Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

Estudo de protocolo baseado em conteúdo e implementação de uma aplicação mensageira para celulares

Autor:

Nelson Rodrigo Pérez Benítez

Orientador:

Prof. Aloysio de Castro Pinto Pedroza, Ph.D.

Orientador:

Prof. Claudio Luis Amorim, Ph.D.

Examinador:

Prof. Marcelo Luiz Drumond Lanza, M.Sc.

DEL

Maio de 2011

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica - Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro - RJ CEP 21949-900

Este exemplar é de propriedade da Universidade Federal do Rio de Janeiro, que poderá incluí-lo em base de dados, armazenar em computador, microfilmar ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do(s) autor(es) e do(s) orientador(es).

DEDICATÓRIA

Aos meus Pais, Nelson e Lidia.

AGRADECIMENTO

Agradecimentos à minha família, principalmente à minha mãe, Lidia Benítez e ao meu pai, Nelson Pérez por te me dado a oportunidade de completar mais essa importante etapa da minha vida. As minhas irmãs Maria Gabriela e Adriana María, pelo apoio brindado em todos esses anos. À minha namorada, Diana Castro por estar sempre presente a pesar da distância.

Também agradecimentos especiais ao meu orientador, o professor Claudio Amorim pelo apoio brindado durante o desenvolvimento deste trabalho. Também ao professor Aloysio Pedroza pela sempre boa vontade de me receber e me orientar. E ao professor Marcelo Lanza, por ter cedido seu tempo à correção deste trabalho e a participação na banca.

Um agradecimento também muito importante ao Héberte de Moraes e o Renato Dutra pelas incansáveis sessões de explicações e análise do meu trabalho; sem dúvida foram de fundamental importância para cumprir com as metas inicialmente propostas.

Obrigado também à Lauro Whately, Diego Dutra, Vinicius Busquet e todos os demais membros LCP pelo apoio brindado em todo momento e pelos conselhos sempre úteis.

Agradecimentos também aos colegas de curso Tiago Bitarelli e Claudio Medeiros, pelas tantas horas de estudos acumuladas que com certeza renderam resultados.

RESUMO

O tema deste trabalho é o estudo de um protocolo de roteamento para redes ad hoc móveis. O protocolo preliminarmente desenvolvido para desktop e utilizando máquina virtual java foi adaptado à plataforma móvel. Inicialmente, o sistema foi portado com as mesmas características e funcionalidades da versão ja existente. Posteriormente, foi introduzida a capacidade de criar e se conectar a redes ad hoc, o que permitiu que os nós operem de maneira completamente distribuída e então alguns testes básicos de desempenho foram efetuados.

Palavras-Chave: ad hoc, MANET, wifi, Android.

ABSTRACT

The purpose of this work is the study of a routing mechanism for Mobile Ad hoc Networks. An existing proof-of-concept application implementing this routing protocol for the desktop platform was modified and adapted to work on a mobile enviroment. Initially ported with the same basic functionalities, the application was later given ad-hoc-network capabilities, allowing the devices to operate in a fully distributed fashion and perform basic performance tests.

Key-words: ad hoc, MANET, wifi, Android.

SIGLAS

UFRJ - Universidade Federal do Rio de Janeiro

API - *Aplication Programming Interface*

SDK - *Standard Development Kit*

NDK - *Native Development Kit*

AODV - *Ad hoc On-demand Distance Vector*

REPI - *Rede Endereçada Por Interesses*

AP - *Access Point - Ponto de Acesso*

ADB - *Android Debug Bridge*

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Tema

O tema deste trabalho é a implementação e testes de uma aplicação de troca de mensagens para celular que utiliza um protocolo original de comunicação baseado em conteúdo para o roteamento de mensagens em um ambiente de rede ad hoc móvel.

1.2 Justificativa

O grande crescimento no número de dispositivos móveis nos últimos anos tem mudado a maneira como as pessoas se comunicam. É particularmente notável a taxa de crescimento do número de assinantes de telefonia móvel, que para o ano 2009 atingiu uma incrível penetração de quase 70 % da população mundial. Não menos significativo é também o fato de que em 2009, o número de assinantes de banda larga móvel tenha superado pela primeira vez o número de assinantes de banda larga fixa [1].

Esses dados revelam a grande disposição que existe por parte dos consumidores em uma escala global de utilizar cada vez mais o dispositivo móvel como fonte e veículo de qualquer tipo de informação.

As Redes Ad-hoc Móveis, em inglés Mobile Ad hoc Network (MANETs) são redes formadas por dispositivos móveis sem elemento centralizador e infraestrutura fixa.

Tradicionalmente as redes móveis ou sem fio mais difundidas envolvem a utilização de algum tipo de processamento central para o encaminhamento das mensagens pela rede. As redes de telefones celulares utilizam as antenas ou estações base, as redes sem fio domésticas geralmente encontram-se ligadas em uma topologia em estrela em torno de um roteador central.

A possibilidade de conexão direta entre dois ou mais nós de uma rede sem necessidade de ponto central abre um novo horizonte de possíveis aplicações para os aparelhos com conexões wireless, além de proporcionar uma maior flexibilidade de uso e uma possível economia de custo.

As MANETs tem sido um tópico de intensa pesquisa nos últimos anos. Vários algoritmos foram propostos para atender ao problema do roteamento das mensagens, destacando-se o AODV[2], Gossip [3] e o OLSR [4]. No entanto, as limitações no desempenho, em particular a baixa taxa de entrega e a alta latência apresentadas por estes protocolos têm resultado em poucas aplicações práticas.

Visando superar estas limitações, pesquisadores do Laboratório de Computação Paralela da COPPE propuseram um original protocolo de roteamento de mensagens para ambientes de redes ad hoc móveis [5]. O protocolo foi amplamente simulado e testado em laboratório, e inclusive um protótipo de aplicação mensageira foi desenvolvido utilizando as regras básicas de encaminhamento adotadas pelo protocolo.

1.3 Objetivos

O objetivo principal deste trabalho é então adaptar esta aplicação para a plataforma móvel e deste modo obter um protótipo funcional que utilize a rede wireless como meio físico de transmissão.

A introdução da conectividade ad hoc foi apresentada como um objetivo secundário, mas de alto interesse a fim de obter um diferencial a mais para o trabalho.

1.4 Delimitação

A proposta inicial do trabalho consistia em adaptar um software já existente e rodando um uma máquina virtual Java para a plataforma móvel. Esta aplicação faz uso da rede IP para o roteamento das mensagens, mas implementa em nível de aplicação a lógica do protocolo, mantendo uma tabela de vizinhos e aplicando as regras que serão definidas mais adiante para a retransmissão ou descarte das mensagens.

Foi também proposto adicionar o recurso de criação de redes ad hoc. Esta alteração permitiria que o aparelho pudesse entrar em contato com outros semelhantes sem a necessidade de um ponto de accesso ou infraestrutura qualquer, demonstrando a capacidade do protocolo de roteamento na prática.

1.5 Metodologia

Na fase de planejamento, foi necessário escolher uma plataforma móvel para a qual a aplicação seria desenvolvida. Levando-se em consideração o grande crescimento, tanto em número de usuários como também na quantidade de dispositivos e a disponibilidade do código fonte, o sistema operacional Android [6] se destacou e foi escolhido como a primeira opção.

Na primeira fase de implementação, o código da aplicação já existente foi revisado e adaptado para funcionar com a API proporcionada pelo Android. O fato de contar com uma aplicação que foi implementada desde o início visando o uso do padrão de engenharia de software conhecido como MVC (Model View Controller) ajudou muito na tarefa de portar grande parte do código para a nova plataforma. De fato, o modelo da aplicação original quase não sofreu alteração, mas o controlador foi bastante modificado enquanto que a parte visual da aplicação foi criada do zero.

Em uma etapa posterior, a possibilidade tanto de detectar como de criar uma própria rede no modo ad hoc foi estudada. Entretanto, esta funcionalidade acabou revelando-se um pouco mais complexa de se implementar do que inicialmente

fora prevista, principalmente devido ao fato dela estar bloqueada via software na distribuição padrão do sistema Android.

Foi constatado que o hardware da maioria (se não de todos) os aparelhos que possuem conexões wifi suporta o modo ad hoc. A ausência de suporte nos aparelhos trata-se então, como já fora mencionado, de uma simples limitação de software[7]. Para contornar este problema, foi necessário desbloquear o acesso ao modo root (Superusuário) que geralmente vem desabilitado quando o aparelho é adquirido.

1.6 Descrição

Este documento, encontra-se organizado da seguinte maneira. No capítulo 2 serão apresentadas com mais detalhes as principais tecnologias envolvidas neste trabalho. Inicialmente o sistema operacional Android e seu funcionamento interno será brevemente descrito. Logo iremos apresentar com mais detalhes o popular padrão IEEE 802.11 e as suas principais versões.

No capítulo 3 serão apresentadas com mais detalhes as Redes Ad Hoc Móveis (MANETs) e protocolos de comunicação representativos. Posteriormente, faremos uma descrição mais detalhada do protocolo de endereçamento por conteúdo que utilizarei neste trabalho.

