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

HEINRICH FELIX MARMITT

Modelos para Injeção de Falhas em Ambientes Móveis

Trabalho de Diplomação

Profª. Drª. Taisy Silva Weber
Orientador

Prof. Dr. Sérgio Luis Cechin
Co-orientador

Porto Alegre, dezembro de 2010

“Education is what survives
when what has been learned
has been forgotten.”
— B. F. SKINNER (1904 - 1990)

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ARP	Address Resolution Protocol
BSS	Basic Service Set
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
ESS	Extended Service Set
FIRMAMENT	Fault Injection Relocatable Module for Advanced Manipulation and Evaluation of Network Transports
FIRMASM	FIRMAMENT Assembly
FIRM_VM	FIRMAMENT Virtual Machine
ICMP	Internet Control Message Protocol
IEEE	Institute of Eletrical and Eletronics Engineers
IETF	The Internet Engineering Task Force
IPv4	Internet Protocol, versão 4
IPv6	Internet Protocol, versão 6
MR-OSI	Open System Interconnection Reference Model
OHA	Open Handset Alliance
WLAN	Wireless Local Area Network

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A validação de softwares por Injeção de Falhas necessita de modelos que representem a carga de falhas com a maior precisão possível. Neste trabalho foi realizada uma revisão sobre os modelos conhecidos de falhas que ocorrem em uma rede IEEE 802.11. Com estes resultados, foram implementadas duas descrições de carga de falhas. Foram executados testes como prova de conceito utilizando as descrições implementadas.

Os resultados apresentam dois modelos adequados para a validação de softwares, com baixa complexidade computacional e baixa intrusividade.

Palavras-chave: Ambientes Móveis, Injeção de Falhas, Redes de Computadores.

Models for Fault Injection in Mobile Networks

ABSTRACT

Fault Injection needs models which accurately describes the fault load used for software validation. This work presents a review about the fault models currently described for an IEEE 802.11 network. With this review, two fault load descriptions were implemented and submitted to testing as a proof of concept.

The results present two adequate models for software validation, with low computational complexity and low intrusivity.

Keywords: Mobile Networks, Fault Injection, Computer Networks.

1 INTRODUÇÃO

1.1 Motivação

A computação móvel está cada vez mais presente no nosso cotidiano. Grande parte dos aparelhos móveis (notebooks, netbooks e smartphones) já saem de fábrica equipados com um ou mais dispositivos para a comunicação em rede. Com a popularização destes aparelhos, a quantidade de aplicativos que necessitam atender a critérios de dependabilidade aumenta. Estes aplicativos podem utilizar técnicas de tolerância a falhas, que precisam ser devidamente testadas.

A dependabilidade de um sistema é definida como a habilidade de evitar defeitos de serviços que são mais frequentes ou mais severos que o aceitável. Um dos meios para atingir esse critério é o uso de mecanismos de tolerância a falhas (AVIZIENIS et al., 2004). Contudo, os mecanismos de tolerância a falhas precisam ser testados e é inviável que o teste seja dependente da ocorrência natural de uma falha específica. A injeção de falhas é um dos métodos aceitos para testar esses mecanismos e está consolidado na literatura (ARLAT et al., 1990) (HSUEH; TSAI; IYER, 1997).

O Grupo de Tolerância a Falhas da Universidade Federal do Rio Grande do Sul (UFRGS) tem mostrado interesse na união destes dois segmentos, tolerância a falhas e computação móvel. Para isso, o injetor de falhas criado pelo grupo (FIRMAMENT, (DREBES et al., 2006)) foi portado para o Android (ACKER; WEBER; CECHIN, 2010). O Android é uma plataforma desenvolvida pela Open Handset Alliance(OHA) para dispositivos móveis e possui um sistema operacional baseado no kernel Linux versão 2.6 (ANDROID-ADG, 2010).

