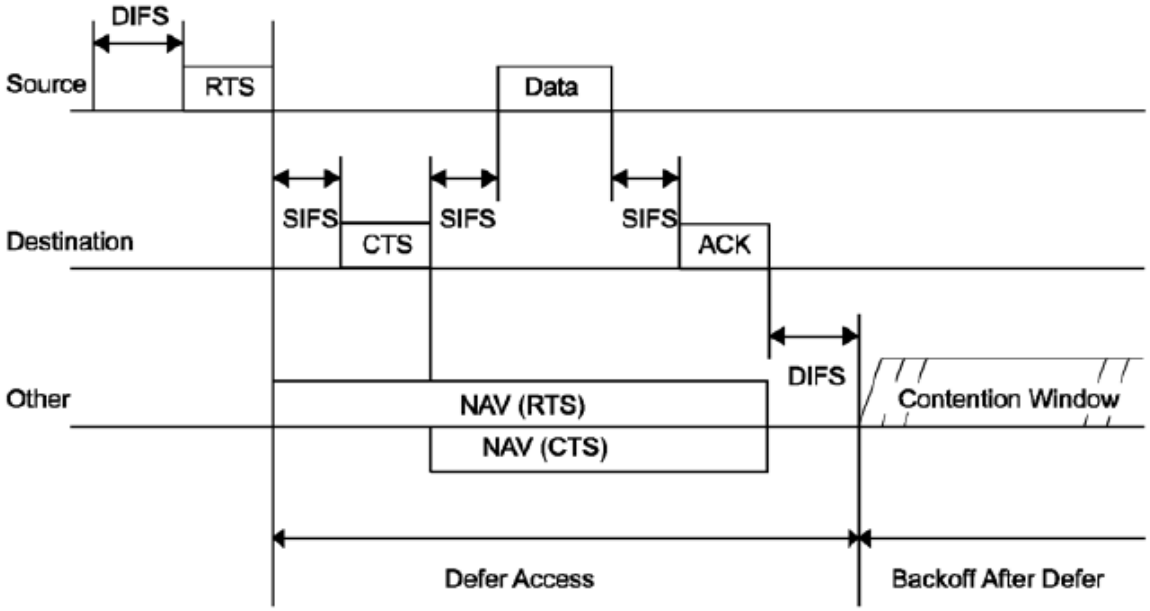

Figura 7. Ilustração do uso de RTS, CTS e NAV. Figura retirada de [6].

Na Figura 7 pode ser visto um exemplo de como ocorre esta reserva do canal com o uso dos pacotes RTS e CTS. Cada pacote contém um NAV (do inglês *Network Allocation Vector*) contendo um número X. O canal é reservado tanto pela fonte quanto pelos receptores da mensagem por um tempo de

X milissegundos após receberem o pacote. A Figura 7 mostra que ao receber o RTS, fonte e destino já reservam o canal pelo tempo NAV solicitado pela fonte. Enquanto para os outros nós da rede, o NAV recebido é mais curto e decorrente apenas do CTS enviado pelo destino.

Apesar de comprovadamente melhorar o desempenho da rede, o uso dos quadros RTS e CTS para resolver o problema do terminal escondido é opcional (pois na topologia da rede pode nem mesmo haver este problema) e pode ser configurado para serem utilizados somente para o envio de grandes quadros de dados, onde uma colisão faria com que ocorresse um grande desperdício de recursos.

2.3.4 A ESTRUTURA DO QUADRO IEEE 802.11

A estrutura completa do quadro IEEE 802.11 pode ser vista na Figura 8. Como um protocolo de camada 2, assemelha-se bastante ao quadro Ethernet, a menos de alguns campos específicos da sua característica de ser sem fio. O campo de carga útil permite o armazenamento de até 2312 bytes e, em geral, possui os dados recebidos através do datagrama IP. O CRC é o campo reservado para armazenar o resultado da execução do código de detecção de erros utilizado assim como no Ethernet.

Os endereços utilizados seguem o mesmo padrão do Ethernet, com 6 bytes, porém no quadro 802.11 existem 4 campos para armazenamento de endereço. O primeiro endereço (endereço 1) é o que referencia a estação sem fio que deve receber o quadro, seja ela um AP ou qualquer dispositivo móvel. O endereço 2 guarda o endereço MAC da estação transmissora. O endereço 3 armazena o MAC do AP da BSS correspondente. Por fim, o endereço 4 é utilizado em redes *ad hoc*.

O restante dos campos são campos de controle, como o número de sequência, a versão do protocolo, e a cifragem utilizada. Os detalhes da utilização desses campos podem ser encontrados na definição do padrão IEEE 802.11 [6].

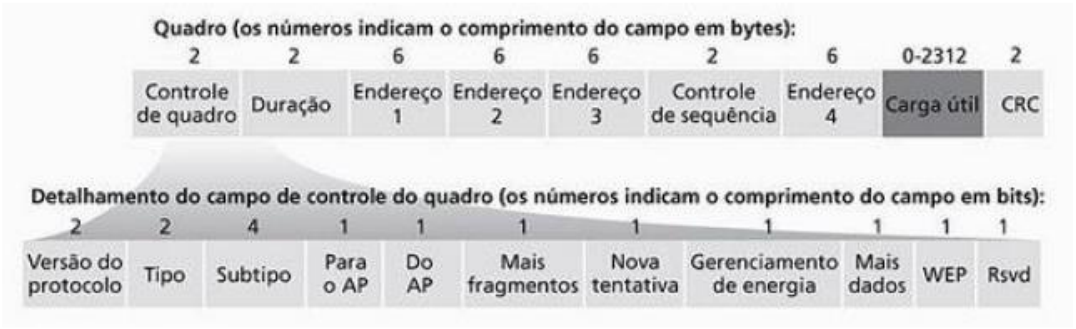

Figura 8. Ilustração dos campos que compõe o quadro do IEEE 802.11. Figura retirada de [7].

2.3.5 CAMADA FÍSICA

O IEEE 802.11 especifica três tipos de implementações na camada física: o FHSS (do inglês Frequency-hopping spread spectrum), o DSSS (do inglês Direct Sequência Spread Spectrum) e o IR (do inglês Infra-Red). Os sistemas baseados em DSSS são os mais populares, devido principalmente ao maior aproveitamento da largura de banda disponível. Na especificação mais básica do padrão IEEE 802.11, esse tipo de sistema usa modulação de banda base DBPSK e QPSK para prover taxas de dados de 1 Mbps e 2 Mbps, respectivamente, e deve operar na faixa de frequências de 2,4 GHz a 2,4835 GHz em um dos 11 canais de 22 MHz de largura de banda

Também existem as extensões de DSSS de alta taxa: o HR-DSSS (do inglês *High Rate-DSSS*), o OFDM (do inglês *Orthogonal Frequency Division Multiplexing*), o ERP (do inglês *Extended Rate PHY*) e o HT PHY (do inglês *High Throughput PHY*). Na especificação do HT PHY, o sistema pode prover taxas de dados de até 600 Mbps.

O DSSS é um dos métodos de espalhamento espectral mais populares. Para prover uma boa robustez diante de interferência e insensibilidade à propagação múltiplo percurso, o DSSS faz uso de espalhamento do sinal, com a utilização de um código pseudoaleatório (PN, do inglês *Pseudo-Noise*)

multiplicado ao sinal. No IEEE 802.11 DSSS PHY, um mesmo código PN é utilizado por todos os usuários da rede.

A técnica DSSS consiste em transmitir, para cada bit, uma sequência Barker (código pseudoaleatório PN) de bits. Assim, cada bit que vale 1 é substituído por uma sequência de bits e cada bit que vale 0 pelo resultado do complemento da primeira sequência.

A camada física da norma 802.11 define uma sequência de 11 bits (10110111000) para representar um 1 e o seu complemento (01001000111) para codificar um 0. Esta técnica, chamada *chipping*, consiste em modular cada bit com a sequência Barker. O *chipping* é responsável por efetuar o espalhamento espectral do sinal e garantir maior robustez contra interferências.

O padrão IEEE 802.11n permite chegar a transmissão de dados em taxas de até 600Mbps, trabalhando nas faixas de frequência de 2,45GHz e de 5GHz, sendo esta última preferível quando se deseja evitar interferências, pois a primeira faixa compartilha o espectro utilizado por outros equipamentos, como telefones sem fio e fornos micro-ondas. Por outro lado a frequência de 5GHz possui menor alcance. Esta tecnologia atinge maiores taxas de transmissão devido ao uso de OFDM (do inglês *Orthogonal Frequency Division Multiplexing*) e ao uso de múltiplas antenas – MIMO (do inglês *Multiple Inputs Multiple Outputs*). Alguns parâmetros típicos nas diferentes versões deste protocolo podem ser vistos na Tabela 1.

Tabela 1. Valores típicos dos parâmetros das diferentes extensões da norma IEEE 802.11.

	Taxa de bits	**Faixa de frequência**	**Largura de banda de cada canal**
802.11b	Até 11 Mbps	2,4-2.485GHz	22MHz
802.11g	Até 54 Mbps	2,4-2.485GHz	22MHz
802.11n	Até 600 Mbps	2,4-2,485GHz e 5GHz	20MHz ou 40MHz

2.4 PROTOCOLO AODV

No contexto de redes *ad hoc*, encontra-se o protocolo reativo de roteamento AODV, muito eficaz pela sua característica simplista e adaptável de roteamento e em consequência amplamente utilizado em implementações de redes *ad hoc*. É simples por não envolver muitos tipos de mensagens e informações de controle na descoberta de rotas e volátil por manter informações de rota apenas enquanto o nó precisa ou faz uso dela [26]. Por ser um protocolo que atua sob demanda, não existem formações de rotas sem a devida requisição. As rotas são descobertas na ocorrência de demanda por parte de qualquer dos nós da rede para o envio de dados.

Dentre os objetivos de sua concepção estão: a eliminação do excesso de mensagens de controle para descoberta de novas rotas, a resolução local de qualquer problema de rotas e a propagação das alterações de rotas localmente aos vizinhos que realmente precisam ser informados da mudança. Existem algumas mensagens de controle usadas para formação e manutenção de rotas pelo protocolo descritas com mais detalhes na seção seguinte.

2.4.1 FUNCIONAMENTO

2.4.1.1 Descoberta de Caminho

Sempre que certo dispositivo necessita se comunicar com outro nó da rede é gerada uma mensagem de *broadcast* RREQ (do inglês *Route Request*) para descobrir uma rota até o destino. O sucesso desta requisição se dá quando o RREQ alcança o alvo da requisição ou nó intermediário com rota para o destinatário depois de ser retransmitido pelos nós. Em seguida, quando uma das duas situações anteriores ocorre é enviado, em modo *unicast*, pelo dispositivo que detêm uma rota para o destino um pacote RREP (do inglês *Route Reply*) até a origem da requisição, novamente através de retransmissões dos nós da rede.

Os nós de uma rede que utiliza o AODV possuem duas informações relevantes para o processo de descoberta de rotas: o número de sequência do nó e o número de identificação de RREQ, o *RREQ_id* [23] e *broadcast_id* [8]. O primeiro serve para validar se a rota é recente ou antiga. Já o segundo, por sua vez, é responsável por identificar unicamente os pacotes RREQ juntamente com o endereço de origem da requisição. São utilizados dois tipos de números de sequência que auxiliam na descoberta de rotas únicas pelo AODV, pois proporcionam rotas livres de laços: de origem e de destino.

O número de sequência do destino é usado pela origem, ou nó intermediário da rota, a fim de diferenciar duas rotas para o mesmo destino. O nó de destino de uma requisição envia em sua mensagem de resposta o número de sequência dele e incrementa este valor a cada nova solicitação de rota. Assim, quando rotas alternativas chegam ao nó de origem, estas são analisadas com relação a este número e é escolhida a rota que possui o maior deles.

Já o número de sequência da origem, é usado na formação do caminho reverso (do destino até a origem), onde cada nó, recebendo um RREQ, salva o IP do vizinho pelo qual recebeu este pacote até um tempo chamado tempo de vida (*lifetime*) necessário para que o RREQ percorra toda a rede e origine uma resposta ao solicitante da rota. Este parâmetro é usado para se saber quão recente é o caminho reverso até a origem.

A formação do caminho, no sentido da origem até o destino – rota direta, se dá através da resposta do destinatário, ou por um nó intermediário da rota para o destino, com o envio de um pacote RREP. Caso um nó intermediário receba mais de um RREP, este apenas irá encaminhar aquele que tiver um número de sequência do destino maior que o número de sequência presente em sua tabela de roteamento. Isto ocorre pois o número de sequência é incrementado a cada nova descoberta de rota para possibilitar a formação de rotas mais atuais para os mesmos nós de destino. No caso de receber dois pacotes RREP com número de sequência iguais, aquele com menor número de saltos é armazenado na tabela de roteamento.

2.4.1.2 Formação do Caminho Reverso

A viagem do pacote RREQ pela rede, até alcançar o alvo, automaticamente forma o caminho reverso da rota até o destino. Para isso, todos os nós responsáveis por encaminhar o pacote RREQ até o destino devem "lembrar" de quem esta mensagem foi recebida. Obviamente, nem todos os nós atingidos pelo pacote de requisição farão parte da rota. Por isso é incluído um temporizador para apagar o caminho reverso não utilizado caso não seja recebido um RREP dentro do tempo determinado (ACTIVE_ROUTE_TIMEOUT). A Figura 9 ilustra como são formadas as rotas no AODV e também o processo descrito de formação do caminho reverso.

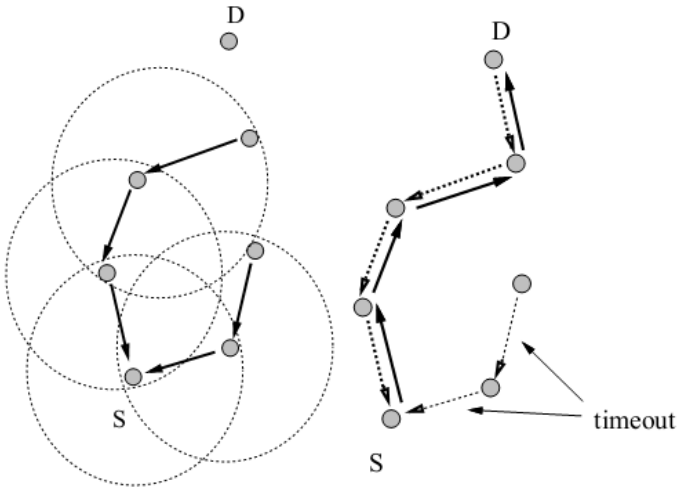

Figura 9. Formação de rota em rede *ad hoc* com AODV onde é mostrado os *broadcast* de RREQ, representados pelos círculos, e consequente formação do caminho reverso à direita, e a formação da rota direta após a resposta RREP de D mostrado na figura à esquerda.

No exemplo da Figura 9, o nó "S" faz uma solicitação de rota para o nó "D". Os círculos na imagem à esquerda representam o *broadcast* de RREQs. À medida que cada nó repassa um RREQ, ele

gera uma entrada em sua tabela para o caminho reverso como na figura da esquerda indicado pelas setas. Ao receber este pacote o nó destino inicia um processo inverso (imagem da direita), desta vez com RREPs enviados por todo o caminho até atingir o nó de origem para formar o caminho S→D representado na figura da direita pelas setas não tracejadas. O percurso D→S, chamado de "caminho reverso" é formado assim que o RREQ alcança o dispositivo de destino já que em cada nó da rota é guardada informação sobre o remente do pacote recebido, salvo se esgote o tempo de vida da mensagem.

Existe ainda uma "lista de precursores" conforme [23], para reportar possíveis quebras de link e enviar RREPs. Constantemente os nós verificam o estado dos enlaces para checar se estes permanecem ativos ou não. Caso haja alguma falha, a lista de precursores é usada e um pacote RERR enviado para cada elemento que usa este enlace para comunicação. No envio de RREP, cada nó busca na sua lista de quem recebeu o RREQ e em seguida responde em *unicast* ao nó antecessor.