No capítulo 4 será dada uma descrição da estrutura básica da aplicação e os resultados obtidos.

Por fim, no capítulo 5, será apresentada a conclusão e sugestões para possíveis trabalhos futuros, seguidas pelas referências bibliográficas consultadas no decorrer do trabalho.

Capítulo 2

O Sistema Operacional Android

2.1 História e precedentes

Tradicionalmente para criar aplicações voltadas a plataformas embarcadas como celulares, o desenvolvedor tinha que possuir um certo grau de intimidade com o hardware para o qual estava desenvolvendo. O software resultante era compatível apenas com um determinado aparelho, ou tal vez um conjunto de aparelhos do mesmo fabricante. Com a massiva popularização dos celulares veio a necessidade de criar software de maneira mais rápida e eficiente; este modelo então rápidamente mostrou suas deficiências.

Um grande avanço nos últimos anos neste sentido foi a introdução dos Java MIDlets. Os MIDlets são aplicações executadas em uma máquina virtual Java otimizada para o ambiente móvel. Este processo abstrai a camada do hardware do programador e permite que as aplicações criadas possam ser executadas em uma maior variedade de dispositivos.

Por outro lado, os MIDlets apresentam uma inconveniente restrição na execução das aplicações. Por motivos de segurança, cada MIDlet encontra-se isolado em um ambiente de execução fechado conhecido popularmente como "sandbox"(do inglês para caixa de areia) desde o qual possui um accesso restrito ao hardware do aparelho e ao resto do sistema. Por exemplo um MIDlet não pode detectar (ou ser avisado) quando há uma ligação a ser recebida ou receber um SMS enviado por outro aparelho [8].

O resultado é que diferente das aplicações nativas criadas pelo fabricante do aparelho, as aplicações de terceiros na forma de MIDlets obtém um menor grau de accesso ao hardware e direitos de execução [9]. O que acaba limitando o potencial das aplicações, já que em geral são incapazes de tirar proveito da mobilidade dos aparelhos.

Mais recentemente vários sistemas operacionais específicos para aparelhos móveis foram introduzidos ampliando drasticamente as possibilidades do desenvolvedor. Dentre alguns dos mais conhecidos, encontram-se o Symbiam OS, Windows Mobile, Palm OS e o iOS (utilizado em no iPhone). No entanto, diferentemente do Android, estes sistemas são proprietários e em alguns casos dão prioridade às aplicações nativas criadas pelo fabricante acima daquelas criadas por terceiros, impõem limitacões na comunicação entre aplicações e dados nativos do aparelho, e controlam ou restringem a distribuição de aplicações criadas por terceiros [9].

O sistema operacional Android tem seu início no ano de 2005[10], quando a Google compra uma pequena empresa de Palo Alto, California denominada Android Inc. e começa a investir no desenvolvimento da plataforma. Para o ano de 2007 um grupo de empresas líderes da indústria anuncia a criação do Open Handset Alliance com o objetivo de promover a inovação no desenvolvimento de produtos para celulares. O primeiro Kit de Desenvolvimento Padrão, ou SDK (Standard Development Kit) foi disponibilizado em novembro de 2007 e em setembro de 2008 foi anunciado o G1, o primeiro dispositivo comercial utilizando a plataforma. No mês seguinte, a Google liberou o código fonte da plataforma sob a licença Apache de código aberto[11].

O sistema operacional Android oferece um ambiente de desenvolvimento aberto que possui como base o sólido Kernel do Linux, um projeto maduro, estável e com amplo suporte internacional. O acesso ao hardware é facilitado por meio de uma completa biblioteca de funções e as interações entre aplicações, apesar de ser estritamente controladas, são completamente suportadas. Além disso, o sistema é completamente democrático no sentido em que todas as aplicações possuem os mesmos direitos de execução e permissões, sejam elas criadas pelo fabricante, ou por um desenvolvedor qualquer. De fato, tanto as aplicações criadas pelo fabricante

como aquelas criadas por terceiros são criadas usando a mesma API e executadas no mesmo ambiente. Isso permite que o usuário substitua facilmente uma aplicação qualquer como o discador ou o administrador de contatos por exemplo, por uma versão terceirizada.

2.2 Arquitetura do sistema

Um dos objetivos fundamentais da arquitetura do sistema Android é facilitar a interação das aplicações entre si, e fornecer mecanismos que permitam a reutilização de componentes de outras aplicações.

Os diversos componentes do sistema Android encontram-se organizados formando uma pilha de componentes de software. Trata-se basicamente de um kernel e uma coleção de bibliotecas C/C++ expostas por meio de um framework que provê serviços e controla a execução das aplicações.

A seguir uma descrição mais detalhada de cada um dos componentes da pilha.

- Kernel do Linux

  No núcleo da plataforma, encontra-se o Kernel do Linux versão 2.6.29, responsável pelos drivers, gestão de processos e memória, gerenciamento de energia e outras tarefas básicas do sistema. Além disso, o Kernel ainda provê uma camada de abstração entre o hardware e o resto da pilha.

- Bibliotecas

  Em um nível acima do Kernel, encontram-se um conjunto básico de bibliotecas nativas C/C++. A seguir uma lista das mais importantes.

  Libc A biblioteca C do sistema é uma versão otimizada para ambientes embarcados.

  Mídia As bibliotecas de mídia provêem funcionalidades básicas para a manipulação de áudio e vídeo.

Gráficos Bibliotecas para a renderização de gráficos 2D e 3D que incluem o SGL e OpenGL.

WebKit Esta biblioteca é responsável pelo suporte ao browser, trata-se da mesma utilizada no Google Chrome e no Safari.

SQLite Trata-se de um banco de dados relacional encapsulado na forma de biblioteca. O SQLite é também um projeto de código aberto independente não relacionado com o Android.

- Android Run time

O que diferencia o Android de mais uma implementação para ambientes embarcados do Linux é o denominado Android Run Time. Trata-se basicamente da máquina virtual Dalvik e bibliotecas que a suportam.

Apesar do desenvolvimento de aplicações para Android ser feito na linguagem de programação Java, a Dalvik não é uma máquina virtual Java. As bibliotecas do núcleo (core libraries) proporcionam a maior parte das funcionalidades existentes nas bibliotecas Java, assim como algumas bibliotecas que são específicas do Android.

A maior parte dos componentes do application framework (e indiretamente das aplicações) accessam a bibliotecas centrais através da máquina virtual Dalvik. Trata-se de uma máquina virtual baseada em registradores que foi otimizada para garantir que várias instâncias possam ser executadas ao mesmo tempo de maneira eficiente.

- Android Framework

O Framework provê o conjunto de classes utilizado na criação de aplicações. Também proporciona um certo nível de abstração no acesso ao hardware gerencia a interface do usuário. Trata-se da mesma API utilizada para a criação das aplicações básicas.

O framework está projetado de maneira a promover a reutilização dos componentes, com facilidades para que as aplicações publiquem e compartilhem suas informações com outras aplicações. Sempre com o accesso autorizado pelo usuário.

- Camada de Aplicação

Na camada de aplicação encontram-se todas as aplicações do usuário. Existe um conjunto básico de aplicações que são criadas pelo fabricante e ja estão instaladas na hora da compra, mas como já foi destacado anteriormente estas podem ser trocadas a qualquer momento se assim o usuário o desejar.

Cada aplicação no sistema Android é executada em um processo separado e sobre uma instância diferente da máquina virtual Dalvik.

2.3 A Máquina Virtual Dalvik

Sendo talvez a peça mais innovadora de toda a plataforma; a máquina virtual Dalvik foi projetada específicamente para o sistema Android.

Existem dois fatores que levaram à criação da máquina virtual Dalvik. O primeiro é a licença. A pesar da linguagem e o compilador Java serem gratuitos, isso não é precisamente verdade com respeito à Maquina Virtual Java. O segundo motivo é o desempenhos, já que apesar de ter sido bastante melhorada desde as primeiras versões, a máquina virtual Java não foi especificamente projetada para ser executada em um ambiente com severas limitações de memória e CPU. Para termos uma idéia das limitações, vale lembrar que o primeiro aparelho comercial com Android, o G1 tinha apenas 192 MB de memória RAM e um processador Qualcomm MSM7201A de 528 MHz. Sendo que muitos outros aparelhos atuais ainda se encontram em uma faixa bem próxima.[11]

Entre as otimizações de maior importância podemos citar a compressão dos arquivos executáveis. Para reduzir ainda mais o tamanho de um arquivo executável, a