1.2 Objetivos

Este trabalho tem como objetivo estudar e implementar modelos adequados para a injeção de falhas de comunicação em ambientes móveis. Estes serão utilizados posteriormente para a validação de aplicações desenvolvidas para o sistema, em conjunto com o injetor de falhas FIRMAMENT. É esperado que os modelos reproduzam as falhas de comunicação que ocorrem na rede IEEE 802.11 com a maior precisão possível, com a ressalva de manter um baixo nível de intrusividade.

1.3 Organização

Este documento será organizado seguindo a seguinte ordem. O capítulo 2 é composto de uma revisão sobre os principais conceitos teóricos abordados neste trabalho. O capítulo 3 apresentará a plataforma Android, o injetor de falhas utilizado no projeto e os métodos

de entrada para os modelos de falhas utilizados pelo injetor. O capítulo 4 apresentará os modelos matemáticos existentes na literatura para falhas em redes IEEE 802.11. O capítulo 5 contém as implementações das cargas de falhas e os testes executados para verificar se atendem as funcionalidades esperadas. Finalmente, serão apresentadas as conclusões obtidas com este trabalho.

2 CONCEITOS

Neste capítulo será apresentado um resumo dos principais conceitos teóricos utilizados neste trabalho. A primeira seção trata dos conceitos relacionados com tolerância a falhas e a seção seguinte apresenta os conceitos relacionados com redes de computadores.

2.1 Tolerância a Falhas e Injeção de Falhas

2.1.1 Tolerância a Falhas / Dependabilidade

O termo *tolerância a falhas* foi proposto por Avizienis em 1967, com o objetivo de nomear toda a área de pesquisa que estuda o comportamento de sistemas computacionais sujeitos à ocorrência de falhas. Com o tempo, ele foi substituído pelo termo *dependabilidade* (WEBER, 2003). A dependabilidade pode ser definida como a habilidade de evitar defeitos em serviços que são mais frequentes ou mais severos que o aceitável (AVIZIENIS et al., 2004). Deste modo, a área de dependabilidade se dedica a estudar os padrões de falhas de serviços e métodos utilizados para que os sistemas tenham um padrão de funcionamento adequado mesmo com a ocorrência destas falhas.

A dependabilidade é composta de cinco atributos principais: disponibilidade (*availability*), confiabilidade (*reliability*), segurança funcional (*safety*), integridade (*integrity*) e facilidade de manutenção (*maintainability*). Também podem ser definidos quatro principais grupos de métodos para garantir a dependabilidade de um sistema: prevenção de falhas, tolerância a falhas, remoção de falhas e prevenção de falhas (AVIZIENIS et al., 2004). Aplicações que necessitem atender a critérios de dependabilidade utilizam um ou mais destes métodos, os quais precisam ser devidamente testados e validados.

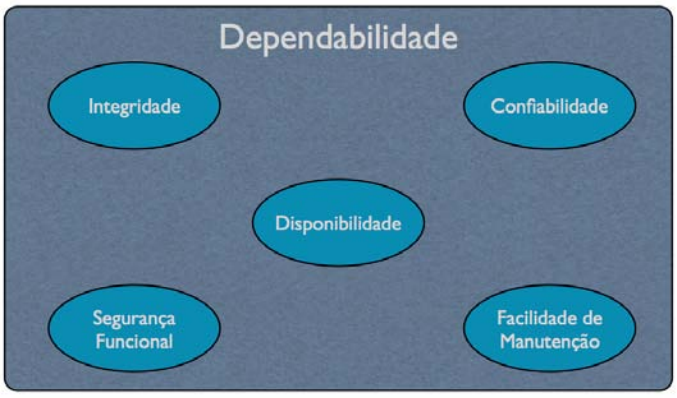

Figura 2.1: Dependabilidade e seus atributos

Contudo, a validação de métodos que garantem a dependabilidade é uma tarefa complexa. Algumas falhas possuem seu padrão de ocorrência bem definido e previsível, mas a grande maioria das falhas ocorrem em tempos aleatórios e, ocasionalmente, possuem uma probabilidade de ocorrência muito baixa. Por esta razão, utilizam-se métodos como modelamento matemático do sistema, simulação da operação ou injeção de falhas.