Através de mensagens do tipo HELLO, o protocolo AODV possibilita aos nós o monitoramento de enlaces que fazem parte de uma rota válida, ou seja, ainda são passíveis de uso, para controle de rotas e readaptação da rede em caso de falhas com a descoberta de caminhos alternativos. Caso haja falha de algum enlace na rede, um pacote de RERR é enviado de forma *unicast* para os dispositivos que utilizam este link para roteamento de tráfego, dando início ao processo de reparo local de rotas. Em geral, uma quantidade tolerável de pacotes HELLO podem ser perdidos antes de um reparo de rota.

2.4.1.3 Reparo Local de Rotas

Caso haja um enlace com falha em rota ativa, existe a detecção pelo nó vizinho da rede (na Figura 10 o nó B) e todo o tráfego passante passa a ser armazenado. O reparo local tem início assim que é percebida a falha com *broadcast* de RREQ na rede. Refeita a rota com o recebimento do RREP, os dados armazenados são novamente enviados usando a nova rota até o destinatário. No exemplo da Figura 10, o nó B percebe a perda de comunicação com C, por não receber HELLO durante um período de guarda (*timeout*) e transmite um RREQ (círculo 1 na figura) para descobrir um caminho alternativo.

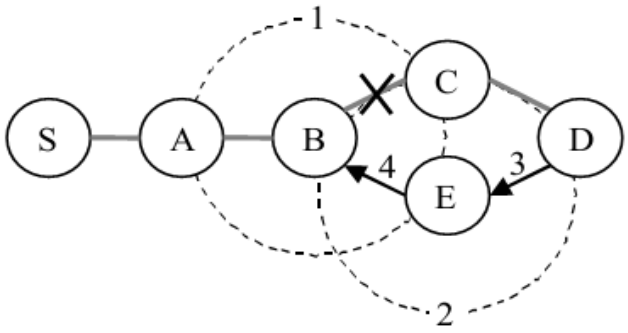

Figura 10. Reparo local de rota ilustrando a perda de comunicação do nó C. Figura retirada de [2].

2.5 A PLATAFORMA ANDROID

2.5.1 HISTÓRICO

Atualmente verificamos uma grande popularidade dos *smartphones* e *tablets* que contam com um crescimento acelerado desde 2007, período que contou com grandes novidades tecnológicas como o lançamento do primeiro iPhone da Apple e a criação do Android, sistema operacional *open-source* para dispositivo móveis.

Devido a esta proliferação, vem se aumentando o interesse pelo estudo em aplicações para este tipo de dispositivo que vem sendo cada vez mais utilizado pelo grande público como ferramenta de lazer e de trabalho. Isso ocorre graças à usabilidade que eles dispõem e da grande quantidade de componentes a eles integrados, incluindo interfaces Wi-Fi baseadas no padrão 802.11.

Além disso, a sua plataforma aberta e flexível para o desenvolvimento de aplicativos chama a atenção também dos desenvolvedores. É disponibilizada uma série de ferramentas e bibliotecas que permitem o acesso a todos os componentes presentes nos aparelhos das mais diferentes arquiteturas

presentes no mercado, tornando viável que uma grande variedade de aplicações sejam desenvolvidas. O sucesso desta plataforma pode ser contatado verificando sua loja virtual de aplicativos, que hoje contém mais de 700 mil aplicações disponíveis, sendo algumas gratuitas enquanto outras são pagas.

O início do projeto Android foi em 2005 e sua primeira versão ficou pronta em novembro de 2007, mas sua primeira versão comercial, o Android 1.0, foi lançada somente em setembro de 2008, tornando-se uma plataforma de código aberto (*open source*) sob a licença Apache. Este sistema utiliza tecnologias de código livre e é baseado no sistema operacional Linux (inicialmente como uma variação do *kernel* versão 2.6). Por ser *open source* seu código está disponível aos desenvolvedores e pode ser consultado por qualquer um que deseja melhorá-lo, documentá-lo ou simplesmente estudar suas funcionalidades. Hoje, o Android é desenvolvido pelo Google (desde 2005, quando fez a compra da marca Android) e pela Open Handset Alliance (um consórcio que conta com a presença de várias grandes empresas, incluindo o próprio Google). Seu desenvolvimento é dado em linguagens como C, C++ e Java.

Curiosamente, os nomes das versões comerciais do Android vem sendo lançados, desde sua versão 1.0, em ordem alfabética e sempre baseados em nomes de doces. São eles Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich e Jelly Bean, que representa a versão 4.3, a mais atual. Já se fala na versão 5.0 (Key Lime Pie) com previsão de lançamento para o segundo semestre de 2013.

Atualmente, a versão mais utilizada do Android ainda é a Gingerbread (v2.3), pois grande parte dos celulares com menos recursos utilizam esta versão e não têm suporte para versões mais atuais, como a Ice Cream Sandwich (ICS, v4.0), por exemplo. Um gráfico com a evolução da proporção de aparelhos, por versão do Android, pode ser visto na Figura 11.

Hoje, o Android já está presente em mais de 900 milhões de aparelhos móveis no mundo. Ele é o sistema operacional de 80% dos *smartphones*, estando bem à frente dos seus concorrentes. Este sistema operacional foi determinante na história para juntar um mercado de tecnologias móveis antes fragmentado.

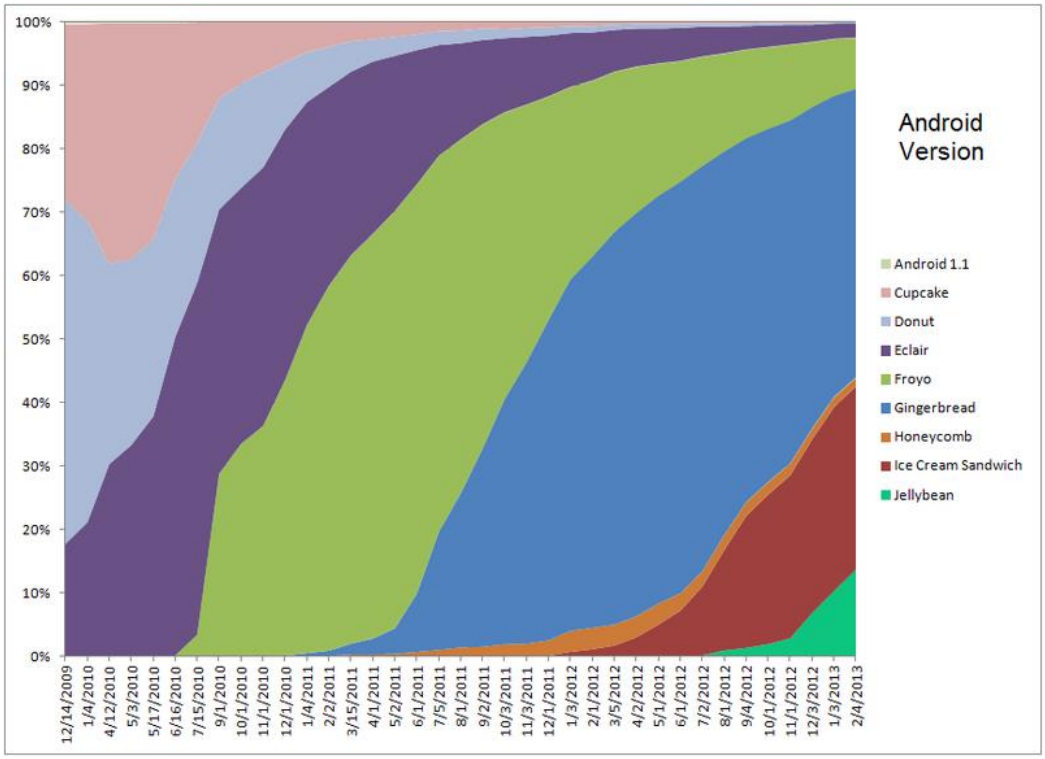

Figura 11. Proporção de versões do Android em aparelhos ao longo do tempo.

2.5.2 ARQUITETURA DA PLATAFORMA ANDROID

O Android é composto por uma pilha de softwares, como pode ser visto na Figura 12. Na camada mais inferior, está o núcleo do sistema operacional Linux, versão 2.6, modificado, incluindo *drivers* como o de Wi-Fi e de câmera. Acima desta camada encontram-se as bibliotecas que fazem o gerenciamento dos componentes físicos e provêm interfaces com os elementos de mais baixo nível, como protocolos. Neste mesmo nível também está a Android Runtime, composta pela Dalvik Virtual Machine (DVM), que será detalhada mais à frente.

Nas camadas superiores encontram-se os *frameworks* do Android, que facilitam o trabalho do desenvolvedor disponibilizando uma interface de alto nível para programação. Na camada mais alta, encontram-se as aplicações que estão disponíveis aos usuários.

A execução de aplicações no sistema operacional Android é feita sobre uma máquina virtual Java semelhante a JVM (do inglês *Java Virtual Machine*) presente comumente nos dispositivos. Dessa forma, o desenvolvimento de aplicativos para Android é feito na linguagem Java e é compilado para arquivos de *bytecode* com a extensão ".dex".

Essa máquina virtual Java presente no Android é denominada *Dalvik Virtual Machine* (DVM), a qual é o ambiente de execução dos programas neste sistema operacional. Ao mesmo tempo, estão disponíveis no Android um conjunto de bibliotecas Java padrão para o provimento de funções simples e também a comunicação com os componentes do aparelho. A DVM é uma modificação da JVM padrão otimizada para funcionar com menos recursos de memória e processamento e também para permitir a presença de várias instâncias da máquina virtual executando ao mesmo tempo. Essa última característica permite que aplicações executem em ambientes completamente separados, garantindo mais estabilidade, escalabilidade e segurança.

Os arquivos ".dex" compilados são executados dentro das DVMs. Além do Java, programas para Android podem ser escritos em C ou C++, utilizando a ferramenta Native Development Kit (NDK). Para determinadas aplicações, é mais recomendado utilizar estas linguagens por permitirem chegar a um nível mais detalhado de desenvolvimento, por desempenho, ou até para utilizar funções e bibliotecas presentes somente nessas linguagens. No entanto, a maior parte dos aplicativos não necessita do uso da NDK e podem ser desenvolvidos exclusivamente em Java. É recomendado que se utilize a NDK somente quando isso seja essencial ao programa e não pela preferência em se programar nessas linguagens (C ou C++).

A estrutura de uma aplicação desenvolvida para Android é baseada em torno de *Activities*, as quais são as classes que, de fato, estão visíveis para o usuário. Por exemplo, uma tela de cadastro será uma classe que herda os componentes da classe *Activity*, enquanto uma segunda classe que trata os dados enviados por esta tela (e os armazena num banco de dados) não é uma classe filha da *Activity*. Somente uma *Activity* de uma aplicação pode ser visualizada ao mesmo tempo. As *Activities* utilizam componentes gráficos de interação com o usuário, como formulários, botões, textos, imagens, etc.

Para se desenvolver aplicativos para Android, deve-se fazer o uso de seu sistema de desenvolvimento (SDK), disponível de forma gratuita para download na própria página de desenvolvedores do Android, juntamente com a interface de desenvolvimento (IDE) Eclipse e uma grande quantidade de documentação tanto para a instalação e preparação do ambiente quanto para o desenvolvimento de aplicações. Para que a SDK funcione corretamente, a máquina utilizada para desenvolvimento deve conter o kit de desenvolvimento Java (JDK) instalado.

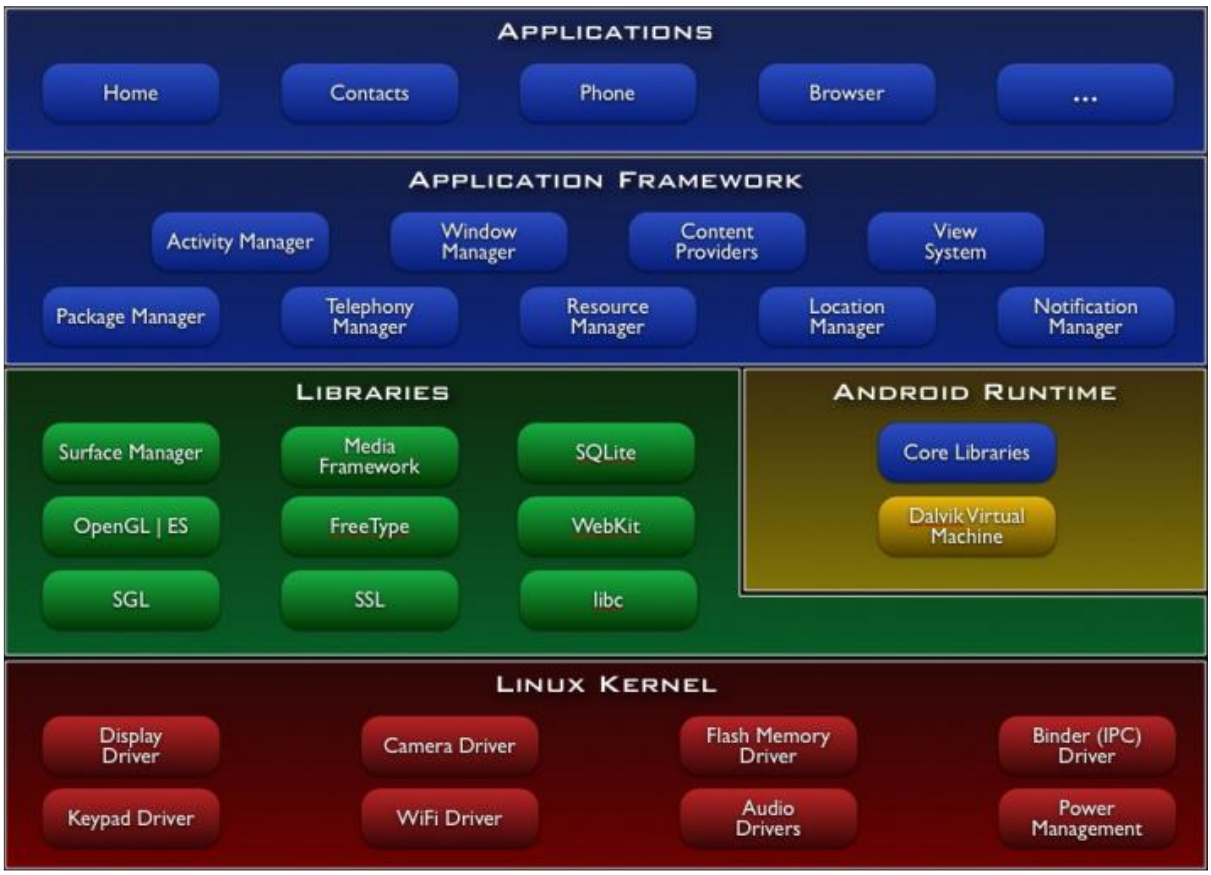

Figura 12. Arquitetura de software do sistema operacional Android organizada por camadas.

2.5.3 O MERCADO

O sucesso dos smartphones se deve pela presença de aplicações e componentes que integram, em um mesmo aparelho, diversos dispositivos tecnológicos bastante populares anteriores a ele e tudo isso com acesso à Internet, que permite a navegação na Web e também que essas aplicações integrem-se com dados em tempo real. Estes aparelhos também permitem que seus usuários comuniquem-se instantaneamente através de seus aplicativos.

Dessa forma, surge-se a proposta de desenvolver sistemas operacionais para estes novos equipamentos móveis de forma a criar-se padrões que facilitariam o trabalho de desenvolvedores de aplicações para estes aparelhos e também para que os usuários acostumassem-se com os sistemas presentes e escolhessem aqueles que possuíssem maior afinidade. Os fabricantes de hardware também ganham vantagens com o estabelecimento de sistemas operacionais, de forma que este é um serviço que ele não precisaria mais se preocupar.

Com tudo isso, nem estes fabricantes de hardware nem os desenvolvedores dos sistemas operacionais também teriam preocupações primárias com a criação de funcionalidades para estes aparelhos, pois os desenvolvedores de aplicações cuidariam disso. Assim, estes ficam mais livres para o melhoramento dos componentes do dispositivo, que estão a cada dia mais avançados, com melhorias de usabilidade e com a criação de ferramentas para os desenvolvedores evoluírem seus aplicativos.