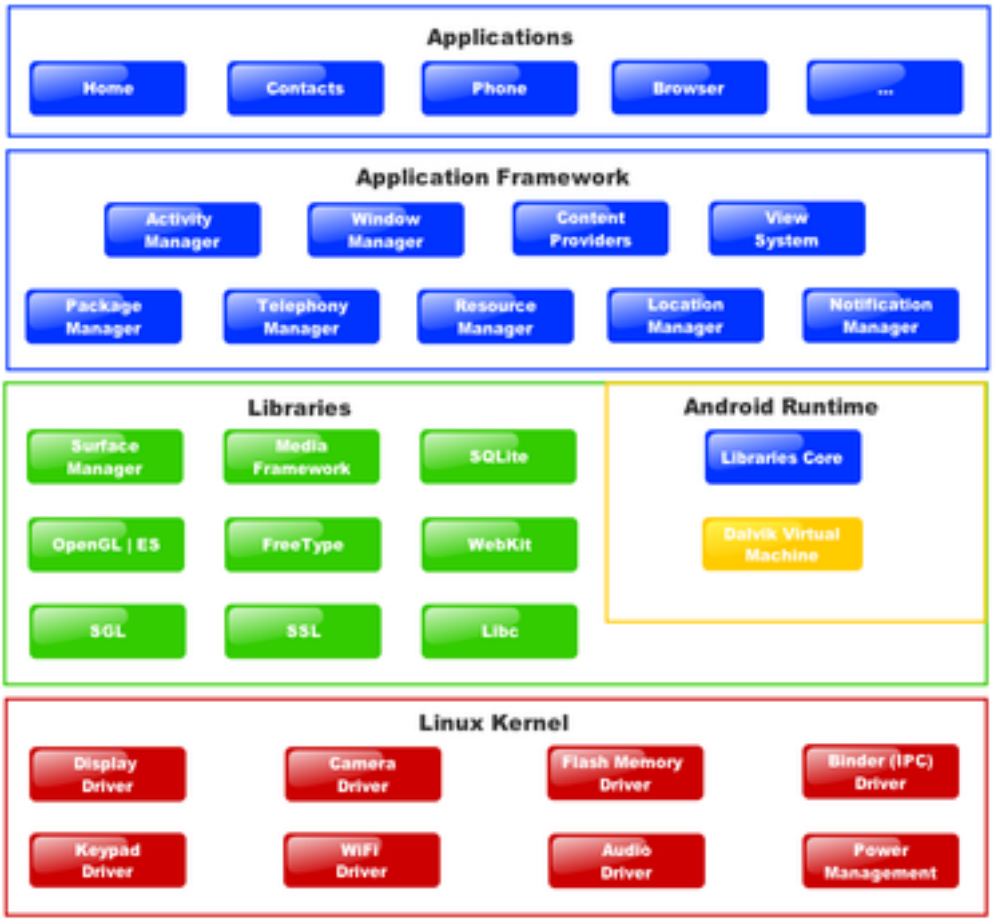

Figura 2.1: Pilha de componentes do sistema Android.

máquina Dalvik parte de um executável gerado pelo compilador Java (Um arquivo JAR, contendo uma ou várias classes comprimidas) e reutiliza as informações duplicadas dentro dos diversos arquivos .class contidos nele. Gerando assim um ou mais arquivos denominados Executáveis Dalvik (Com a extensão .dex). O Executável Dalvik geralmente é da ordem da metade do tamanho do arquivo JAR original [11].

2.4 O kit de Desenvolvimento Nativo

Apesar das aplicações no sistema Android geralmente estarem contidas em instâncias da máquina virtual Dalvik, desde meados de 2009 a Google apresentou o Kit de Desenvolvimento Nativo, ou NDK (Native Development Kit).

Apesar das ferramentas para compilar código nativo ja existirem antes disso [12], este kit de ferramentas, embora um pouco limitado em sua versão inicial, apresentou várias simplificações para o que seria de outro modo um processo bastante complexo e pouco documentado.

O NDK basicamente provê:

- Um conjunto de ferramentas e Makefiles utilizados para gerar bibliotecas nativas geradas em C ou C++.
- Um conjunto de cabeçalhos e bibliotecas que serão suportadas em todas as futuras versões do sistema, a partir da versão 1.5.
- Uma maneira de inserir as bibliotecas geradas dentro do pacote da aplicação, o arquivo .apk.
- Documentação, tutoriais e exemplos de como compilar uma aplicação que utilize código nativo.

A tecnología utilizada para criar uma ponte entre o código escrito em Java e o código nativo em C ou C++ é conhecida como JNI (Java Native Interface) do inglês para Interface Nativa Java. Trata-se de um framework independente do sistema Android e disponível normalmente para aplicações escritas em Java que precisem utilizar a CPU de maneira intensiva.

Apesar disso, as aplicações que utilizam código nativo continuam rodando dentro de uma máquina virtual e são distribuídas nos pacotes APK. O modelo fundamental do sistema não muda. O código nativo é inserido como uma biblioteca dentro do pacote APK e se relaciona com a aplicação via chamadas JNI.

Pequenos trechos de código nativo foram utilizados no desenvolvimento deste trabalho para possibilitar a criação de redes sem fio no modo ad hoc.

Capítulo 3

Redes Ad Hoc Móveis

3.1 Wifi

De acordo com Kurose [13] as redes sem fio podem ser classificadas de acordo com dois critérios: (i) se um pacote de informação atravessa um ou mais de um enlace sem fío (single ou multi-hop), e (ii) se existe alguma infraestrutura, ou seja se alguma estação cumpre o papel centralizador na rede.

- Single-hop, infraestruturada. São redes organizadas em uma topologia estrela, na qual um único ponto central provê accesso e roteamento para todos os demais, além de oferecer, em geral, conexão para uma rede maior (a Internet). As redes 802.11 mais frequentemente utilizadas em instituções de ensino, ambientes de trabalho e locais públicos estão nesta categoria.

- Single-hop, sem infraestrutura. São redes nas quais não existe ponto central, mas os pacotes geralmente passam por um único enlace sem fio antes de chegar ao destino. Entraria nesta categoria por exemplo, uma rede ad hoc criada temporalmente por dois dispositivos querendo trocar informações na ausência de infraestrutura de rede.

- Multi-hop, infraestruturada. São redes onde existe um ponto central ligado a uma rede cabeada, mas onde alguns nós da rede devem repassar as suas mensagens para outros nós wireless para se comunicarem com o ponto central.

- Multi-hop, sem infraestrutura. Neste tipo de rede sem fio não existe estação base e as mensagens devem ser repassadas a mais de um nó para chegar ao

destino. Os nós podem estar em movimento, neste caso a rede é considerada uma "rede ad hoc móvel"ou "Mobile Ad hoc Network"(MANET).

3.2 MANET

As MANETs são sistemas autônomos constituídos de um conjunto de nós móveis que se comunicam entre si por meio de enlaces sem fio e realizando, se for preciso múltiplos "saltos"entre nós antes de alcançar o destino.

Em outros termos isso quer dizer que uma rede ad hoc móvel permite a criação de interconexões entre dispositivos sem a necessidade de uma administração central ou infraestrutura preestabelecida. O próprio termo ad hoc provém do latim e quer dizer "agora e para agora", referindo-se a algo que é criado de maneira improvisada. O principal propósito de uma rede ad hoc móvel é prover autonomia e flexibilidade aproveitando princípios da auto organização.

Exemplos de possíveis aplicações variam desde redes de micro sensores distribuidos, um conjunto de satélites em órbita, ou sistemas de transporte modernos, onde carros e sinais de trânsito inteligentes seriam os nós.

Tais redes têm sido o objeto de intenso estudo nos últimos anos, com vários protocolos de encaminhamento sendo propostos e avaliados de acordo com métricas específicas.

Da maneria geral, podemos identificar três categorias de técnicas de endereçamento dentro de uma rede ad hoc móvel: encaminhamento proativo, reativo ou híbrido.

No endereçamento proativo ou orientado a tabelas, os nós mantêm uma lista atualizada dos destinos e de suas respectivas rotas de encaminhamento. Estas tabelas são construídas por meio de mensagens de controle que inundam a rede. O custo das inundações em geral é diminuido pela sua relativamente baixa frequência. Uma desvantagem dessa técnica no entanto, é o tempo que a rede pode necessitar para assimilar uma reestruturação de seus nós.

No endereçamento reativo por outro lado, o encaminhamento é feito sob demanda. Quando um nó deseja enviar uma mensagem, ele inunda a rede com pacotes descobridores de rota. Essa técnica tem a clara desvantagem de produzir inundações desnecessárias além de proporcionar uma alta latência em descobrir a melhor rota.

Visando unir o melhor dos dois mundos, as técnicas de endereçamento híbridas combinam as duas técnicas de maneira a minimizar as desvantagens de cada uma.

Um gênero de endereçamento híbrido bem conhecido é o assim denominado endereçamento hierárquico. Neste tipo de protocolos a utilização de técnicas proativas ou reativas depende do nível hierárquico onde o nó se encontre. O endereçamento em geral é iniciado com alguma rota determinada proativamente, e é assistido na sua etapa final, se necessário, por meio de alguma técnica reativa dentro do nível onde o nó destino se encontra. A vantagem deste tipo de solução é que a inundação reativa utilizada para localizar o nó destino, fica contida dentro do nível hierárquico do nó destino. O problema é que dependendo da aplicação, a rede pode mudar tão rapidamente que a própria estrutura hierárquica possui um tempo de vida muito curto.