2.1.2 Injeção de Falhas

Arlat define a injeção de falhas como a introdução deliberada de falhas em um sistema e que é um método aplicável sempre que as noções de falhas e/ou erros forem motivos de preocupação no desenvolvimento de um sistema (ARLAT et al., 1990). Arlat também define um conjunto de atributos principais que podem ser utilizados para caracterizar a injeção de falhas. Este conjunto é composto pelos seguintes atributos: um conjunto de falhas (F), um conjunto de ativações ou dados de entrada (A), um conjunto de dados de saída (R) e um conjunto de medidas derivadas da injeção (M). Elas podem ser facilmente lembradas com o acrônimo *FARM*.

Segundo Hsueh, uma das técnicas de injeção de falhas é baseada em simulações, que assume que os erros ocorrem seguindo uma determinada distribuição de probabilidade (HSUEH; TSAI; IYER, 1997). Esta técnica é útil para avaliar a eficácia dos mecanismos de tolerância a falhas e a dependabilidade do sistema, contudo é importante que as entradas reproduzam fielmente o modelo de falhas do sistema. A definição da distribuição de falhas em um sistema é normalmente realizada de forma empírica, ou seja, o sistema-alvo (ou similar) é observado em funcionamento e procuram-se modelos matemáticos que reproduzam a distribuição destas falhas no tempo com uma confiabilidade aceitável.

No capítulo 4 serão apresentados alguns modelos já existentes para falhas em redes sem fio, que servirão de base para as implementações realizadas no capítulo 5.

2.2 Redes de Computadores

2.2.1 Modelo MR-OSI

O modelo MR-OSI (*Open System Interconnection Reference Model*) foi proposto inicialmente no final da década de 70 (FOROUZAN, 2008). Este modelo foi desenvolvido para facilitar a compreensão e o projeto de uma arquitetura de redes. É composto por uma estrutura em camadas, relacionadas entre si, que definem como a transferência de informações em uma rede deve ocorrer. A figura 2.2 mostra as 7 camadas do modelo.

As camadas do modelo MR-OSI podem ser agrupadas em três subgrupos: Camadas de suporte à rede (Física, Enlace e Rede), Camada de transporte (Transporte) e Camadas de suporte ao usuário (Sessão, Apresentação e Aplicação). O subgrupo de suporte à rede é o de maior interesse neste trabalho, e suas camadas serão detalhadas abaixo.

A camada Física é responsável pelas funções necessárias para o transporte do fluxo de bits através do meio físico. Entre outros, esta camada define como os bits serão representados (codificação), qual o tipo do meio de transmissão, qual a taxa em que os dados serão transmitidos e qual a topologia física da rede.

A camada de Enlace é responsável por entregar os dados conforme o endereçamento físico e filtrar alguns dos erros que ocorrem durante a transmissão do fluxo de bits. Assim, fica sob responsabilidade da camada de enlace a divisão do fluxo de bits em pacotes, o endereçamento físico destes pacotes, controle de erros, controle de fluxo.

A camada de Rede é responsável pela entrega dos pacotes entre o dispositivo de ori-

Figura 2.2: Organização das camadas do modelo MR-OSI

gem e o dispositivo de destino, seguindo a visão lógica da rede. A camada de rede executa o endereçamento lógico e o roteamento dos pacotes para que eles cheguem ao local correto.

2.2.2 Protocolo IP

O Internet Protocol (IP) é o protocolo de rede utilizado pela sistema TCP/IP, que domina a Internet atualmente. O IP é um protocolo de datagramas sem conexão e não-confiável (FOROUZAN, 2008). Desta maneira, o IP é definido como um serviço de entrega *best-effort*, ou seja, faz o melhor possível para entregar o pacote, mas fornece nenhuma garantia.

Como o IP é um protocolo sem conexão, cada datagrama é tratado de forma independente, podendo seguir por rotas distintas até o destino. Essa característica pode ocasionar em datagramas sendo entregues em uma ordem diferente da qual foram enviados. Deste modo, se o usuário estiver interessado em garantir confiabilidade ou ordem de entrega, deve utilizar protocolos das camadas superiores que disponibilizem esses serviços.