Atualmente o grande concorrente do Android, o iOS da Apple que roda nos seus aparelhos (iPhone, iPad, etc), é uma plataforma bastante reconhecida pela sua usabilidade e pelo seu desempenho sobre o hardware desta mesma organização. Entretanto, esta sistema operacional possui menos mercado devido aos preços mais caros dos aparelhos que o embarcam, porque sua loja virtual é menos aberta a desenvolvedores e seus aplicativos são na maioria pagos, enquanto no Android boa parte deles é gratuito.

Outros sistemas operacionais para celulares são o Blackberry, cujo desenvolvedor é empresa homônima e o Windows Phone, desenvolvido pela Microsoft. Cada um com suas limitações seja de disponibilidade de aplicativos, ou recursos nos seus aparelhos, possuem uma fatia do mercado ainda

insignificante. Alguns outros sistemas operacionais como o Symbian, detido pela Nokia, deixou de ser produzido devido a sua limitação de recursos. Assim a Nokia passou a implementar em seus aparelhos o sistema operacional da Microsoft.

Atualmente fala-se no lançamento de mais dois sistemas operacionais de código aberto ainda para o ano de 2013. O Ubuntu Touch, pela já reconhecida desenvolvedora de sistemas operacionais Linux para computadores, e o Firefox OS, pela Mozilla, que atualmente é bastante popular pelo seu navegador de Internet, o Firefox. Estes sistemas operacionais já estão disponíveis para testes em alguns aparelhos, mas ainda de forma limitada.

Além de todas as vantagens já citadas do Android sobre os outros sistemas operacionais, este possui bastante popularidade por vir de fábrica instalado em celulares e tablets da maior parte das grandes empresas de produção de hardware atuais, como Samsung, LG e Motorola. Além disso, está presente nos aparelhos das mais diversas capacidades de processamento, desde os mais básicos com poucos recursos até mesmo nos carros-chefes destas empresas.

2.5.4 O ANDROID COMO FERRAMENTA DE PESQUISA

Esta proliferação de *smartphones* faz com que diversos projetos de pesquisa sejam voltados para o desenvolvimento de componentes e aplicações para estes dispositivos. Por ser uma plataforma de código aberto e com ferramentas de desenvolvimento disponibilizadas de forma gratuita, o Android é a escolha natural dos pesquisadores para a implementação dos seus projetos em ambiente real para a obtenção de dados. Além disso, essa é uma plataforma bastante simples e com bastante documentação, podendo-se inclusive utilizar aplicações auxiliares presentes, até mesmo gratuitamente, em sua loja virtual.

Dada sua característica de código aberto e por ser baseado em Linux, plataforma bem consolidada quando se fala em sistemas operacionais para computadores, seu núcleo está suscetível a mudanças pelo desenvolvedor para implementar funcionalidades que não estão disponíveis por padrão, como a alteração das formas de comunicação, do consumo de energia, da forma de processamento, etc.

3 REVISÃO BIBLIOGRÁFICA

3.1 APRESENTAÇÃO

Uma gama extensa de trabalhos tem surgido pela consciência energética a respeito do meio ambiente mas principalmente pela demanda de capacidade das baterias de suportarem massivas cargas de processamento ao rodarem muitos aplicativos simultaneamente. Isto requer que cada componente dos sistemas operacionais destes aparelhos sejam projetados para consumir a menor energia possível o que, obviamente, não se trata apenas de diminuir a quantidade de energia a ser gasta mas sim, de otimizar esse gasto fazendo com que os dispositivos permaneçam mais tempo em funcionamento.

3.2 TÉCNICAS DE ECONOMIA DE ENERGIA

Visando estender o tempo de vida da rede, foram criadas algumas métricas para guiar algoritmos de roteamento e evitar assim que a topologia da rede *ad hoc* seja alterada por questões energéticas. Sabe-se que a energia é ponto chave no desenvolvimento para dispositivos móveis, por serem atualmente equipados com baterias de pequena carga.

Como explicado anteriormente, uma das caraterísticas principais das redes *mesh* (*ad hoc*) é sua topologia, daí a motivação para abordar o aspecto energia no processo de roteamento, para evitar ao máximo uma quebra de link ou "morte" de nó da rede mantendo a topologia inalterada pelo maior tempo possível. Abaixo são descritas algumas métricas de roteamento propostas em redes *ad hoc* considerando a energia dos aparelhos:

3.2.1 MINIMUM TOTAL TRANSMISSION POWER ROUTING (MTPR)

Utiliza-se neste caso o percurso com menor energia total, ou seja, é calculada entre cada nó da rota a energia necessária para uso do enlace e por fim este valor é somado aos demais consumos ao longo do percurso. Sabe-se que este cálculo deve levar em conta o nível de ruído de cada enlace, a taxa de erro de bit (Ψ_j) e a distância entre os nó para obter a energia gasta nos enlaces participantes da rota descoberta. A relação sinal ruído obtida pela equação abaixo envolve todos estes elementos para o cálculo da potência de transmissão:

$$SNR = \frac{P_i G_{i,j}}{\sum_{k \neq i} P_k G_{k,j} + \eta_j} \Psi_j \ , \tag{2}$$

$$R_l = \sum_{i=0}^{D-1} R(n_i, n_{n+1}), \text{ para todo } n_i \in \text{rota}, \tag{3}$$

$$R_k = min P_l, \quad l \in A, \tag{4}$$

onde em meios de comunicação sem fio, o ganho do canal ($G_{i,j}$) é modelado por $1/d^n$ (d sendo a distância entre os nós e n igual a 2 para pequenas distancias e 4 para grandes distâncias), onde $R(n_i, n_{n+1})$ é o custo de energia relativo ao enlace entre n_i *e* n_{n+1} e A o conjunto de todas as rotas possíveis. R_l (3) é portanto a soma de todos os custos de enlace de uma rota com D nós e R_k (4) a rota escolhida com o método.

3.2.2 MINIMUM BATTERY COST ROUTING (MBCR)

A métrica anterior não é tão eficiente quando analisada individualmente já que não leva em conta o nível de bateria dos nós individualmente. Para tal, o MBCR, proposto por S. Singh, M. Woo e C. S. Raghavendra [24] , calcula o custo de energia relativo ao nó i e faz uma somatória dos custos de bateria c_i envolvidos na rota R_j, assim

$$f_i(c_i^t) = \frac{1}{c_i^t}, \tag{5}$$

$$R_j = \sum_{i=0}^{D_j - 1} f_i(c_i^t) \ , \tag{6}$$

$$R_i = \min\{R_j \mid j \in A\}, \quad (7)$$

onde A é o conjunto de todas as rotas possíveis. O parâmetro c_i^t corresponde ao nível de bateria do dispositivo i no instante t e f_i ao custo relacionado à c_i^t como mostra a Equação (5). A Equação (7) mostra a escolha do caminho com menor valor R_j entre as rotas calculadas em (6), ou seja, o caminho R_i no qual os dispositivos apresentam maiores níveis de energia.

Na situação hipotética ilustrada na Figura 13 a origem possui duas rota até o destino para escolher. Dependendo da estratégia de roteamento usada a rota escolhida será diferente visto que os custos de energia são distintos. Na rota 1, o nó 3 tem alto custo, mas a soma dos custos referentes a esta rota é menor em relação a soma de custos da rota 2. Portanto se a métrica MBCR fosse usada na rede, a rota 1 seria a escolhida.

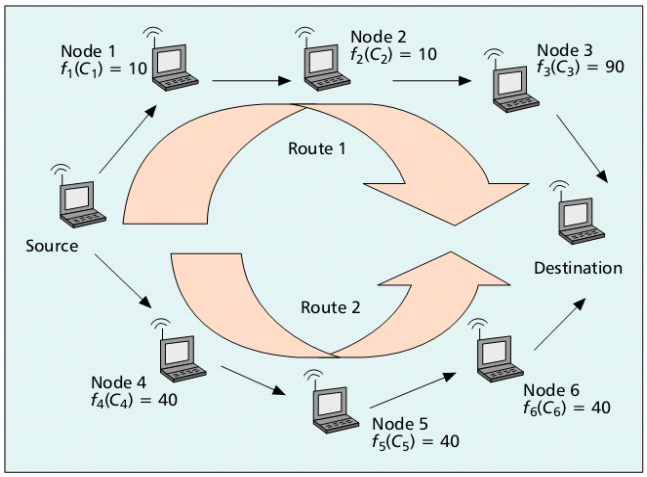

Figura 13. Exemplo de topologia com duas rotas possíveis a serem escolhidas através de parâmetros energéticos.

3.2.3 MIN-MAX BATTERY COST ROUTING (MMBCR)

Uma desvantagem do MBCM é que mesmo que um nó esteja em estado crítico, como somente a soma dos custos é levada em consideração, há chances de uma rota ser formada através deste nó. Neste caso, também desenvolvido por Singh, Woo e Raghavendra [24], eles calculam como o custo da rota através do "menor maior" custo de bateria dentre todos os nós que compõe a rota. Na Equação (8), *Rj* está relacionado desta vez com o valor de bateria do nó com menor nível presente na rota obtida, ou melhor, o maior custo. Assim

$$R_j = \max f_i\left(c_i^t\right), \text{ i} \in \text{rota j, onde} \quad (8)$$

$$R_i = \min\{R_j \mid j \in A\}. \quad (9)$$

Agora a rota selecionada R_i com a Equação (9) contém nós com níveis de bateria superiores ao menor valor encontrado em nós de todas as rotas possíveis.

3.2.4 CONDITIONAL MAX-MIN BATTERY CAPACITY ROUTING

Há ainda outra técnica proposta por Toh [9] onde é proposto o uso de um limiar para o MMBCR. Consiste em definir um valor *gama* (ɣ) para limitar a escolha de rotas em que todos os nós tenham níveis de bateria superiores a esta referência e além disso, caso qualquer das rotas para determinado destino estejam além do limiar proposto, é feita a escolha da rota com menor energia de transmissão total (MTPR).

3.2.5 MINIMUM DRAIN RATE

As métricas anteriores apenas utilizam dados de bateria remanescente, sem atentar para o fato de a bateria estar sendo drenada com muita intensidade pelo tráfego de rede. Baseado nas condições de tráfego atuais e visando prever o tempo de vida dos dispositivos, o *Minimum Drain Rate* – MDR de Dongkyun Kim, J.J Garcia-Luna-Aceves, Katia Obraczka, Juan-Carlos Cano e Pietro Manzoni [13] é um mecanismo com o intuito de avaliar as informações de tráfego de rede e usá-las juntamente com as informações de bateria usadas pelos métodos anteriores, porém a correlação entre os dados de tráfego de rede e de bateria não é algo simples de ser resolvido. Ao invés disso, a taxa de consumo de bateria é quantificada para indicar a utilização do dispositivos, seja transmitindo, recebendo ou escutando o canal para estimar o consumo de energia.

A taxa de consumo de bateria mencionada é obtida pelo dispositivo monitorando o gasto de energia médio por segundo. Este valor DR_i (fórmula para cálculo presente em [13]) refere-se a taxa média de consumo energético do nó n_i. É possível obter, a partir desta taxa, o tempo de vida C_i do nó fazendo a seguinte razão:

$$C_i = \frac{RBP_i}{DR_i}, \tag{10}$$

onde RBP_i representa o dreno de bateria do dispositivo i no tempo. Deste modo, podemos também calcular o tempo de vida máximo de uma rota r sabendo o menor tempo de vida C_i de nó integrante do caminho. Seja

$$L_p = min_{\forall n_i \in r} C_i. \tag{11}$$

O mecanismo MDR então calcula a melhor rota r_M, entre todas as possíveis r_*, que apresentar o maior tempo de vida, logo

$$r_M = r = max_{\forall r_i \in r_*} L_i. \tag{12}$$

3.3 PROPOSTAS DE AODV PARA PLATAFORMA ANDROID

A utilização de *smartphones* Android em redes *ad hoc* foi implementada por Jradi e Reedtz [11], por meio de uma biblioteca *ad hoc* desenvolvida para habilitar estes dispositivos à formarem uma rede *ad hoc* roteada através do protocolo AODV. Para testar o funcionamento da arquitetura foi criado um aplicativo de chat simulando o envio de dados para os dispositivos da rede. Como plataforma para o experimento os pesquisadores utilizaram o Android 2.1 em aparelhos HTC Hero e Google Nexus One.

Seguindo a mesma linha desenvolvida por Jradi e Reedtz, Corriero et al [12] implementaram uma rede MANET (do inglês *Mobile Ad Hoc Network*) com uso do protocolo FB-AODV (do inglês *Flow-based Ad hoc On Demand Distance Vetor*), utilizando o aspecto *flow-based* do protocolo para preservar a bateria dos aparelhos telefônicos. Isto é alcançado em grande parte pelo uso da métrica WCIM (do inglês *Weighted Contention and Interference Routing Metric*) pois seleciona rotas pelas características do fluxo de dados e da qualidade do enlace de transmissão.

Neste sentido e relacionando eficiência energética com o roteamento AODV para redes *ad hoc*, Jing [10] simulou o uso de informações de bateria e o gasto de energia para transmissão no enlace como dados adicionados no corpo das mensagens de HELLO afim de usar tais informações no controle e formação de rotas. A abordagem foi simulada usando o OPNET em um cenário onde 30 nós ficaram dispostos em uma área de aproximadamente 1200x500m² seguindo o modelo aleatório de mobilidade *waypoint*. O autor também criou uma técnica de alerta para níveis baixos de bateria afim de acionar uma atualização de rota e evitar que nós da rede sejam exauridos.

As mensagens de HELLO foram alteradas como mostrado na Figura 14 (HELLO RREQ) [9]. No lugar do *destination sequence number* está o nível de bateria verificado no dispositivo, f(s). A alteração não compromete o funcionamento do protocolo AODV por não perder o sentido principal de notificação de nó ativo.

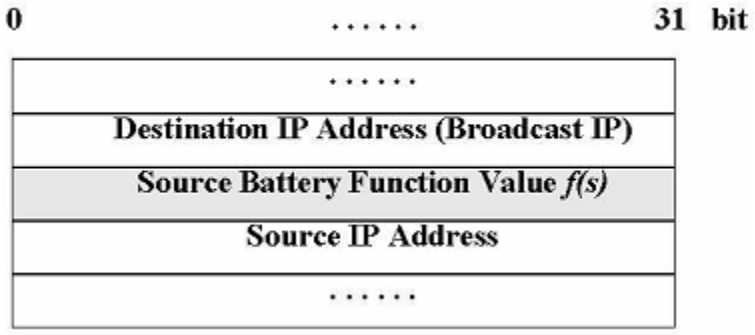

Figura 14. Cabeçalho do pacote HELLO com inclusão de informação de energia. Figura retirada de [10].

A abordagem desenvolvida por Gupta e Das [14] para um AODV *energy-aware* fundamenta-se na criação de uma métrica de roteamento em função da carga de bateria residual e do número de vizinhos do dispositivo móvel. Além disso, sempre que no modo *idle* (quando não há atividade de transmissão ou recepção de pacotes) é atingido, o rádio Wi-Fi é desligado evitando gastos desnecessários de energia usando um esquema de temporização, sendo T_{active} o tempo no qual o aparelho permanece em modo ativo e T_{sleep} o tempo em modo *idle* (*sleep*). Os nós são classificados em 3 classes: normal, *warning* e *danger zone*. Esta classificação visa associar cada nó da rede a um custo de roteamento relativo ao nível de bateria do dispositivo. No caso de haver troca na zona de classificação de certo dispositivo, este se encarrega de enviar para a origem um pacote RWARN a fim de iniciar um novo processo de formação de rota. O RWARN se diferencia do RERR por não apagar a rota ativa e dessa forma evita interrupções na comunicação fim a fim.