3.2.1 AODV

O AODV (Ad hoc On-demand Distance Vector) é um protocolo de roteamento reativo, o que significa que as rotas são obtidas apenas quando necessárias. Ele também garante que a rota obtida não contenha ciclos e tenta sempre encontrar o caminho mais curto.

O protocolo AODV opera basicamente utilizando um método de "inundação"controlado. Cada vez que um nó deseja enviar uma mensagem para um outro nó fora do alcance do enlace sem fio, ele envia uma mensagem para os vizinhos que ele conhece. Este tipo de mensagem é denominado RREQ (Route Request - Pedido de rota). Uma RREQ contém vários bits de infomação úteis: a origem, o destino, o tempo de vida e um número de sequência que serve como um identificador único da mensagem.

O identificador serve para garantir que cada nó que receba a mensagem RREQ somente a reenvie uma vez. Isso é importante considerando-se que em um ambiente onde as estações se deslocam continuamente, algumas podem acabar criando ciclos, e uma mesma estação pode vir a receber a retransmissão de uma mensagem RREQ que já havia sido enviada anteriormente. O tempo de vida da mensagem é outro mecanismo de controle que evita o uso desnecessário de recursos de rede.

Caso o nó emissor não receba uma rota dentro de um intervalo de tempo preestabelecido, ele vai retransmitir a mensagem RREQ, mas desta vez com um tempo de vida maior e com um número identificador diferente.

3.2.2 Protocolos epidêmicos

Os protocolos epidêmicos diferem do AODV na maneira em que uma mensagem se espalha pela rede. Existem varias versões de protocolos epidêmicos, também conhecidos como protocolos Gossip (Rumor em inglês). Mas a ideia principal é que cada nó que receba uma mensagem tenha associado a ele uma certa probabilidade de retransmitir a mesma.

Uma mensagem então se espalha pela rede de maneira análoga ao modo de transmissão de uma epidemia em uma população de indivíduos.

Os protocolos epidêmicos são claramente mais eficientes que o AODV, pois tentam minimizar o número de retransmissões desnecessárias. Porém a probabilidade de retransmissão associada a cada nó deve ser cuidadosamente ajustada, pois uma probabilidade muito alta conduz a um excesso de mensagens, enquanto que uma probabilidade baixa demais provoca o desvanecimento prematuro da mensagem.

3.3 Redes Endereçadas Por Interesses - REPI

O trabalho proposto em [14] introduz um novo modelo de comunicação para dispositivos móveis denominado Modelo de Comunicação endereçada por Interesses. Neste contexto, os interesses do usuário são utilizados de maneira ativa pelo sistema para o encaminhamento das mensagens.

O conceito de Rede Endereçada Por Interesses então surge da aplicação do modelo em uma rede de dispositivos móveis. Neste contexto, o interesse é definido como qualquer termo, ou conjunto de termos que possa vir a ter algum significado para o usuário.

Toda mensagem enviada pelo protocolo está composta por um cabeçalho (ou prefixo) e um campo de comprimento variável destinado à mensagem em si. O elemento central do protocolo implementado neste trabalho é o cabeçalho da mensagem. Este cabeçalho, como mostrado na Figura 3.1 foi dividido em duas partes denominadas C (Características) e I (Interesses). A parte C contêm campos cujos dados são gerados por variáveis aleatórias com uma distribuição normal. Já a segunda parte denominada I, contém a lista dos interesses do usuário e assume que eles sigam uma distribuição Zipf.

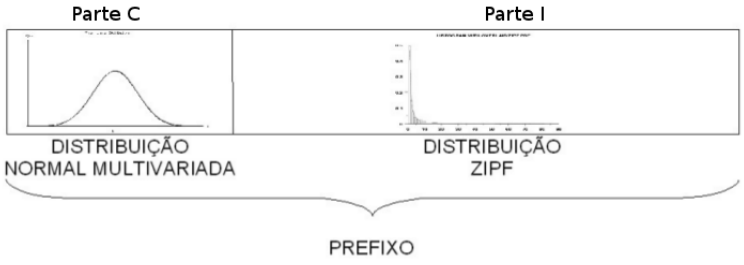

Figura 3.1: Prefixo da mensagem REPI.

De acordo com [15] a parte I, com distribuição Zipf, permite que ocorra formação de grupos. Enquanto que a parte C, com distribuição Normal Multivariada, permite a identificação de um usuário na rede e o encaminhamento da mensagem.

Quando uma mensagem é recebida por algum nó, a parte C do prefixo da mensagem é comparada com as características locais do nó. Várias políticas de encaminhamento podem então ser adotadas; desde a mais estrita exigindo um casamento idêntico entre todos os campos, até uma política mais aberta requerindo apenas o casamento de um dos campos.

Para este trabalho adotamos a política mais relaxada possível, exigindo o casamento de um único campo do prefixo C para o encaminhamento da mensagem.

Nó	**Nó vizinho**	**Prefixo C**					**Prefixo I**	**Enc**	**End**
		a	b	c	d	e			
n_1	n_2	0110	1000	M	75	25	Cinema		
n_2	n_3	0110	1010	F	65	28	Cozinha	V	F
n_3	n_3,n_5	1100	1000	M	70	28	Cinema	V	V
n_4	n_3	1100	1110	F	65	27	Política	F	F
n_5	n_3	1111	1110	F	80	40	Cinema	F	V

Tabela 3.1: Exemplo de encaminhamento de uma mensagem REPI

Para exemplificar o funcionamento do encaminhamento de uma mensagem, considere a situação descrita na tabela 3.1. Aqui o no n_1 quer entrar em contato com quem tenha os mesmos interesses que ele, neste caso o interesse é "Cinema". O nó n_2 encaminha a mensagem recebida de n_1 por haver casamento no campo C_a=0110. O nó n_3 encaminha a mensagem por haver casamento no campo C_b=1000 e ele ainda é destino da mensagem por possuir o interesse "Cinema"em comum com o nó n_1. O nó n_4 apesar de ser vizinho do nó n_3 não possui nenhum casamento nos campos de características, motivo pelo qual descarta a mensagem. O mesmo acontece com o nó n_5, mas neste caso como ele possui o interesse em comum, a mensagem é recebida.

Desta maneira, todos os nós com o interesse "Cinema"foram alcançados e armazenam o Prefixo do nó de origem, permitindo um endereçamento fim-a-fim.

Um papel importante da parte C do prefixo é reduzir o isolamento dos nós devido aos diferentes interesses ao permitir a colaboração no encaminhamento das mensagens. Além disso, reduz a intensidade de uma inundação (flooding) no caso de vários nós possuirem os mesmos interesses.

Com uma operação de casamento simples como a descrita acima, é suposto que não existem nós isolados lógicamente do resto da rede. Por nó isolado entendemos um nó que não possui casamento algum nos diferentes campos de características. Neste caso o nó isolado receberá todas as mensagens dos nós vizinhos, mas não encaminhará nenhuma mensagem e não terá nenhuma mensagem encaminhada. Uma solução para este problema seria que o nó isolado verificasse os prefixos dos seus vizinhos,

por meio das mensagens recebidas e alterasse assim algumas de suas características para permitir o encaminhamento de suas mensagens por um ou mais vizinhos.

Capítulo 4

Aplicação Mensageira

4.1 Aplicação na versão desktop

A aplicação original sobre a qual este trabalho foi desenvolvido foi escrita em Java e possui uma interface gráfica como apresentada na figura 4.1. O desenvolvimento desta aplicação não forma parte do trabalho, mas partes do código foram reutilizadas para o trabalho.

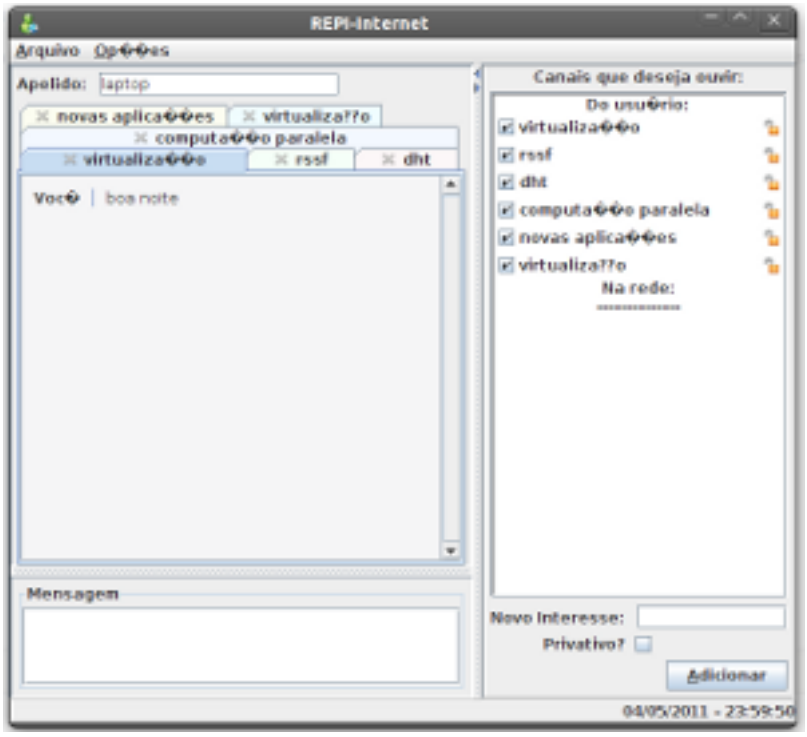

Figura 4.1: Captura de tela mostrando a interface gráfica da aplicação mensageira em seu formato para desktop.