O datagrama IP é formado por duas partes: Cabeçalho e Dados. O cabeçalho tem comprimento variável entre 20 e 60 bytes e contém as informações necessárias para o roteamento e a entrega do pacote (FOROUZAN, 2008). A figura 2.3 mostra um diagrama de um cabeçalho do IPv4.

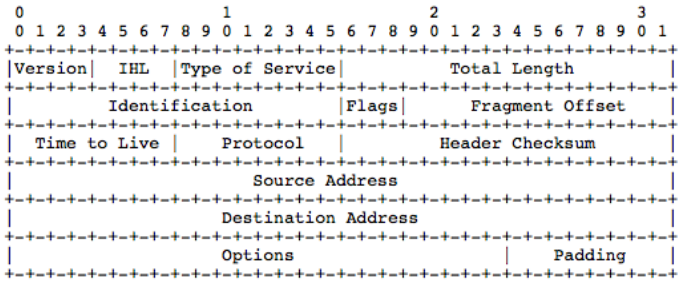

Figura 2.3: Cabeçalho IPv4 (IETF, 1981)

Dois campos serão utilizados posteriormente nas implementações dos modelos de

falhas: Protocolo (*Protocol*, byte 9) e Endereço de Destino (*Destination Address*, byte 16). O campo protocolo contém o tipo de protocolo que está sendo carregado na área de dados do datagrama IP. Os tipos mais comuns são: ICMP (valor 1), TCP (valor 6) e UDP (valor 17). O campo Endereço de Destino contém o endereço IP (lógico) do dispositivo destinatário.

2.2.3 *Wireless* - IEEE 802.11

O IEEE 802.11 é um padrão para comunicação sem fio definido pelo Institute of Electrical and Electronics Engineers (IEEE) que abrange as camadas física e de enlace do modelo MR-OSI. Este padrão é amplamente utilizado em redes locais sem fio (*Wireless Local Area Networks*, WLANs). A recomendação do IEEE define dois tipos de serviços: *Basic Service Set* - BSS (conjunto básico de serviços) e *Extended Service Set* - ESS (conjunto estendido de serviços) (FOROUZAN, 2008).

O conjunto básico de serviços (BSS) é a base de uma WLAN. Ele é formado por estações sem fio fixas ou móveis e, opcionalmente, por um ponto de acesso (*Access Point*, AP). Uma BSS sem um AP é chamada de *rede ad-hoc*, enquanto uma BSS com um AP é chamada de *rede de infra-estrutura*.

O conjunto estendido de serviços (ESS) é formado por duas ou mais BSSs com APs (infra-estrutura). Estes APs são interconectados por meio de um sistema de distribuição, normalmente utilizando uma conexão com fios. O padrão IEEE 802.11 permite que o sistema de distribuição utilize qualquer tipo de rede padrão IEEE, como o IEEE 802.3 (Ethernet), por exemplo. Deste modo, o ESS permite a construção de arquiteturas de rede maiores, onde o sistema de distribuição se encarrega da comunicação entre duas estações, se elas não estiverem em uma mesma BSS.