O custo de uma rota com *n* saltos é dado por C =$\sum_{i=1}^{n} c(x_i)$, onde o custo em cada nó x_i equivale a valores pré-definidos de $C_{normal,}\, C_{warning}\, ou\, C_{danger}$ com $C_{normal} < C_{warning} < C_{danger}$. A proposta leva em consideração ainda o número de vizinhos no cálculo dos custos para diminuir a probabilidade de formação de rotas com nós pertencentes às regiões $C_{warning}\ e\, C_{danger}$ e forçar assim que novas rotas sejam formadas pelo dispositivos de sua vizinhança. Para isto foi adicionado um fator k_i para descrever o número de vizinhos no cálculo de custo de cada nó que passa a ser $c(x_i) = C_{normal,}\ k_i C_{warning}\ ou\, k_i C_{danger}$.

O resultado destas modificações no protocolo AODV e no funcionamento do rádio Wi-Fi observado em simulação foi de um aumento no tempo de vida da rede de mais de 65% para tráfegos de baixa intensidade e entre 8% e 26% para níveis de tráfego mais elevados.

Por outro lado, existe ainda uma tática mais abrangente de relacionar eficiência energética e roteamento. Trata-se do uso de informações da camada de aplicações para auxiliar no processo de roteamento da camada de rede na forma do algoritmo, e chamada de *Application-Aware Energy Efficient Routing Algorithm* – AWERA [15]. Para tanto, uma arquitetura em camadas formada por componentes presentes nestas camadas anteriormente citadas foram acrescidas de 5 novos módulos, quais sejam: Application Monitor (AM), Remaining Energy Monitor (REM), Energy Constraint Computation module (ECCM), Routing Table (RT) e Routing Table Maintenance Module (RTM) como mostra a Figura 15.

O módulo AM coleta informações da aplicação em processamento no *smartphone* referentes ao tipo da aplicação e utilização da rede. Já o REM monitora a quantidade de energia remanescente do dispositivo. Estes dados coletados pelos AM e REM são enviados ao ECCM para calcular através de um modelo energético uma métrica que reflita a demanda de energia da aplicação.

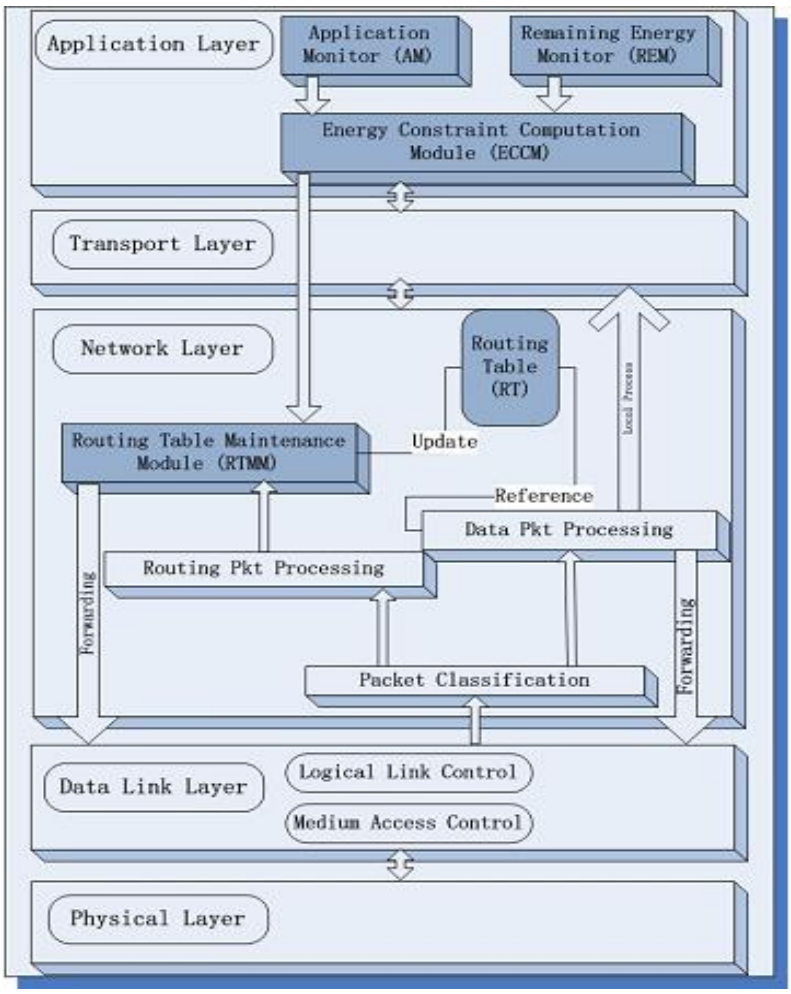

Figura 15. Arquitetura em camadas proposto pelo AWERA. Figura retirada de [15].

3.4 ANÁLISE DOS TRABALHOS APRESENTADOS

Dentre os trabalhos analisados, percebe-se que parte deles propõe métricas teóricas, independentes do AODV, de roteamento baseadas em informações energéticas, como quantidade remanescente de bateria, dreno de bateria e quantidade de saltos. Outros trabalhos fazem implementações do AODV para Android sem preocupação com o consumo de energia do protocolo, tendo como objetivo apenas o teste das características originais do protocolo. Por fim, alguns trabalhos fazem adaptações no AODV para economia energética e testam-nas. Entretanto, estes testes são feitos apenas em ambientes simulados.

Neste trabalho propomos adaptações no protocolo AODV para que este tenha maior eficiência energética e realizamos experimentos em dispositivos reais dotados de sistema operacional Android. A preocupação com uma implementação de forma eficiente em termos energéticos se dá devido a quantidade limitada de capacidade de bateria dos *smartphones*.

4 PROPOSTA E METODOLOGIA

Este projeto baseia-se em uma adaptação da implementação de AODV para Android desenvolvida por Jradi e Reedtz [11]. Em seu trabalho, eles desenvolveram uma aplicação que faz o papel de um protocolo da camada de roteamento para redes *ad hoc* dentro do sistema operacional Android. A biblioteca desenvolvida é capaz de criar uma rede *ad hoc* em dispositivos Android e repassar dados nesta rede utilizando o protocolo AODV. Em seus testes eles mostram que a biblioteca funciona e é suportada nos dispositivos HTC Hero e Google Nexus One. Para realizar os testes, eles desenvolveram uma aplicação simples de troca de mensagens de texto.

O trabalho de Jradi e Reedtz é um marco importante para o desenvolvimento de diversas aplicações que trabalharão sobre redes *ad hoc*. A partir dele, podem ser desenvolvidas aplicações de compartilhamento de arquivos, por exemplo, entre participantes de uma rede *ad hoc*. Entretanto, apesar da popularização dos *smartphones* Android, uma limitação destes dispositivos é a sua capacidade de bateria. É comum os usuários destes aparelhos se queixarem pela drenagem rápida de energia que o dispositivo consome. Isso se deve à alta quantidade de aplicações que são executadas simultaneamente.

Dessa forma, a implementação de redes *ad hoc* nestes dispositivos pode não se tornar popular principalmente por este fator. Em uma rede *ad hoc* o protocolo de roteamento será executado mesmo que o usuário esteja apenas repassando informações de terceiros, consumindo sua bateria sem que ele esteja ativo na rede. Assim, os usuários não vão querer deixar seus dispositivos conectados enquanto não mandam ou recebem informação, o que não é bom para a operação da rede. Esse cenário torna a rede inviável.

Tendo em vista este problema especificamente, este trabalho propõe mudanças na biblioteca desenvolvida em [11] e, consequentemente, no protocolo AODV para que parâmetros de consumo de energia sejam levados em consideração no processo de roteamento. Estas modificações têm por objetivo aumentar o tempo de vida da rede e reduzir o consumo de bateria dos dispositivos sobrecarregados ou com baixos níveis de bateria, evitando criar novas rotas através deles.

Basicamente, isso é feito associando como custo de cada salto algum valor resultado de uma função inversa da quantidade remanescente de bateria do nó destino deste salto. No protocolo original, a métrica é baseada na quantidade de saltos, então cada salto sempre possui custo unitário. A escolha dessa função e como a escolha da rota é feita dados estes valores são objetos deste trabalho.

4.1 PROPOSTAS DE MUDANÇAS

A ideia principal do nosso trabalho é que os nós armazenem a informação da quantidade de bateria remanescente da maior parte dos outros nós da rede de forma mais atualizada possível. Essa informação será armazenada em um arquivo de texto no próprio dispositivo, que denominamos de NEDB (do inglês *Network energy database*). Essas informações serão repassadas entres os nós através das próprias mensagens de controle do AODV (RREP, HELLO, etc).

Para que esta informação seja adicionada aos cabeçalhos dos pacotes de controle do protocolo, adicionamos, em cada um deles, um campo multidimensional que armazena o endereço IP, a quantidade remanescente de bateria e o instante de tempo daquela informação. Para que esta informação de tempo esteja correta, é necessário que os dispositivos sejam previamente configurados com a data e hora previamente. Também pode-se criar um mecanismo de sincronismo de relógio distribuído para se fazer esta sincronização. Antes de enviar qualquer pacote, um nó adicionará estas informações em seu cabeçalho. O nó que receber este pacote, deve inicialmente ler estas informações, comparar com as presentes no seu NEDB e atualizá-lo caso as informações recebidas sejam mais recentes. Caso seja um pacote que o nó deverá repassar, ele adicionará ao final do pacote as suas próprias informações de energia e então irá passá-lo adiante.

Por exemplo, em um RREQ, que é repassado na forma de *broadcast* através da rede, os nós intermediários que recebem-no adicionarão suas próprias informações de energia ao final do cabeçalho antes de repassá-lo. Já no pacote de controle HELLO, que também é uma mensagem de *broadcast*,

porém com TTL (do inglês *Time to* live) igual a um, os nós somente farão o processo de atualizar as suas NEDBs, caso necessário.

Essas múltiplas informações de energia que trafegam através dos pacotes, serão armazenadas na forma de um Array de objetos que será serializado antes de ser enviado. A serialização serve como um mecanismo para transformar estes objetos em uma String para armazenamento ou transmissão, de modo que tal String possa depois ser transformada novamente em um objeto.

Com estas informações trafegando na rede, diversas decisões podem ser tomadas por um nó no momento de escolher a rota mais apropriada em termos de economia de energia e aumento do tempo de vida da rede. No caso, por exemplo, de dois RREPs consecutivos chegarem com o mesmo número de sequência, ou seja, são rotas equivalentes, o nó da origem pode escolher a rota a ser armazenada baseado nas medidas de energia de cada uma das rotas. A forma como esta escolha é feita pode ser baseada em qualquer um dos métodos descritos em [9] e [13]. Em nosso trabalho, utilizamos o MMBCR, pois dentre as métricas estudadas, esta apresenta uma complexidade intermediária, tornando-o simples de implementar, porém apresentando boa eficiência.

Para a escolha da melhor rota, o MMBCR calcula uma função inversa da quantidade remanescente de bateria do nó destino e atribui este valor como o custo daquele salto. Dessa forma, quanto menos bateria o nó destino possuir, maior será o custo de se traçar uma rota por aquele nó. Quando tem-se todos os nós presentes numa possível rota, o MMBCR atribui como custo da rota o valor do salto com maior custo, ou seja, o salto mais crítico em termos de quantidade restante de energia. Dessa forma, comparam-se as rotas entre si e escolhe-se aquela com menor custo energético. Em suma, o MMBCR vai escolher dentre as rotas aquela em que o nó mais crítico em níveis de energia, ou seja, aquele com menos bateria remanescente, é o com mais bateria quando comparado ao nó com menos bateria de cada uma das outras rotas. Assim, ele evita que os nós em piores condições participem de novas rotas.

Com isso, quando ocorre a chegada de dois ou mais RREPs com mesmo número de sequência, a escolha que antes era feita tão somente baseada no número de saltos (quanto menos saltos, melhor a rota), agora será feita utilizando o MMBCR: a rota com menor custo energético será a escolhida. Este custo da rota será armazenado na tabela de roteamento para que possa ser comparado posteriormente com os RREPs que chegarem.

Por um lado, nossa implementação introduz mais informação do que necessário para esta funcionalidade. Para esta decisão, bastava que no encaminhamento do pacote RREP, fosse atualizado o custo da rota. Isso seria feito da seguinte maneira: o nó destino introduziria o seu custo através da função inversa da sua quantidade remanescente de bateria, adicionaria esta informação no pacote e transmitiria adiante. O nó seguinte, compararia a informação do pacote e caso seu custo fosse maior, atualizaria a informação antes de repassar o pacote; caso seu custo fosse menor ou igual, somente repassaria o pacote. Assim em diante até que o pacote chegasse à fonte (que originou o RREQ) que teria a informação do custo da rota.

Com esta abordagem, não seria necessária a implementação das NEDBs e a informação de energia individual de cada nó não seria transmitida em cada pacote de controle do AODV. Entretanto, introduzimos a maior parte de informação possível dentro da rede para que o trabalho pudesse ser facilmente adaptado e, por exemplo, outro parâmetro de energia que não o MMBCR fosse utilizado. Fizemos assim pensando em nosso código como uma fonte de estudo e pesquisa e que pudesse ser utilizado como plataforma para trabalhos futuros. Assim, apesar de abdicarmos de eficiência, por exemplo, um segundo trabalho na mesma área facilmente seria desenvolvido utilizando o código que propomos. Por outro lado, adaptar o código para que se tornasse mais eficiente nestes aspectos e focasse em uma determinada funcionalidade também seria simples.

Também adotamos algumas medidas complementares para aumentar o tempo de vida da rede e diminuir a sobrecarga em dispositivos com pouca bateria remanescente. Partimos do pressuposto de que a partir de um determinado limiar de bateria, a energia no dispositivo deve passar a ser economizada para que o seu usuário posso utilizar seu aparelho para outras funções caso ele necessite.

Criamos dois limiares em que o nosso código tomará diferentes medidas para economizar energia. Primeiramente, num primeiro limiar que inicialmente determinamos como 15% de bateria restante, o nó passará a evitar completamente que rotas passem por ele. Isso foi pensado pois o tráfego

de informação de terceiros não é interessante para nós que estão com um nível já crítico de bateria. Para que isso ocorresse, ao detectar que sua bateria chegou nesse nível, o nó força o envio de um RERR para todos os nós que o antecedem em alguma rota. Com isso, os nós receberão esta mensagem e, segundo o procedimento do protocolo, identificarão essa rota como uma rota quebrada e procurarão outros caminhos.

Ao mesmo tempo, no mesmo limiar, o nó deve parar de enviar mensagens HELLO. Assim os nós que dependem dele identificarão que ele não está mais disponível e as rotas que passam por ele serão quebradas. O resultado dessas duas ações é o mesmo. Entretanto o primeiro ocorre imediatamente assim que o próprio nó descobre que atingiu o limiar. Já o segundo levará alguns instantes até que os seus vizinhos percebam que ele deixou de existir, entretanto possui um efeito mais duradouro e confiável.

Para evitar a formação de novas rotas após essas medidas, o nó passará a não retransmitir pacotes RREQ que passarem por ele os quais ele não é o destino. No caso dele ser o destino, ele responderá o RREP normalmente. Dessa forma, nenhuma nova rota será feita através dele, porém ele ainda poderá ser fonte ou destino de alguma transmissão.

No segundo limiar, o nó será forçado a sair da rede para economizar bateria. Dessa forma, ele não poderá mais exercer nem mesmo o papel de fonte ou destino no tráfego de dados. Inicialmente este valor foi determinado em 5% para que a partir de então seu dispositivo focasse apenas em funções essenciais do sistema e fosse utilizado apenas para emergência. Tanto este quanto o primeiro limiar poderiam estar disponíveis como parâmetro de configuração ao usuário, que poderia determinar a partir de quando quer economizar energia. Entretanto este funcionalidade não fez parte do escopo do nosso trabalho.

Por fim, uma melhoria adicional do nosso trabalho em relação a [11] foi que o nosso aplicativo desenvolvido faz a configuração automática da rede. Em [11] a rede *ad hoc* é inicializada manualmente e os IPs dos dispositivos são configurados estaticamente dentro do código. Em nosso trabalho, o próprio aplicativo faz a configuração do dispositivo para que ele rode em modo *ad hoc* e o IP é configurado ainda estaticamente, porém em um arquivo de configuração dentro do dispositivo. Assim não é necessário nenhum conhecimento em redes ou programação para executar estas tarefas. No momento em que o aplicativo é iniciado, o dispositivo já faz parte da rede.

