UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DA COMPUTAÇÃO

RODRIGO JAUREGUY DOBLER

Injeção de Falhas de Comunicação para Validação de Aplicações no Ambiente Android

Trabalho de Diplomação.

Profa. Dr. Taisy Silva Weber
Orientador

Prof. Dr. Sérgio Luis Cechin
Co-orientador

Porto Alegre, dezembro de 2010.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-Reitor: Prof. Rui Vicente Oppermann
Pró-Reitora de Graduação: Profa. Valquiria Link Bassani
Diretor do Instituto de Informática: Prof. Flávio Rech Wagner
Coordenador do ECP: Prof. Gilson Inácio Wirth
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

AGRADECIMENTOS

Agradeço aos meus orientadores, professores Sérgio e Taisy por toda a ajuda, idéias e disposição durante o desenvolvimento deste trabalho.

Gostaria de agradecer também aos colegas Bruno Fiss, Heinrich Marmitt e Jorel Settin pela ajuda, conversas e argumentações durante a elaboração do trabalho.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

FIRMAMENT	Fault Injection Relocatable Module for Advanced Manipulation and Evaluation of Network Transports
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
DECnet	Digital Equipment Corporation networking protocol
ARP	Address Resolution Protocol
API	Application Programming Interface
SGL	Scalable Graphics Library
SSL	Secure Sockets Layer
JIT	Just In Time
OpenGL	Open Graphics Library
SDK	Software Development Kit
XML	eXtensible Markup Language
ADT	Android Development Tools
ADB	Android Debug Bridge
AVD	Android Virtual Device
IP	Internet Protocol
IGMP	Internet Group Management Protocol
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
MSN	Microsoft Network Messenger

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Dispositivos móveis como celulares, palm tops e smartphones estão cada vez mais presentes em nossas vidas. Eles estão evoluindo muito depressa e, a cada nova versão, os aparelhos são lançados com muito mais recursos. Isto proporciona novos horizontes para os desenvolvedores de software.

Hoje, devido a iniciativas de alguns fabricantes, muitas empresas e desenvolvedores independentes estão lançando programas para celulares. Porém, nem sempre são tomados cuidados com relação à tolerância a falhas e, desse modo, muitas aplicações podem apresentar problemas.

Assim, neste trabalho será portada uma ferramenta de injeção de falhas, Firmament, a qual foi desenvolvida para o sistema operacional Linux, para que ela possa ser utilizada no ambiente Android. Também será feita uma avaliação de aplicações com esta ferramenta, para mostrar como falhas de comunicação podem afetar o funcionamento de uma aplicação.

Esta ferramenta irá permitir que seja feita injeção de falhas na troca de mensagem sobre o protocolo IP de algumas aplicações executando no emulador do Android. Isto irá permitir que se possa analisar o comportamento dessas aplicações na presença de falhas e, deste modo, verificar se são ou não tolerantes a falhas.

Palavras-Chave: tolerância a falhas, injeção de falhas, Firmament, Android.

Communication Fault Injection for Validation of Applications in Android Environment

ABSTRACT

Mobile devices such as cell phones, palmtops and smartphones are every day more present in our lives. They are evolving very quickly, and at each new version, the devices are released with a lot more resources. This provides new horizons for software developers.

Nowadays, due to the initiative of some manufacturers, many companies and independent developers are launching programs to mobile phones. However, not always the proper care is taken with respect to fault tolerance and then, many applications can present problems.

Thus, this work will present the port of a fault injection tool, Firmament, which was developed for the Linux operating system, so it can be used in the Android environment. Also, some applications will be evaluated with this tool to show how communication faults can affect the behavior of an application.

This tool will allow fault injection in the exchange of messages over the IP protocol on some applications running on the Android emulator. This will allow the analysis of the behavior of these applications in the presence of faults and thus determine whether they are fault tolerant.

Keywords: fault tolerance, fault injection, Firmament, Android.

1 INTRODUÇÃO

Os dispositivos móveis trouxeram muitos benefícios para as nossas vidas. Os aparelhos celulares, por exemplo, permitem que nós possamos nos comunicar com outras pessoas ou acessar informação de quase qualquer lugar em que estejamos. Estes aparelhos melhoraram muito nos últimos anos, o que possibilitou a criação de excelentes sistemas operacionais para eles.

Alguns fabricantes, como a Apple e a Google, lançaram ferramentas de desenvolvimento de software, para permitir que outras empresas e desenvolvedores independentes possam desenvolver aplicativos para as respectivas plataformas. Porém, apenas a Google tornou público o código fonte do Android e não impôs restrições ao desenvolvimento de aplicações. Essa estratégia da Google incentivou o uso do Android nos celulares e hoje um grande número de fabricantes o adota como sistema operacional de seus aparelhos, como Motorola, Samsung, Sony Ericsson, LG e outros. Deste modo, o Android começou a ganhar grande destaque no mercado de aparelhos móveis.

De acordo com a recente pesquisa publicada pela empresa Canalys (CANALYS), nos Estados Unidos, o maior mercado de smartphones do mundo, os dispositivos com Android representaram juntos 34% do mercado norte-americano no 2º trimestre de 2010. No mundo todo, nesse mesmo período, o número de smartphones que utilizam o Android cresceu 886% e isso se deveu as operadoras de telefonia celular, as quais estão promovendo amplamente os dispositivos Android e também ao grande número de aplicações feitas para este sistema, sendo muitas delas de excelente qualidade e sem custo algum para o usuário. Esse grande número de aplicações desenvolvidas é realmente um grande atrativo, contudo, nem todas elas possuem implementações consistentes para detectar e corrigir erros e, desta forma, quando falhas acontecem, elas podem causar erros na aplicação, causando transtornos para o usuário.

Segundo (ARLAT, 2003) e (CARREIRA, 1998), uma falha é um fenômeno que pode causar um desvio em um componente de hardware ou software a partir de suas funções pretendidas. A injeção de falhas, por acelerar a ocorrência de erros e falhas, torna-se importante para testar os mecanismos de tolerância a falhas de um sistema. Assim, a injeção de falhas pode ser aplicada em um modelo de simulação do sistema tolerante a falhas alvo ou em uma implementação de hardware e software. A simulação baseada em injeção de falhas é desejada, pois ela pode prover checagens no início do projeto de mecanismos de tolerância a falhas.

Desse modo, seria necessário testar as aplicações desenvolvidas, injetando-se falhas, para que seja possível analisar como as aplicações irão se comportar. Isto permitiria fazer uma prevenção nas aplicações para que elas fossem capazes de contornar um estado de falha sem causar maiores problemas ao usuário ou sistema operacional. Para que isso seja viável, é necessário que as aplicações tenham alta dependabilidade.

Avizienis (AVIZIENIS, 2004) define dependabilidade, como sendo a habilidade de prestar serviços em que se pode justificadamente confiar. Este conceito engloba uma série de atributos como disponibilidade, confiabilidade, facilidade de manutenção, segurança funcional crítica, testabilidade, entre outros.

Normalmente, utilizam-se ferramentas de injeção de falhas para se poder testar um sistema eficientemente e, deste modo, encontrar soluções para os eventuais problemas, garantindo desta forma um aumento na dependabilidade do sistema. Hoje não existem muitas ferramentas de injeção de falhas e, destas, a maioria não está mais disponível para uso.

Falhas de comunicação são relevantes em ambientes móveis, pois eles dependem do envio e recebimento de mensagens para poderem se comunicar. Deste modo, o uso de uma boa ferramenta de injeção de falhas torna-se essencial para que seja possível fazer testes eficientes e, desse modo, obter conclusões corretas.

Este trabalho irá mostrar um passo a passo de como se fazer o porte para o ambiente Android de uma ferramenta de injeção de falhas, o Firmament, a qual foi desenvolvida para o sistema operacional Linux. Além disso, serão feitos testes em um sistema cliente-servidor para verificar o correto funcionamento da ferramenta e em aplicações reais do Android para avaliar como essas aplicações se comportam na presença de falhas de comunicação.

2 ESPECIFICAÇÃO DO PROJETO

Aqui será feita uma explicação das ferramentas utilizadas no desenvolvimento do trabalho.

2.1 Netfilter

O Netfilter (RUSSEL; WELTE, 2002) é uma ferramenta para manipulação de pacotes que define pontos específicos, ou ganchos, na pilha de protocolos onde funções de callback podem ser registradas. Quando os pacotes alcançam esses pontos, eles são passados para essas funções, as quais possuem acesso completo ao seu conteúdo. Assim, elas podem processar os pacotes e decidir se os mesmos podem passar pela pilha de protocolos ou serem descartados.

O Netfilter permite o registro de funções de callback para as famílias de protocolos IPv4, IPv6, DECnet e ARP. Os ganchos estão disponíveis em diversos pontos de interesse: no recebimento e envio de pacotes, no encaminhamento de pacotes roteados e na entrega e recebimento de pacotes para os níveis superiores.

2.2 Firmament

É uma ferramenta de injeção de falhas, desenvolvida por Roberto Jung Drebes (DREBES, 2006) (SIQUEIRA, 2009), a qual visa permitir a especificação de cenários de falhas de comunicação complexos, com baixa intrusividade, para o teste de protocolos de comunicação e sistemas distribuídos.

Esta ferramenta pode ser utilizada para aplicações de rede, nas quais o uso de implementações reais pode evidenciar deficiências da implementação que possivelmente escapariam à simulação. Ela também pode ser usada para o estudo de novos protocolos, pois a ferramenta permite colocar o sistema em estados incomuns ou de difícil reprodução.

O Firmament utiliza a arquitetura da interface de programação Netfilter, a qual está disponível a partir da versão 2.4 do kernel do Linux. Por utilizar apenas interfaces de programação de alto nível do kernel, a ferramenta é independente da arquitetura da máquina e, desta forma, pode ser utilizada em servidores, desktops e dispositivos embarcados que utilizem o Linux.

2.2.1 Faultlets

Um faultlet é uma aplicação que emula o comportamento de falha. Ele é executado

sobre cada pacote que cruza um dos fluxos de transporte, podendo duplicar, atrasar, descartar ou modificar o conteúdo do pacote.

O fluxo de execução de um faultlet pode ser alterado a partir de dados lidos do pacote, permitindo assim, a criação de estruturas avançadas de controle, como laços e desvios, tornando Firmament apropriado ao teste de qualquer protocolo.

Os faultlets são configurados independentemente para os fluxos de entrada e saída do protocolo IP, e a sua execução é feita pela máquina virtual Firm_vm.

2.2.2 Firm_vm

A máquina virtual Firm_vm é o módulo responsável pela interpretação e processamento das instruções que especificamos cenários de falhas. Firm_vm trabalha sobre os fluxos de entrada e saída dos protocolos IPV4 e IPV6.

2.3 Android

O Android (ANDROID) é um ambiente de desenvolvimento para celulares, inicialmente desenvolvido pela Android Inc., empresa que foi adquirida pela Google em 2005, e depois foi lançado oficialmente pela Open Handset Alliance, um grupo formado por 78 empresas de tecnologia que atuam em diferentes áreas como operadoras de telefonia, empresas de software, empresas de semicondutores, empresas de comercialização e fabricantes de celulares. Este grupo foi criado com o objetivo de desenvolver padrões abertos para dispositivos móveis, cujo primeiro produto foi o Android.

O ambiente de desenvolvimento Android está disponível sob uma licença de software livre/código aberto desde o seu lançamento, em 21 de outubro de 2008. O código fonte do Android foi disponibilizado sob a licença Apache, segundo a qual, os fornecedores podem adicionar extensões proprietárias sem ter que submetê-las a comunidade do código aberto.

Atualmente, estão disponíveis 4 versões do Android, que são lançadas com atualizações e correções de bugs com relação a versão anterior. Elas são citadas a seguir:

- Cupcake (versão 1.5): baseada no kernel 2.6.27 do Linux e lançada em 30 de abril de 2009;
- Donut (versão 1.6): baseada no kernel 2.6.29 do Linux e lançada em 15 de setembro de 2009;
- Eclair (versão 2.0/2.1): baseada no kernel 2.6.29 do Linux e lançada em 26 de outubro de 2009;
- Froyo (versão 2.2): baseada no kernel 2.6.32 do kernel do Linux e lançada em 20 de maio de 2010;

Estas versões estão distribuídas no mercado de acordo nas seguintes proporções, mostradas na figura 2.1 e na tabela 2.1, de acordo com o número de dispositivos Android que acessaram o Android Market, num período de 14 dias de coleta de dados, terminado em 1º de Novembro de 2010 de acordo com o site (ANDROID-PLATFORM):

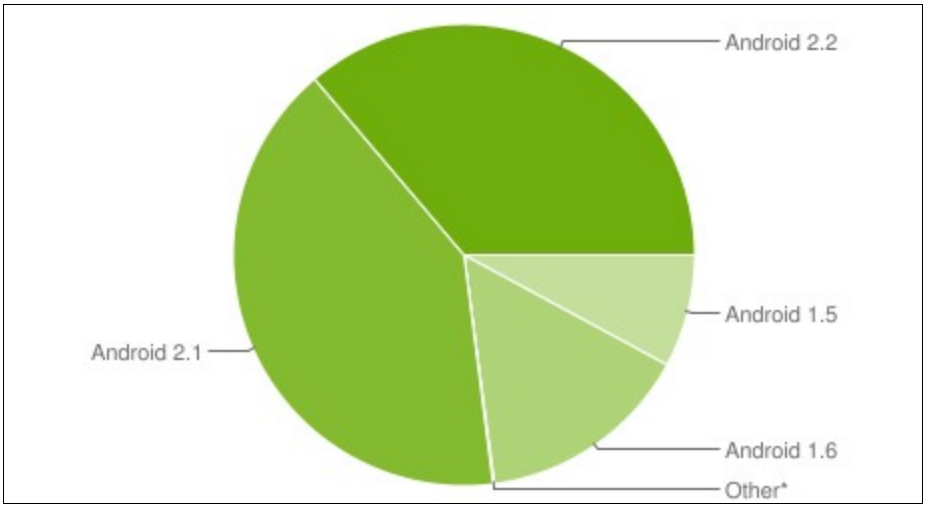

Figura 2.1: Participação no mercado das diferentes versões do Android

Na figura 1, o campo “others” corresponde a 0,1% dos dispositivos executando versões obsoletas do Android.

Tabela 2.1: Percentagem das diferentes versões do Android no mercado

Platform	API Level	Distribution
Android 1.5	3	7.9%
Android 1.6	4	15.0%
Android 2.1	7	40.8%
Android 2.2	8	36.2%

O sistema operacional do Android (ANDROID-SISTEM) foi construído com base numa versão modificada do kernel 2.6 do Linux, a qual foi ajustada pela Google fora da árvore principal do kernel do Linux. O Android não tem um sistema Window X nativo e nem suporta o conjunto completo de bibliotecas padrão GNU e isso torna difícil o porte de aplicações ou bibliotecas Linux/GNU existentes para o Android. Contudo, o suporte para o sistema Window X é possível.

A pilha de software do sistema operacional consiste de aplicações Java executando em um objeto com base em Java framework, orientado a aplicação, em cima de bibliotecas Java rodando na máquina virtual Dalvik com compilação JIT. Bibliotecas escritas em C incluem o gerenciador de superfície, quadro de mídia OpenCore, sistema de gestão de banco de dados relacional SQLite, API gráfica OpenGL ES 2.0 3D, mecanismo de layout WebKit, motor gráfico SGL, SSL, e libc Bionic.

2.3.1 Android SDK

O Software Development Kit, SDK, inclui um conjunto abrangente de ferramentas de desenvolvimento. Estas incluem um debugger, bibliotecas, um emulador de telefone (com base no QEMU), documentação, exemplos de código e tutoriais. Atualmente as plataformas suportadas incluem o desenvolvimento de computadores de arquitetura x86

executando Linux (qualquer distribuição moderna desktop Linux), Mac OS X 10.4.8 ou posterior, Windows XP ou Vista. Os requisitos incluem o Java Development Kit, o Apache Ant e o Python 2.2 ou posterior. O ambiente de desenvolvimento integrado, IDE, apoiado oficialmente é o Eclipse (versão 3.2 ou superior), utilizando o plugin Android Development Tools, ADT, embora os desenvolvedores possam utilizar qualquer editor de texto para criar arquivos Java e XML e, em seguida, usar ferramentas de linha de comando para criar, construir e depurar os aplicativos do Android, bem como controles ligados a dispositivos Android, como por exemplo, provocar uma reinicialização, instalar pacotes de software remotamente.

2.3.2 Máquina Virtual Dalvik

Dalvik é o nome da máquina virtual utilizada pelo Android e como ele, ela também é um software open-source e foi originalmente escrita por Dan Bornstein.

Antes da execução das aplicações no Android, elas são convertidas para o formato Dalvik Executable, .dex, o qual foi projetado para ser adequado para sistemas que são limitados em termos de memória e velocidade de processador.

A máquina virtual Dalvik é baseada numa arquitetura de registradores, sendo dessa forma diferente das máquinas virtuais Java, as quais são máquinas de pilhas. Geralmente, as máquinas baseadas em pilhas devem usar instruções para carregar dados na pilha e manipular esses dados e, portanto, exigem mais instruções do que as máquinas de registradores para implementar o mesmo nível de código. Contudo, as instruções em máquinas de registradores devem codificar os registros de origem e destino e, desse modo, tentem a serem maiores. Esta diferença é significativa para os interpretadores da máquina virtual, pois para eles, o envio de códigos de operação tende a ser custoso, juntamente com outros fatores igualmente relevantes para a compilação em tempo de execução.

Uma ferramenta chamada dx é utilizada para converter alguns, mas não todos, os arquivos Java .class no formato .dex. Múltiplas classes são incluídas em um único arquivo .dex e strings duplicadas e outras constantes utilizadas em vários arquivos .class são incluídas somente 1 vez para economizar espaço. O bytecode de Java é também convertido para um conjunto de instruções alternativo utilizado pela máquina virtual Dalvik.

Os arquivos executáveis Dalvik podem ser modificados novamente quando instalados em um dispositivo móvel. Para se obter otimizações adicionais, ordem de bytes podem ser trocadas em alguns dados, estruturas de dados simples e bibliotecas de funções podem ser ligados em linha, e objetos de classe vazia pode ser curto-circuitados.