Nela o usuário pode navegar entre várias abas escolhendo o tema de interesse. É possível criar novos interesses e os interesses criados pela rede aparecerão no painel direito.[1]

Esta aplicação utiliza um servidor de inicio (Também denominado servidor de bootstrap) para poder atravesar os limites de uma NAT (Network Address Translator) e estabelecer assim uma conexão peer-to-peer entre as duas instâncias da aplicação. A técnica utilizada para criar uma conexão entre duas aplicações localizadas por trás de uma NAT é conhecida popularmente como "hole punching"(Do inglês "criação de buraco"). Esta técnica não oferece garantia de que uma conexão possa ser estabelecida em todas as situações e com todas as NATs, já que as características das NATs não estão padronizadas.

4.2 Aplicação na versão móvel

4.2.1 Arquitetura da Aplicação

A proposta básica do trabalho consiste em implementar como prova de conceito o protocolo REPI na camada de aplicação. O protocolo de transporte utilizado é o UDP, e na camada de rede teríamos o protocolo IP. A pesar disso, como todas as mensagens são enviadas por broadcast, na prática temos que a funcionalidade de roteamento do protocolo IP é contornada e as únicas regras válidas acabam sendo às do protocolo REPI. A Figura 4.2 ilustra a pilha de protocolos utilizada na aplicação desenvolvida neste trabalho.

A aplicação possui 3 threads principais, que são:

- Thread da Interface gráfica
- Thread Servidor
- Thread Cliente

[1]Para uma descrição mais detalhada da interface com o usuário e suas funções, ver o Apêndice B

Figura 4.2: Figura da pilha de protocolos utilizados na aplicação para plataforma móvel.

Inicialmente a única thread iniciada é a encarregada da interface gráfica. Existem dois eventos que podem disparar a criação das threads cliente e servidor. O primeiro é o usuário decidir criar uma nova conexão wireless no modo ad hoc. O outro evento é a detecção feita pelo próprio sistema da existência de uma conexão com uma rede estruturada. O que indica ao programa que ele encontra-se no modo internet e deve tentar entrar em contato com o servidor de bootstrap para receber uma lista de vizinhos.

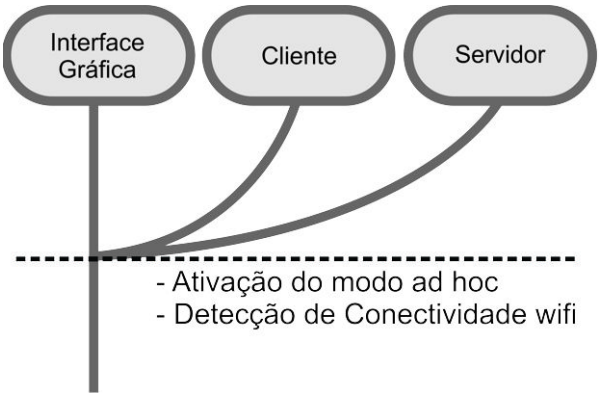

Figura 4.3: Figura esquemática das tres principais linhas de execução da aplicação.

A relação entre as threads é ilustrada na Figura 4.4. Ambas as threads servidor e cliente compartilham um mesmo socket não bloqueante. Quando uma mensagem em é detectada pela thread servidor, esta informa à um componente da thread principal denominado MessageManager. O MessageManager, ou gerenciador de mensagens, é então o encarregado de informar à todos os "ouvintes"(listeners) da recepção da mensagem.

No modo de operação atual, o MessageManager então envia a mensagem para a atividade principal, encarregada de atualizar a interface com o usuário, e também para uma outra entidade dentro da própria thread principal chamada MatchFilter. O MatchFilter tem a responsabilidade de comparar o prefixo do tipo C (Prefixo das características) do emissor com o prefixo do próprio nó. Como já foi dito anteriormente, a política de roteamento adotada neste trabalho foi a mais permissiva de todas, exigindo o casamento de um único campo do prefixo C para decidir reenviar a mensagem para os vizinhos.

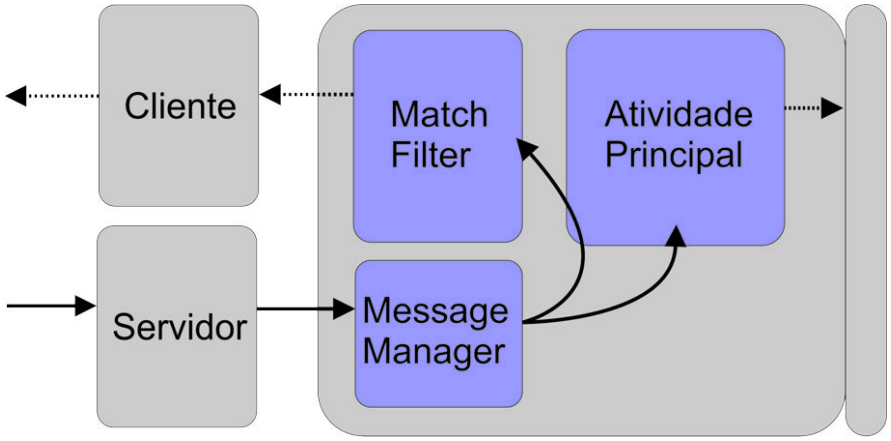

Figura 4.4: Figura esquemática das tres principais linhas de execução da aplicação.

Em versões futuras, somente o MatchFilter irá ser notificado da recepção de uma mensagem, e ele então implementará além do casamento do prefixo C, também o casamento com o prefixo I, decidindo assim se mostrar ou não a mensagem ao usuário. O motivo pelo qual na versão atual temos uma notificação indo diretamente para a atividade principal é que dessa maneira foi mais fácil depurar falhas no código na fase de desenvolvimento.

É importante ressaltar no entanto, que esta alteração é de natureza extremadamente simples, e que inclusive o método de casamento dos prefixos do tipo I, encontra-se já implementado dentro do MatchFilter.

4.2.2 Interface Gráfica

A interface da aplicação móvel foi projetada para ser simples. A atividade principal mostra ao usuário um campo de texto e um botão para enviar as mensagens. A

lista de interesses é carregada com uma lista padrão, mas o usuário pode adicionar ou retirar os interesses se assim preferir. Os interesses aparecem no topo da tela, como uma lista deslizante horizontalmente.

Figura 4.5: Captura de tela mostrando a interface gráfica da aplicação mensageira em seu formato para celular.

O botão menu da aplicação como é padrão no sistema Android, aparece de baixo e mostra uma lista de opções, entre as quais estão uma segunda atividade de configurações, onde o usário pode trocar o nome que irá mostrar para os outros membros da rede. Existe também um botão para listar alguns dos outros membros da rede, e claro, a opção de habilitar uma rede wifi ad hoc.

4.2.3 Conexões ad hoc

Por padrão, um aparelho rodando uma versão oficial do sistema operacional Android até pelo menos a versão 2.2 não detecta redes wifi no modo ad hoc. Uma solução parcial para este problema foi descrita em [16] e consiste em um patch para um arquivo denominado wpa_supplicant. O wpa_supplicant é um executável proje-

tado para rodar como "demônio", ou seja funciona como um "backend"encarregado de controlar as conexões wireless. Nas distribuições de linux normais, ele geralmente funciona em conjunto com outros processos executáveis como "frontends". Eles são o wpa_cli (versão em linha de comando) e o wpa_gui (versão com interface gráfica).

Apesar do que foi discutido no capítulo 2, foi constatado que além de poder criar bibliotecas escritas em código nativo, o NDK também nos permite a criação de executáveis completos a partir de código nativo. O que permite que o patch mencionado anteriormente seja aplicado ao código fonte do wpa_supplicant e uma nova versão então seja utilizada.

Com esta versão alterada do wpa_supplicant[2] , foi então possível que o sistema deixe de filtrar as redes wireless ad hoc da tabela de resultados. Em lugar disso, elas são apresentadas com um asterisco, de maneira a indicar sua condição de redes ad hoc.

Esta solução porém é parcial, pois permite que o aparelho enxergue as redes ad hoc diponíveis, mas nada nos ajuda se quisermos criar uma.

O primeiro passo para criar uma rede ad hoc foi verificar a existência de uma aplicação de código aberto que aparentemente cria uma rede no modo ad hoc[17]. Analisando o funcionamento do código foi constatado que ela utiliza versões próprias dos utilitários conhecidos como ifconfig e iwconfig.