4.2 METODOLOGIA E DESENVOLVIMENTO

Para iniciar o desenvolvimento de uma aplicação no Android, primeiramente é necessário baixar a SDK (do inglês *Software Development Kit*) para esta plataforma. Este SDK é baixado juntamente com a IDE (do inglês *Integrated Development Enviroment*) Eclipse adaptada. Estas ferramentas podem ser baixadas no próprio site para desenvolvedores do Android em [16]. Além das ferramentas já mencionadas, alguns utilitários adicionais são encontrados no mesmo link para download. Como por exemplo um emulador de máquina virtual Android para que as aplicações possam ser testadas. Esta máquina virtual é conhecida como AVD (do inglês *Android Virtual Device*).

Após extrair a pasta do download efetuado em [16], deve-se abrir a IDE Eclipse para iniciar o desenvolvimento. A programação para Android segue os mesmos procedimentos da programação Java. Assim é necessário criar um projeto, que conterá pacotes, classes, etc. Para iniciar o desenvolvimento do nosso aplicativo, criamos um projeto Android com o código-fonte do projeto que serviu de base para a nossa aplicação [11].

Fizemos diversos testes para verificar a conformidade da aplicação desenvolvida em [11] antes de iniciarmos a modificar os nossos códigos. Uma ferramenta importante nesse estágio foi o ADB (do inglês *Android Debug Bridge*). O ADB é encontrado juntamente com as aplicações baixadas em [16] dentro do diretório /platform-tools. Com essa ferramenta, podemos acessar enviar comandos a um dispositivo físico conectado ao computador através de cabo USB (do inglês *Universal Serial Bus*). Para que ele funcione, é necessário acessar as configurações do celular e ativar a opção “Depuração de USB” em “ferramentas do desenvolvedor” conforme pode ser visto na Figura 16.

O ADB funciona através de comandos enviados via terminal e está presente tanto no sistema operacional Linux quanto no Windows. Com o ADB é possível enviar diversos comandos para o

dispositivo. Pode-se, por exemplo, instalar e desinstalar aplicações, enviar e extrair arquivos do celular e enviar comandos diretamente a aplicações que estão em execução no dispositivo. Também é possível acessar o terminal do dispositivo, que possui comandos semelhantes ao terminal Unix, pois o Android é baseado no Kernel do Linux versão 2.6.

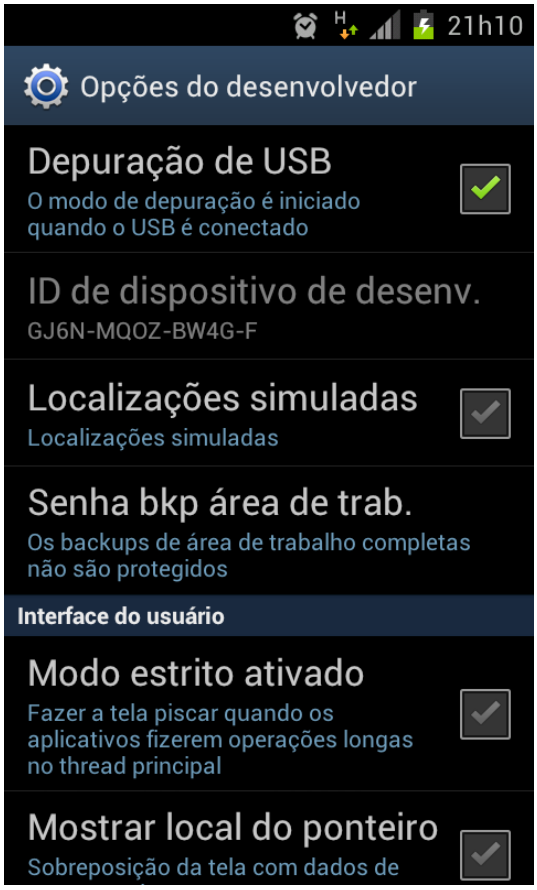

Figura 16. Ativar depuração de USB nas configurações de dispositivo Android.

Através do comando adb shell o desenvolvedor terá a sua disposição diversos comandos bem conhecidos para controlar o sistema operacional, tais como, chmod, mkdir, dentre outros. Assim, é possível conhecer toda a estrutura de diretórios e controlar boa parte das funções do sistema operacional.

Por outro lado, diversas operações não são permitidas por padrão para o usuário padrão do ADB. Para possuir controle total do sistema e gerenciar as permissões, é preciso obter acesso de super usuário do sistema operacional, equivalente ao root do Linux. Em geral, os celulares não vêm com esta função. Para cada dispositivo, deve ser seguido um procedimento específico para obter este privilégio.

Para obter acesso de super usuário no dispositivo, Samsung Galaxy Y (GT-5360), por exemplo, basta seguir o tutorial encontrado em [17]. Na maioria dos celulares da Samsung da linha Galaxy o procedimento é bem semelhante. No geral, é bem simples encontrar tutoriais na Internet ensinando o procedimento para cada modelo de aparelho. Apesar de ser um procedimento simples, o processo envolve riscos e só deve ser feito quando se tem certeza do que está fazendo. Os tutoriais mais sérios farão este alerta e é importante confiar na fonte do tutorial que se está seguindo.

Após obter acesso de super usuário no dispositivo, basta digitar o comando su no terminal do ADB. A partir de então todas as funções do aparelho estão habilitadas e todos os comandos tornam-se disponíveis, incluindo aqueles que alteram a permissão dos sistema de arquivos. No nosso experimento, parte das funções que executamos requerem o acesso de super usuário no sistema.

Antes de testar a aplicação desenvolvida em [11] precisamos colocar a interface Wi-Fi do dispositivo no modo *ad hoc*. Isso é feito com o auxílio de uma aplicação que requer o acesso de super

usuário no aparelho. O Wi-Fi Tether [18] é um aplicativo para Android que permite gerenciar a interface Wi-Fi dos aparelhos com o objetivo de interconectar celulares e PCs e compartilhar a conexão de dados do celular. Dessa forma, ele possui, dentre outras, a funcionalidade de colocar o celular no modo *ad hoc*.

O download da versão utilizada do Wi-Fi Tether pode ser feito em [19] e após instalado o procedimento para colocar o aparelho no modo *ad hoc* é simples. Basta acessar as configurações do aplicativos e definir na opção "*Change Setup Method*" o item "WEXT (*ad hoc*)". Nas configuração há outras diversas opções que podem ser personalizadas a critério do usuário, como o canal a ser utilizado, a potência a ser transmitida e o SSID da rede. De volta à tela inicial da aplicação (Figura 17) basta clicar no centro da tela e aguardar o procedimento ser feito. Uma mensagem será apresentada na tela informando o status da operação. Os detalhes do resultado podem ser vistos no *log* dentro da própria aplicação.

Figura 17. Tela inicial do aplicativo Wi-Fi Tether [18] onde inicializa-se a interface sem fio no modo *ad hoc*.

É importante ressaltar que apesar da aplicação informar que suporta as versões mais recentes do Android (JB e ICS), em nossos testes só conseguimos colocar dispositivos com a versão Gingerbread – 2.3 - do Android para operar no modo *ad hoc*. Por consequência, nossa aplicação só foi testada em dispositivo com esta versão do sistema operacional.

Com o Wi-Fi Tether ativado, pudemos executar a aplicação desenvolvida em [11]. A partir desse ponto demos início ao desenvolvimento das funcionalidades propostas em nosso trabalho descritas neste capítulo.

Para instalar o arquivo .apk baixado em [19], basta utilizar o comando `install` do ADB.

4.3 IMPLEMENTAÇÃO DA PROPOSTA

4.3.1 INICIALIZAÇÃO DA REDE *AD HOC*

A primeira funcionalidade implementada foi a que coloca o dispositivo no modo *ad hoc* automaticamente ao iniciar o aplicativo atribuindo um endereço IP fixo definido em arquivo de

configuração. Para isso, estudamos como o aplicativo Wi-Fi Tether executa esta ação e identificamos que isso é feito utilizando o comando tether.

Criamos o arquivo de configuração chamado adhoc.ini e colocamos o mesmo na raiz do sistema de arquivos. Isto foi feito através do comando push do ADB. Neste arquivo atualmente temos apenas um parâmetro que é o endereço IP que será atribuído àquele nó, chamado de ipv4_address. Neste parâmetro deve ser atribuído um número entre 2 e 254 para ser o identificador do dispositivo. O IP final atribuído será o IP da rede (192.168.2.0) onde o último número será substituído por este identificador. Novos parâmetros podem ser adicionados no futuro para inicializar ou gerenciar a rede.

Ao iniciar a execução do aplicativo, o programa lê o valor atribuído neste arquivo e atribui o valor do endereço IP enviando comandos de terminal ao sistema operacional. Esta operação é feita através do método runCommand(cmd) da classe Activity, onde cmd é um comando comum de terminal que deverá ser executado. Para atribuir o endereço IP, rodamos o seguinte comando:

su -c "ifconfig wlan0 *ip*",

onde *ip* é o valor lido no arquivo de configuração.

Com a interface Wi-Fi do dispositivo ligada, deve-se configurá-la para utilizar o modo *ad hoc* através do seguinte comando (para que este comando esteja disponível, é importante que o Wi-Fi Tether tenha sido executado manualmente ao menos uma vez):

su - c "/data/data/com.googlecode.android.wifi.tether/bin/iwconfig wlan0 mode ad hoc".

Para forçar a inicialização da interface Wi-Fi, basta utilizar o comando:

su -c "svc wifi enable".

E, analogamente, para desativar a interface Wi-Fi:

su -c "svc wifi disable".

Para iniciar a rede no modo *ad hoc* utilizamos o comando:

su - c "/data/data/com.googlecode.android.wifi.tether/bin/tether start".

Analogamente, para sair do modo *ad hoc*, basta executar o comando:

su - c "/data/data/com.googlecode.android.wifi.tether/bin/tether stop".

É importante notar que em alguns dispositivos a interface wlan0 é chamada de eth0. Por isso, nosso aplicativo tenta acessar ambas as interfaces todas as vezes que isso é necessário.

Com esta sequência de comandos o dispositivo fará parte da rede *ad hoc* com o IP especificado no arquivo de configuração e pronto para executar o protocolo de roteamento AODV. Entretanto, notamos que em alguns dispositivos esta sequência de comandos não produz efeito. Assim, o modo *ad hoc* deve ser inicializado manualmente utilizando o aplicativo Wi-Fi Tether nestes dispositivos.

4.3.2 INFORMAÇÕES DE ENERGIA

Para gerenciar as informações relacionadas a quantidade de bateria, criamos uma classe chamada EnergyAware dentro do pacote aodv.ea. Nesta classe o primeiro método a ser destacado é aquele que retorna a quantidade remanescente de bateria do próprio nó em porcentagem, onde 0 significa sem bateria e 100 que a bateria está completa. Chamamos este método de getOwnBatteryLevel().

Esta informação é obtida com o auxílio da classe BatteryManager nativa do próprio Android que além de nos trazer esta informação, também pode seu utilizada para indicar se o dispositivo está sendo carregado, distinguindo se pela rede elétrica ou através de cabo USB. Esta informação também pode ser utilizada em trabalhos futuros para melhorar o protocolo. Dispositivos com baixo nível de bateria, mas que estão conectados à rede elétrica possuem vida útil mais longa. Também pode-se afirmar que rotas onde todos os dispositivos estão sendo carregados são melhores em termos de tempo de vida. Estas informações utilizadas juntamente com o MDR [13] podem melhorar a eficiência da rede e deve ser fruto de trabalhos futuros.

Na mesma classe EnergyAware, gerenciamos o acesso e o armazenamento de informações na NEDB através dos métodos getBatteryLevel(ip), isMoreRecent(ip, timestamp) e store(ip, batteryLevel, timestamp), que respectivamente, retornam a quantidade de bateria de um dado nó a partir de seu IP, checa se um valor recém recebido de bateria de dado nó é mais recente do que aquele já armazenado em sua NEDB e armazena os valores recebidos na NEDB (atualizando caso ele já exista).

A NEDB é um arquivo de texto puro armazenado na raiz do sistema de arquivos organizado de forma que cada linha corresponde a uma entrada de bateria de um nó. Numa mesma linha está armazenado o IP do dispositivo em questão, a quantidade remanescente de bateria e o instante de tempo do momento em que este valor de bateria foi obtido. Estes valores estão separados pelo caractere ponto-e-vírgula.

Para fazer o tráfego de informação de energia através da rede, adicionamos uma classe chamada BatteryInformation, que possui os atributos battery_level, ipv4_address e timestamp. Essa classe é responsável por manipular as informações de bateria de cada nó da rede. Adicionalmente, incluímos em cada pacote de controle do AODV o atributo energy_info do tipo String, o qual contém o valor serializado utilizando JSON (do inglês *JavaScript Object Notation*) do Array de objetos de BatteryInformation que está trafegando.

Ao fazer a construção do pacote, o nó cria uma instância de BatteryInformation com seus próprios parâmetros de energia. Ele então cria um Array contendo este objeto. Este Array é convertido para String utilizando JSON de forma que este resultado é inserido no pacote dentro do campo energy_info. Quando o próximo nó receber o pacote, ele fará a conversão de String para Array utilizando o decodificador JSON e então salvará as informações recebidas na NEDB caso seja necessário. Se este for um pacote que deva ser repassado (como o RREQ, por exemplo), o nó adicionará um novo elemento ao final do Array contendo as suas próprias informações de bateria. Converterá novamente o Array para String, atualizará o valor de energy_info e então passará o pacote adiante.

Este procedimento é seguido para todos os pacotes de controle do AODV. Uma limitação deste processo para a formação de rota através da mensagem RREQ, é que um nó intermediário que conhece uma rota válida para o destino, retornará a mensagem RREP com informações de energia a partir de si próprio. As informações de energia dos nós posteriores não serão informadas ao nó da origem. Dessa forma, sua decisão de escolha de rota será baseada somente em uma informação parcial da rota. Entretanto, devido às outras medidas tomadas pela nossa proposta, assumimos que este nó intermediário escolheu aquela rota, pois em algum momento ela era a melhor disponível em termos de quantidade de energia. Sabemos também que se aquela ainda é uma rota ativa, todos os nós presentes nela, possuem um mínimo de energia necessária para permanecerem na rede por mais algum tempo.

Para armazenar a informação de custo de rota em termos de energia, modificamos a classe RouteEntry que gerencia as informações na tabela de roteamento do AODV. Adicionamos o atributo cost à esta classe que armazenará o custo de cada uma das rotas. Lembramos que o custo é dado com o uso do MMBCR. Ao receber um RREP, o custo é calculado utilizando este método e então armazenado na tabela de roteamento. Também modificamos todas as chamadas ao método isIncomingRouteInfoBetter() que recebia a informação hopCount e agora recebe esta nova informação de custo em termos de energia.

Todas estas ações descritas são tomadas dentro da classe Receiver, que é a responsável por tratar os pacotes recebidos e montar os pacotes que deverão ser repassados. Os limiares descritos foram criados dentro da classe Constants. Criamos os atributos PRIMEIRO_LIMIAR e SEGUNDO_LIMIAR e os valores deles são inteiros que representam a porcentagem de bateria remanescente em que deseja-se executar as ações pré-determinadas.

Fazemos o envio forçado da mensagem RERR para todos os nós precursores de um nó que atinja o primeiro limiar de 15% de bateria. Isso foi feito através de um Thread que checa se este limiar foi atingido e, quando detectado, lê-se a tabela de roteamento, identifica-se todos os precursores e então envia um RERR para todos eles como se os nós destino tivessem desaparecidos. Isso força estes precursores a apagar o registro de sua tabela de roteamento e então procurar novas rotas caso seja necessário.