Por ser otimizada para as necessidades de pouca memória, Dalvik tem algumas características específicas que a diferenciam de outras máquinas virtuais padrão:

- A máquina virtual foi diminuída para ocupar menos espaço;
- A constante pool foi modificada para utilizar somente índices de 32-bit para simplificar o interpretador;
- Ela usa o seu próprio bytecode e não o do Java;

Além disso, a Dalvik foi concebida de modo que um dispositivo pode executar

várias instâncias da máquina virtual de forma eficiente.

2.3.3 Android ADB

O Android ADB, Android Debug Bridge, é uma ferramenta de depuração que vem integrada ao SDK, e que permite gerir o estado de uma instância do Emulador ou de dispositivos ligados através de um cabo USB.

Inclui um daemon que executa em segundo plano, podendo redirecionar conexões de socket entre o host e o emulador ou dispositivo, assim como uma interface de linha de comando pela qual se pode controlar o daemon, o emulador e o dispositivo. Além disso, ainda pode realizar atualizações de código (como aplicações e atualizações do próprio Android), executar comandos no shell do dispositivo, gerir o redirecionamento de portas e copiar arquivos de e para um emulador ou dispositivo.

2.3.4 Arquitetura do Android

A arquitetura do Android (ANDROID-ARCHITECTURE) é dividida em 5 camadas, as quais são: as Aplicações, o Framework de Aplicações, as Bibliotecas, o Android Runtime e o kernel do Linux. Cada camada é descrita a seguir em mais detalhes:

- Aplicações: o Android possui um conjunto de aplicativos básicos, como um cliente de e-mail, SMS programa, calendário, mapas, navegador, contatos e outros. Todos os programas são escritos utilizando a linguagem de programação Java. Uma aplicação Android pode ser construída utilizando-se quatro blocos principais de construção:
  - Activity (Atividade);
  - Intent Receiver (Receptor de Intenção);
  - Service (Serviço);
  - Content Provider (Provedor de Conteúdo).

  Uma aplicação Android deverá ser feita com um ou mais desses blocos de construção.
- Framework de Aplicações: por fornecer uma plataforma de desenvolvimento aberta, o Android oferece aos desenvolvedores a capacidade de criar aplicações extremamente ricas e inovadoras. Os desenvolvedores podem aproveitar o hardware do dispositivo, as informações de localização de acesso, execução de serviços de fundo, definir alarmes, notificações para adicionar a barra de status, etc.

  Os desenvolvedores têm acesso total às mesmas APIs do quadro utilizado pelos aplicativos do núcleo. A arquitetura do aplicativo é projetada para simplificar a reutilização dos componentes. Desse modo, qualquer aplicação pode publicar suas capacidades e qualquer outra aplicação pode então fazer uso dessas capacidades (sujeito a restrições de segurança impostas pelo quadro). Este mesmo mecanismo permite que componentes sejam substituídos pelo usuário.
- Bibliotecas: o Android possui um conjunto de bibliotecas C/C++ utilizadas por diversos componentes do sistema Android. Estas capacidades são expostas a desenvolvedores através da estrutura de aplicativos do Android.

- Android Runtime: o Android possui um conjunto de bibliotecas que fornece a maioria das funcionalidades disponíveis nas principais bibliotecas da linguagem de programação Java.

  Cada aplicação Android executa em seu próprio processo, com sua própria instância da máquina virtual Dalvik. Dalvik foi escrita de forma que um dispositivo pode executar várias instâncias eficientemente. A máquina virtual Dalvik executa os arquivos em Dalvik executável, .dex, formato que é otimizado para um consumo mínimo de memória. A máquina virtual é baseada em registradores, e executa classes compiladas por um compilador da linguagem Java que foram transformadas para o formato dex, através da ferramenta dx.

  A Dalvik invoca o kernel do Linux para a funcionalidade subjacente como encadeamento e de baixo nível de gerenciamento de memória.

- Kernel do Linux: o Android se baseia na versão 2.6 do kernel do Linux para o sistema central de serviços, como segurança, gerenciamento de memória, gerenciamento de processos, rede de pilha e modelo de driver.

  O kernel também atua como uma camada de abstração entre o hardware e o restante da pilha de software;

  A figura 2 mostra uma imagem da organização da arquitetura do Android:

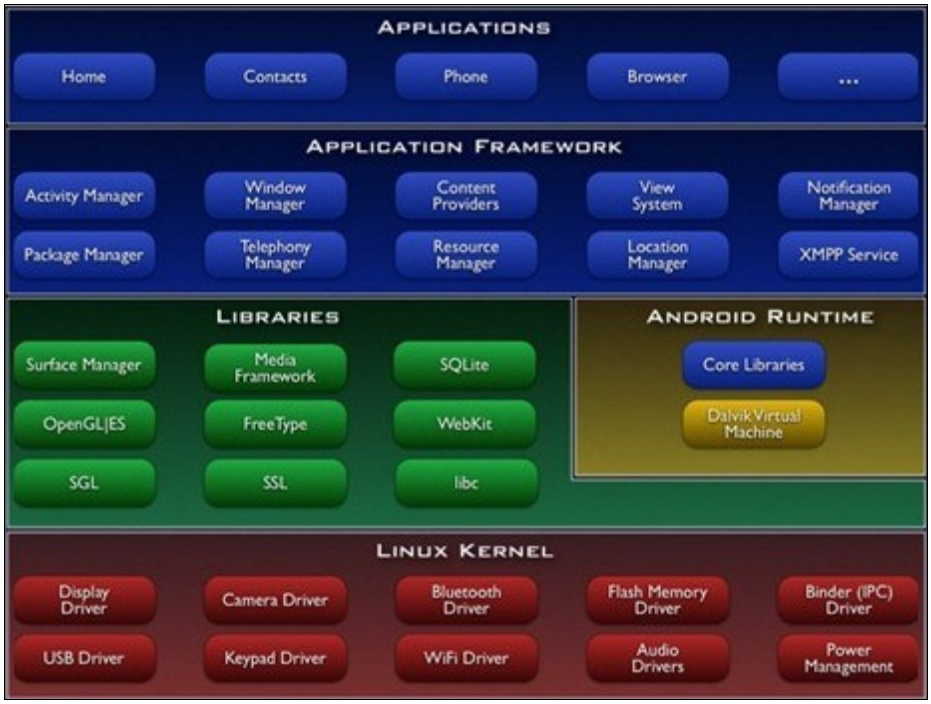

Figura 2.2: Divisão da arquitetura do Android em 5 camadas

3 DESENVOLVIMENTO

Aqui será mostrado o passo a passo do porte da ferramenta Firmament para o Android, e a recompilação do kernel e passagem dos arquivos do Firmament para dentro do emulador do Android.

3.1 Obtenção dos arquivos fontes do Android

Para que se possa construir os arquivos fontes do Android, precisa-se utilizar os sistemas operacionais Linux ou Mac OS. O sistema operacional Windows não é suportado atualmente. Então, foi utilizada uma distribuição do Linux, Ubuntu 9.10 Karmic Koala, num computador com processador Opteron 180 com 2GB de memória RAM no desenvolvimento deste trabalho.

Para obter-se os arquivos fonte e depois compilá-los, precisa-se seguir os passos do tutorial mostrado no próprio site do Android. O tutorial encontra-se na página (ANDROID-SOURCE).

3.2 Recompilando o kernel do Android

O kernel do Android necessita ser recompilado para que se possa ativar o módulo do Netfilter, o qual está desativado por padrão. A ativação deste módulo torna-se necessária para que o Firmament, ferramenta de injeção de falhas, possa ser utilizado para executar os faultlets nos pacotes das mensagens que irão trafegar pela rede.

Também será necessário ativar o carregamento de módulos, para que o módulo recompilado do Firmament possa ser executado no emulador do Android e possa, assim, realizar a injeção de falhas de acordo com o faultlet selecionado.

No site do Android, na seção do SDK, será feito o download do SDK para Linux na página (ANDROID-SDK) e, depois se segue o link (ANDROID-SDK-INSTALLING) para se obter o tutorial de instalação do SDK, o qual descreve como obter plataformas e outros componentes.

Depois este link (ADT-PLUGIN) ensina como instalar o Android Development Tools plugin para o Eclipse e esse outro link (HELLO-WORLD), descreve como criar um Android Virtual Devices e rodar o emulador.

Para se obter e recompilar o kernel do Android, ativando-se os módulos necessários, seguem-se os passos explicados nos sites (CROSS-COMPILATION).

Esse kernel recompilado é que será utilizado no emulador do Android para que seja possível carregar o Firmament e executar os testes de injeção de falhas no emulador do

Android.

3.3 Recompilando o Firmament para o Android

Esta parte foi conduzida utilizando-se como referência o artigo (ACKER, 2010), o qual foi muito útil para compilar os arquivos firm_asm e msa_mrif, os quais pertencem ao módulo do Firmament. A solução apresentada no site (LOADABLE-kernel-MODULE) serviu de apoio para realizar a compilação cruzada da máquina virtual firm_vm. A figura 3.1 mostra como isso deve ser feito:

O primeiro passo é compilar os arquivos **firm_asm** e **msa_mrif**. Obter o Firmament versão 2.6.29 e extrair em uma pasta no diretório **home**. Os dois arquivos citados estão dentro da pasta **asm**, dentro da pasta principal do **Firmament**.

1. Agora, entrar no diretório **mydroid**, criado de acordo com o tutorial para obtenção dos arquivos fontes do Android, digitando no terminal os comandos **$cd** e depois **$cd mydroid**. Dentro desse diretório, encontramos várias pastas e dentre elas a pasta **external**. Essa pasta contém os programas que estarão disponíveis no sistema.

2. Assim, entrar na pasta **external** com o comando **$cd external** e criar uma pasta, com o nome de **myapps** executando **$mkdir myapps** no terminal. A seguir, copiar uma das pastas originais que existem dentro da pasta **external**, como a **ping** por exemplo. Depois se renomeia a pasta **ping**, a qual foi copiada para dentro da pasta **myapps**, para **firm_asm**.

3. O próximo passo é entrar na pasta **firm_asm** e excluir todos os outros arquivos, deixando apenas o arquivo **Android.mk** dentro da pasta. Voltar para a pasta **asm** dentro do **Firmament** e copiar os seus arquivos fontes para dentro da pasta **firm_asm**.

4. A seguir, editar o arquivo **Android.mk** e alterar as variáveis **LOCAL_SRC_FILES e LOCAL_MODULE** para o nome do programa e o nome do executável a ser gerado, isto é, **LOCAL_SRC_FILES:= firm_asm.c** e **LOCAL_MODULE := firm_asm**.

5. Agora precisa-se setar as variáveis de compilação com o comando: **$source /home/nome_usuário/mydroid/build/envsetup.sh** e depois executar o comando **$mm** para compilar o módulo **firm_asm**.

6. O executável resultante da compilação está no diretório **/home/nome_usuário/mydroid/out/target/product/generic/system/bin/**.

7. Para compilar o **msa_mrif.c**, repete-se os passos com a devida alteração do nome do arquivo a ser compilado.

8. Para a compilação da máquina virtual **firm_vm**, precisa-se ajustar as bibliotecas e o kernel a ser utilizado na compilação.

   Deve-se executar os seguintes comandos no terminal:
   $cd
   $ firmament-2.6.29/module

9. Depois, entrar na pasta module, dentro do diretório do **Firmament**. Dentro dessa pasta, encontra-se o arquivo **Makefile**. Para alterar esse arquivo, digitar **$gedit Makefile**. Precisa-se adicionar essas duas linhas logo abaixo da linha **obj-m = firm_vm.o**:

   CROSS_COMPILE=/home/nome_usuário/mydroid/prebuilt/linux-x86/toolchain/arm-eabi-4.3.1/bin/arm-eabi-
   kernel_DIR ?=/home/nome_usuário/kernel/

10. A seguir, alterar as linhas:
    all: build
    build:
    make -C $(KSRC) SUBDIRS=`pwd` modules

    para apenas

    all:
    make -C $(kernel_DIR) M=`pwd` ARCH=arm CROSS_COMPILE=$(CROSS_COMPILE) modules

    Isso garante que o Firmament vai ser compilado para a arquitetura ARM e a versão 2.6.29 do kernel goldfish do Android.

11. A parte seguinte é a retirar esta parte do código, pois não tem função alguma:
    install:
    @# completely broken ATM - removes the whole dir
    @#make -C $(KSRC) SUBDIRS=`pwd` modules_install
    @echo "Install it yourself ..."

12. Salvar e fechar o arquivo **Makefile**.

13. Agora é preciso entrar na pasta do **Firmament**, a qual está no diretório **home**. Para isso utiliza-se os comandos a seguir:
    $cd
    $cd firmament-2.6.29/

14. Depois, digitar no terminal os seguintes comandos:
    $export ARCH=arm

$export CROSS_COMPILE=/home/nome_usuário/mydroid/prebuilt/linux-x86/toolchain/arm-eabi-4.3.1/bin/arm-eabi-
$make

15. Assim encontra-se o executável **firm_vm.ko**, o qual é utilizado no emulador do Android junto com os outros dois arquivos que já foram compilados anteriormente: o **firm_asm** e o **msa_mrif**.

Figura 3.1: Passos para recompilação do Injetor de Falhas Firmament

3.4 Copiando os arquivos do Firmament para o Android

A figura 3.2 mostra como se deve proceder para copiar os arquivos compilados do Firmament para dentro do emulador do Android, utilizando-se para isso o Android Debug Bridge ou simplesmente ADB, o qual está referenciado no site (ANDROID-DEBUG-BRIDGE).

1. Agora abrir uma janela do terminal e digitar os seguintes comandos para garantir que o terminal está no nosso diretório **home** e, também, para criar uma pasta chamada **firmament** ali.
   $cd
   $mkdir firmament

2. Copiar para dentro dela os arquivos compilados do Firmament: os arquivos **firm_asm** e **msa_mrif** e o **firm_vm.ko**, os quais estão no local especificado anteriormente.

3. Agora, executar o emulador com o kernel que foi recompilado da seguinte maneira:
   No tutorial do hello world no site do Google, foi ensinado a criar um AVD com o nome de my_avd. Assim, digitar no terminal:

   $emulator -avd my_avd -kernel /home/nome_usuário/kernel/arch/arm/boot/zImage

   Assim, o emulador será executado com o kernel que foi recompilado.

4. Agora abrir outra janela do terminal e digitar o seguinte comando para entrar no adb do emulador:
   $adb shell

5. O próximo passo é entrar na pasta **data** e criar dentro dela uma pasta chamada de **firmament**:

#cd data
#mkdir firmamet

6. Executar o comando a seguir para voltar ao terminal do Linux:
   #exit

7. A seguir, executar o seguinte comando para copiar os arquivos, os quais estão dentro da pasta **firmament** no diretório **home**, para dentro da pasta **firmament**, a qual foi criada dentro da pasta data no diretório do emulador:
   $adb push /home/nome_usuário/firmament data/firmament

Assim, os arquivos recompilados do Firmament agora estão dentro do emulador do Android, no caminho **/data/firmament**.

Figura 3.2: Copiando os arquivos do Firmament para o diretório do Android

3.5 Restrições

Os arquivos fontes do Android só podem ser compilados no sistema operacional Linux ou Mac OS. Assim, pode-se utilizar uma máquina virtual no Windows para executar uma versão do Linux ou fazer uma instalação independente de uma distribuição do Linux.

Na hora de fazer a compilação cruzada para recompilar o kernel do Android para a arquitetura ARM, Advanced Risc Machine, foi encontrada dificuldade, pois foi necessário buscar material adequado em sites e fóruns da internet para que fosse possível realizar esta etapa do trabalho. Esses mesmos problemas ocorreram na hora de fazer a compilação cruzada para o módulo da máquina virtual firm_vm do Firmament.

4 TESTES COM O FIRMAMENT

Neste capítulo são apresentados os testes feitos com a ferramenta Firmament no emulador do Android.

A primeira parte dos testes consiste em verificar o correto funcionamento do Firmament no ambiente Android. Para isso, serão utilizados os faultlets de cão de guarda, perda, duplicação e atraso de pacotes em um sistema cliente-servidor de troca de mensagens. O modelo cliente-servidor corresponde a aplicação alvo do teste, os faultlets representam a carga de falhas e as mensagens trocadas são a carga de trabalho.

A segunda parte do teste consiste em injetar falhas em aplicações reais do Android e verificar se elas possuem ou não mecanismos para contornar estados de falhas.

Para a realização dos testes foi utilizado um computador com um processador AMD Opteron 180 dual-core, 2GB de memória RAM e sistema operacional Linux Ubuntu 9.10 Karmic Koala.

4.1 Modelo Cliente Servidor

Nesta etapa foi utilizada a comunicação entre cliente-servidor, feita com base no modelo mostrado no site (CLIENTE_SERVIDOR), o qual foi utilizado para fazer o envio e o recebimento das mensagens.

Quando o emulador é iniciado, ele executa atrás de um serviço roteador/firewall virtual que o isola das configurações e interfaces de rede da máquina de desenvolvimento e da internet. Desta forma, o emulador não consegue ver a máquina de desenvolvimento ou outros emuladores na rede, vendo apenas que ele está conectado através da Ethernet a um roteador/firewall.

Desta maneira, o roteador/firewall virtual gerencia o espaço de endereçamento 10.0.2/24, na forma 10.0.2.<xx>, onde <xx> é um número. Os endereços são mostrados na tabela 4.1:

Tabela 4.1: Endereços de rede do Emulador do Android

Endereço de Rede	Descrição
10.0.2.1	Endereço do Roteador/gateway
10.0.2.2	Endereço para a interface de rede de comunicação com o host (127.0.0.1 na máquina de desenvolvimento)
10.0.2.3	Primeiro servidor DNS
10.0.2.4 / 10.0.2.5 / 10.0.2.6	Segundo, terceiro e quarto servidores DNS opcionais (se houver)

10.0.2.15	A interface de rede de comunicação do Emulador com o host
127.0.0.1	A interface própria do Emulador

O Emulador suporta os protocolos de transporte TCP e UDP para a troca de mensagens. O protocolo IGMP não é suportado e nem Multicast. O protocolo ICMP, utilizado pela aplicação Ping, não recebe as mensagens enviadas do Linux para o Emulador, apenas do Emulador para ele mesmo e dele para o Linux. Devido a essa limitação, torna-se necessária a criação de um sistema cliente-servidor para a troca de mensagens através do protocolo UDP ou TCP.