O ifconfig é um programa disponível na maioria das distribuições de Linux e que permite a configuração das interfaces de rede de maneira rápida e simples. Vários parâmetros como o endereço IP e a máscara de rede podem ser repassados pela linha de comando. Se o ifconfig for chamado sem parâmetros, um resumo da configuração de cada interface é mostrado.

O iwconfig é um programa parecido que oferece funcionalidades específicas de redes wireless. Pelo uso de parâmetros podemos configurar a frequência desejada, a

[2]Para este trabalho foi utilizada uma versão ja compilada por terceiros e disponibilizada na internet

potência de transmissão, o BSSID da rede, entre outras coisas.

Esses dois utilitários são indispensáveis na hora de criar uma rede no modo ad hoc. Porém o NDK só nos permite a criação de bibliotecas de código nativo. Foi necessário então criar dentro da biblioteca um método que receba uma string e execute uma chamada ao sistema com a string como argumento.

Antes de utilizar estes dois utilitários no entanto, é preciso executar um passo a mais. Aparentemente o driver da interface wireless encontra-se implementado como um módulo do kernel. É preciso então inserir este módulo de maneira a permitir a ativação da interface de rede. Sem essa inserção, o comando ifconfig falha reportando a falta da interface de rede. A inserção do módulo é feita por meio da função descrita acima, mas o módulo e localização do mesmo variam de aparelho para aparelho.

Felizmente, o código disponível no repositório público da aplicação mencionada apresenta uma lista bastante completa de marcas e modelos de aparelhos, seus respectivos drivers e suas localizações no sistema de arquivos. Além disso foi constatado que geralmente os aparelhos pertencentes a um mesmo fabricante compartilham o mesmo driver, o que simplifica bastante a tarefa de identificação do aparelho e a conseqüente configuração do mesmo.

O estabelecimento da rede no modo ad hoc pela aplicação então foi testada com sucesso em todos os aparelhos disponíveis no laboratório, que são os seguintes:

- Samgung Galaxy I-7500
- Samgung Galaxy 5500
- Motorola MB-502
- Samsung Galaxy S (I-9000)

Um detalhe de implementação importante neste ponto é o fato de termos que disparar uma thread separada da thread encarregada de renderizar a interface gráfica na hora de criar a rede wifi. Isso devido à natureza extremadamente rígida do sistema Android, que não hesita em assumir que uma aplicação haja apresentado uma falha

caso fique sem oferecer resposta por um período de uns poucos segundos. No caso é oferecido ao usuário a chance de fechar a aplicação, dando a falsa impressão que a mesma teve um problema.

4.2.4 Distribuição

As aplicações feitas para o Sistema Operacional Android geralmente são distríbuidas por meio de um pacote comprimido com a extensão APK. Este tipo de arquivo nada mais é que uma variação do formato JAR e contém as classes compiladas em Dalvik bytecode, os recursos da aplicação e opcionalmente alguma biblioteca de código nativo.

Na Figura 4.6 temos uma representação gráfica do APK da aplicação. O componente da direita representa as bibliotecas de código nativo, e está desenhado com um contorno pontilhado para indicar que ele é opcional.

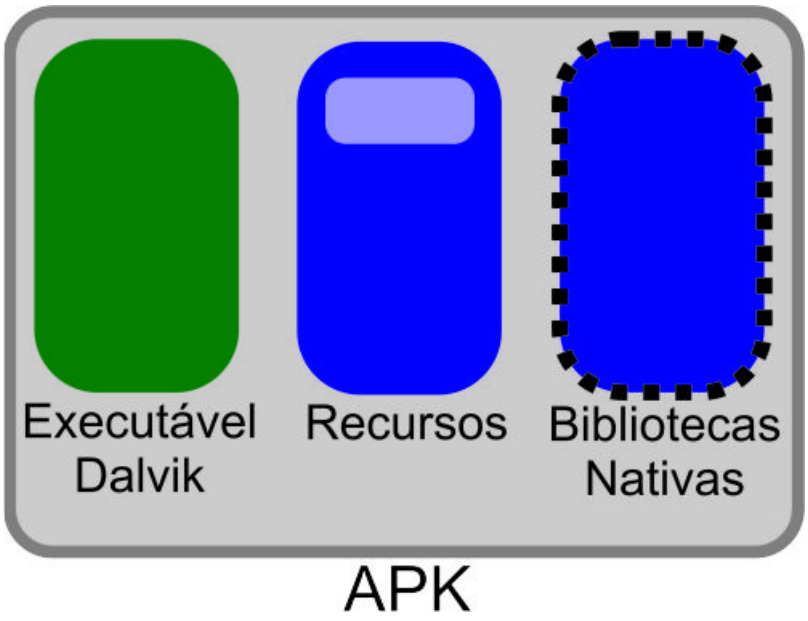

Figura 4.6: Figura ilustrando o conteúdo de um arquivo APK.

É classificado como recurso da aplicação tudo aquilo que não é código fonte, ou seja todos os arquivos de imagens, som, vídeo ou outros formatos de arquivo que a aplicação possa precisar para funcionar corretamente.

Para esta aplicação, foi criada uma biblioteca de código nativo. Como foi explicado mais acima, esta biblioteca possui uma função que permite executar uma chamada ao sistema. Nesta chamada ao sistema, iniciamos a execução de um arquivo binário compilado em código nativo para a arquitetura ARM. Este arquivo executável; denominado tether, é o encarregado de efetuar a inserção de um módulo no kernel via o comando insmod. Uma vez inserido éste módulo (que nada mais é que o próprio driver da interface de rede do celular), é possível executar o ifconfig e o iwconfig de maneira a configurar adequadamente a interface de rede.

Tanto o tether, como os utilitarios ifconfig e iwconfig encontram-se entre os arquivos de recursos no APK. Uma vez instalada, a aplicação copia estes binários para o seguinte diretório no sistema de arquivos: /data/data/com.ufrj.nelson.pg/bin.

Estes arquivos são executados assim que as threads cliente e servidor forem iniciadas se estivermos iniciando o modo ad hoc.

4.3 Ferramentas utilizadas

O SDK do sistema Android provê um conjunto bem amplo de ferramentas que nos auxiliam no desenvolvimento de aplicações. Entre estas ferramentas, as mais utilizadas para o desenvolvimento deste trabalho foram as seguintes:

Dalvik Debug Monitor Server Trata-se de uma ferramenta de depuração de código que oferece captura de telas, informações sobre threads, estado da memória, processos e eventuais logs que o desenvolvedor venha a criar.

Emulador O emulador é um aparelho celular virtual utilizado para os testes básicos da aplicação. Trata-se de uma ferramenta bastante útil e completa que nos permite abrir mão de um aparelho real em vários tipos de situações. A pesar disso, existem funcionalidades que não são oferecidas e por isso para este trabalho foi preciso o uso de aparelhos reais.

Android Esta ferramenta nos permite administrar varias instâncias de emuladores, criar e atualizar projetos e atualizar o SDK.

Android Debug Bridge O ADB é uma versátil ferramenta que nos permite acessar e manipular o estado de um emulador ou aparelho. Trata-se de um programa cliente-servidor composto por três elementos:

Um cliente Que é executado na máquina de desenvolvimento

Um servidor Também executado na máquina de desenvolvimento e encarregado da comunicação com o aparelho ou emulador.

Um processo demonio Executado como processo de fundo em cada emulador ou aparelho.

Todas estas ferramentas são agrupadas de maneira bastante útil mediante um plugin projetado para ser utilizado com Ambiente de Desenvolvimento Integrado conhecido como Eclipse. Este plugin estende as capacidades do eclipse e nos permite criar componentes, interfaces gráficas, depurar o código e até exportar o pacote compilado para uma instância do emulador ou para algum aparelho real.

Além dessas ferramentas, foi utilizado durante todo o processo de desenvolvimento o sistema de controle de versões conhecido como Git. Com ele podemos manter um registro completo de todos os progressos no desenvolvimento da aplicação.

4.4 Limitações

As manipulações descritas acima, mais especificamente a operação de inserção do módulo no kernel só pode ser realizada uma vez que o usuário executando a aplicação tenha permissão de root (superusuário). Esta necessidade pode em principio ser uma importante limitação; já que o usuario root geralmente encontra-se bloqueado por segurança nos telefones recém adquiridos, e a operação de desbloqueio convencional geralmente envolve passos complicados e até arriscados (a operação tem o potencial de deixar o celular definitivamente inoperante) que o usuário normal dificilmente estaria disposto a correr.

Por outro lado, existem hoje aplicações muito úteis que automatizam o processo e tornam um pouco mais seguro. Além disso, várias aplicações no Android Market

são publicadas com necessidade de acesso a superusuário, o que não as impede de ter altos números de instalações.

A segunda, e tal vez mais importante limitação desta aplicação encontra-se no fato que uma vez que usamos a técnica descrita acima para criar uma rede ad hoc, o aparelho perde a capacidade de utilizar a interface wifi para se conectar a internet. O que pode acabar criando uma situação desconfortável para o usuário, que teria que escolher entre usar a aplicação e navegar pela internet. O uso da rede 3G estaria ainda disponível e seria um atenuante à esta condição.