Para que as mensagens HELLO parassem de ser enviadas ao atingir o primeiro limiar de 15%, alteramos dinamicamente o valor do parâmetro keepBroadcasting para false dentro do seu loop no arquivo Sender.java dentro da classe NeighbourBroadcaster. Assim que o limiar for atingido entramos num loop que fará a checagem de tempos em tempos se a quantidade mínima de bateria foi restabelecida. Caso isso ocorra, reiniciamos o processo de broadcast de mensagens HELLO alterando novamente o parâmetro keepBroadcasting para true.

Ainda dentro do mesmo loop, verificamos se o segundo limiar é atingido, e caso isso ocorra, finalizamos a execução do programa através do comando:

```
android.os.Process.killProcess(android.os.Process.myPid());
```

Novamente na classe Receiver quando uma mensagem RREQ é recebida, uma condição foi adicionada para que o nó primeiramente cheque seu status de bateria e se ele estiver abaixo do primeiro limiar e ele não for o destino final do RREQ, ele interrompe a execução e consequentemente não repassa o pacote. Isso é feito para evitar que novas rotas sejam formadas através dele.

Fora feitas estas mudanças no código de [11] para o teste da nossa proposta. Todo a descrição do experimento realizado e os resultados obtidos que mostram que estas implementações de fato funcionaram estão dispostas no capítulo seguinte.

Para que nossa proposta possa ser reproduzida e melhorada, todo o código-fonte do nosso projeto está disponível na plataforma de versionamento Github [20]. Também está disponível o código utilizado para efetuar o nosso experimento descrito no Capítulo 5.

5 EXPERIMENTOS E RESULTADOS

5.1 CONFIGURAÇÃO

Para efetuar os experimentos que demonstram o funcionamento do protocolo AODV com as adaptações propostas dispomos de cinco aparelhos celulares com sistema operacional Android. A Tabela 2 mostra o ID associado a cada nó, os modelos utilizados, a versão do Android e o endereço IP atribuído a cada um deles dentro da rede.

Tabela 2. Celulares utilizados nos experimentos.

ID	Celular	Número do modelo	IP	Versão do Android
2	Samsung Galaxy Y	GT- S5369	192.168.2.2	2.3.6
3	Samsung Galaxy Y	GT-S6350B	192.168.2.3	2.3.5
4	Samsung Galaxy Y	GT-S6350B	192.168.2.4	2.3.6
5	Samsung Galaxy Y	GT-S6350B	192.168.2.5	2.3.6
6	Samsung Galaxy Ace Duos	GT-S6802B	192.168.2.6	2.3.6

Estes são celulares bastante populares por seu baixo custo e executam a versão do Android mais popular até o momento, o Gingerbread. A diferença no número do modelo que ocorre entre alguns aparelhos com o mesmo nome diz respeito à cor ou origem (fabricação em país diferente), enquanto que as funcionalidades e recursos são praticamente os mesmos.

Para preparar cada um destes celulares antes da execução dos testes, executamos o procedimento para obtenção do acesso como super usuário. Como mencionado anteriormente, para cada modelo há um método diferente para acesso no modo super usuário. Entretanto, como todos os aparelhos utilizados foram do mesmo fabricante e de modelos muito semelhantes (Galaxy), houve poucas diferenças neste procedimento entre cada um deles.

O passo seguinte foi colocar na raiz do sistema de arquivos do celular os arquivos NEDB.txt e adhoc.ini. Neste último, alteramos o parâmetro ipv4_address com o identificador de acordo com cada aparelho. Em seguida, instalamos o aplicativo Wi-Fi Tether [18] em todos os celulares e fizemos sua configuração inicial.

Para auxiliar nos testes, foi feita também a instalação de outro aplicativo: o Shark For Root [25]. Este aplicativo utiliza o comando tcpdump para capturar todos pacotes enviados e recebidos por cada celular. Após iniciarmos nosso programa, iniciávamos este aplicativo, o qual salva na raiz do sistema de arquivos um arquivo com extensão .pcap. O arquivo com extensão .pcap é utilizado posteriormente para fazer a análise do tráfego gerado com a ferramenta Wireshark, que interpreta este formato de arquivos e mostra todos os pacotes que trafegaram na interface de rede durante o período monitorado. É importante que no campo parâmetro, no aplicativo Shark for Root, seja inserido o valor –vv –s 0 para que pacotes de qualquer tamanho sejam capturados. Caso contrário, os pacotes com campo de dados maiores que 54 bytes são truncados.

Para os nossos testes, criamos também uma aplicação que faz uso do protocolo UDP para envio de tráfego através da rede *ad hoc* a partir de um dos nós da rede e com destino para outro nó. Programamos esta aplicação para enviar tráfego a uma taxa de bits constante a 440Kbps durante um determinado tempo configurável. Cada pacote gerado possui 8,8Kb compostos por uma longa cadeia de caracteres e foram gerados a cada 20ms.

5.2 TESTES E RESULTADOS

5.2.1 TOPOLOGIA LINEAR

A primeira topologia da rede utilizada para teste foi uma topologia linear, conforme a Figura 18, para testar a caracterização do funcionamento da rede. Esta topologia foi utilizada para verificar se o protocolo AODV funcionava corretamente, verificar o consumo de bateria e analisar alguns parâmetros de desempenho. Para descrição dos experimentos, associamos os números 2 à fonte (identificada pela letra S) e o número 6 ao destino (identificado pela letra D). O nó S (2) enviava pacotes de dados para o nó D (6). Para garantir a topologia linear, foi tomado o cuidado para que cada nó só conseguisse detectar a presença dos nós antecessor e sucessor, respectivamente.

Figura 18. Representação da topologia linear utilizada nos experimentos.

5.2.1.1 Caracterização da distância máxima

A rede de dispositivos celulares foi implantada em uma área externa. Após vários testes com relação à máxima distância permitida para comunicação dos nós, percebeu-se que, para estes dispositivos, os nós conseguem mantes comunicação se localizados a uma distância máxima de 12,4m. Assim, os cinco dispositivos da Figura 18 foram dispostos em uma área externa (em uma rua) com o cuidado de respeitar esta distância máxima. A Tabela 3 mostra as distâncias entre cada um dos nós na disposição linear.

Tabela 3. Distâncias fixadas entre os nós da topologia linear.

Do/para	S (2)	3	4	5	D (6)
S (2)	-	8m	16m	24m	32m
3	8m	-	8m	16m	24m
4	16m	8m	-	8m	16m
5	24m	16m	8m	-	8m
D (6)	32m	24m	16m	8m	-

5.2.1.2 Perda de conectividade devido a inatividade

Identificamos um problema durante os testes iniciais: depois de 30 segundos, em geral, a rede parava de funcionar após os dispositivos apagarem suas telas devido à inatividade (interatividade do usuário). Nosso diagnóstico é que, ao apagar a tela, o celular entra em modo de economia de energia e, consequentemente, reduz consideravelmente a sua potência de transmissão/sensibilidade de recepção ou mesmo desliga a interface Wi-Fi. Para evitar a inoperância da rede devido a este problema, configuramos os celulares para apagar a tela somente após 30 minutos de inatividade. Desta forma, os dispositivos puderam permanecer ativos por todo o tempo de teste.

O problema desta medida é que, com o visor ligado, o celular consome muita bateria, o que reduz o tempo de vida da rede. Por outro lado, também não é uma medida interessante na prática, pois os dispositivos têm que estar sempre ativos para que façam parte da rede. Ao mesmo tempo, os usuários com celulares no modo de economia de energia não participam da rede e, consequentemente, diminuem

a conectividade da rede. Entretanto, essa foi a única solução encontrada até o momento, e foi suficiente para realização dos testes. Para amenizar seu efeito, os celulares foram configurados para tela com brilho mínimo.

5.2.1.3 Consumo de bateria

5.2.1.3.1 Transferência de dados a partir do nó 2 com destino ao nó 6

Com este problema resolvido, colocamos a rede para rodar na topologia linear durante 20 minutos. Este teste tem como objetivo analisar o dreno de bateria durante este período. Os dispositivos não encontravam-se com bateria cheia no início dos testes. Cada um deles possuía um nível de bateria distinto. A quantidade de bateria de cada nó antes do início do teste pode ser visto na Tabela 4 na coluna Bateria inicial. Ao final dos 20 minutos de teste, a quantidade de bateria de cada nó foi registrada conforme mostra a coluna Bateria final da Tabela 4.

Com os valores de bateria inicial e final, calcula-se o dreno de bateria de cada nó no período do teste subtraindo a quantidade de bateria inicial da quantidade de bateria final. O dreno por hora pode ser obtido multiplicando o dreno do período por três, pois o teste foi realizado durante um terço de hora. Por fim, o tempo de vida estimado de cada um dos nós é dado a partir do dreno/hora encontrado. O valor do tempo estimado de vida de cada nó representa o tempo em que a bateria de um nó duraria desde o momento em que ele entra na rede, com 100% de bateria, até que sua bateria seja completamente esgotada. Esse valor foi obtido através de uma aproximação que considera que o gasto de bateria do dispositivo seja linear.

Tabela 4. Consumo de bateria na topologia linear com o nó 2 atuando como fonte e o nó 6 atuando como destino.

Nó	Bateria inicial	Bateria final	Dreno no período	Dreno/hora	Tempo de vida estimado (h)
S (2)	37%	33%	4%	12%	8,3
3	40%	31%	9%	27%	3,7
4	78%	69%	9%	27%	3,7
5	62%	51%	11%	33%	3,0
D (6)	92%	82%	10%	30%	3,3

A informação de bateria remanescente é disponibilizada pelo dispositivo apenas em números inteiros. Logo, a precisão da análise está limitada a esta restrição. De qualquer forma, percebemos que todos os nós intermediários obtiveram o mesmo dreno de bateria, praticamente. Há somente uma pequena diferença no nó 5, mas podemos atribuir isto à falta de precisão dos dados obtidos ou mesmo à diferença no tipo ou na capacidade de bateria destes aparelhos. Entretanto, consideramos que não foi uma diferença significativa e os resultados parecem coerentes.

Percebe-se que o nó destino obtém o mesmo consumo de energia se comparado com os nós intermediários. Enquanto durante o experimento ele consumiu 10% de bateria, os nós intermediários consumiram entre 9% e 11%. Por outro lado, verifica-se uma diferença considerável no consumo do nó fonte. Nota-se que o consumo deste nó foi menor do que a metade do consumo de todos os outros aparelhos, consumindo apenas 4% de bateria no período de testes.

5.2.1.3.2 Transferência de dados a partir do nó 6 com destino ao nó 2

Para checar se esta grande diferença de consumo de bateria é uma característica do dispositivo analisado, o sentido do tráfego foi invertido para mais um teste. No caso, a topologia da rede foi mantida, mas o nó 6 passou a ser o transmissor de dados, enquanto que o nó 2 passou a ser o destinatário dos pacotes. Deixamos a rede ativa por 10 minutos, metade do tempo do teste anterior, e analisamos os

resultados obtidos. Para compará-los, o dreno de bateria obtido neste teste foi dobrado para ter a mesma magnitude do teste de 20 minutos.

O resultados obtidos estão dispostos na Tabela 5. Novamente os nós intermediários tiveram consumo aproximadamente igual. Entretanto, o nó 6, atuando como fonte de dados, consumiu menos bateria do que todos os outros nós. Ainda, constatamos que o nó 2, que neste experimento fez papel de destino, teve consumo semelhante ao dos nós intermediários. Com este fato, chegamos à conclusão de que o procedimento de recebimento de grande volume de dados consome mais bateria do que o processo de envio.

Tabela 5. Consumo de bateria na topologia linear com o nó 6 atuando como fonte e o nó 2 atuando como destino.

Nó	Bateria inicial	Bateria final	Dreno no período	Dreno/hora	Tempo de vida estimado (h)
D (2)	30%	26%	4%	24%	4,2
3	25%	20%	5%	30%	3,3
4	68%	63%	5%	30%	3,3
5	49%	44%	5%	30%	3,3
S (6)	80%	78%	2%	12%	8,3

5.2.1.3.3 Teste da rede sem tráfego de dados

Para finalizar a análise da quantidade de bateria consumida, colocamos a rede ativa por mais 10 minutos na mesma topologia, porém sem o tráfego de dados. Dessa forma, as únicas mensagens trocadas foram pacotes de controle HELLO, enviados a cada 1 segundo por cada nó. Os dados obtidos neste teste podem ser vistos na Tabela 6.

Como resultado deste teste, verifica-se que, no período analisado, os nós consumiram uma quantidade desprezível de bateria. Considerando ainda a falta de precisão na obtenção da informação de energia, apenas número inteiros, pode-se afirmar que cada nó consumiu, no máximo, 1% de bateria neste teste. Assim percebe-se que daquele consumo de bateria dos dois primeiros testes, a parte considerável dele foi referente ao envio de grandes pacotes de dados.

Tabela 6. Consumo de bateria na topologia linear sem tráfego de dados.

Nó	Bateria inicial	Bateria final	Dreno no período
D (2)	38%	37%	1%
3	35%	35%	0%
4	62%	62%	0%
5	41%	40%	1%
S (6)	75%	75%	0%

Com tudo isso, podemos afirmar que a manutenção da rede através do AODV com suas mensagens de controle sem tráfego de dados, ou com baixo tráfego de dados, não consome muita bateria do celular. Inclusive, podemos afirmar que parte da bateria consumida em todos os testes anteriores se deve ao visor ligado e a outras atividades comuns do *smartphone*. Entretanto, quando a rede *ad hoc* está em operação, e sob uma alta taxa de geração de tráfego, tanto os dispositivos finais quanto os intermediários consomem uma quantidade significativa de bateria.

5.2.1.3.4 Análise dos resultados obtidos no consumo de bateria da topologia linear

Verifica-se que, através da análise da Tabela 4 e da Tabela 5, que pode-se atingir um tempo de vida da rede de, no mínimo, 3 horas a uma taxa constante de 440kbps. A esta taxa, o tráfego gerado no período é de 4,5Gb de dados. Este valor é bastante elevado se considerarmos o tráfego de mensagens de texto, imagens ou até mesmo vídeos de curta duração. Em uma rede pequena, entendemos que a transferência desta quantidade de dados em curto período de tempo (3 horas) não é comum para os padrões atuais de aplicações. Dessa forma, verificamos que, dado o padrão de navegação dos usuários atuais, a rede poderia alcançar um tempo de vida mais longo.

5.2.1.4 Análise do tráfego gerado na topologia linear

Após cada um dos testes, foram extraídos os arquivos .pcap gerados pelo aplicativo Shark for Root para se fazer a análise dos resultados obtidos. Para o teste de 20 minutos de operação, foram gerados 60 mil pacotes de dados a uma taxa de bits constante. Cada pacote continha 8,8mil bits e foram gerados na taxa de 50 pacotes por segundo, resultando em 440Kbps. Isso totaliza 528Mb de dados transferidos no período total de testes. Cada um dos pacotes possui ainda 520 bits de cabeçalho, incluindo aqueles introduzidos pelo próprio AODV. Com isso, foram ainda transferidos 31,2Mb de cabeçalho juntamente com os pacotes de dados. Tudo isso sem contar todas as mensagens de controle do AODV, como as mensagens HELLO, durante todo o período. Dessa forma, a rede suportou a entrega de pelo menos 466Kbps.

Estes 528Mb de dados gerados foram entregues em aproximadamente 20 minutos exatos. Desde o primeiro pacote de dados até o último, se passaram 1201,95s (quase 20 minutos e 2 segundos). Com estes dados, chegamos a uma vazão da rede igual a 439,286Kbps. Como o tráfego foi gerado a uma taxa de 440Kbps, abaixo da taxa mínima do 802.11, mesmo com os atrasos da camada MAC devido à contenção com nós vizinhos, atrasos de repasse e de propagação, para esta rede pequena, a taxa entregue está muito próxima da taxa gerada.