O servidor é iniciado como uma aplicação Android no eclipse, a qual inicia o Emulador do Android e fica esperando o envio das mensagens pelo cliente, o qual também executa no Eclipse, mas independentemente do emulador.

No experimento de injeção de falhas, em que o sistema cliente-servidor é o alvo e os faultlets geram a carga de falhas, foram utilizados os faultlets de cão de guarda, perda, duplicação e atraso de pacotes, os quais são mostrados nas figuras 8, 9, 12 e 15 na sequência do texto. O servidor possui código para contar o número total de pacotes recebidos e o número de pacotes duplicados, os quais são utilizados para comparar com os valores dos registradores R10 e R11 nos faultlets de perda e duplicação para verificar se os resultados obtidos são condizentes.

Em cada execução do cliente, a carga de trabalho corresponde ao envio de 50.000 pacotes através do protocolo de transporte UDP para o servidor. O tamanho do pacote de dados UDP utilizado foi de 1472 bytes de dados + 28 bytes de cabeçalho. Cada pacote enviado contém um número de sequência, permitindo saber se o pacote recebido é duplicado ou se há falta de algum pacote.

O envio das mensagens pelo cliente foi ajustado de forma que o servidor pudesse receber todos os pacotes quando um faultlet que apenas aceitava os pacotes estivesse executando. Isso foi feito para que as perdas de pacotes fossem causadas unicamente pelo faultlet perda e não por outro motivo.

Quando o servidor já estiver em execução, precisa-se fazer um redirecionamento de portas, pois o emulador do Android se comunica unicamente com a máquina host, por padrão, na porta 5554.

Assim, deve-se abrir uma janela do terminal do Linux e executar:

- telnet localhost <porta do Emulador>

Depois fazer um redirecionamento de portas com o comando:

- redir add<protocolo>:<porta do host>:<porta do Emulador>.

No caso deste trabalho:

- telnet localhost 5554 e redir add udp:5000:4444.

Os programas application Android, servidor e cliente, os quais compõe o sistema cliente-servidor, são descritos a seguir e os seus respectivos códigos são apresentados nas figuras A.1, A.2 e A.3 no apêndice A deste volume.

Application Android: é um programa que executa uma thread que tem o servidor, o qual é iniciado e fica esperando o envio dos pacotes do cliente.

Servidor: é uma aplicação Java que executa no emulador do Android, com IP

“10.0.2.15”, na porta 4444. Ao ser iniciado, ele fica esperando pelo envio de pacotes do cliente através do fluxo de transporte UDP. O servidor possui código para analisar o número de sequência de cada pacote recebido e, assim, determinar se há pacotes duplicados, pacotes fora de ordem e o total de pacotes recebidos.

Cliente: é uma aplicação Java, a qual executa no Linux e se comunica com o servidor que está executando no emulador do Android. Ao ser iniciado, ele faz o envio de 50.000 pacotes de dados para o servidor através do fluxo de transporte UDP. Cada pacote enviado pelo cliente possui um número de sequência, o qual é utilizado pelo servidor para fazer o controle dos pacotes que ele recebeu.

Os arquivos faultlets usados nos testes foram copiados para a pasta Firmament, dentro do diretório do emulador do Android e depois executados com os seguintes comandos mostrados na figura 4.1:

1. **insmod firm_vm.ko:** comando utilizado para colocar o Firmament no kernel do Emulador do Android para que ele possa atuar sobre os fluxos de transporte, a partir dos faultlets, injetando falhas;

2. **dmesg -c:** comando utilizado para ver o log do kernel. Serve para ver se o Firmament já está carregado, para ver as mensagens do mecanismo de cão de guarda e para ver o valor contido nos registradores R10 e R11, os quais irão contar o número de mensagens duplicadas, perdidas e o total recebido para os faultlets de duplicação e perda de mensagens;

3. **echo "stopflow all" > /proc/net/firmament/control:** comando utilizado para parar todos os fluxos. É necessário para que se possa montar o faultlet desejado no Firmament;

4. **./firm_asm nome-do-faultlet /proc/net/firmament/rules/ipv4_in:** comando utilizado para montar o faultlet no Firmament, deixando ele pronto para a execução. A identificação “ipv4_in” estabelece que a injeção de falhas se dará no fluxo IPv4 de entrada do Emulador. O campo “nome-do-faultlet” corresponde ao nome do arquivo faultlet utilizado;

5. **echo "startflow all" > /proc/net/firmament/control:** comando utilizado para iniciar todos os fluxos, inclusive o IPv4_in, o qual recebeu o faultlet desejado e, assim, a injeção de falhas começa;

6. **echo "showregister ipv4_in R10" > /proc/net/firmament/control:**comando utilizado para mostrar o conteúdo do registrador R10 depois da execução dos faultlets de perda e duplicação. Nos faultlets de perda e de duplicação, esse registrador contém o número total de pacotes recebidos no fluxo IPv4_in;

7. **echo "showregister ipv4_in R11" > /proc/net/firmament/control:**comando utilizado para mostrar o conteúdo do registrador R11 depois da execução dos faultlets de perda e duplicação. No faultlet de perda, esse registrador contém o número total de pacotes perdidos no fluxo IPv4_in e no faultlet de duplicação, esse registrador contém o número total de pacotes duplicados no fluxo IPv4_in;

Figura 4.1: Comandos utilizados nas execuções dos faultlets

4.1.1 Mecanismo cão de guarda (watch-dog)

O objetivo deste teste é verificar se o mecanismo de cão de guarda, watch-dog, está funcionando corretamente no Android.

O mecanismo de cão de guarda é um mecanismo de segurança, o qual entra em ação quando se executa um faultlet que faz com que a máquina virtual entre em um loop infinito.

Nestes casos, este mecanismo detecta o tempo elevado de execução do faultlet, interrompe o seu processamento e permite a entrega do pacote. O tempo de timeout padrão do Firmament é de aproximadamente 20 ms.

A visualização da execução do mecanismo de cão de guarda é feita através do sistema de registro de eventos (log) do kernel do Android, através do comando dmesg.

O funcionamento do faultlet criado para ser executado no Firmament e, deste modo, acionar o mecanismo de cão de guarda é feito da seguinte maneira: o faultlet verifica se o protocolo de transporte é UDP e se for, ele testa para ver se o destino é o emulador do Android (IP: 10.0.2.15). Se sim, ele desvia a execução para o loop1, o qual desvia a execução para o loop2 que por sua vez, desvia para o loop1, fazendo com que a máquina virtual execute um laço infinito e, assim, o mecanismo cão de guarda entra em ação, interrompendo a execução do faultlet e permitindo a entrega do pacote. O faultlet utilizado neste teste é mostrado na figura B.1, no Apêndice B deste volume.

Depois que o mecanismo de cão de guarda entrou em ação e permitiu a entrega dos pacotes, verificou-se o log de mensagens do kernel do Android e o resultado foram inúmeras mensagens de aviso, indicando que o cão de guarda foi chamado diversas vezes para o fluxo IPv4_in. As mensagens são mostradas na figura 4.2:

```
firm_vm: watchdog called for flow ipv4_in.
firm_vm: watchdog called for flow ipv4_in.
firm_vm: watchdog called for flow ipv4_in.
firm_vm: watchdog called for flow ipv4_in.
.
.
.
firm_vm: watchdog called for flow ipv4_in.
firm_vm: watchdog called for flow ipv4_in.
firm_vm: watchdog called for flow ipv4_in.
firm_vm: watchdog called for flow ipv4_in.
```

Figura 4.2: Algumas das mensagens contidas no log do kernel do Android

A partir destas mensagens, conclui-se que o mecanismo está funcionando de maneira correta, pois elas mostram que ele foi acionado devido ao laço infinito contido no faultlet, estando de acordo com a especificação descrita no Firmament.

4.1.2 Descarte de Pacotes

O teste apresentado a seguir tem por objetivo mostrar a capacidade do Firmament de fazer com que pacotes sejam perdidos, simulando falhas de comunicação que podem ocorrer na rede.

O faultlet para descarte de pacotes foi configurado no Firmament, estando pronto para fazer a injeção de falhas no protocolo de transporte UDP. A seguir, o sistema cliente servidor foi iniciado, e o cliente iniciou o envio dos 50.000 pacotes de dados para o servidor.

O faultlet verifica se o protocolo de transporte é UDP e se for, ele verifica se o pacote tem como destino o emulador do Android (IP: 10.0.2.15). Caso afirmativo, o faultlet faz um sorteio, estabelecendo o número 1000 como base. Depois do sorteio, o número terá valor entre -1000 e 1000. Então, soma-se 500 ao número resultante do sorteio, o que resultará num valor entre -500 e 1500. Desse modo, a probabilidade do número ser negativo é de 25%, pois na faixa de -500 a 1500, a quantidade de números negativos é de ¼. Essa taxa é então utilizada para fazer o descarte de pacotes. Assim, se o número for negativo, a execução desvia para a parte do faultlet responsável pelo descarte de pacotes, a qual irá incrementar em 1 o valor do registrador R11, o qual corresponde ao número de pacotes perdidos e irá descartar o pacote. Caso contrário, o faultlet incrementa em 1 o valor do registrador R10, o qual corresponde ao número total de pacotes recebidos, e aceita o pacote.

Os registradores R10 e R11 são utilizados para fazer a comparação com os valores obtidos no programa servidor para verificar a consistência dos resultados.

O faultlet criado para fazer o descarte de pacotes é mostrado na figura B.2, no Apêndice B deste volume.

O resultado da execução deste faultlet sobre o envio das 50.000 mensagens pelo cliente foi registrado no programa servidor pelo recebimento de 37535 pacotes de dados.

Esse número de mensagens recebidas equivale a 75,07% do total de mensagens enviadas, o que corresponde a uma perda de 24,93%. Essa taxa é muito próxima da taxa de descarte de pacotes estabelecida no faultlet. Isso ocorre porque o descarte de pacotes é feito de forma pseudo-aleatória e não determinística. Desse modo, nesse tipo de teste, a taxa definida raramente é igual a resultante, podendo ser maior ou menor.

A verificação do arquivo de log do kernel do Android mostrou os valores armazenados nos registradores R10 e R11. Os valores contidos nestes registradores estão na numeração hexadecimal e deste modo precisam ser convertidos para a base decimal para proporcionar um melhor entendimento. O registrador R10, o qual registra o total de mensagens recebidas, contém o valor, em hexadecimal, de R10 = 0x929f, o qual corresponde a 37535 em decimal. Este valor corresponde ao mesmo valor encontrado no programa servidor para a contagem das mensagens recebidas. O registrador R11, o qual registra o total de mensagens descartadas, contém o valor, em hexadecimal, de R11 = 0x30b1, o qual corresponde a 12465 em decimal. Se somarmos este valor com o valor de R10, ou com o valor obtido no programa servidor, iremos obter o valor de 50.000, o qual corresponde ao número total de mensagens enviadas.

O resultado do envio dos pacotes do cliente para o servidor e do faultlet são mostrados nas figuras 4.3 e 4.4:

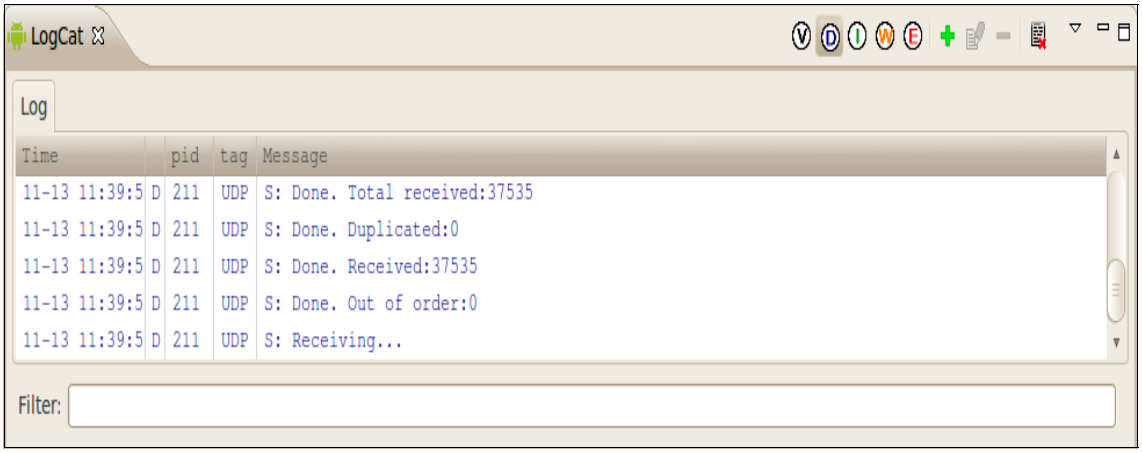

Figura 4.3: Resultado do envio dos pacotes do cliente para o servidor

```
firm_vm: stopping flow ipv4_in.
firm_vm: stopping flow ipv6_in.
firm_vm: stopping flow ipv4_out.
firm_vm: stopping flow ipv6_out.
firm_vm: flow ipv4_in already stopped.
firm_vm: flow ipv6_in already stopped.
firm_vm: flow ipv4_out already stopped.
firm_vm: flow ipv6_out already stopped.
firm_vm: starting flow ipv4_in.
firm_vm: starting flow ipv6_in.
firm_vm: starting flow ipv4_out.
firm_vm: starting flow ipv6_out.
firm_vm: ipv4_in.r10: 0x929f
firm_vm: ipv4_in.r11: 0x30b1
```

Figura 4.4: Visualização do arquivo de log do kernel do Android

4.1.3 Duplicação de Pacotes

O teste apresentado a seguir tem por objetivo mostrar a capacidade do Firmament de fazer com que pacotes sejam duplicados, reproduzindo uma falha comum de comunicação, típica do protocolo UDP.

O faultlet para duplicar os pacotes foi configurado no Firmament, estando pronto para fazer a injeção de falhas no protocolo de transporte UDP. A seguir, o sistema cliente servidor foi iniciado, e o cliente iniciou o envio dos 50.000 pacotes de dados para o servidor.

O faultlet verifica se o protocolo de transporte é UDP e se for, ele verifica se o pacote tem como destino o emulador do Android (IP: 10.0.2.15). Caso afirmativo, o faultlet faz um sorteio, estabelecendo o número 1000 como base. Depois do sorteio, o número terá valor entre -1000 e 1000. Então, soma-se 500 ao número resultante do sorteio, o que resultará num valor entre -500 e 1500. Desse modo, a probabilidade do número ser negativo é de 25%, pois na faixa de -500 a 1500, a quantidade de números negativos é de ¼. Essa taxa é então utilizada para fazer a duplicação de pacotes. Assim, se o número for negativo, a execução desvia para a parte do faultlet responsável pela

duplicação de pacotes, a qual irá incrementar em 1 o valor do registrador R11, o qual corresponde ao número de pacotes duplicados, irá incrementar em 2 o valor do registrador R10, o qual corresponde ao número total de pacotes recebidos (os enviados e mais os duplicados) e irá duplicar o pacote. Caso contrário, o faultlet incrementa em 1 o valor de R10 e aceita o pacote.

Os registradores R10 e R11 são utilizados para fazer a comparação com os valores obtidos no programa servidor para verificar a consistência dos resultados.

O faultlet criado para fazer a duplicação de pacotes é mostrado na figura B.3, no Apêndice B deste volume.

O resultado da execução deste faultlet sobre o envio das 50.000 mensagens pelo cliente foi registrado no programa servidor pelo recebimento de 62423 pacotes de dados no total. O servidor também registrou a chegada das 50.000 mensagens originais enviadas pelo cliente.

Esse número total de mensagens recebidas equivale a 124,846% do total de mensagens enviadas, o que corresponde a um aumento de 24,846%. Essa taxa é muito próxima da taxa de duplicação de pacotes estabelecida no faultlet. Isso ocorre porque a duplicação de pacotes é feita de forma pseudo-aleatória e não determinística. Desse modo, nesse tipo de teste, a taxa definida raramente é igual a resultante, podendo ser maior ou menor.

A verificação do arquivo de log do kernel do Android mostrou os valores armazenados nos registradores R10 e R11. Os valores contidos nestes registradores estão na numeração hexadecimal e deste modo precisam ser convertidos para a base decimal para proporcionar um melhor entendimento. O registrador R10, o qual registra o total de mensagens recebidas, contém o valor, em hexadecimal, de R10 = 0xf3d7, o qual corresponde a 62423 em decimal. Este valor corresponde ao mesmo valor encontrado no programa servidor para a contagem do total de mensagens recebidas. O registrador R11, o qual registra o total de mensagens duplicadas, contém o valor, em hexadecimal, de R11 = 0x3087, o qual corresponde a 12423 em decimal. Se subtrairmos este valor do valor de R10, ou do valor obtido no programa servidor, iremos obter o valor de 50.000, o qual corresponde ao número total de mensagens enviadas pelo cliente.