E por último, vale a pena mencionar que apesar da solução apresentada neste trabalho ter sido testada com sucesso, ela necessariamente abre mão de um recurso muito útil disponível na plataforma que é a ativação de Atividades e Serviços em tempo de execução[3]. Por ter desabilitado o serviço nativo encarregado de administrar as conexões wifi, temos que ter as threads correspondentes ao servidor e ao cliente da aplicação ligadas a um serviço que necessariamente precisa estar rodando em background, mesmo quando o servidor esta esperando mensagens e o cliente encontra-se esperando mensagens para colocar na fila.

O serviço nesta condição não consome uso do processador, mas a própria existência do processo neste caso é um desperdício de recursos de memória. A plataforma oferece funcionalidades para ativar componentes de uma aplicação tais como Serviços e Atividades sob demanda, e em resposta a ocorrências externas. E neste caso estamos abrindo mão de um recurso extremamente útil.

[3]Ver ApendiceA

Capítulo 5

Conclusão e Trabalhos Futuros

Este trabalho apresentou um protótipo funcional de aplicação mensageira utilizando uma Rede Endereçada por Interesses. O software resultante apresenta ainda vários aspectos que podem ser melhorados. Mas apesar disso, ele se comporta de maneira estável e pode ser utilizado em demonstrações.

A principal dificuldade encontrada no desenvolvimento deste trabalho foi sem dúvidas a falta de suporte oficial da plataforma Android ao wifi em modo ad hoc. Os motivos que levaram aos projetistas da plataforma (No caso a Google e seus parceiros) a desabilitar tal funcionalidade são no mínimo controversos.[7] De acordo com este post[18] o wifi no modo ad hoc foi visto como uma solução relativamente fraca em comparação com outras tecnologias emergentes como o Wi-Fi Direct[19].

A lógica do protocolo foi implementada no nível da aplicação por questão de simplicidade. Uma proposta para ampliar este trabalho, com o objetivo de melhorar a sua eficiência e mitigar algumas de suas limitações consiste na implementação da lógica do protocolo na camada de rede. Isto poderia ser feito por meio de um módulo do kernel que contenha as principais regras do Modelo de Comunicação endereçada por Interesses. Este módulo seria então incluído pela aplicação em tempo de execução.

Com isso ganharíamos não somente em eficiência, ja que o protocolo estaria sendo executado em código nativo, mas também seria permitido o uso da interface wireless tanto para as comunicações da rede endereçada por interesses como para o acesso a

APs normais conectados à internet.

Referências Bibliográficas

[1] *Measuring the Information Society*, Report 1.01, International Telecommunication Union, CH-1211 Geneva, Switzerland, 2010.

[2] PERKINS, C., BELDING-ROYER, E., DAS, S., "Ad hoc On-Demand Distance Vector (AODV) Routing", RFC 3561 (Experimental), Jul. 2003.

[3] ZYGMUNT HAAS, JOSEPH Y. HALPERN, L. L., "Gossip-Based Ad Hoc Routing", *Infocom*, v. 3, 2002.

[4] CLAUSEN, T., JACQUET, P., "Optimized Link State Routing Protocol (OLSR)", RFC 3626 (Experimental), Oct. 2003.

[5] GRANJA, R. S., *Protocolos para redes de comunicação ad hoc endereçadas por interesses*. M.Sc. dissertation, COPPE, Junho 2010.

[6] "Android", http://developer.android.com, (Acesso em 12 de Abril 2011).

[7] "wifi - support ad hoc networking", http://code.google.com/p/android/issues/detail?id=82, (Acesso em 12 de Abril 2011).

[8] "J2ME platform Limitations and ways around these limitations", http://www.mangospring.com/w/developers/j2me-platform-limitations-and-ways-around-these-limitations/, (Acesso em 22 de Abril 2011).

[9] MEIER, R., *Professional Android 2 Application Development*. 10475 Crosspoint Boulevard. Indianapolis, IN 46256, Wiley Publishing Inc., 2010.

[10] "Google buys Android for its mobile arsenal", http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm, (Acesso em 22 de Abril 2011).

[11] HASHIMI, S. Y., KOMATINENI, S., MACLEAN, D., *Pro Android 2*. Apress, 2010.

[12] SILVA, V., *Pro Android Games*. Apress, 2009.

[13] KUROSE, J. F., ROSS, K. W., *Computer Networking*. 501 Boylston Street. Suite 900, Boston, MA.USA, Pearson, 2009.

[14] DUTRA, R. C., GRANJA, R. S., AMORIM, C. L., *et al.*, "REPI: Rede de comunicação Endereçada Por Interesses", *VI Workshop de Redes Dinâmicas e Sistemas Peer-to-Peer. Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, , 2010.

[15] DUTRA, R. C., AMORIM, C. L., *Modelo de Comunicação Endereçada por Interesses*, Report ES-733, COPPE, Março 2010.

[16] "Ad-hoc Wifi in Android", http://szym.net/2010/12/adhoc-wifi-in-android/, (Acesso em 4 de Maio 2011).

[17] "Android Wifi Tether", http://code.google.com/p/android-wifi-tether/, (Acesso em 5 de Maio 2011).

[18] "Android AdHoc IBSS Wifi support", https://groups.google.com/forum/#!topic/android-platform/tLLspmSySbY, (Acesso em 7 de Maio 2011).

[19] ALLIANCE, W.-F., *Wi-Fi Certified Wi-Fi Direct*, Report, Wi-Fi Alliance, Outubro 2010.

Apêndice A

Atividades, Serviços e componentes do sistema Android

A.1 Atividades

No contexto da API do sistema Android, uma atividade é um componente fundamental que oferece uma interface gráfica e foco em uma única tarefa que o usuário este realizando em um dado instante.

Uma atividade geralmente possui uma interface que utiliza a tela completa do aparelho, a pesar de ser possível criar atividades que ocupem somente uma parcela da tela. Uma aplicação geralmente é composta por uma ou várias atividads ligadas fracamente umas com as outras por meio de componentes denominados Intents

As atividades possuem um ciclo de vida estabelecido e bem documentado. Este ciclo de vida, a modo de máquina de estados, apresenta várias etapas nas quais a Atividade pode encontrar-se visivel, parcialmente visivel, ou invisível e mesmo assim continuar existindo. As atividades que encontram-se no estado visível são aquelas com as quais o usuário está interagindo, e portanto são as mais resguardadas pelo sistema. Atividades parcialmente visíveis possuem uma prioridade um pouco mais baixa, mas aínda assim alta. As atividades invisíveis tem uma prioridade baixa, e podem ser candidatas a terminação pelo sistema, caso haja necessidade de liberar mais recursos.

Métodos específicos são executados nas transições de estados e podem ser sobrecarregados pelo programador, dando assim um maior controle sobre o comportamento da atividade.

A.2 Intents

Os *Intents* podem ser tidos como a representação dentro do programa das mensagens sendo trocadas entre os diferentes componentes do sistema. Eles oferecem uma grande versatilidade, ja que podem ser explícitos, em cujo caso o componente que os cria especifica detalhadamente para qual componente o *Intent* está sendo destinado. Ou podem também ter uma natureza implícita, em cujo caso o componente que os cria somente especifica as características gerais do que precisa, e deixa o ambiente de execução encontrar um componente ideal para cumprir com a tarefa asignada.

Por exemplo, uma aplicação de contatos pode administrar uma lista de nomes e endereços de email dos contatos do usuário. A aplicação porém não possui mais funcionalidades além de mostrar seus dados para o usuário. Ela pode no entanto, oferecer a possibilidade de enviar um email para cada contato que apresenta. Neste caso a atividade que mostra o contato, irá criar um *Intent* e especificar nele que precisa de alguma outra atividade que possa enviar email, passando como informação o nome e o endereço de email do contato. Se o *Intent* for criado no modo implícito, o sistema irá decidir qual aplicação de email usar. No caso de haverem múltiplas aplicações com essa capacidade, uma lista é mostrada ao usuário, que deve escolher a mais apropriada.

A.3 Serviços

Os serviços são básicamente atividades sem interface gráfica encarregados de alguma tarefa que deve ser efetuada em background. O exemplo clássico de uso para um serviço é uma reprodutor de música, que deve permanecer rodando mesmo depois que sua atividade for destruida.

É importante ressaltar que mesmo assim, os serviços continuam sendo executados na thread principal do processo. Ou seja, se tivermos uma aplicação que faz uso de um serviço para alguma atividade de fundo e possui uma atividade principal, este serviço deve explícitamente disparar threads secundárias se deseja efetuar operações potencialmente custosas. Se isso não for feito, estas operações podem vir a interferir com a renderização da interfáce gráfica, empobrecendo assim a experiência do usuário.

A.4 O Runtime

Uma aplicação sendo executada dentro do sistema Android portanto não possui toda a libertade

que uma aplição convencional para plataformas mais robustas (PCs) geralmente possui. No sistema Android somente os componentes que estão interagindo diretamente com o usuário possuem garantías de não serem exterminados. Todos os outros devem ser projetados de forma a poder ser eliminados em qualquer momento. Pois caso haja necessidade, o sistema irá destruindo componentes para liberar recursos. Estos componentes destruidos podem ou não ser restaurados posteriormente de maneira automática pelo sistema; isso depende das suas características.