Como utilizamos UDP, alguns dos pacotes poderiam não ter sido entregues. Entretanto, na execução dos testes em que houve tráfego de dados, obtivemos taxa de entrega de 100%. Isso provavelmente aconteceu devido à ausência de interferência e a simplicidade da rede. Cada nó possuía, no máximo, dois vizinhos e a quantidade de saltos que um pacote de dados percorreu foi de apenas 4 saltos. Com isso, haviam poucos motivos para a rede falhar devido a erro de canal ou a ocorrerem erros na entrega de pacotes.

5.2.1.5 Informações de energia nas mensagens de controle do AODV

A fim de verificar a correta implementação das modificações nas mensagens do AODV sobre a informação de energia, analisamos os cabeçalhos dos pacotes de controle trocados pelo AODV modificado. A Figura 19 ilustra as informações visualizadas no Wireshark relativas a uma mensagem HELLO. Verificamos no pacote HELLO gerado pelo nó 6 e enviado através de *broadcast* na rede que a informação de energia introduzida por nosso trabalho estava presente dentro do pacote.

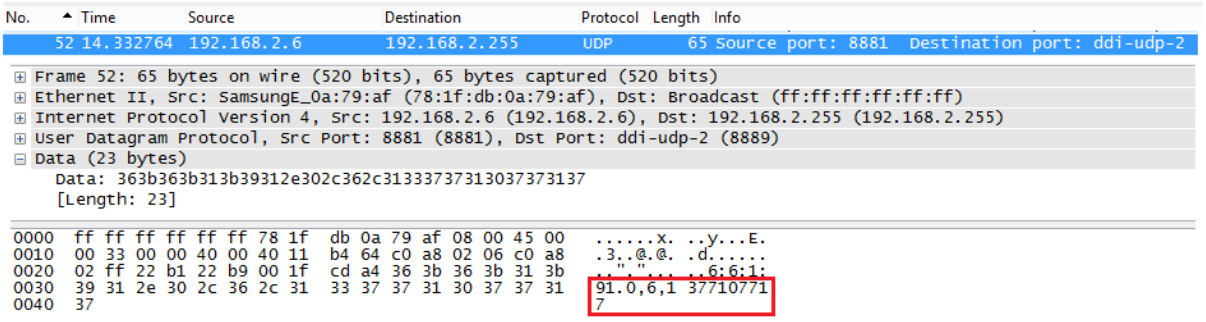

Figura 19. Informação de energia enviada no pacote HELLO pelo nó 6.

No exemplo da Figura 19, o detalhe em destaque informa que nó 6 estava com 91% de bateria no instante de tempo 1377107717. As informações foram inseridas separadas por vírgulas. A primeira

delas, o 91.0, representa a quantidade de bateria. Em seguida, o 6 representa o nó a qual aquela informação pertence. Por fim, o 1377107717 significa o instante de tempo em que a informação de energia foi coletada. Como previsto, os nós que recebem este pacote, verificam se esta informação é mais atualizada do que a que eles já possuem em sua NEDB para devida atualização. Na Figura 20 ilustramos o estado da NEDB de um dos nós da rede em um determinado momento.

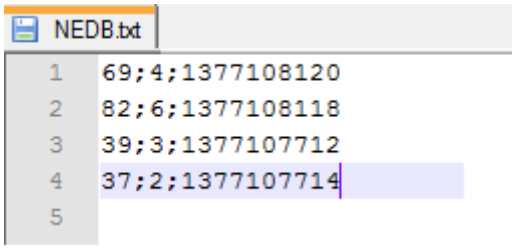

Figura 20. NEDB do nó 5 em um dado instante após a finalização da rede.

Essa figura mostra um exemplo de NEDB gerada pelo nosso aplicativo. Ela foi gerada pelo nosso aplicativo e extraída do nó 5. Na NEDB as informações são organizadas por linha e separadas por ponto-e-vírgula. Cada linha representa a informação de um nó da rede. Em uma linha o primeiro elemento representa a quantidade de bateria. A segunda informação é o nó ao qual esta informação pertence. Já o último elemento significa o instante de tempo em que a informação foi coletada. Pode-se perceber que o NEDB contém a informação de bateria de todos os outros nós da rede, pois os pacotes de controle RREQ e RREP passaram por todos eles. Entretanto, verifica-se que as informações mais atualizadas são as de seus vizinhos que enviavam as informações de bateria através das mensagens HELLO constantemente, enquanto o RREQ e o RREP foram transmitidos somente no início do experimento e o valor ficou obsoleto. Analogamente, todos os outros nós da rede possuíam NEDBs semelhantes.

Na Figura 21 mostramos que as tabelas de roteamento passam a possuir a informação de custo da rota além da informação de quantidade de saltos. Esta imagem foi retirada com o auxílio da ferramenta de depuração Logcat presente na SDK do Android. Com esta ferramenta, ao conectar um dispositivo através de cabo USB ao computador, podemos visualizar com detalhes as mensagens, erros e avisos gerados pelos aplicativos. O exemplo a seguir é extraído de um exemplo simples, em que a topologia contém apenas os nós S (2) e D (6). Em um dado instante, identificamos que o custo da rota do nó 6 para o nó 2 era de 1,25.

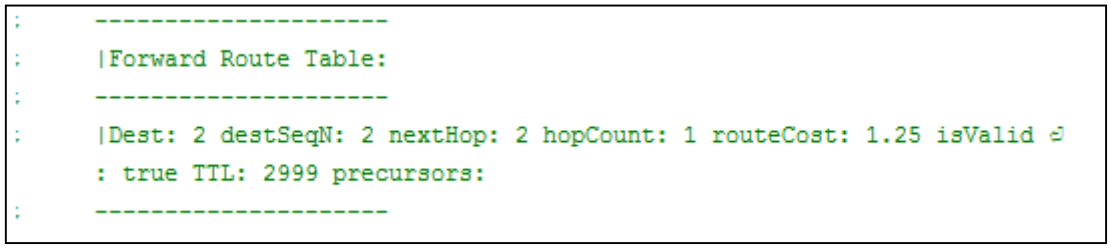

Figura 21. Tabela de roteamento a nova informação de custo da rota calculado com informações de energia.

Como usamos o MMBCR, e a função $100/BR_i$ para representar o custo de um salto, sendo BRi a porcentagem de bateria remanescente no nó i, esse custo de 1.25 significa que, na rota ilustrada, o nó com pior nível remanescente de bateria (e, neste caso, único nó) estava com 80% de bateria, o que foi constado ao consultar as configurações do aparelho. Também vemos que a quantidade de saltos é igual a 1, pois havia somente dois nós no momento em que esta imagem foi capturada. Nesta figura, também percebemos que esta ainda era uma rota válida por, no mínimo, 2999ms, podendo ser renovada caso o nó D (6) receba um HELLO do nó S (2).

5.2.2 TOPOLOGIA ANEL

Nesta seção, apresentamos a verificação da operação do AODV com relação à troca de informação de energia e consequente escolha de rota. Para isso, definimos uma segunda topologia, no formato de anel, conforme ilustrada na Figura 22. Novamente, o nó S (2) envia dados para o nó D (6) sobre UDP a uma taxa de 440Kbps.

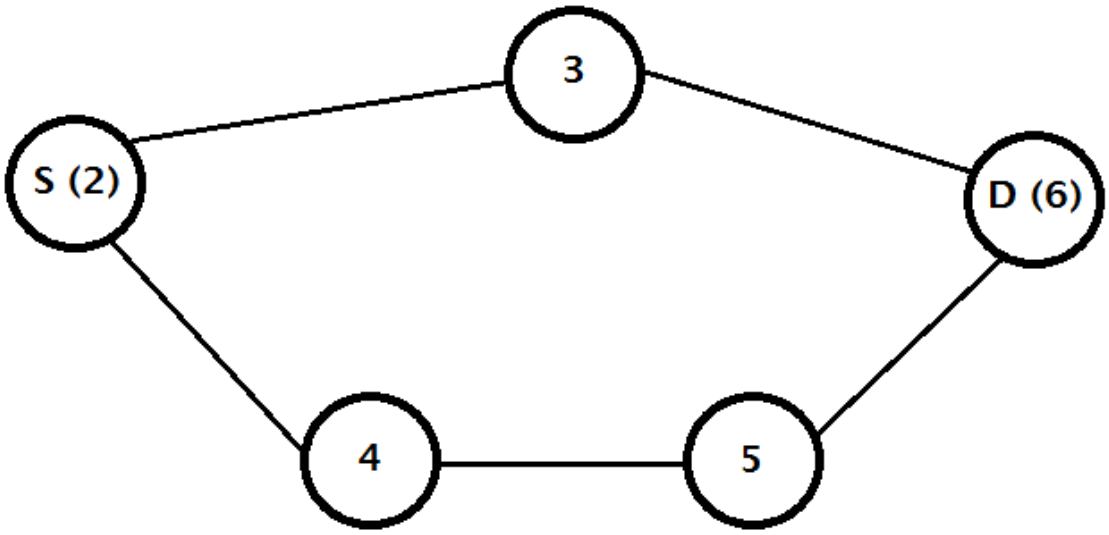

Figura 22. Ilustração da topologia anel utilizada nos experimentos.

Nesta topologia, determinamos as distâncias de modo que a fonte de dados S (2) pudesse alcançar o destino D (6) somente através de duas rotas válidas: através do nó 3 e através dos nós 4 e 5, respectivamente. Com isso, as distâncias lineares de cada um dos nós em relação aos outros pode ser vista na Tabela 7.

Tabela 7. Distâncias fixadas entre os nós da topologia anel.

	S (2)	3	4	5	D (6)
S (2)	-	8m	11m	13m	15m
3	8m	-	14m	14m	8m
4	11m	14m	-	5m	13m
5	13m	14m	5m	-	11m
D (6)	15m	8m	13m	11m	-

Dadas estas distâncias, cada nó alcança somente 2 outros nós. Com isso, somente as duas rotas poderiam ser formadas. Esta topologia foi montada em uma área externa e aberta. Os testes também duraram 20 minutos com a geração de tráfego a partir de S (2) com destino D (6). Dessa vez, quase todos os nós iniciaram o experimento com bateria completa com exceção do nó 3, que iniciou com bateria remanescente de 56%.

Esta configuração inicial de bateria permite mostrar que a modificação do AODV implementada considera, de fato, a função de bateria remanescente para a escolha da rota em detrimento da quantidade de saltos. Pode-se perceber da topologia escolhida que a rota através do nó 3 é melhor em relação à quantidade de saltos do que a rota através dos nós 4 e 5. Na versão do AODV tradicional, a rota escolhida seria aquela através do nó 3. Entretanto, dada a função inversa da quantidade de bateria remanescente

determinada ($100/BR_i$), a rota através do nó 3 possuía custo de aproximadamente 2 no início dos testes. Por outro lado, a rota através dos nós 4 e 5, possuía custo de aproximadamente 1, sendo este a melhor rota a ser escolhida.

Neste teste também fizemos um ajuste no primeiro limiar do nó 4. Este limiar determina que o nó, ao atingi-lo, deverá quebrar todas as rotas através dele e depois evitar a formação de novas rotas. Neste teste, desejamos mostrar também a implementação da funcionalidade através da qual são evitadas rotas que contêm nós cujos níveis estão abaixo de um limiar pré-estabelecido. Foi estabelecido um limiar de 95% para o nó 4. Ou seja, quando ele atingir esta quantidade remanescente de bateria, ele forçará a quebra das rotas as quais ele participa e deixará de enviar mensagens HELLO. Além disso, ele também passará a rejeitar RREQ para os quais ele não é destino final. Todo o consumo de bateria durante o teste pode ser visualizado na Tabela 8.

Tabela 8. Consumo de bateria na topologia anel.

Nó	Bateria inicial	Bateria final	Dreno no período	Dreno/hora	Tempo de vida estimado (h)
S (2)	100%	96%	4%	12%	8,3
3	56%	52%	4%	12%	8,3
4	100%	95%	5%	15%	6,7
5	100%	94%	6%	18%	5,6
D (6)	100%	90%	10%	30%	3,3

Nota-se que o consumo de bateria da fonte e do destino mantém-se praticamente o mesmo do caso da topologia linear. O nó fonte continua gastando menos bateria do que o nó destino. A semelhança ocorre porque o papel de ambos, nesta topologia, foi basicamente o mesmo da topologia anterior. Entretanto, verificamos diferenças nos nós intermediários. Isso ocorre, pois os nós 4 e 5 participaram do tráfego de dados na primeira metade do experimento apenas. Após quase 10 minutos de testes, o nó 4 atingiu o limiar de 95%. Com isso, ele enviou uma mensagem RERR ao nó 2 e parou de enviar mensagens HELLO, fazendo com que ocorresse a quebra da rota. Dessa forma, o nó 5 também deixou de participar da rota do fluxo de dados. Após este evento, por deixar de transmitir mensagens HELLO, o consumo de bateria do nó 4 foi praticamente nulo. Enquanto o nó 5, que ainda continuou a mandar mensagens HELLO até o fim do experimento, teve um consumo de bateria um pouco maior que o do nó 4.

Por outro lado, no período inicial do testes o nó 3 ficou ocioso. Durante aproximadamente metade do teste, o tráfego ocorreu todo através da rota que passa pelos nós 4 e 5. Então o nó 3, durante estes 10 minutos iniciais apenas enviava mensagens HELLO, o que consome pouca bateria se comparado ao alto fluxo de dados que estava sendo gerado. Quando a fonte S (2) percebeu que não havia mais rotas válidas até o destino D (6), fez a busca por outras rotas e então estabeleceu a nova rota através do nó 3. Somente neste período o consumo de bateria de 3 foi representativo.

Notamos que as implementações propostas por nosso trabalho efetivamente funcionaram nessa rede. A escolha inicial da rota se deu pelo caminho com melhores parâmetros de energia e não somente pela quantidade de saltos. O nó 4, ao atingir o limiar estabelecido, forçou a quebra da rota que passava através dele. Com isso, o nó S (2) então teve que buscar outra rota e então encontrou a rota através do nó 3, pois o nó 4 deixou de repassar pacotes RREQ. Representamos todos estes eventos através do diagrama de tempo da Figura 23 que é visto na perspectiva do nó S (2).

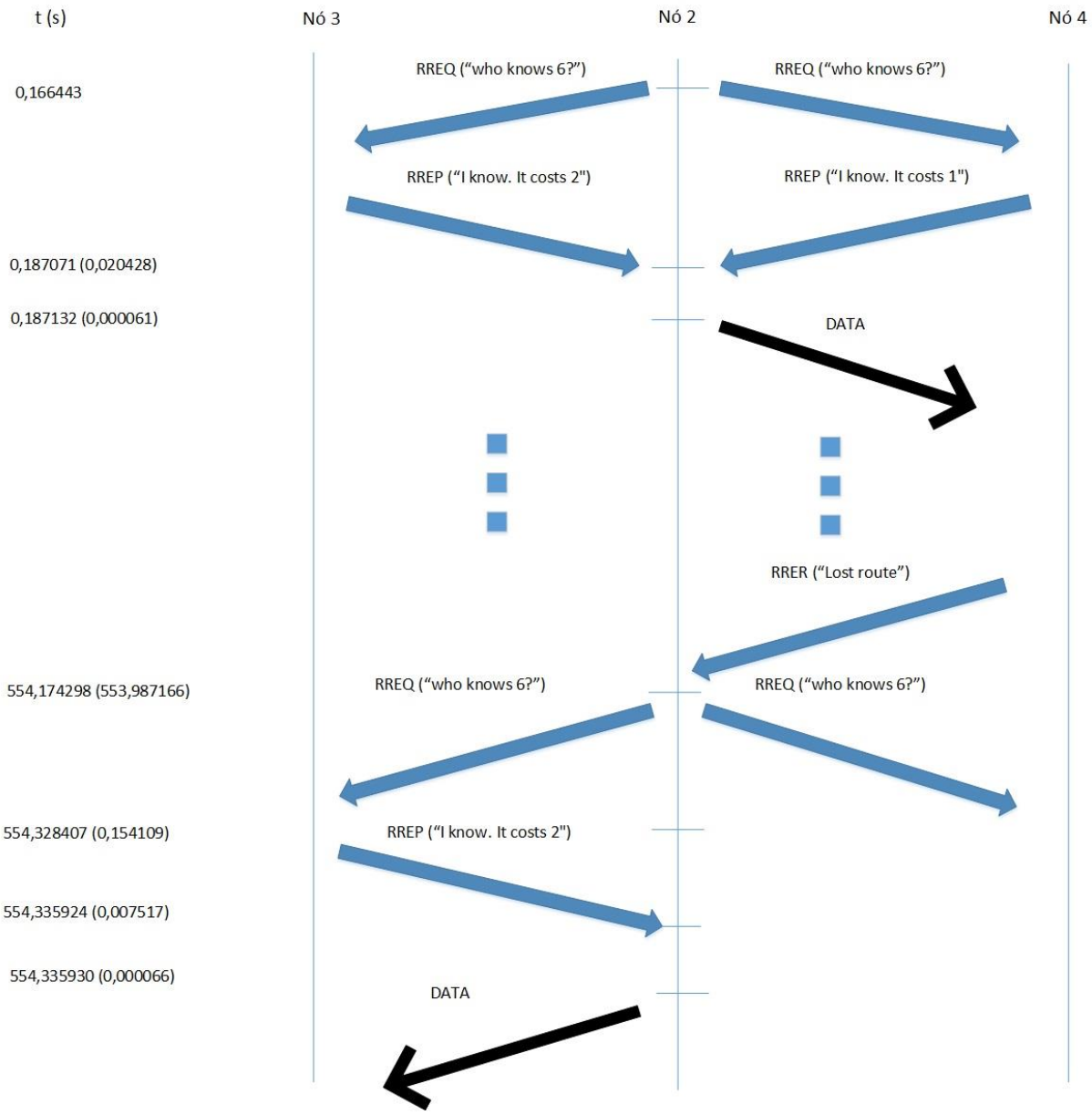

Figura 23. Diagrama de tempo apresentando os instantes de eventos ocorridos durante o experimento.