O resultado do envio dos pacotes do cliente para o servidor e do faultlet são mostrados nas figuras 4.5 e 4.6:

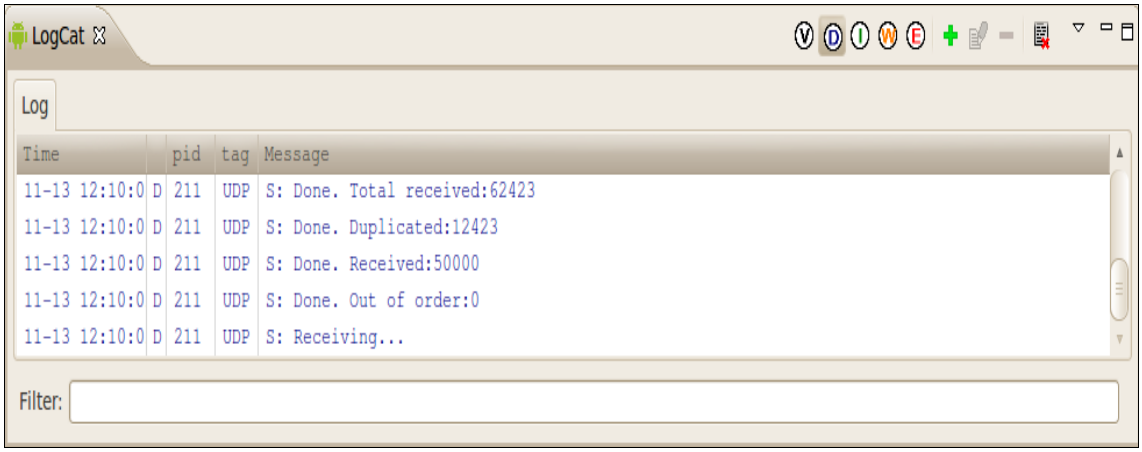

Figura 4.5: Resultado do envio dos pacotes do cliente para o servidor

```
firm_vm: stopping flow ipv4_in.
firm_vm: stopping flow ipv6_in.
firm_vm: stopping flow ipv4_out.
firm_vm: stopping flow ipv6_out.
firm_vm: flow ipv4_in already stopped.
firm_vm: flow ipv6_in already stopped.
firm_vm: flow ipv4_out already stopped.
firm_vm: flow ipv6_out already stopped.
firm_vm: starting flow ipv4_in.
firm_vm: starting flow ipv6_in.
firm_vm: starting flow ipv4_out.
firm_vm: starting flow ipv6_out.
firm_vm: ipv4_in.r10: 0xf3d7
firm_vm: ipv4_in.r11: 0x3087
```

Figura 4.6: Visualização do arquivo de log do kernel do Android

4.1.4 Atraso de Pacotes

O teste apresentado a seguir tem por objetivo mostrar a capacidade do Firmament de fazer com que pacotes sejam atrasados, simulando falhas de comunicação que podem ocorrer na rede devido a congestionamentos ou rotas diferentes por exemplo.

O faultlet para atraso de pacotes foi configurado no Firmament, estando pronto para fazer a injeção de falhas no protocolo de transporte UDP. A seguir, o sistema cliente servidor foi iniciado, e o cliente iniciou o envio dos 50.000 pacotes de dados para o servidor.

O faultlet verifica se o protocolo de transporte é UDP e se for, ele verifica se o pacote tem como destino o emulador do Android (IP: 10.0.2.15). Caso afirmativo, o faultlet faz um sorteio, estabelecendo o número 1000 como base. Depois do sorteio, o número terá valor entre -1000 e 1000. Então, soma-se 500 ao número resultante do sorteio, o que resultará num valor entre -500 e 1500. Desse modo, a probabilidade do número ser negativo é de 25%, pois na faixa de -500 a 1500, a quantidade de números negativos é de ¼. Essa taxa é então utilizada para fazer o descarte de pacotes. Assim, se o número for negativo, a execução desvia para a parte do faultlet responsável pelo atraso de pacotes, a qual irá incrementar em 1 o valor do registrador R11, o qual corresponde ao número de pacotes atrasados e irá atrasar o pacote. Caso contrário, o faultlet incrementa em 1 o valor do registrador R10, o qual corresponde ao número total de pacotes recebidos sem atraso, e aceita o pacote.

Os registradores R10 e R11 são utilizados para fazer a comparação com os valores obtidos no programa servidor para verificar a consistência dos resultados.

O faultlet criado para fazer o atraso de pacotes é mostrado na figura B.4, no Apêndice B deste volume.

O resultado da execução deste faultlet sobre o envio das 50.000 mensagens pelo cliente foi que o atraso de pacotes não funcionou. No momento em que as mensagens são enviadas e aproximadamente 25% delas vão sofrer atraso, o programa servidor pára de funcionar. Uma possível causa disso pode ser porque a versão do kernel do Android é

distinta da versão do kernel do Linux para a qual a ferramenta Firmament foi feita. Recentemente, as atualizações do kernel tem alterado bastante a estrutura da pilha de protocolos do sistema operacional, em especial a da função NF_QUEUE do Netfilter, que emula atraso de mensagens.

4.2 Aplicações reais do Android

Nesta etapa foram realizados testes com aplicações reais feitas para o Android. O objetivo dos testes é verificar como se comportam as aplicações escolhidas na presença de falhas de comunicação e, assim, verificar se elas possuem algum mecanismo para tentar contornar estas falhas ou avisar o usuário de que falhas estão acontecendo. Serão feitos testes com as seguintes aplicações, obtidas nos sites (ANDROID-APLICATION):

Cestos, um jogo online multiplayer;

Traffic Cam Viewer, uma aplicação que permite a visualização do trânsito em estradas e ruas de alguns lugares no mundo;

MSN Talk, um cliente de mensagens instantâneas que permite se conectar a rede MSN;

As aplicações de jogo online e de conversa instantânea serão instaladas no smartphone Motorola Backflip, o qual possui a versão 1.5 do Android e a versão 2.6.27 do kernel, e no emulador do Android, o qual possui a versão 2.1 do Android e versão 2.6.29 do kernel, o qual foi alterado anteriormente para permitir a execução do módulo do Firmament.

O faultlet atua tanto para pacotes do fluxo de transporte UDP como TCP, garantindo desta maneira que a injeção de falhas estará sendo feita corretamente, independentemente do protocolo de transporte utilizado pelas aplicações. Ele faz o sorteio de um número, sendo que a probabilidade desse número se negativo depende da taxa de descarte de pacotes escolhida. Assim, se o número for negativo, a execução desvia para a parte do faultlet responsável pelo descarte de pacotes, a qual irá incrementar em 1 o valor do registrador R11, o qual corresponde ao número de pacotes perdidos e irá descartar o pacote. Caso contrário, o faultlet incrementa em 1 o valor do registrador R10, o qual corresponde ao número total de pacotes recebidos, e aceita o pacote.

O faultlet criado para fazer o descarte de pacotes é mostrado na figura B.5, no Apêndice B deste volume.

4.2.1 Traffic Cam Viewer

É uma aplicação para o Android, desenvolvida por Robert Chou, que permite visualizar o tráfego em estradas e ruas de vários lugares do mundo, como 41 dos 50 estados dos Estados Unidos, Europa, Austrália, Canadá, Hong Kong, Singapura e outros.

Esta aplicação é excelente para o teste de descarte de pacotes, pois ela sendo um streaming de vídeo, o fluxo de pacotes de dados é constante e, desta forma, pode-se observar facilmente quando as falhas acontecem.

Para a realização do teste foi utilizada a versão v2.8.4 do programa e escolhida a câmera 12th ST NW and Constitution Ave NW, a qual fica em Washington DC, nos

Estados Unidos. Foi escolhida a parte da tarde para fazer os testes para melhor visualização das falhas nas imagens, pois à noite as imagens ficaram muito escuras.

O teste consistirá em aplicar uma injeção de falhas no programa, através da perda de pacotes, durante um tempo em torno de 3 minutos e será observado o comportamento da aplicação nestas condições de falha.

4.2.1.1 *Injetando-se falhas no fluxo IPv4_in*

Os faultlets serão aplicados em ordem crescente de descarte de pacotes, começando com uma taxa de descarte igual a 10% e aumentando até que se consiga visualizar as falhas nas exibições das imagens. Os registradores R10 e R11 contém respectivamente o número de pacotes aceitos e descartados.

A seguir são apresentados os resultados dos testes realizados:

Taxa de 10 % de descarte de pacotes: as imagens fluem quase normalmente, com um atraso de 1 a 3 segundos na atualização das imagens, porém sem erros visíveis;

Taxa de 20 % de descarte de pacotes: o atraso na atualização da tela aumentou, sendo de 5 a 12 segundos, contudo, ainda sem erros visíveis;

Taxa de 30 % de descarte de pacotes: a demora na atualização da tela passou a ser entre 20 e 25 segundos, e por um momento muito rápido, a tela ficou preta. Essa foi a única falha visualizada neste teste;

Taxa de 40 % de descarte de pacotes: a demora na atualização da tela passou a ser bem variada, levando 5, 18 ou 40 segundos, e foi possível visualizar uma falha na montagem da imagem, a qual é mostrada na figura 4.7:

Figura 4.7: Falha na imagem para taxa de 40 % no descarte de pacotes

Taxa de 50 % de descarte de pacotes: o atraso na atualização da tela passou a ser 40 a 50 segundos, e também foi possível visualizar uma falha na montagem da imagem, a qual é mostrada na figura 4.8:

Figura 4.8: Falha na imagem para taxa de 50 % no descarte de pacotes

A figura 4.9 mostra a imagem da câmera quando não ocorre perda de pacotes:

Figura 4.9: Imagem normal da câmera sem perda de pacotes

A realização do teste mostrou como o aumento da taxa de perda de pacotes trouxe problemas para a aplicação, como um maior tempo na taxa de atualização da imagem obtida pela câmera e a ocorrência de falhas na montagem das telas devido à perda de pacotes.

A aplicação, porém, não emitiu um aviso de que estava tendo problemas para receber os pacotes e montar as telas. Desse modo, essa aplicação não deve ter uma implementação de um mecanismo que alerte que falhas de comunicação estão ocorrendo. Contudo, a aplicação apresentou-se bem robusta, pois só apresentou falhas na montagem das imagens para taxas de 40% e 50% de descarte de pacotes, as quais são taxas altas de descarte de pacotes.

4.2.2 Cestos

É um jogo online multiplayer, o qual foi desenvolvido pela empresa Chicken Brick Studios e que permite partidas entre 2 e 4 pessoas ao mesmo tempo. O programa foi instalado no smartphone Motorola BackFlip e no Emulador do Android para permitir partidas online entre uma aplicação em um dispositivo real e outra no emulador com o injetor de falhas.

Os dois programas se comunicam através de um servidor central, o qual mantém a lista de usuários online que podem jogar entre si. A versão utilizada do programa foi a versão 1.5, lançada em 26 de agosto de 2010.

A imagem no início da partida é mostrada na figura 4.10:

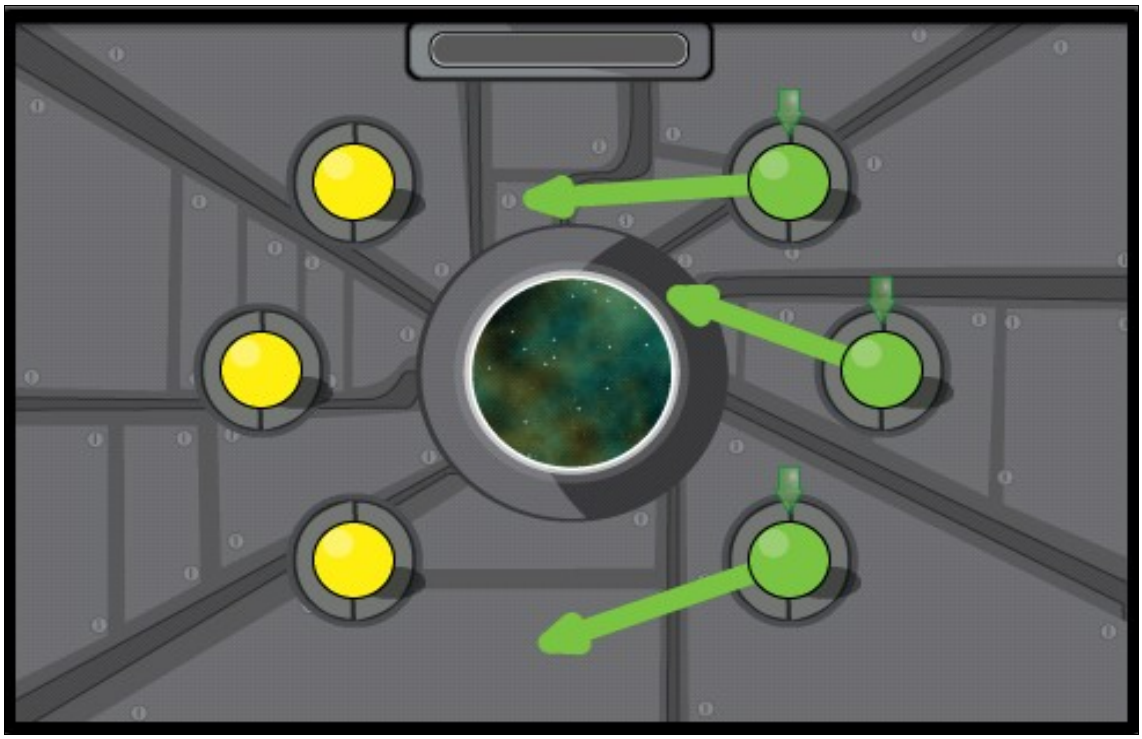

Figura 4.10: Imagem do início da partida entre o emulador e o telefone

Os testes realizados foram divididos em duas partes: na primeira, foram injetadas

falhas no fluxo IPv4_in, começando com taxa de 10% de perda de pacotes e aumentando ela até ocorrer uma falha no jogo. Na segunda parte, foram injetadas falhas no fluxo IPv4_out e, da mesma maneira, com uma taxa de falha inicial igual a 10% e aumentando ela até o jogo apresentar falhas.

Os dois programas se comunicam através de um servidor central, o qual mantém a lista de usuários online que podem jogar entre si. A versão utilizada do programa foi a versão 1.5, lançada em 26 de agosto de 2010.

4.2.2.1 *Injetando-se falhas no fluxo IPv4_in*

Neste teste foram injetadas falhas de comunicação no fluxo IPv4_in, as quais atuam sobre os pacotes enviados da rede do servidor do jogo Cestos para a aplicação no emulador. Foram testadas configurações de injeção de falhas diferentes, com taxas de descarte de pacotes começando em 10% e aumentando até o jogo apresentar falhas.

A seguir são apresentados os resultados dos testes realizados:

Taxa de 10 % de descarte de pacotes: o início do jogo ficou um pouco lento, mas foi possível jogar, contudo, o jogo estava mais lento com relação ao que estava sendo executado no telefone em alguns instantes;

Taxa de 20 % de descarte de pacotes: teve um maior atraso na inicialização do jogo e depois a execução da partida estava bem mais adiantada no telefone do que no emulador, mesmo o jogo tendo um mecanismo de sincronização.

Taxa de 30 % de descarte de pacotes: também ocorreu certo atraso no início da partida e depois, o jogo no emulador começou a ficar muito atrasado com relação ao telefone, até que em um determinado momento, o emulador foi desconectado da partida. Isso é mostrado na figura 4.11:

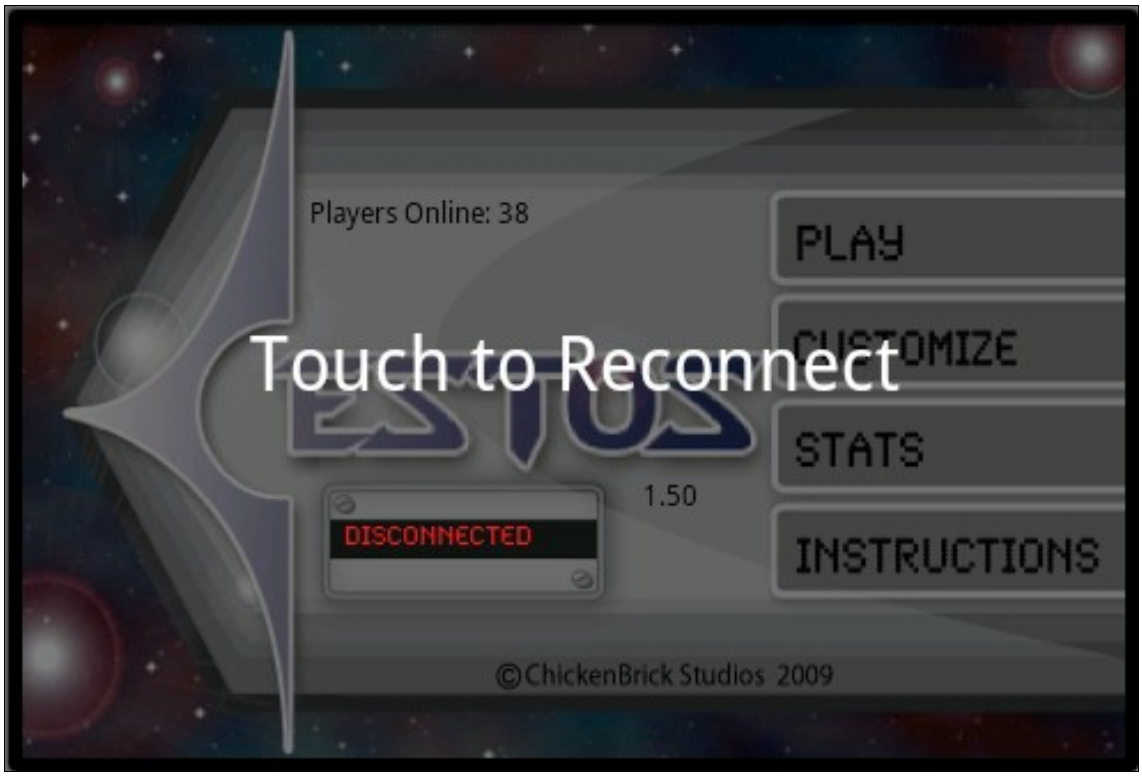

Figura 4.11: Imagem mostrada no emulador após ser desconectado da partida

Com estes testes, pode-se concluir que o aumento da taxa de descarte de pacotes no fluxo IPv4_in fez com que o jogo apresentasse problemas como o aumento do atraso entre as duas aplicações, o que fez com que o emulador fosse desconectado da partida, pois a partida já estava totalmente fora de sincronismo.

Tanto a aplicação no telefone quanto a no emulador recebem do servidor mensagens com os movimentos dos outros jogadores e, quando essas mensagens não chegam , caso do emulador, a aplicação fica parada, esperando por essas informações.

4.2.2.2 *Injetando-se falhas no fluxo IPv4_out*

Neste teste foram injetadas falhas de comunicação no fluxo IPv4_out, as quais atuam sobre os pacotes enviados da aplicação executando no emulador para a rede do servidor do jogo. Foram testadas configurações de injeção de falhas diferentes, com taxas de descarte de pacotes começando em 10% e aumentando até que o jogo falha-se.