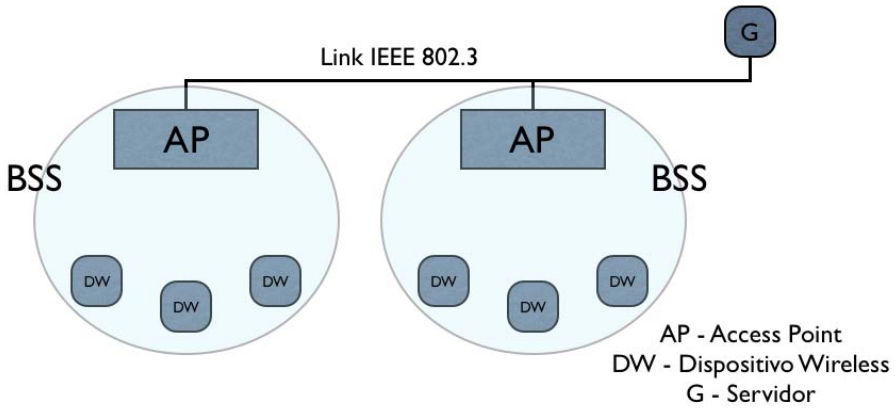

Figura 2.4: Uma ESS formada por duas BSSs unidas por uma conexão Ethernet

O IEEE 802.11 opera em diferentes frequências de transmissão e utilizando diferentes métodos de multiplexação e modulação. O IEEE 802.11a utiliza a frequência de 5,725 GHz e alcança taxas de transmissão que variam entre 6 e 54 Mbps. O IEEE 802.11b utiliza a frequência de 2.4 GHz e trabalha com taxas de transmissão de 5,5 Mbps e 11 Mbps. O IEEE 802.11g também utiliza a frequência de 2.4 GHz, alcança taxas de transmissão de até 54 Mbps e pode operar em um modo de compatibilidade com o IEEE 802.11b.

Como o IEEE 802.11 utiliza as faixas de frequências disponíveis para aplicações ISM (industrial, científica e médica), vários outros aparelhos de rádio-frequência podem gerar interferência em suas transmissões. Por exemplo, fornos de micro-ondas domésticos operam na faixa de 2.45 GHz e alguns telefones sem fio operam em 2.4 GHz.

Figura 2.5: Faixas de frequência ISM (industrial, científica, médica)

Uma característica importante das redes IEEE 802.11 é o seu mecanismo de controle de acesso ao meio. Este mecanismo utiliza um protocolo chamado CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*). Este protocolo implementa funções para que as colisões de transmissão sejam evitadas. Como uma rede sem fio não é capaz de uma detecção eficaz de colisões durante a transmissão de um pacote, a colisão só seria detectada ao término da transmissão completa deste pacote. Esta limitação prejudica o desempenho da rede, aumentando a importância de que as colisões sejam evitadas.

O protocolo CSMA/CA define que cada estação começará cada transmissão verificando se o canal de transmissão está livre. É um mecanismo similar ao utilizado por redes Ethernet (IEEE 802.3), mas que possue um comportamento de evitar colisões ao invés de detectá-las. Ao contrário do mecanismo CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*) utilizado na rede Ethernet, a transmissão não se iniciará imediatamente. O protocolo impõe que o dispositivo espere por um tempo chamado de DIFS (*Distributed Inter-frame Space*) e por um intervalo aleatório (cujos limites são definidos pelo tamanho da janela de contenção). Durante este tempo o dispositivo continua ”ouvindo”o meio e se este continuar livre até o final da contagem a transmissão pode ser executada.

Se a estação receptora não enviar um pacote sinalizando que a transmissão foi recebida corretamente durante um intervalo *timeout* após a recepção, é interpretado que o pacote sofreu uma colisão durante o envio. Neste caso, as estações envolvidas na colisão entram em um estado de *backoff*. Neste estado o dispositivo incrementa o tamanho da sua janela de contenção exponencialmente. Caso o tamanho da janela ultrapasse um limite definido, a transmissão é abortada e o pacote é descartado.

Embora o protocolo CSMA/CA faça um esforço extra em relação ao protocolo CSMA/CD para evitar colisões, estas ainda acontecem, contudo com uma probabilidade menor. O CSMA/CA ainda oferece um segundo mecanismo, opcional, que consiste em uma negociação da utilização do meio através de uma negociação extra utilizando os pacotes RTS/CTS.

O mecanismo RTS/CTS difere do CSMA/CA padrão a partir do momento em que o dispositivo percebe que o meio está disponível. Ele então espera pelo tempo DIFS e pelo tempo aleatório definido pela janela de contenção. Em seguida, envia um pacote de controle (RTS - *Request to Send*) que contém uma requisição para o envio do pacote.