Obtivemos êxito inclusive na entrega de pacotes de dados. Estes dados, que foram gerados com o mesmo padrão realizado na topologia linear, eram compostos pela mesma quantidade de pacotes com os mesmo tamanhos descritos na Seção anterior. Assim como ocorreu naquele primeiro experimento, nenhum pacote deixou de ser entregue ao destino. Novamente, como temos uma topologia simples, não haviam muitos motivos para que isso ocorresse. Entretanto, poderiam haver possíveis perdas no momento em que o nó 4 deixasse de fazer parte da rede. Desde que ele forçou a quebra da rota e isto foi percebido pelo nó S (2), a transmissão de pacotes de dados foi suspensa pela fonte, que então passou a enviar pacotes buscando uma nova rota. O tráfego de pacotes de dados foi retomado somente quando uma nova rota foi estabelecida. Verificamos ainda que uma pequena quantidade pacotes de dados foram enviados através de 4 mesmo após ele atingir o seu limiar (enviados pelo nó S (2) antes que ele fosse notificado deste evento), porém o nó 4 continuou repassando estes pacotes, que chegaram naturalmente ao destino.

6 CONCLUSÃO

Este trabalho apresentou uma modificação do protocolo AODV com o objetivo de melhorar a eficiência energética no roteamento de redes *ad hoc* em dispositivos reais com a plataforma Android. Para isso foram efetuadas pesquisas a respeito do padrão IEEE 802.11 (Wi-Fi), pois os dispositivos testados executam a rede sobre esta interface. Estudou-se as características das redes *ad hoc* e, especificamente, o protocolo AODV. Também foram feitas pesquisas em técnicas de economia de energia para roteamento para serem aplicadas na nossa proposta.

Foram estudados diversos trabalhos que tinha o mesmo objetivo: adaptações do protocolo AODV para levar em consideração aspectos energéticos. Vários aspectos destes trabalhos puderam ser aproveitados, apesar de que, na maior parte dos casos, seus testes foram feitos apenas em ambientes de simulação. Também nos baseamos em trabalhos de implementação do AODV para Android, os quais foram adaptados por nossa proposta.

Foi desenvolvida uma melhoria na biblioteca de AODV para Android utilizada. Anteriormente o processo de inicialização e configuração da rede *ad hoc* era feito manualmente. Após as mudanças efetuadas por este trabalho, todo este procedimento é feito na inicialização da própria biblioteca. Foram introduzidas informações energéticas nos pacotes de controle do AODV a fim de propagar estas informações através da rede. Com isso, a escolha de rotas pode ser feita utilizando técnicas que associam custos às rotas baseados em função das quantidades de bateria dos nós que compõe cada rota, dreno de bateria, quantidade de saltos, dentre outros parâmetros.

Outra contribuição proposta foi a determinação de limiares que impedem a formação de rotas através de nós que estão com níveis críticos de bateria. Ao atingirem estes limiares, os nós deixam de participar de todas as rotas as quais eles fazem parte ao enviarem uma mensagem RERR aos seus precursores. Ao mesmo tempo ele deixa de enviar mensagens HELLO para não ser notado na rede. Ao receberem mensagens RREQ as quais ele não é o destino final, ele os ignora impedindo a formação de novas rotas que passem por ele.

Foram realizados testes em duas topologias com a utilização de cinco dispositivos Android. Um gerador de tráfego a uma taxa de bits constante foi utilizado para simular o envio de dados através da rede. O objetivo destes testes foi analisar o consumo de bateria nestes dispositivos sobre o protocolo AODV modificado. Também foi objetivo destes testes a verificação da formação e manutenção de rotas. Além do mais, desejou-se verificar se os limiares determinados seriam respeitados pela aplicação.

Verificou-se através destes experimento, que as modificações propostas no protocolo AODV funcionaram corretamente. Estas propostas contribuíram para a escolha mais inteligente de rotas através de parâmetros de energia e que o tempo de vida da rede obtido foi admissível. Apesar do custo extra em controle inserido, os valores obtidos de vazão ainda ficaram dentro de valores que consideramos aceitáveis. Estes valores podem ainda ser melhorados especializando o programa em uma única técnica de economia de energia.

Os experimentos efetuados possuíam suas limitações devido a quantidade de dispositivos disponíveis para testes. Em trabalhos futuros devem buscar variar a topologia para obter resultados em diferentes cenários. Para a validação dos resultados obtidos neste trabalho, os experimentos efetuados poderiam ser repetidos algumas vezes para tirar a média dos valores obtidos e obtê-los com maior precisão. Os valores obtidos também podem ser comparados com a execução de experimentos semelhantes feitos com o protocolo AODV original, que mostrarão quantitativamente a melhoria obtida.

Não foi inspecionada a potência em que os dispositivos realizavam a transmissão de informações. Em um próximo trabalho, deve-se estudar este valor e variá-lo a fim de encontrar sua influência no consumo de energia e até mesmo nas distâncias de transmissão alcançadas, que atualmente são curtas. Deve-se inspecionar se a potência pode ser configurada para um valor maior para aumentar a abrangência da rede.

A fim de reduzir o custo em controle introduzido, podemos limitar o valor do Array que transmite as quantidades de energia nos pacotes de controle do AODV. Atualmente, em uma rede muito grande, eles crescerão indeterminadamente. Para otimizar este aspecto, num trabalho futuro, este Array poderia ser limitado a uma certa quantidade de bytes, preferencialmente contendo informações dos nós

mais próximos do destino e evitando ter pacotes com cabeçalho muito maior que a quantidade de dados. Para resolver o mesmo problema, também pode-se fazer a compressão dos cabeçalhos de controle do AODV.

Outra melhoria futura que pode ser feita em nosso trabalho é a substituição do NEDB de um arquivo de texto simples para uma estrutura de dados mais eficiente para ordenação e busca, como árvore e heap. Esta mudança poderia melhorar a eficiência das buscas e inserção de dados para redes maiores. Como nossos testes houve impacto negativo na implementação. Entretanto, para redes envolveram cenários com no máximo cinco dispositivos, não maiores esta simplificação pode se tornar um problema.

Neste trabalho só utilizamos dispositivos com Android versão 2.3 – Gingerbread. Apesar de ainda ser a versão atualmente mais popular deste sistema operacional, em breve ele se tornará obsoleto. Para trabalhos futuros, deve-se investir em pesquisa para colocar dispositivos com as versões mais recentes do Android no modo *ad hoc*. Atualmente a versão mais nova deste sistema é o Jelly Bean versão 4.3.

Para que este paradigma de redes *ad hoc* para *smartphones* se torne popular, o consumo energético é um problema muito importante a ser resolvido. Levando em consideração o nosso trabalho, ele deve especializar-se em alguma das técnicas de economia de energia como, por exemplo, o MMBCR que utilizamos. Reforçamos que nosso objetivo foi criar um ambiente onde as técnicas pudessem ser comparadas e, quando escolhida uma delas, um trabalho futuro poderia implementar da forma mais eficiente a técnica escolhida.

Por outro lado, muitos outros aspectos devem ser levados em consideração para a popularização das redes *ad hoc* em *smartphones* e devem também ser objeto de estudo. Destacamos, por exemplo, as distâncias que as interfaces atuais alcançam, que ainda são curtas, e também formas de incentivo aos usuários de celulares para que eles façam parte da rede mesmo quando não estão enviando ou recebendo dados, o que é essencial para a formação e abrangência da rede. Quando estes, dentre outros aspectos, estiverem esclarecidos, estas redes poderão ser implantadas nativamente em dispositivos para a comunicação entre os usuários e até mesmo para resolver problemas de sobrecarga nas redes de dados de telefonia [21].

Atualmente está sendo discutida a versão 2 do protocolo AODV [22]. Ainda não existem implementações deste protocolo, mas ele já apresenta diversas melhorias em relação a versão utilizada neste trabalho. Uma das mais importantes para nosso foco é que nesta nova versão ele suportará nativamente métricas de roteamento alternativas e não mais apenas a quantidade de saltos. Exemplos citados na própria RFC [22] deste protocolo incluem métricas de energia, além de latência, atraso e métricas financeiras. Um trabalho futuro a ser considerado é a implementação desta versão do protocolo para dispositivos Android.

O Android ainda é um dos únicos sistemas operacionais para dispositivos móveis com código aberto. Dessa forma, a maior parte dos trabalhos nesta área foca apenas neste sistema. Entretanto, na iminência de surgirem novos sistemas operacionais abertos para dispositivos móveis, como o Firefox OS e o Ubuntu Touch, em pouco tempo devem surgir trabalhos de implementação para estes novos sistemas. Espera-se que, com a popularização e amadurecimento deste paradigma de rede, estas funcionalidades desenvolvidas, por exemplo, em nosso trabalho já venham embutidas nativamente dentro dos sistemas operacionais.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] HAAS Z. J., DENG J., PAPADIMITRATOS P. and SAJAMA S., "Wireless ad hoc networks", in Wiley Encyclopedia of Telecommunications, John G. Proakis, Ed. Wiley, December 2002.

[2] CLAUSEN T. and JACQUET P., "Optimized Link State Routing Protocol (OLSR)". RFC 3626 (Experimental), October 2003, disponível em http://www.ietf.org/rfc/rfc3626.txt, acesso em 10 de julho.

[3] OGIER R., TEMPLIN F. and LEWIS M., "Topology dissemination based on reverse-path forwarding (TBRPF)", IETF RFC 3684, February 2004, disponível em http://tools.ietf.org/html/rfc3684, acesso em 20 de julho.

[4] PERKINS C.E. and BHAGWAT P., "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers", *In Proceedings of the SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, August 1994.

[5] MURTHY S. and GARCIA-LUNA-ACEVES J., "An Efficient Routing Protocol for Wireless Networks", *Mobile Networks and Applications Journal*, Special Issue on Routing in Mobile Communication Networks, Oct. 1996, pp. 183–97.

[6] "IEEE standard for information technology – telecommunications and information exchange between systems – local and metropolitan area networks – specific requirements part 11: Wireless lan medium access control (MAC) and physical layer (PHY) specifications," IEEE Std 802.11-2012, 2012.

[7] KUROSE, J.; ROSS, K. Computer networks: A top down approach featuring the internet. Pearson/Addison Wesley, 2010.

[8] PERKINS C.E., and ROYER E.M., "Ad hoc on-demand distance vector routing", *in Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

[9] TOH C. K., "Maximum battery life routing to support ubiquitous mobile computing in wireless ad hoc networks", IEEE Comm. Mag., June 2001, pp. 138-147.

[10] JING X. and. LEE M. J, "Energy-aware algorithms for AODV in Ad hoc networks," *Proceedings of Mobile Computing and Ubiquitous Networking*, pp. 466-468, Yokosuka, Japan, Jan. 2004.

[11] JRADI R. K. and REEDTZ L. S., "Ad hoc Networks on Android" Technical report, Technical University of Denmark, 2010.

[12] CORRIERO N., COVINO E. and MOTTOLA A., "An approach to use FB-AODV with Android", *Digital Investigation*, vol. 5, pp. 336-343, 2011.

[13] KIM D., GARCIA-LUNA-ACEVES, J.J., OBRACZKA K., CANO J. and MANZONI P., "Power-Aware Routing Based on The Energy Drain Rate for Mobile Ad Hoc Networks", In *Proceedings of the IEEE International Conference on Computer Communication and Networks*, October 2002.

[14] GUPTA N. and DAS S. R., "Energy-aware on-demand routing for mobile ad hoc networks", In Proc. IWDC, pages 164-173, 2002.

[15] DING R. and MUNTEAN G.-M., "A novel device and application-aware energy efficient routing algorithm for WLANs," in *GLOBECOM Workshops* (GC Wkshps), 2012 IEEE, Dec. 2012.

[16] Página para desenvolvedores da plataforma Android, disponível em http://developer.android.com/develop/index.html, acesso em 19 de agosto de 2013.

[17] Tutorial para fazer root em celulares Galaxy Y, disponível em http://www.tecmundo.com.br/galaxy/42409-como-fazer-root-no-galaxy-y-video-.htm, acesso em 14 de agosto de 2013.

[18] Google Code da aplicação WiFi-Tether, disponível em https://code.google.com/p/android-wifi-tether/, acesso em 14 de agosto de 2013.

[19] Download da versão do aplicativo WiFi-Tether utilizado, disponível em https://code.google.com/p/android-wifi-tether/downloads/detail?name=wifi_tether_v3_2-beta2.apk&can=2&q=, acesso em 14 de agosto de 2013.

[20] Repositório do código-fonte da aplicação desenvolvida no trabalho, disponível em https://github.com/augustoraphael/ea-aodv-for-android, acesso em 19 de agosto de 2013.

[21] HAN B., HUI P., KUMAR V., MARATHE M., SHAO J. and SRINIVASAN A., "Mobile data offloading through opportunistic communications and social participation", *IEEE Transactions on Mobile Computing*, vol. 11, no. 5, pp. 821-834, 2011.

[22] PERKINS C.E., RATLIFF S. and DOWDELL J., "Dynamic MANET On-demand (AODVv2) Routing", draft-ietf-manet-dymo-26, disponível em http://tools.ietf.org/html/draft-ietf-manet-dymo-26, acesso em 15 de agosto de 2013.

[23] PERKINS C.E. and ROYER E.M, "Ad hoc On-Demand Distance Vector (AODV) Routing", IETF RFC 3561, Julho 2003, disponível em http://www.ietf.org/rfc/rfc3561.txt, acesso em 15 de agosto de 2013.

[24] SINGH S., WOO M. and RAGHAVENDRA C.S., "Power-aware routing in mobile ad hoc networks", *Proceedings of ACM MobiCom'98*, Dallas, Texas, 1998, pp. 181–190.

[25] Página de download da aplicação Shark for Root, disponível em https://play.google.com/store/apps/details?id=lv.n3o.shark&hl=pt_BR, acesso em 19 de agosto de 2013.

[26] Descrição do protocolo AODV pelo grupo MOMENT Lab. (líder Elizabeth M. Royer), disponível em http://moment.cs.ucsb.edu/AODV/, acesso em 22 de agosto de 2013.

[27] JOHNSON D. and MALTZ D., "Dynamic source routing in ad hoc wireless networks", In Computer Communications Review - *Proceedings of SIGCOMM '96*, Aug. 1996.