Os resultados dos testes realizados são mostrados a seguir:

Taxa de 10, 20, 30, 40, 50, 60, 70 e 80 % de descarte de pacotes: para todas essas taxas, o início do jogo ficou um pouco lento, mas depois foi possível jogar normalmente, com raros atrasos entre o emulador e o telefone;

Taxa de 90 % de descarte de pacotes: o início do jogo também foi lento e depois começou a acontecer certo atraso e no início da terceira jogada, aconteceu uma falha no envio dos pacotes e o emulador foi desconectado da partida. A imagem mostrada no emulador é a mesma da figura 4.19 mostrada acima. A aplicação no telefone mostrou uma tela de encerramento da partida, avisando que o emulador estava desconectado.

Com estes testes, pode-se concluir que o aumento da taxa de descarte de pacotes no fluxo IPv4_out não teve efeito significativo até a taxa de 80% de descarte de pacotes, pois a aplicação continuava funcionando. Ela só foi apresentar falha para 90% de descarte de pacotes, o que fez com que o emulador fosse desconectado da partida, por não conseguir enviar os pacotes ao servidor.

4.2.3 MSN Talk

Este aplicativo é um cliente do MSN Messenger desenvolvido para dispositivos Android pela empresa Talk Builder. Ele permite conversas entre grupos de amigos e suporta o envio de mensagens offline.

A versão utilizada foi a 2.40 lançada em 29 de outubro de 2010, a qual possui como destaque as seguintes atualizações no software:

- janelas de conversa separadas;
- mostra mensagem de sistema quando um contato muda de status na janela de conversa;
- correções de bugs.

Os testes foram divididos em duas partes: na primeira, a injeção de falhas com o faultlet para perda de pacotes era iniciada antes de se executar o MSN Talk e na segunda parte, a aplicação foi iniciada antes de se iniciar a injeção de falhas com o faultlet. A injeção de falhas sempre é feita no emulador executando a aplicação de troca de mensagens.

4.2.3.1 *Iniciando o MSN Talk depois de carregar o faultlet perda_udp_tcp no Firmament*

O objetivo deste teste é mostrar como a aplicação se comporta quando falhas de comunicação já estão ocorrendo antes da aplicação ser executada. Este teste foi realizado injetando-se falhas tanto para o fluxo IPv4_in como para o fluxo IPv4_out.

4.2.3.1.1 *Injetando-se falhas no fluxo IPv4_out*

Neste teste foram injetadas falhas de comunicação no fluxo IPv4_out, as quais atuam sobre os pacotes enviados pela aplicação no emulador para o servidor do MSN. Foram testadas três situações distintas, para falhas com 95%, 75% e 50% de chances de descarte de pacotes.

Para 95% e 75% de chance de descarte de pacotes, o MSN Talk tentou por um bom tempo se conectar com o servidor e acabou não conseguindo, exibindo uma mensagem de que não foi possível estabelecer uma conexão com o servidor e pedindo para verificar a conexão com a internet. A mensagem exibida na tela do emulador é mostrada na figura 4.12:

Figura 4.12: Mensagem mostrada na tela do emulador indicando que não foi possível se conectar ao servidor do MSN

Para 50% de descarte de pacotes, foi possível se conectar ao servidor e enviar mensagens para a aplicação executando no telefone com o Android. O recebimento de mensagens da aplicação executando no telefone foi normal. As mensagens trocadas são mostradas nas figuras 4.13 e 4.14:

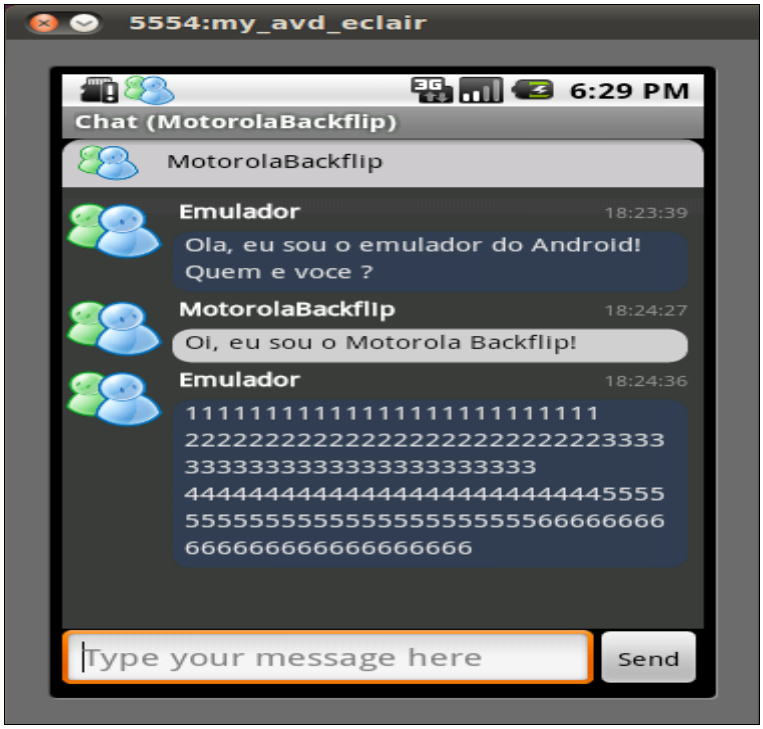

Figura 4.13: Janela de mensagens do emulador para taxa de 50% de descarte de pacotes no fluxo IPv4_out

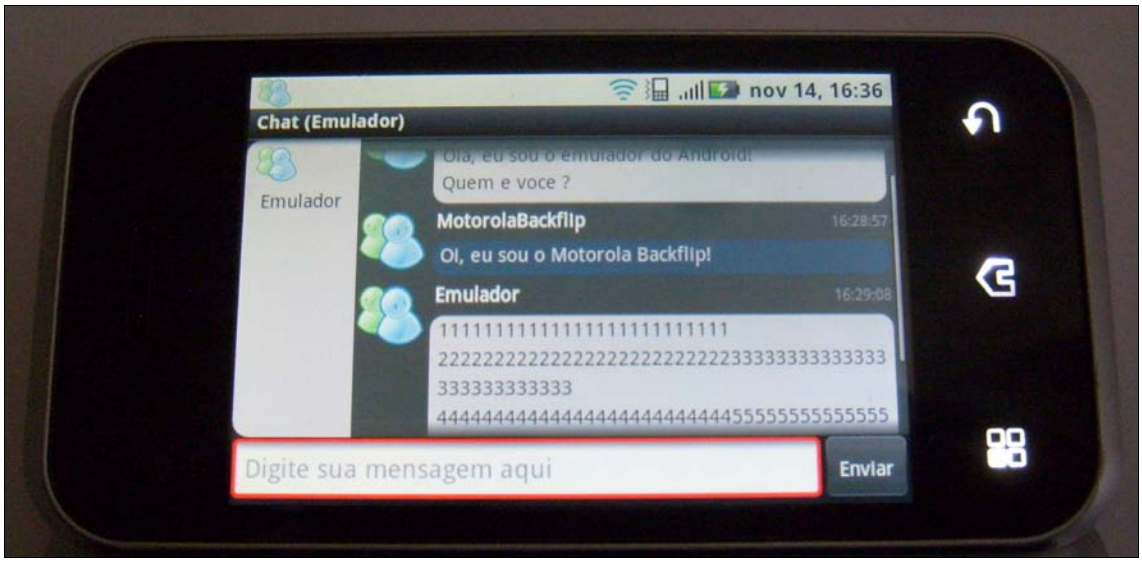

Figura 4.14: Janela de mensagens do telefone para taxa de 50% de descarte de pacotes no fluxo IPv4_out

4.2.3.1.2 *Injetando-se falhas no fluxo IPv4_in*

Neste teste foram injetadas falhas de comunicação no fluxo IPv4_in, as quais atuam sobre os pacotes enviados da rede do servidor do MSN para a aplicação no emulador. Foram testadas quatro situações distintas, para falhas com 95%, 75%, 50% e 25% de chances de descarte de pacotes.

Para 95% e 75% de chance de descarte de pacotes, o MSN Talk não conseguiu se conectar com a rede do MSN.

Para 50% de descarte de pacotes, foram feitas três tentativas de conexão, sendo que duas delas falharam. A aplicação executando no telefone enviou 2 mensagens, as quais demoraram por volta de 1 minuto para serem recebidas no telefone. As outras 4 mensagens enviadas pelo telefone foram recebidas depois de 2 minutos de espera. As mensagens trocadas são mostradas nas figuras 4.15 e 4.16 a seguir:

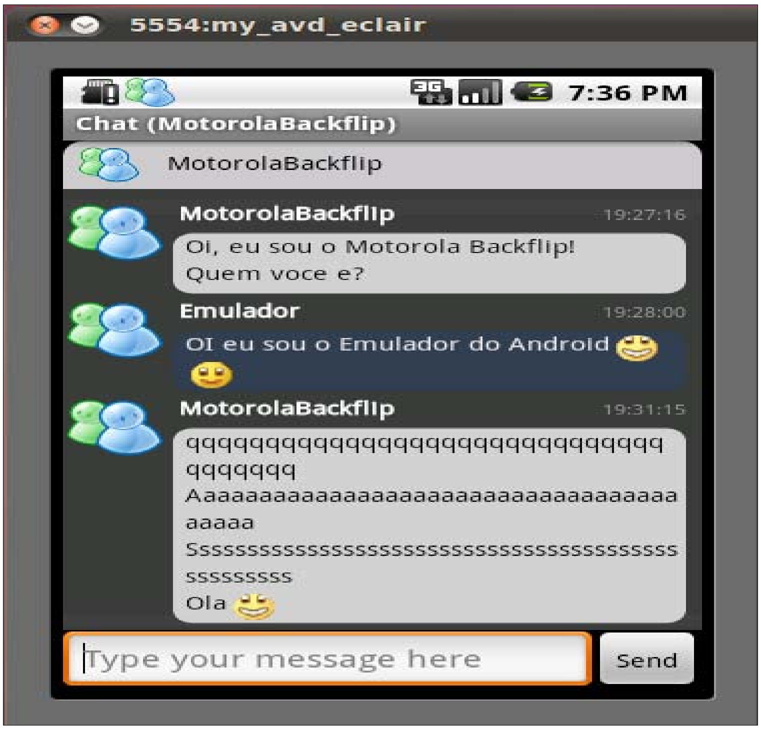

Figura 4.15: Janela de mensagens do emulador para taxa de 50% de descarte de pacotes no fluxo IPv4_in

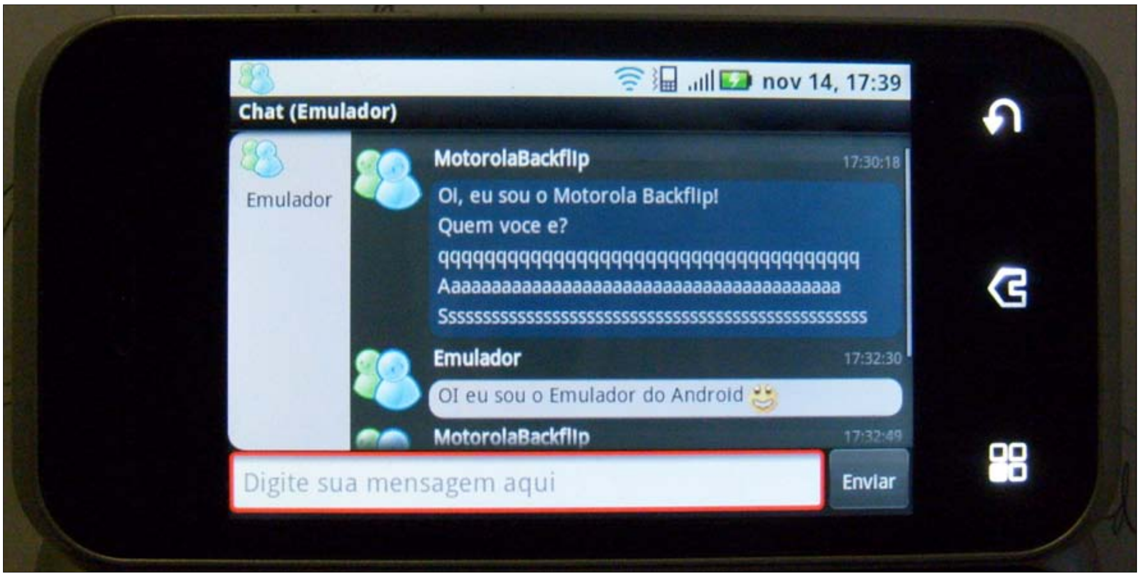

Figura 4.16: Janela de mensagens do telefone para taxa de 50% de descarte de pacotes

no fluxo IPv4_in

Para 25% de chance de descarte de pacotes, a injeção de falhas permitiu uma conversa quase normal, com exceção de certo atraso em algumas mensagens enviadas da aplicação no telefone para a aplicação no emulador. As mensagens trocadas são mostradas nas figuras 4.17 e 4.18 a seguir:

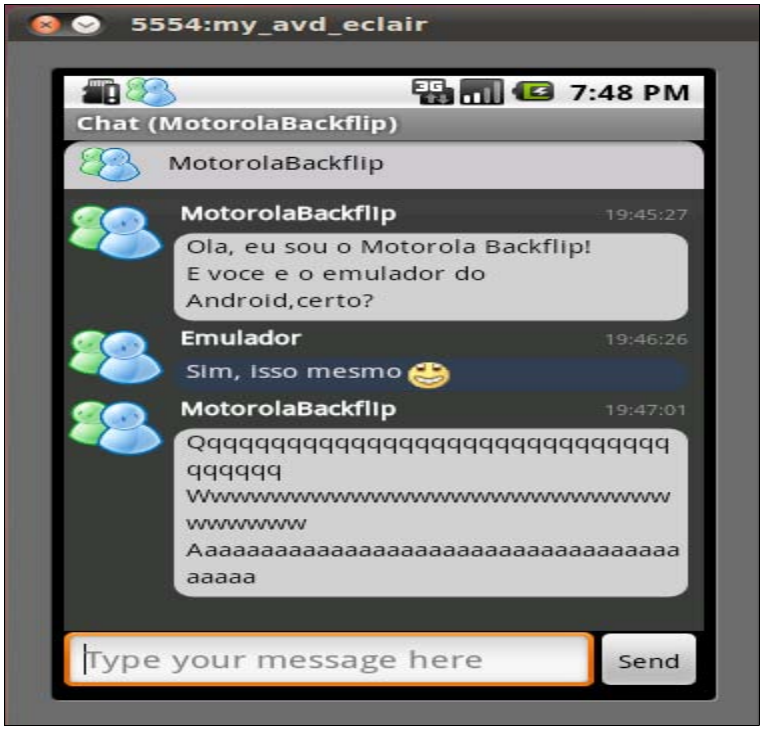

Figura 4.17: Janela de mensagens do emulador para taxa de 25% de descarte de pacotes no fluxo IPv4_in

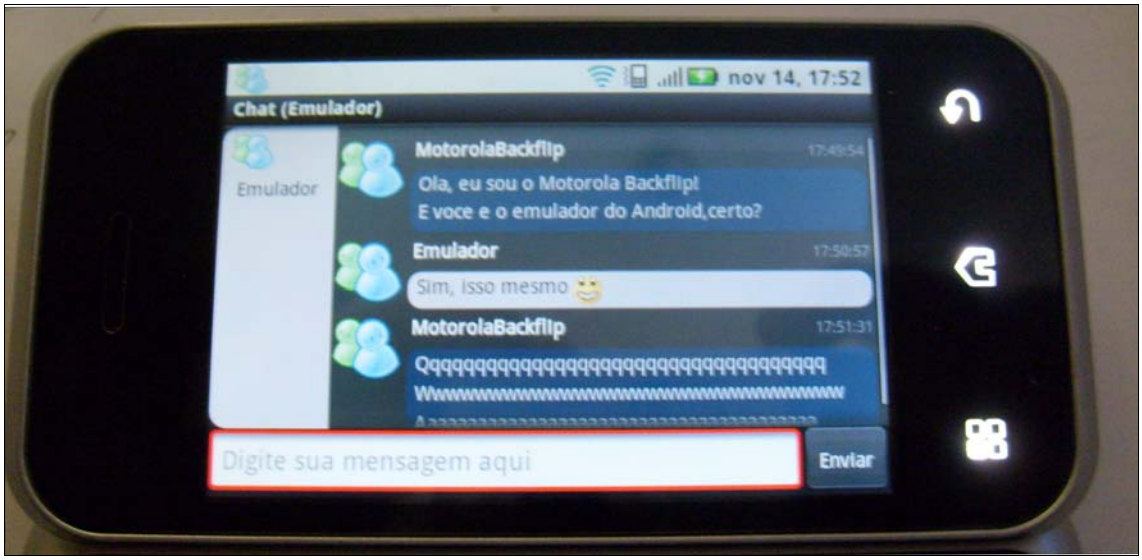

Figura 4.18: Janela de mensagens do telefone para taxa de 25% de descarte de pacotes no fluxo IPv4_in

4.2.3.2 *Iniciando o MSN Talk antes de carregar o faultlet perda_udp_tcp no Firmament*

O objetivo deste teste é mostrar como a aplicação se comporta quando falhas de comunicação acontecem depois que a aplicação já está sendo executada. Este teste foi realizado injetando-se falhas tanto para o fluxo IPv4_in como para o fluxo IPv4_out no programa executando no emulador.

4.2.3.2.1 *Injetando-se falhas no fluxo IPv4_out*

Neste teste foram injetadas falhas de comunicação no fluxo IPv4_out, as quais atuam sobre os pacotes enviados pela aplicação no emulador para o servidor do MSN. Foram testadas duas situações distintas, para falhas com 99,5% e 95% de chances de descarte de pacotes.

Para 95% de chance de descarte de pacotes, a aplicação no emulador consegue enviar as mensagens para o telefone, mas elas demoram a chegar, o que permite que se escreva uma mensagem no telefone e que ela chegue ao emulador antes que as mensagens do emulador cheguem ao telefone.