Quando o pacote de controle RTS for enviado corretamente, o destinatário da mensagem (AP, em uma rede de infra-estrutura) envia um outro pacote de controle CTS (*Clear to Send*) para todas as estações que estiverem ao seu alcance, sinalizando que o meio deve ser reservado por um espaço de tempo definido para que a estação que requisitou possa enviar sua mensagem sem colisões. Se o pacote de resposta (CTS) não for recebido corretamente no tempo esperado, o dispositivo emissor considera que o pacote RTS tenha sofrido uma colisão e passa para o estado de *backoff*.

Esta negociação, no entanto, não garante que nenhuma colisão vá ocorrer ou que a mensagem será recebida corretamente pelo destinatário. Alguns fatores como estações

fora do alcance do AP que não receberam o pacote de controle CTS ou interferências físicas durante a transmissão da mensagem podem fazer com que ela sofra um atraso ou até que seja perdida completamente.

No capítulo 4 serão apresentados os modelos que tentam prever estatisticamente se uma mensagem foi perdida ou sofreu um atraso na sua entrega.

Se o leitor desejar um maior aprofundamento, a especificação completa do padrão pode ser encontrada na literatura especializada (IEEE, 2007).

2.2.4 *Bluetooth* - IEEE 802.15

O *Bluetooth* é uma tecnologia de WLAN de curto alcance, especificada pelo padrão IEEE 802.15.1, que define uma PAN (*Personal Area Network*). É uma tecnologia muito difundida em telefones celulares, notebooks e outros dispositivos móveis (FOROUZAN, 2008). A rede Bluetooth é uma rede que opera apenas no modo *ad-hoc*, chamada de piconet. Uma piconet pode ter até 8 estações, onde apenas uma é primária e as demais são chamadas de estações secundárias. Entretanto uma estação secundária de uma piconet pode ser a estação primária de outra piconet, criando assim uma configuração chamada de scatternet.

A pilha de protocolos utilizada pelo *Bluetooth* é diferente da pilha utilizada na internet (TCP/IP). A versão 1.2 do padrão *Bluetooth* alcançava taxas de transmissão de até 1 Mbps. Esta taxa de transmissão chega a 24 Mbps nas versões mais recentes (com a ressalva de que essa velocidade é alcançada utilizando um link IEEE 802.11 para a transmissão dos dados). O *Bluetooth* opera na frequência de 2.4 GHz, utilizando multiplexação por frequência e tempo para executar a transmissão dos dados.

Como o *Bluetooth* opera na mesma faixa de frequências dos padrões IEEE 802.11b e IEEE 802.11g, ele pode ocasionar interferências nas comunicações.

Devido a baixa taxa de transmissão utilizando o *Bluetooth* em si e pelo baixo alcance dos transmissores, este padrão não é tão atrativo para a validação em relação ao padrão IEEE 802.11.

3 SISTEMA E FERRAMENTA

Este capítulo apresenta o sistema (Android) que executa as aplicações-alvo da injeção de falhas e as ferramentas utilizadas para este fim.

3.1 Android

O sistema Android (ANDROID-ADG, 2010) é uma plataforma para ambientes móveis. Desenvolvido inicialmente pela Google, atualmente o projeto é coordenado pela Open Handset Alliance (OHA). A OHA é formada por um conjunto de empresas de comunicação e tecnologia que tem por objetivo criar padrões para a telefonia móvel.

O Android é baseado no núcleo Linux e possibilita que desenvolvedores independentes criem aplicações que utilizem diretamente todas as funcionalidades do aparelho móvel. Deste modo, as aplicações têm acesso direto a funcionalidades como: fazer chamadas telefônicas, utilizar a câmera fotográfica, utilizar a rede sem fio, entre outras. Em maio de 2010, foi lançada a versão 2.2 rev 1, que utiliza o kernel Linux 2.6.32.