Outra grande diferença do ambiente de execução é o fato dos diferentes componentes poderem ser ativados de forma assíncrona por meio de *Intents*. Isso quer dizer que podemos ter uma Atividade ou Serviço por exemplo que sejam accionadas por um evento externo, a chegada de um SMS por exemplo. O componente desta maneira encontrava-se desativado, mas passa a existir como um processo e se for bem comportado deixa a cena assim que terminar a sua tarefa.

Esta possibilidade permite evitar o desperdício de recursos por meio de componentes que de outra maneira deveríam ficar "dormentes"esperando a ocorrência de um evento.

Apêndice B

Interface Gráfica da Aplicação

B.1 Interface Gráfica da Aplicação Mensageira

B.1.1 Aplicação na versão Desktop

A continuação descrevemos brevemente a interface gráfica das versões desktop e móvel da aplicação mensageira.

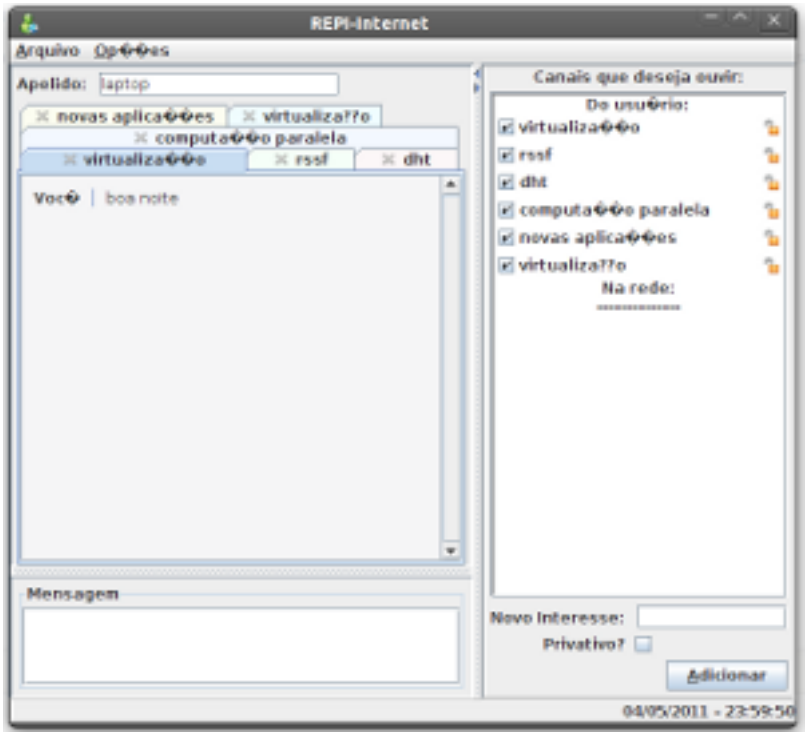

Figura B.1: Captura de tela mostrando a interface gráfica da aplicação mensageira em seu formato para desktop.

Como podemos apreciar na Figura B.1 a aplicação original para desktop que inspirou este trabalho apresenta um painel principal onde as mensagens correspondentes a cada interesse são mostrados. Em cima deste painel temos uma série de abas mostrando o interesse atual e cujos tamanhos são adaptados dinamicamente.

Finalmente a direita, temos um painel vertical onde aparece a lista dos interesses atualmente presentes na rede. O usuário pode marcar ou desmarcar os interesses, e também pode adicionar mais interesses.

B.1.2 Aplicação na versão Android

Figura B.2: Captura de tela da interface gráfica da aplicação.

O reduzido espaço disponível na tela de um aparelho móvel exige que o espaço seja aproveitado da maneira mais eficiente possivel. Por isso foi preciso projetar uma interface bem diferente daquela presente na versão original da aplicação.

É importante manter a simplicidade no design para não sobrecarregar a experiência do usuário. Porém queremos apresentar todas as funcionalidades presentes

na versão original da aplicação.

A solução encontrada, aparece na Figura B.2. No canto superior da tela aparecem os interesses em uma lista deslizavel horizontal. O grande espaço vazio no meio é reservadp para as mensagens, que vão sendo acumuladas de baixo para cima como em uma janela de chat normal. Se o número de mensagens for muito grande, a lista pode ser deslizada para baixo mostrando as mensagens mais antigas.

A lista de interesses permanece estática no topo da tela. Para cada interesse selecionado, a lista de mensagens é atualizada correspondentemente.

Uma opção, no final da lista de interesses permite ao usuário adicionar interesses à sua lista.

Para ativar o modo ad hoc, alterar as preferências do usuário e acessar mais funcionalidades da aplicação, a plataforma oferece um sistema de menus padrão simples, porém bastante flexivel. Apertando a tecla do menu o usuário então e apresentado com a tela de um menu. A Figura B.3 mostra o menu que é apresentado ao usuário.

Figura B.3: Captura de tela da interface gráfica da aplicação mostrando o menu principal da aplicação.

Apêndice C

Encadernação do Projeto de Graduação

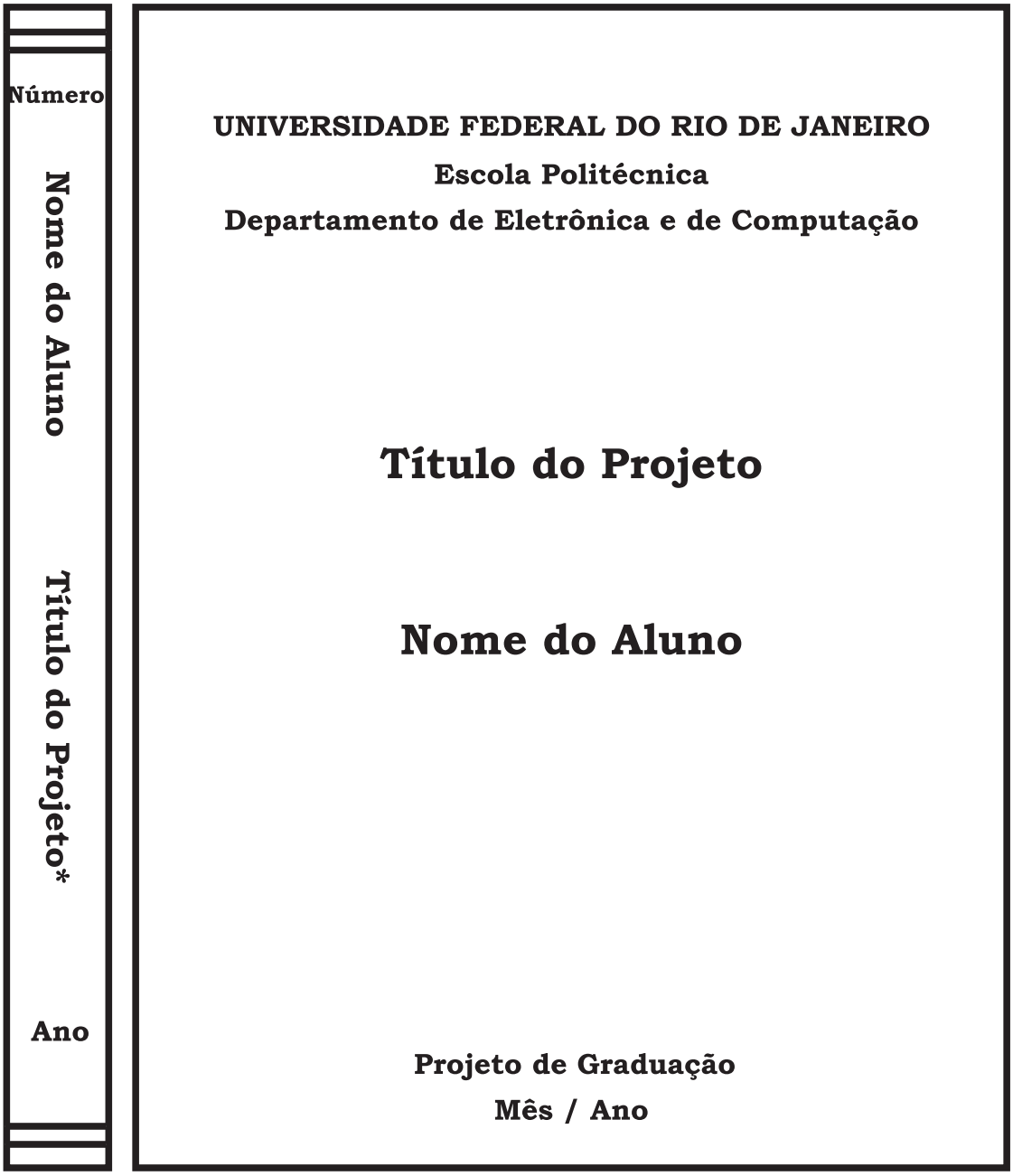

Figura C.1: Encadernação do projeto de graduação.