Para 99,5% de descarte de pacotes, as mensagens enviadas pelo emulador demoram muito para chegar no telefone e as últimas mensagens enviadas chegaram depois de dois minutos e não houve qualquer aviso na aplicação que estava executando no emulador sobre o fato de não estar conseguindo enviar as mensagens devido a uma falha de comunicação na rede (faultlet com taxa de descarte de pacotes muito elevada). As mensagens trocadas são mostradas nas figuras 4.19 e 4.20 a seguir:

Figura 4.19: Janela de mensagens do emulador para taxa de 99,5% de descarte de pacotes no fluxo IPv4_out

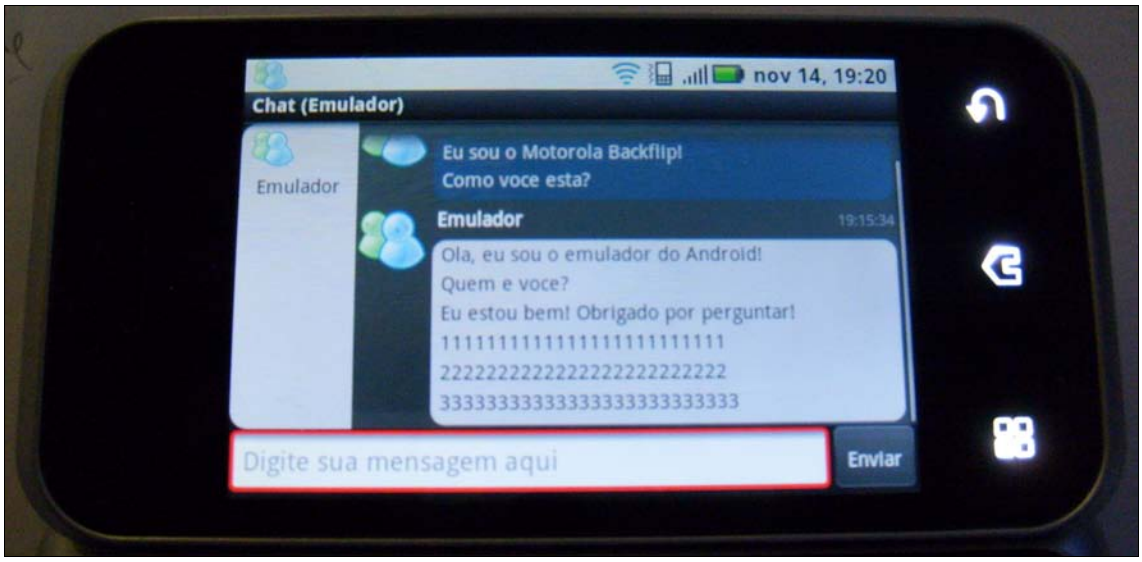

Figura 4.20: Janela de mensagens do telefone para taxa de 99,5% de descarte de pacotes no fluxo IPv4_out

4.2.3.2.2 *Injetando-se falhas no fluxo IPv4_in*

Neste teste foram injetadas falhas de comunicação no fluxo IPv4_in, as quais atuam sobre os pacotes enviados da rede do servidor do MSN para a aplicação no emulador. Foram testadas duas situações distintas, para taxas com 99,5% e 95% de descarte de pacotes.

Para 95% de chance de descarte de pacotes, foram enviadas mensagens da aplicação no telefone para o emulador, mas elas não foram recebidas e as mensagens enviadas do emulador para o telefone demoraram por volta de 2 minutos para chegarem ao telefone.

Para 99,5% de descarte de pacotes, foram enviadas mensagens do telefone para o emulador e do emulador para o telefone, porém, nada foi recebido por nenhum dos dois. Não houve qualquer mensagem de aviso de qualquer uma das partes por não conseguir enviar as mensagens. As mensagens trocadas são mostradas nas figuras 4.21 e 4.22 a seguir.

A realização destes testes mostrou que a aplicação MSN Talk é extremamente robusta, pois com índices altos de injeção de falhas ela conseguiu enviar as mensagens, mesmo com atraso. Ela só foi incapaz de entregar as mensagens quando se usou uma taxa de 99,5% de descarte de pacotes no fluxo IPv4_in. Quando foram utilizadas taxas menores de descarte de pacotes, a troca de mensagem foi feita sem problemas ou com um pequeno atraso. Para taxas maiores houve casos em que o atraso no recebimento ou envio das mensagens foi muito grande. Quando a aplicação foi iniciada depois que as falhas já estavam ocorrendo, ela não conseguiu se conectar na rede do MSN para taxas de descarte de pacotes de 75 e 95%.

Contudo, a aplicação que estava executando no emulador não emitiu nenhum aviso de que não estava conseguindo enviar as mensagens para o servidor do MSN para depois serem entregues a aplicação no telefone. Seria interessante se esta aplicação possuísse um mecanismo para avisar o usuário de que as mensagens não estão sendo entregues, algo semelhante ao que a aplicação Windows Live Messenger da Microsoft

para o sistema operacional Windows possui para estes casos. Isto poderia evitar possíveis transtornos ao usuário.

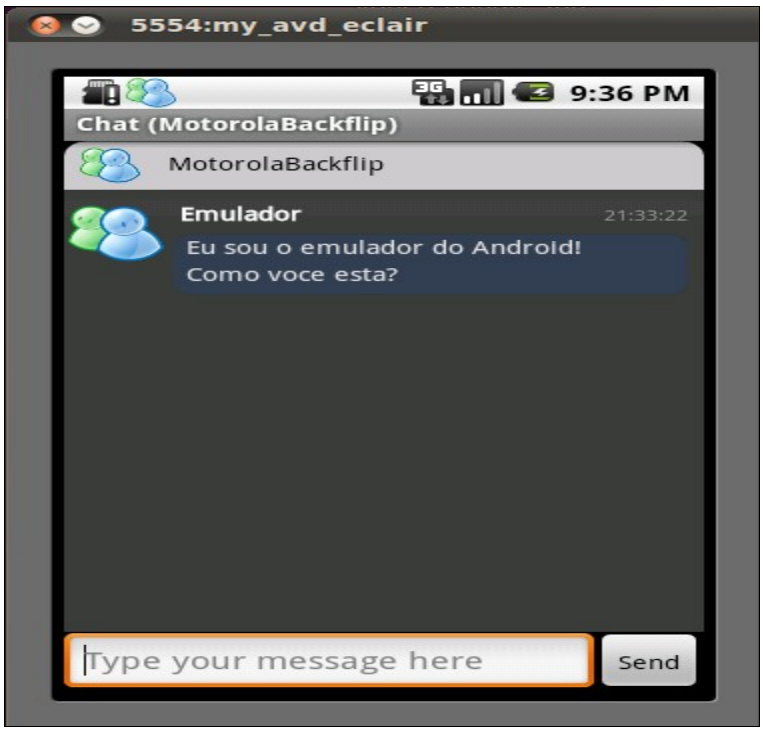

Figura 4.21: Janela de mensagens do emulador para taxa de 99,5% de descarte de pacotes no fluxo IPv4_in

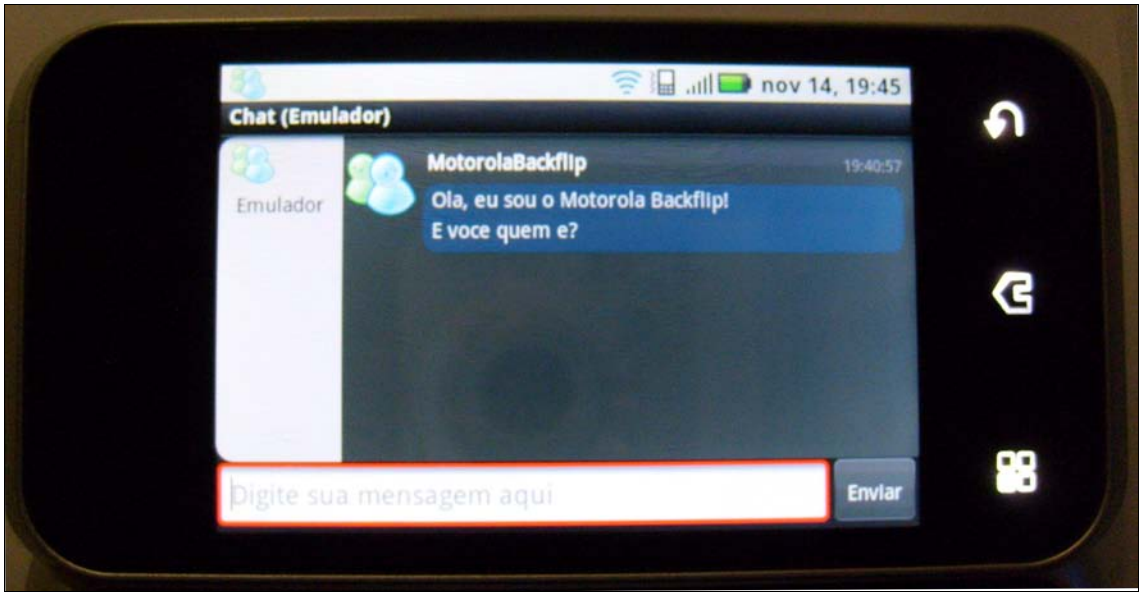

Figura 4.22: Janela de mensagens do telefone para taxa de 99,5% de descarte de pacotes no fluxo IPv4_in

4.3 Comentários sobre os testes das aplicações

O Android possui mais de 100.000 aplicações disponíveis e desse modo, muitas delas seriam interessantes para a realização dos testes. Porém, das aplicações escolhidas para os testes, nem todas elas puderam ser instaladas no emulador do Android ou funcionaram nele. A seguir é feito um comentário sobre o que aconteceu com essas aplicações testadas:

Android Television V1.5.3: instalou com sucesso no emulador, mas um canal de televisão foi selecionado, o vídeo não funcionou, apenas o áudio.

Youtube: instalou com sucesso no emulador, mas os vídeos não funcionam. A tela fica parada na imagem inicial e só o áudio funciona.

ProjectINF_1.2.4: jogo online multiplayer. Instalou com sucesso, mas não inicializou no emulador.

Googletalk: não instalou no emulador, mostrando uma mensagem de que faltava uma biblioteca compartilhada.

Opera_Mobile_10.1: instalou normalmente, a navegação pelas páginas é excelente, mas não conseguiu exibir vídeos de páginas na web, nem com a instalação do Adobe Flash Player. O mesmo ocorreu no navegador padrão que vem com o emulador.

No Motorola BackFlip, telefone com o Android que foi utilizado nos testes com as aplicações MSN Talk e com o jogo online Cestos juntamente com o emulador, essas aplicações que tiveram problemas no emulador do Android funcionaram perfeitamente.

4.4 Testes de aplicações com o telefone com o Android

Era um objetivo inicial deste trabalho colocar o Firmament dentro do diretório do Android do telefone Motorola BackFlip e executar os mesmos testes feitos com o emulador. Para isso seria utilizado o ADB shell. O qual permite executar comandos no emulador e em dispositivos móveis.

Porém, para que isso fosse possível, seria necessário utilizar uma versão do kernel que possuísse o módulo do Netfilter ativado e que permitisse a execução de módulos compilados. Essas duas características estão presentes no kernel para a versão 2.1 do Android. A versão do Android presente no telefone é a 1.5 com kernel 2.6.27 e a Motorola até agora não disponibilizou a atualização para a versão 2.1 do Android para este modelo de telefone. Esse foi o motivo de não ter sido possível testar as aplicações com injeção de falhas diretamente no telefone.

5 CONCLUSÃO

A realização do trabalho proporcionou o aprendizado do funcionamento e da arquitetura do ambiente de desenvolvimento Android, de uma ferramenta de injeção de falhas e como proceder para fazer o porte de aplicações do Linux para o Android.

Por ser baseado na versão 2.6 do kernel do Linux, a princípio, o porte de ferramentas do Linux para o Android funcionaria normalmente. Porém, o Android utiliza uma versão modificada deste kernel, a qual não suporta o conjunto completo de bibliotecas padrão GNU. Isso torna difícil o porte de aplicações ou bibliotecas Linux/GNU existentes para o Android.

Isso pode explicar o fato de que o porte do injetor de falhas Firmament não ter sido um sucesso completo. Nos testes realizados com um sistema cliente-servidor, a ferramenta manteve a total funcionalidade para as funções de cão de guarda, descarte e duplicação de pacotes. Porém a função de atraso, que utiliza a função NF_QUEUE do Netfilter para emular atraso de mensagens não funcionou.

O estudo da ferramenta Firmament e o fato de a maioria das aplicações desenvolvidas para o Android utilizarem a troca de pacotes para comunicação, evidencia o fato de que essas aplicações devem ser construídas com algum mecanismo de tolerância a falhas para evitar maiores transtornos ao usuário quando falhas de comunicação acontecem.

Assim, foram realizados testes de injeção de falhas através do descarte de pacotes em aplicações reais para o Android. Isto permitiu analisar como três aplicações distintas, um programa de streaming de vídeo, um jogo multiplayer online e uma aplicação de troca de mensagens instantânea se comportam na presença de falhas de comunicação.

No teste feito com a aplicação Traffic Cam Viewer, a qual mostra imagens do tráfego de veículos em ruas e estradas de alguns lugares no mundo, as falhas visíveis na montagem das imagens só aconteceram para taxas de descarte de pacotes de 40% e 50%. Para 30% de descarte de pacotes, a tela ficou totalmente preta por menos de 1 segundo.

Isso mostrou que a aplicação é bem robusta, pois só apresentou falhas para taxas maiores de perda de pacotes.

O jogo online multiplayer Cestos foi testado em 2 etapas, sendo que na primeira, foi injetado falhas no fluxo IPv4_in, no qual o servidor envia as mensagens para a aplicação executando no emulador. Para as taxas de 10% e 20% de descarte de pacotes, ocorreu certa demora na inicialização da partida e no geral, o jogo no telefone estava mais adiantado do que no emulador, principalmente para a tava de 20%. Para a taxa de 30% de descarte de pacotes, o jogo no emulador ficou muito atrasado e ele acabou

sendo desconectado do servidor do jogo.

Na segunda etapa foram injetadas falhas no fluxo IPv4_out, no qual o emulador envia as mensagens para o servidor do jogo. Para as taxas de descarte de pacotes de 10% a 80%, foi possível jogar normalmente, com raros atrasos entre o emulador e o telefone. A aplicação apresentou falhas para a taxa de 90 % de descarte de pacotes.

Os resultados destes testes mostraram que o programa Cestos também possui uma boa tolerância a falhas, pois ele falhou apenas para elevadas taxas de descarte de pacotes, porém sem causar maiores problemas.

A aplicação de troca de mensagens, MSN Talk, foi testada em 2 etapas distintas: quando falhas de comunicação já estão ocorrendo antes da aplicação ser executada e quando as mesmas falhas ocorrem depois que a aplicação já está sendo executada.

Nestes testes, foram injetadas diferentes taxas de falhas de comunicação para os fluxos IPv4_in e IPv4_out para ambas as situações descritas anteriormente e, no geral, as mensagens continuaram sendo enviadas ou recebidas, mesmo que em alguns casos ocorreu um atraso bem grande. As mensagens trocadas entre as aplicações só não foram recebidas quando foi utilizada uma taxa de 99,5% de falha no fluxo IPv4_in, para a situação em que as falhas acontecem depois que a aplicação já está executando.

Portanto, a realização destes testes mostrou que as aplicações do Android que foram testadas possuem algum mecanismo para contornar estados de falha, pois todas elas suportaram taxas de falhas razoáveis com problemas mínimos, como atraso no jogo, na atualização da tela ou no envio de mensagens. Essas aplicações só apresentaram falhas quando foi aplicada uma taxa muito alta de perda de pacotes.

Por fim, a Ferramenta Firmament se mostrou muito eficiente para o teste de aplicações reais do Android, embora não tenha sido possível utilizar a sua função de atraso de pacotes, o que proporcionaria uma análise ainda melhor dos mecanismos de tolerância a falhas utilizados pelas aplicações do Android.

Neste trabalho foi utilizado como referência o artigo (DOBLER, 2010), no qual foi feito o porte da ferramenta Firmament para o ambiente Android e os testes iniciais para a verificação do funcionamento da ferramenta no Android.

REFERÊNCIAS

CANALYS. Disponível em: < http://www.canalys.com/index.html >. Acessado em: agosto, 2010.

ARLAT. **Comparison of Physical and Software-Implemented Fault Injection Techniques**. 2003. IEEE Transactions on Computers, Vol. 52, Nº. 9.

CARREIRA. **Comparison of Physical and Software-Implemented Fault Injection Techniques**. 1998. Software Engineering Notes Vol 23, Nº 1, p 42.

AVIZIENIS, A.; Laprie, J.; Randell B.; Landwehr C. **Basic Concepts and Taxonomy of Dependable and Secure Computing**. IEEE trans. on dependable and secure computing, V. 1, n. 1, jan 2004, pp 11-33.

RUSSEL, R.; Welte, H. (2002). **Linux net filter hacking HOWTO**. 2002. Disponível em: < http://www.netfilter.org/documentation/>. Acessado em: abril, 2010.

DREBES, Roberto Jung; JACQUES-SILVA, Gabriela; TRINDADE, Joana Matos Fonseca da; WEBER, T. S.. **A kernel-based Communication Fault Injector for Dependability Testing of Distributed Systems**. In: First International Haifa Verification Conference, 2006, Haifa - Israel. Hardware and Software, Verification and Testing, Revised Selected Papers. Berlin- Heidelberg: Springer-Verlag, 2006. v. 3875. p. 177-190.

SIQUEIRA, Tórgan Flores de; Fiss, B. C.; Menegotto, C. C.; CECHIN, Sergio Luis; WEBER, T. S.. **Avaliação experimental de estratégias de tolerância a falhas do protocolo SCTP por injeção de falhas**. In: Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, 2009, Recife. v. 1. p. 915-929.

SIQUEIRA, Tórgan Flores de; Fiss, B. C.; WEBER, Raul Fernando; CECHIN, Sergio Luis; WEBER, T. S.. **Applying FIRMAMENT to test the SCTP communication protocol under network faults**. In: Test Workshop, 2009. LATW '09. 10th Latin American, 2009, Buzios.. v. 1. p. 1-6.

ANDROID. Disponível em: < http://www.android.com/ >. Acessado em: abril, 2010.

ANDROID-PLATFORM. Disponível em: <http://developer.android.com/resources/dashboard/platform-versions.html>. Acessado em: novembro, 2010.

ANDROID-SISTEM. Disponível em: <http://en.wikipedia.org/wiki/Android_(operating_system) >. Acessado em: novembro, 2010.

ANDROID-ARCHITECTURE. Disponível em: < http://developer.android.com/guide/basics/what-is-android.html >. Acessado em novembro, 2010.

ANDROID-SOURCE. Disponível em: < http://source.android.com/source/download.

html >. Acessado em: maio, 2010.

ANDROID-SDK. Disponível em: < http://developer.android.com/sdk/index.html >. Acessado em: abril, 2010.

ANDROID-SDK-INSTALLING. Disponível em: < http://developer.android.com/sdk/installing.html >. Acessado em: abril, 2010.

ADT-PLUGIN. Disponível em < http://developer.android.com/sdk/eclipse-adt.html >. Acessado em: abril, 2010.

HELLO-WORLD. Disponível em < http://developer.android.com/resources/tutorials/hello-world.html >. Acessado em: abril, 2010.

CROSS-COMPILATION. Disponível em: < http://wiki.strongswan.org/projects/strongswan/wiki/Android , http://my.opera.com/otaku_2r/blog/download-and-build-the-google-android e http://linuxclues.blogspot.com/2010/05/build-compile-linux-kernel-android.html >. Acessados em: maio, 2010.

ACKER, E. V.; WEBER, T. S.; CECHIN, Sergio Luis. **Injeção de falhas para validar aplicações em ambientes móveis**. In: Workshop de Testes e Tolerância a Falhas, 11., 2010, Gramado. SBC, 2010. p. 61-74.

LOADABLE-kernel-MODULE. Disponível em: < http://groups.google.com/group/android-kernel/msg/a43f2b1db7a5c3dc?pli1 >. Acessado em: maio, 2010.

ANDROID-DEBUG-BRIDGE. Disponível em: < http://developer.android.com/guide/developing/tools/adb.html >. Acessado em: abril, 2010.

CLIENTE_SERVIDOR. Disponível em: < http://www.anddev.org/udp-networking_-_within_the_emulator-t280.html e http://www.anddev.org/socket_programming-t325-s15.html>. Acessados em: outubro, 2010.

ANDROID-APLICATION. Disponível em: < http://www.freewarelovers.com/android e http://www.filecrop.com/android-apk.html >. Acessados em: novembro, 2010.

DOBLER, R. J.; WEBER, T. S.; CECHIN, Sergio Luis. **Porte Injetor de Falhas Firmament para o Ambiente Android**. In: Escola Regional de Redes de Computadores, 2010, Alegrete, 2010.

ANEXO – ARTIGO DA PROPOSTA DESTE TRABALHO

Porte do Injetor de Falhas Firmament para o Ambiente Android

Rodrigo J. Dobler[1], Taisy S. Weber[1], Sérgio L. Cechin[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{rjdobler,taisy,cechin}@inf.ufrgs.br

Resumo. *Dispositivos móveis como celulares, palms e smartphones, estão cada vez mais presentes em nossas vidas. Eles estão evoluindo muito depressa e, a cada nova versão, os aparelhos são lançados com muito mais recursos. Isto proporciona novos horizontes para os desenvolvedores de software. Hoje, devido a iniciativas de alguns fabricantes, muitas empresas e desenvolvedores independentes estão lançando programas para celulares. Porém, nem sempre são tomados cuidados com relação à tolerância a falhas e, desse modo, muitas aplicações podem apresentar problemas. Assim, nesse trabalho será portada uma ferramenta de injeção de falhas, Firmament, a qual foi desenvolvida para o sistema operacional Linux, para que ela possa ser utilizada no ambiente Android. Esta ferramenta irá permitir injeção de falhas na troca de mensagem sobre o protocolo IP de algumas aplicações rodando no emulador do Android. Isto irá permitir que se possa analisar o comportamento dessas aplicações na presença de falhas e assim, verificar se são ou não tolerantes a falhas.*

1. Introdução

Os dispositivos móveis trouxeram muitos benefícios para as nossas vidas. Os aparelhos celulares, por exemplo, permitem que nós possamos nos comunicar com outras pessoas ou acessar informação de quase qualquer lugar em que estejamos. Estes aparelhos melhoraram muitos nos últimos anos, o que possibilitou a criação de excelentes sistemas operacionais para eles.

Alguns fabricantes, como a Apple e a Google, lançaram ferramentas de desenvolvimento de software, para permitir que outras empresas e desenvolvedores independentes possam desenvolver aplicativos para as respectivas plataformas. Porém, apenas a Google tornou público o código fonte do Android e não impôs restrições ao desenvolvimento de aplicações. Essa estratégia da Google incentivou o uso do Android nos celulares e hoje um grande número de fabricantes o adota como sistema operacional de seus aparelhos, como Motorola, Samsung, Sony Ericsson, LG e outros. Assim, o Android começou a ganhar grande destaque no mercado de aparelhos móveis.

De acordo com a recente pesquisa publicada pela empresa Canalys (CANALYS), nos Estados Unidos, o maior mercado de smartphones do mundo, os dispositivos com Android representaram juntos 34% do mercado norte-americano no 2º trimestre de 2010.

No mundo todo, nesse mesmo período, o número de smartphones que utilizam o Android cresceu 886% e isso se deveu as operadoras de telefonia celular, as quais estão promovendo amplamente os dispositivos Android e também ao grande número de aplicações feitas para este sistema, sendo muitas delas de excelente qualidade e sem

custo algum para o usuário. Esse grande número de aplicações desenvolvidas é realmente um grande atrativo, contudo, nem todas elas possuem implementações consistentes para detectar e corrigir erros e, assim, quando falhas acontecem, elas podem causar erros na aplicação, causando transtornos para o usuário.

Desse modo, seria necessário testar as aplicações desenvolvidas, injetando-se falhas, para que seja possível analisar como as aplicações irão se comportar. Isto permitiria fazer uma prevenção nas aplicações para que elas fossem capazes de contornar um estado de falha sem causar maiores problemas ao usuário ou sistema operacional. Para que isso seja viável, é necessário que as aplicações tenham alta dependabilidade.

Avizienis (Avizienis, 2004) define dependabilidade, como sendo a habilidade de prestar serviços em que se pode justificadamente confiar. Este conceito engloba uma série de atributos como disponibilidade, confiabilidade, facilidade de manutenção, segurança funcional crítica, testabilidade, entre outros.

Normalmente, utilizam-se ferramentas de injeção de falhas para se poder testar um sistema eficientemente e, assim, encontrar soluções para os eventuais problemas, garantindo assim um aumento na dependabilidade do sistema. Hoje não existem muitas ferramentas de injeção de falhas e, destas, a maioria não utiliza métodos eficientes para realizar bons testes e assim, pode-se pensar que o aparelho possui um sistema de comunicação confiável quando na verdade pode não ter.

Falhas de comunicação são relevantes em ambientes móveis, pois eles dependem do envio e recebimento de mensagens para poderem se comunicar. Assim, o uso de uma boa ferramenta de injeção de falhas torna-se essencial para que seja possível fazer testes eficientes e assim obter conclusões corretas.

Assim, este trabalho irá mostrar um passo a passo de como se fazer o porte para o ambiente Android de uma ferramenta de injeção de falhas, o Firmament, a qual foi desenvolvida para o sistema operacional Linux. A idéia é que esse trabalho possa ser utilizado como referência para outras pessoas que tenham interesse em portar ferramentas de injeção de falhas, ou de outros tipos, do Linux para o Android.

2 Especificação do Projeto

2.1 Netfilter

O Netfilter (RUSSEL; WELTE, 2002) é uma ferramenta para manipulação de pacotes que define pontos específicos, ou ganchos, na pilha de protocolos onde funções de callback podem ser registradas. Quando os pacotes alcançam esses pontos, eles são passados para essas funções, as quais possuem acesso completo ao seu conteúdo. Assim, elas podem processar os pacotes e decidir se os mesmos podem passar pela pilha de protocolos ou serem descartados.

O Netfilter permite o registro de funções de callback para as famílias de protocolos IPv4, IPv6, DECnet e ARP. Os ganchos estão disponíveis em diversos pontos de interesse: no recebimento e envio de pacotes, no encaminhamento de pacotes roteados e na entrega e recebimento de pacotes para os níveis superiores.

2.2 Firmament

É uma ferramenta de injeção de falhas, desenvolvida por Roberto Jung Drebes (DREBES, 2005), (SIQUEIRA, 2009), a qual visa permitir a especificação de cenários de falhas de comunicação complexos, com baixa intrusividade, para o teste de protocolos de comunicação e sistemas distribuídos.

Esta ferramenta pode ser utilizada para aplicações de rede, nas quais o uso de implementações reais pode evidenciar deficiências da implementação que possivelmente escapariam à simulação. Ela também pode ser usada para o estudo de novos protocolos, pois a ferramenta permite colocar o sistema em estados incomuns ou de difícil reprodução.

O Firmament utiliza a arquitetura da interface de programação Netfilter, a qual está disponível a partir da versão 2.4 do kernel do Linux. Por utilizar apenas interfaces de programação de alto nível do kernel, a ferramenta é independente da arquitetura da máquina e, assim, pode ser utilizada em servidores, desktops e dispositivos embarcados que utilizem o Linux.

2.3 Android

O Android (ANDROID) é um ambiente de desenvolvimento que foi inicialmente desenvolvido pela Android Inc, empresa que foi adquirida pela Google, e depois foi lançado oficialmente pela Open Handset Alliance, um grupo formado por 65 empresas de tecnologia que atuam em diferentes áreas como operadoras de telefonia, empresas de software, empresas de semicondutores, fabricantes de celulares, etc.

O sistema operacional do Android foi construído com base na versão 2.6 do kernel do Linux e a utiliza para os serviços centrais do sistema, tais como segurança, gestão de memória, gestão de processos, etc. O kernel também atua como uma camada de abstração entre o hardware e o software.

O Software Development Kit, SDK, inclui um conjunto abrangente de ferramentas de desenvolvimento. Estas incluem um debugger, bibliotecas, um emulador de telefone (com base no QEMU), documentação do código da amostra, e tutoriais. Atualmente as plataformas suportadas para desenvolvimento incluem computadores de arquitetura x86 rodando Linux, Mac OS X 10.4.8 ou posterior, Windows XP ou Vista.

3 Desenvolvimento

3.1 Obtenção dos arquivos fontes do Android

Para se obter os arquivos fonte e depois compilá-los, precisa-se seguir os passos do tutorial mostrado no próprio site do Android. O tutorial encontra-se na página (ANDROID-SOURCE).

3.2 Recompilando o kernel do Android

O kernel do Android necessita ser recompilado para que se possa ativar o módulo do Netfilter. A ativação deste módulo torna-se necessária para que o Firmament, ferramenta de injeção de falhas, possa ser utilizado para atuar sobre as mensagens na rede. Também será necessário ativar o carregamento de módulos, para que o módulo recompilado do Firmament possa ser executado no emulador do Android.

No site do Android, na seção do SDK, será feito o download do SDK para Linux na página (ANDROID-SDK) e, depois segue-se o link (ANDROID-SDK-INSTALLING) para se obter o tutorial de instalação do SDK, o qual descreve como obter plataformas e outros componentes. Depois este link (ADT-PLUGIN) ensina como instalar o Android Development Tools plugin para o Eclipse e esse outro link (HELLO-WORLD), descreve como criar um Android Virtual Devices e rodar o emulador.

Para se obter e recompilar o kernel do Android, ativando-se os módulos necessários, segue-se os passos explicados nos sites (CROSS-COMPILATION).

Esse kernel recompilado é que será utilizado no emulador do Android para que seja possível carregar o Firmament e executar os testes de injeção de falhas no emulador do Android.

3.3 Recompilando o Firmament para o Android

Esta parte foi conduzida utilizando-se como referência o artigo (ACKER 2010), o qual foi muito útil para compilar os arquivos firm_asm e mas_mrif, os quais pertencem ao módulo do Firmament. A solução apresentada no site (LOADABLE-KERNEL-MODULE) serviu de apoio para realizar a compilação cruzada da máquina virtual firm_vm. A figura 1 mostra como isso deve ser feito:

O primeiro passo é compilar os arquivos firm_asm e msa_mrif. Obter o Firmament versão 2.6.29 e extrair em uma pasta no diretório ***home***. Os dois arquivos citados estão dentro da pasta ***asm***, dentro da pasta principal do ***Firmament***.

1. Agora, entrar no diretório ***mydroid***, criado de acordo com o tutorial para obtenção dos arquivos fontes do Android, digitando no terminal os comandos ***$cd*** e depois ***$cd mydroid***. Dentro desse diretório, encontramos várias pastas e dentre elas a pasta ***external***. Essa pasta contém os programas que estarão disponíveis no sistema.
2. Assim, entrar na pasta ***externa***l com o comando ***$cd external*** e criar uma pasta, com o nome de ***myapps*** executando ***$mkdir myapps*** no terminal. A seguir, copiar uma das pastas originais que existem dentro da pasta ***external***, como a ***ping*** por exemplo. Depois renomeia-se a pasta ***ping***, a qual foi copiada para dentro da pasta ***myapps***, para ***firm_asm***.
3. O próximo passo é entrar na pasta ***firm_asm*** e excluir todos os outros arquivos, deixando apenas o arquivo ***Android.mk*** dentro da pasta. Voltar para a pasta ***asm*** dentro do diretório do ***Firmament*** e copiar os seus arquivos fontes para dentro da pasta ***firm_asm***.
4. A seguir, editar o arquivo ***Android.mk*** e alterar as variáveis ***LOCAL_SRC_FILES e LOCAL_MODULE*** para o nome do programa e o nome do executável a ser gerado, isto é, ***LOCAL_SRC_FILES:= firm_asm.c e LOCAL_MODULE := firm_asm*** .
5. Agora precisa-se setar as variáveis de compilação com o comando:

   $source /home/nome_usuário/mydroid/build/envsetup.sh e depois executar o comando ***$mm*** para compilar o módulo ***firm_asm***.
6. O executável resultante da compilação está no diretório ***/home/nome_usuário/mydroid/out/target/ product/generic/system/bin/***.
7. Para compilar o ***msa_mrif.c***, repete-se os passos anteriores com a devida alteração do nome do arquivo a ser compilado.
8. Para a compilação da máquina virtual ***firm_vm***, precisa-se ajustar as bibliotecas e o kernel a ser utilizado na compilação.

   Deve-se executar os seguintes comandos no terminal:

   $cd

   $ firmament-2.6.29/module

9. Depois, entrar na pasta ***module***, dentro do diretório do ***Firmament***. Dentro dessa pasta, encontra-se o arquivo ***Makefile***. Para alterar esse arquivo, digitar ***$gedit Makefile*** . Precisa-se adicionar essas duas linhas logo abaixo da linha ***obj-m = firm_vm.o***:

   CROSS_COMPILE=/home/nome_usuário/mydroid/prebuilt/linux-x86/toolchain/arm-eabi-4.3.1/bin/arm-eabi-

   KERNEL_DIR ?=/home/nome_usuário/kernel/

10. A seguir, alterar as linhas

    all: build

    build:

    make -C $(KSRC) SUBDIRS=`pwd` modules

    para apenas

    all:

    make -C $(KERNEL_DIR) M=`pwd` ARCH=arm CROSS_COMPILE=$(CROSS_COMPILE) modules

    Isso garante que o Firmament vai ser compilado para a arquitetura ARM e a versão 2.6.29 do kernel goldfish do Android.

11. A parte seguinte é a retirar esta parte do código pois não tem função alguma:

    install:

    @# completely broken ATM - removes the whole dir

    @#make -C $(KSRC) SUBDIRS=`pwd` modules_install

    @echo "Install it yourself ..."

12. Salvar e fechar o arquivo ***Makefile***.

13. Agora é preciso entrar na pasta do ***Firmament***, a qual está no diretório ***home***. Para isso utiliza-se os comandos a seguir:

    $cd

    $cd firmament-2.6.29/

14. Depois, digitar no terminal os seguintes comandos:

    $export ARCH=arm

    $export CROSS_COMPILE=/home/nome_usuário/mydroid/prebuilt/

    linux-x86/toolchain/arm-eabi-4.3.1/bin/arm-eabi-

    $make

15. Assim, encontra-se o executável ***firm_vm.ko***, o qual é utilizado no emulador do Android junto com os outros dois arquivos que já foram compilados anteriormente: o ***firm_asm*** e o ***msa_mrif***.

Figura 1: Passos para recompilação do Injetor Firmament

3.4 Copiando os arquivos do Firmament para o Android

A figura 2 mostra como se deve proceder para copiar os arquivos compilados do Firmament para dentro do emulador do Android, utilizando-se para isso o Android Debug Bridge ou simplesmente ADB, o qual está referenciado no site (ANDROID-DEBUG-BRIDGE).

1. Agora abrir uma janela do terminal e digitar os seguintes comandos para garantir que o terminal está no nosso diretório ***home*** e, também, para criar uma pasta chamada ***firmament*** ali.

   $cd

   $mkdir firmament

2. Copiar para dentro dela os arquivos compilados do Firmament: os arquivos ***firm_asm*** e ***msa_mrif*** e o ***firm_vm.ko***, os quais estão no local especificado anteriormente.

3. Agora, executar o emulador com o kernel que foi recompilado da seguinte maneira:

   No tutorial do hello world no site do Google, foi ensinado a criar um AVD com o nome de ***my_avd***. Assim, digitar no terminal:

   $emulator -avd my_avd -kernel /home/nome_usuário/kernel/arch/

   arm/boot/zImage

   Assim, o emulador será executado com o kernel que foi recompilado.

4. Agora abrir outra janela do terminal e digitar o seguinte comando para entrar no adb do emulador:

   $adb shell

5. O próximo passo é entrar na pasta ***data*** e criar dentro dela uma pasta chamada de ***firmament***:

   #cd data

   #mkdir firmament

6. Executar o comando a seguir para voltar ao terminal do Linux

   #exit

7. A seguir, executar o seguinte comando para copiar os arquivos, os quais estão dentro da pasta ***firmament*** no diretório ***home***, para dentro da pasta ***firmament***, a qual foi criada dentro da pasta ***data*** no diretório do emulador:

   $adb push /home/nome_usuário/firmament data/firmament

   Assim, os arquivos recompilados do Firmament agora estão dentro do emulador do Android, no caminho ***/data/firmament***.

Figura 2: Copiando os arquivos do Firmament para dentro do Android

3.5 Verificando o Firmament dentro do Android

Depois disso foi criado um arquivo de teste apenas para ter certeza de que o Firmament estava funcionando dentro do emulador do Android. O arquivo criado tem o nome de teste, o qual foi copiado de forma semelhante para dentro do emulador do Android.

A figura 3 mostra como o arquivo é composto:

```
add r0 r1
acp
```

Figura 3: Conteúdo do arquivo utilizado para testar o Firmament dentro do Emulador do Android

Ele foi executado com os seguintes comandos da figura 4 para assegurar o funcionamento do Firmament dentro do Android:

```
#cd data
# cd firmament
# dmesg -c
# ./firm_asm teste
# ./firm_asm teste /proc/net/firmament/rules
# echo "stopflow all"
# stopflow all
# echo "stopflow all" > /proc/net/firmament/control
# ./firm_asm teste /proc/net/firmament/rules/ipv4_out
# echo "startflow ipv4_out" > /proc/net/firmament/control
```

Figura 4: Comandos utilizados para testar o funcionamento do Firmament

Isso apenas assegurou que o Firmament estava respondendo, porém falta fazer testes com perdas e atrasos de pacotes para que se possa fazer uma melhor análise.

3.6 Restrições

Os arquivos fontes do Android só podem ser compilados no sistema operacional Linux ou Mac OS. Assim, pode-se utilizar uma máquina virtual no Windows para executar uma versão do Linux ou fazer uma instalação independente de uma distribuição do Linux.

Na hora de fazer a compilação cruzada para recompilar o kernel do Android para a arquitetura ARM, Advanced Risc Machine, foi encontrada dificuldade, pois foi necessário buscar material adequado para que fosse possível realizar esta etapa do trabalho. Esses mesmos problemas ocorreram na hora de fazer a compilação cruzada para o módulo da máquina virtual firm_vm do Firmament.

4 Conclusão

Por seu código ser aberto, o Android já tem uma grande quantidade de aplicações desenvolvidas para ele e esse número tende a crescer ainda mais. Assim, é necessário que essas aplicações sejam testadas para ver como elas se comportam na presença de falhas, o que permitiria aos desenvolvedores tomarem medidas para evitar transtornos aos usuários.

Firmament permite executar testes que explorem falhas que normalmente só aparecem em situações reais, pois só assim se terá certeza de que uma aplicação está realmente funcionando bem. Esse tipo de teste é uma das principais características desta ferramenta.

O passo a passo descrito aqui será de grande ajuda para quem desejar portar outras ferramentas para o ambiente Android, pois a compilação cruzada de módulos do Linux para o Android foi apresentada de forma objetiva e bem detalhada.

Referências

CANALYS. Disponível em: < http://www.canalys.com/index.html >. Acessado em: agosto, 2010.

AVIZIENIS, A.; Laprie, J.; Randell B.; Landwehr C. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE trans. on dependable and secure computing, V. 1, n. 1, jan 2004, pp 11-33.

RUSSEL, R.; Welte, H. (2002). Linux net filter hacking HOWTO. 2002. Disponível em: < http://www.netfilter.org/documentation/ >. Acessado em: abril, 2010.

DREBES, R. J. FIRMAMENT: Um módulo de injeção de falhas de comunicação para linux. 2005. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

SIQUEIRA, Tórgan Flores de ; Fiss, B. C. ; WEBER, Raul Fernando ; CECHIN, Sergio Luis ; WEBER, T. S. . Applying FIRMAMENT to test the SCTP communication protocol under network faults. In: Test Workshop, 2009. LATW '09. p. 1-6.

ANDROID. Disponível em: < http://www.android.com/ >. Acessado em: abril, 2010.

ANDROID-SOURCE. Disponível em: < http://source.android.com/source/download.html >. Acessado em: maio, 2010.

ANDROID-SDK. Disponível em: < http://developer.android.com/sdk/index.html >. Acessado em: abril, 2010.

ANDROID-SDK-INSTALLING. Disponível em: < http://developer.android.com/sdk/installing.html >. Acessado em: abril, 2010.

ADT-PLUGIN. Disponível em < http://developer.android.com/sdk/eclipse-adt.html >. Acessado em: abril, 2010.

HELLO-WORLD. Disponível em < http://developer.android.com/resources/tutorials/hello-world.html >. Acessado em: abril, 2010.

CROSS-COMPILATION. Disponível em: < http://wiki.strongswan.org/projects/strongswan/wiki/Android , http://my.opera.com/otaku_2r/blog/download-and-build-the-google-android e http://linuxclues.blogspot.com/2010/05/build-compile-linux-kernel-android.html >. Acessados em: maio, 2010.

ACKER, E. V. ; WEBER, T. S. ; CECHIN, Sergio Luis . Injeção de falhas para validar aplicações em ambientes móveis. In: Workshop de Testes e Tolerância a Falhas, 11., 2010, Gramado. SBC, 2010. p. 61-74.

LOADABLE-KERNEL-MODULE. Disponível em: < http://groups.google.com/group/android-kernel/msg/a43f2b1db7a5c3dc?pli1 >. Acessado em: maio, 2010.

ANDROID-DEBUG-BRIDGE. Disponível em: < http://developer.android.com/guide/developing/tools/adb.html >. Acessado em: abril, 2010.

APÊNDICE A

Aqui são mostrados códigos dos programas application Android, servidor e cliente, os quais constituem o sistema cliente-servidor.

```
package packbench.packbench;

import android.app.Activity;
import android.os.Bundle;
public class PackBench extends Activity {

	/* Chamado quando a atividade é criada pela primeira vez  */

	public void onCreate(Bundle savedInstanceState) {

		super.onCreate(savedInstanceState);

		/* Inicia o servidor (Server) e ele ficará esperando por pacotes do cliente */
		new Thread(new Server()).start();

		/* Espera o servidor Server iniciar */
		try {
			Thread.sleep(500);

		} catch (InterruptedException e)
		{
		}
	}
}
```

Figura A.1: Application Android

```
package packbench.packbench;

import java.io.ByteArrayInputStream;
import java.io.DataInputStream;
import java.net.DatagramPacket;
```

```
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketTimeoutException;
import java.util.BitSet;
import android.util.Log;

public class Server implements Runnable {

	public static final String SERVERIP = "10.0.2.15"; // endereço do emulador
	public static final int SERVERPORT = 4444;   //porta do servidor

	public void run() {
	try {
		/* Busca o nome do Servidor */
		InetAddress serverAddr = InetAddress.getByName(SERVERIP);

		Log.d("UDP", "S: Connecting...");
		/* cria um novo socket UDP */
		DatagramSocket socket = new DatagramSocket(SERVERPORT, serverAddr);

		/* define o tamanho máximo do pacote udp */
		byte[] buf = new byte[1472];

		/* Prepara o pacote UDP que irá conter o dado que será recebido */
		DatagramPacket packet = new DatagramPacket(buf, buf.length);
	do {
		/* Cada pacote a ser recebido possui um número de sequência */
		/* Os BitSets abaixo armazenam o status de cada pacote */

		/* Bits indicadores de pacotes recebidos */
		BitSet receivedPackets = new BitSet(50000);

		/* Bits indicadores de pacotes duplicados */
		BitSet duplicatedPackets = new BitSet(50000);

		/* Bits indicadores de pacotes recebidos fora de ordem */
		BitSet outOfOrderPackets = new BitSet(50000);

		Log.d("UDP", "S: Receiving...");
		/* Número de sequência do último pacote recebido */
		/* Utilizado para detectar pacotes fora de ordem */

		int currentPacketNumber = -1;
		/* Recebe o pacote UDP */
		{
			socket.receive(packet);
			ByteArrayInputStream bis = new ByteArrayInputStream(packet.getData());

			DataInputStream dis = new DataInputStream(bis);
			/* Lê o número de sequência do pacote */
			int numeroDeSequencia = dis.readInt();

			/* Verifica se pacote já foi recebido */
			if (receivedPackets.get(numeroDeSequencia))
				/* Marca pacote como duplicado */
				duplicatedPackets.set(numeroDeSequencia);
```

```
                    else

                            /* Marca pacote como recebido */
                            receivedPackets.set(numeroDeSequencia);

                    /* Caso um pacote de número de sequência mais alto já tenha sido recebido,
                                            Marca como fora de ordem */

                    if (numeroDeSequencia < currentPacketNumber)
                            outOfOrderPackets.set(numeroDeSequencia);

                    currentPacketNumber = Math.max(currentPacketNumber,
                                                    numeroDeSequencia);

            }
            /* Seta time-out para finalizar a recepção de pacotes */
            socket.setSoTimeout(3000);

            /* Realiza as mesmas operações citadas anteriormente */
            int pacotesRecebidos = 1;
            try {
                    do {
                            socket.receive(packet);
                            ByteArrayInputStream bis = new ByteArrayInputStream(
                                                            packet.getData());
                            DataInputStream dis = new DataInputStream(bis);
                            int numeroDeSequencia = dis.readInt();

                            if (receivedPackets.get(numeroDeSequencia))
                                    duplicatedPackets.set(numeroDeSequencia);
                            else
                                    receivedPackets.set(numeroDeSequencia);
                            if (numeroDeSequencia < currentPacketNumber)
                                    outOfOrderPackets.set(numeroDeSequencia);
                            currentPacketNumber = Math.max(currentPacketNumber,
                                                            numeroDeSequencia);
                            pacotesRecebidos++;

                    } while (true);

            } catch (SocketTimeoutException e) {
                    /* Emite Relatório no Log do Android */
                    Log.d("UDP", "S: Done. Total received:" + pacotesRecebidos);
                    Log.d("UDP", "S: Done. Duplicated:"+ duplicatedPackets.cardinality());
                    Log.d("UDP", "S: Done. Received:"+ receivedPackets.cardinality());
                    Log.d("UDP", "S: Done. Out of order:"+ outOfOrderPackets.cardinality());
                            socket.setSoTimeout(0);
                    }
                    } while (true);

            } catch (Exception e) {
                    Log.e("UDP", "S: Error", e);
            }
    }
}
```

Figura A.2: Código do programa servidor

```
import java.io.ByteArrayOutputStream;
import java.io.DataOutputStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class SendMessages {

	static int packetsToSend=50000; /* Total de mensagens a serem enviadas */
	public static void main(String[] args) {
		try {
			InetAddress serverAddr = InetAddress.getByName("localhost");
			DatagramSocket socket = new DatagramSocket();

			/* Prepara os dados a serem enviados */
			byte[] buf;
			DatagramPacket packet;
			ByteArrayOutputStream bos;
			DataOutputStream dos;
			byte[] numeroSequencia;
			for (int i = 0; i < packetsToSend; i++) {
				buf = new byte[1472];
				/* Cria pacote UDP com dado e destino(url+porta) */
				packet = new DatagramPacket(buf, buf.length, serverAddr, 5000);

				bos = new ByteArrayOutputStream();
				dos = new DataOutputStream(bos);
				/* Escreve número de sequência no pacote */
				dos.writeInt(i);
				numeroSequencia = bos.toByteArray();
				System.arraycopy(numeroSequencia, 0, buf, 0,
						numeroSequencia.length);

				/* Envia o pacote*/
				socket.send(packet);

				/* Aguarda 5 milisengundos para envio do próximo pacote */
				Thread.sleep(5, 1000);

			}

			/* Fecha o socket*/
			socket.close();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

Figura A.3: Código do programa cliente

APÊNDICE B

Aqui são apresentados os faultlets utilizados para o acionamento do mecanismo cão de guarda e para a realização da injeção de falhas com taxa de 25% de descarte, duplicação e atraso de pacotes.

```
SET 9 R0                ;deslocamento do pacote a ser lido
READB R0 R1             ;R1=pacote[R0](pacote[9])
SET 17 R0               ;protocolo a ser selecionado
SUB R0 R1               ;(6 para TCP, 17 para UDP)
JMPZ R1 UDP             ;se igual a (UDP) vai para UDP
ACP                     ;se diferente, o pacote é aceito

UDP:
SET 16 R0               ;lê word (4 bytes) na posição 16 contendo IP
READW R0 R1             ;(12 é a origem e 16 o destino)
SET 0x0a00020f R0       ;verifica se pacote tem como destino o android
SUB R0 R1               ;ou seja, o IP: 10.0.2.15
JMPZ R1 loop1           ;caso sim, então vai para LOOP1
ACP                     ;se não, o pacote é aceito

loop1: JMP loop2        ;força a execução de um loop infinito
                        ;para que o mecanismo do watch-dog
loop2: JMP loop1        ;(cão de guarda) entre em ação
```

Figura B.1: Faultlet executado para acionar o mecanismo cão de guarda

```
SET 9 R0                ;lê o byte na posição 9 do pacote
READB R0 R1             ;que contém o tipo de protocolo
SET 17 R0               ;verifica se é pacote UDP:
SUB R0 R1               ;(6 para TCP, 17 para UDP)
JMPZ R1 UDP             ;se igual a (UDP) vai para UDP
ACP                     ;se diferente, o pacote é aceito
```

```
UDP:
SET 16 R0           ;lê word (4 bytes) na posição 16 contendo IP
READW R0 R1         ;(12 é a origem e 16 o destino)
SET 0x0a00020f R0   ;verifica se pacote tem como destino o android
SUB R0 R1           ;ou seja, o IP: 10.0.2.15
JMPZ R1 DESC        ;caso sim, então vai pro DESC
ACP                 ;se não, o pacote é aceito

DESC:
SET 1000 R0         ;módulo do nº a ser sorteado
RND R0 R1           ;sorteio
SET 500 R0          ;soma 500 com o nº sorteado
ADD R0 R1           ;probabilidade de ser negativo = 25%
JMPN R1 DROP        ;se for negativo desvia para DROP
SET 1 R0            ;senão
ADD R0 R10          ;soma 1 ao nº total de pacotes recebidos
ACP                 ;aceita o pacote

DROP:
SET 1 R0            ;
ADD R0 R11          ;soma 1 ao nº de pacotes perdidos
DRP                 ;descarta o pacote
```

Figura B.2: Faultlet utilizado para fazer o descarte de pacotes

```
SET 9 R0            ;lê o byte na posição 9 do pacote
READB R0 R1         ;que contém o tipo de protocolo
SET 17 R0           ;verifica se é pacote UDP:
SUB R0 R1           ;(6 para TCP, 17 para UDP)
JMPZ R1 UDP         ;caso seja, vá para rotina UDP
ACP                 ;se diferente, o pacote é aceito

UDP:
SET 16 R0           ;lê word (4 bytes) na posição 16 contendo IP
READW R0 R1         ;(12 é a origem e 16 o destino)
SET 0x0a00020f R0   ;verifica se pacote tem como destino o android
SUB R0 R1           ;ou seja, o IP: 10.0.2.15
JMPZ R1 DESC        ;caso sim, então vai pro DESC
ACP                 ;se não, o pacote é aceito

DESC:
SET 1000 R0         ;módulo do nº a ser sorteado
```

```
RND R0 R1               ;sorteio
SET 500 R0              ;soma 500 com o nº sorteado
ADD R0 R1               ;probabilidade de ser negativo = 25%
JMPN R1 DUPL            ;se for negativo desvia para DUPL
SET 1 R0                ;senão
ADD R0 R10              ;soma 1 ao registrador 10
ACP                     ;que contém o numero total de pacotes

DUPL:
SET 1 R0                ;
SET 2 R1                ;
ADD R0 R11              ;soma 1 ao nº de pacotes duplicados
ADD R1 R10              ;soma 2 ao nº total de pacotes
DUP                     ;duplica o pacote selecionado
```

Figura B.3: Faultlet utilizado para fazer a duplicação de pacotes

```
SET 9 R0                ;lê o byte na posição 9 do pacote
READB R0 R1             ;que contém o tipo de protocolo
SET 17 R0               ;verifica se é pacote UDP:
SUB R0 R1               ;(6 para TCP, 17 para UDP)
JMPZ R1 UDP             ;se igual a (UDP) vai para UDP
ACP                     ;se diferente, o pacote é aceito

UDP:
SET 16 R0               ;lê word (4 bytes) na posição 16 contendo IP
READW R0 R1             ;(12 é a origem e 16 o destino)
SET 0x0a00020f R0       ;verifica se pacote tem como destino o android
SUB R0 R1               ;ou seja, o IP: 10.0.2.15
JMPZ R1 DESC            ;caso sim, então vai pro DELAY
ACP                     ;se não, o pacote é aceito

DESC:
SET 1000 R0             ;módulo do nº a ser sorteado
RND R0 R1               ;sorteio
SET 500 R0              ;soma 500 com o nº sorteado
ADD R0 R1               ;probabilidade de ser negativo = 25%
JMPN R1 DELAY           ;se for negativo desvia para DELAY
SET 1 R0                ;senão
ADD R0 R10              ;soma 1 ao nº total de pacotes sem atraso
ACP                     ;aceita o pacote
```

```
DELAY:
SET 1 R0                ;
ADD R0 R11              ;soma 1 ao nº de pacotes atrasados
SET 15 R0               ;
DLY R0                  ;atrasa o pacote em 15 milisegundos
```

Figura B.4: Faultlet utilizado para fazer o atraso de pacotes

```
SET 1000 R0             ;módulo do nº a ser sorteado
RND R0 R1               ;sorteio
SET 800 R0              ;adiciona 800 com o nº sorteado
ADD R0 R1               ;probabilidade de ser negativo = 10%
JMPN R1 DROP            ;se for negativo desvia para DROP
SET 1 R0                ;senão
ADD R0 R10              ;soma 1 ao nº de pacotes recebidos
ACP                     ;aceita o pacote

DROP:
SET 1 R0                ;
ADD R0 R11              ;soma 1 ao nº de pacotes perdidos
DRP                     ;descarta o pacote
```

Figura B.5: Faultlet para o descarte de pacotes nas aplicações do Android