Segundo a IDC (*International Data Corporation*), o Android alcançou cerca de 16.8% do mercado de sistemas operacionais de smartphones em 2010, com a previsão de ocupar cerca de 24% do mercado em 2014 (IDC, 2010). O grande crescimento de participação do mercado do Android, a expansão do mercado de smartphones e a base em código aberto da plataforma servem de grande incentivo para que muitas aplicações sejam criadas por diferentes desenvolvedores nos próximos anos. A figura 3.1 mostra os dados da AdMob sobre a participação no mercado mundial dos principais sistemas operacionais de smartphones entre Maio de 2009 e Maio de 2010.

Verificando alguns aparelhos disponíveis no mercado em 2010 (como o Nexus One, Motorola Droid e HTC Hero), é possível constatar que a grande maioria deles possue mais de uma interface de comunicação. O padrão IEEE 802.11 e o *Bluetooth* (IEEE 802.15.1) são largamente utilizados. Pelas popularidade das redes IEEE 802.11 e pelas limitações do *Bluetooth* (citadas na seção 2.2.4), este trabalho se restringirá aos modelos de falhas para as interfaces IEEE 802.11.

3.2 Netfilter

O Netfilter é um componente da *framework* para filtragem de pacotes disponível nos núcleos Linux 2.4.x e 2.6.x (RUSSELL; WELTE, 2002). Ele consiste em um conjunto de ganchos que permitem que módulos desenvolvidos para o núcleo Linux possam registrar chamadas de funções à pilha de rede do núcleo. Esta função é chamada sempre que um pacote for capturado por este gancho. Entre os usos do Netfilter se incluem: construção

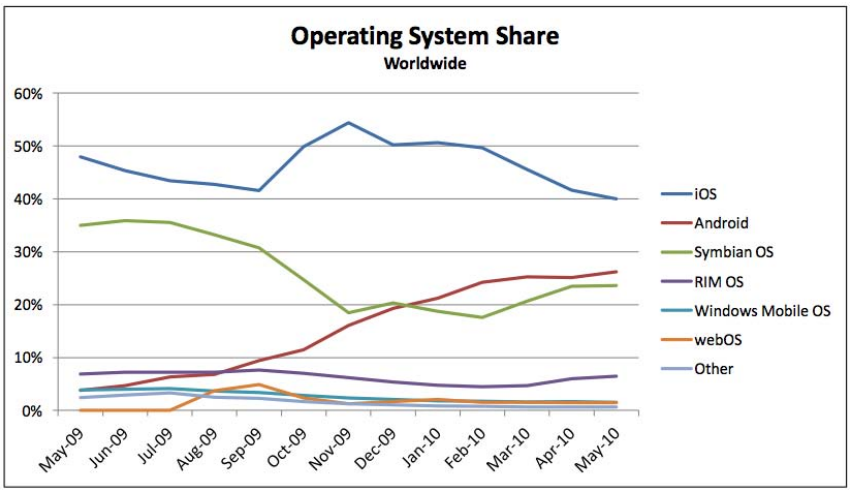

Figura 3.1: Participação no mercado dos principais sistemas operacionais para smartphones. (ADMOB, 2010)

de *Firewalls*, NAT (*Network Address Translation*, Tradução de Endereços de Rede) e alteração de pacotes para *QoS* ou injeção de falhas.

Estão disponíveis ganchos para os protocolos IPv4, IPv6, DECnet e ARP. Para o protocolo IPv4 estão disponíveis ganchos antes e depois do roteamento do pacote (PRE_ROUTING e POST_ROUTING), ganchos para quando o pacote é destinado à máquina que está executando o Netfilter (LOCAL_IN), gancho para quando o pacote é direcionado para uma interface diferente (FORWARD) e por fim, gancho que captura pacotes criados localmente (LOCAL_OUT).

3.3 FIRMAMENT

O FIRMAMENT (DREBES et al., 2006) é um injetor de falhas que utiliza a interface de programação do Netfilter, apresentado na seção anterior. Ele foi projetado para ser um injetor de falhas por software, atuando principalmente sobre o protocolo de rede IP. Segundo Drebes, devido às características do protocolo IP (não-confiável, sem retransmissão), este é um ponto ideal para a injeção de falhas de comunicação, já que falhas ocorridas nas camadas inferiores (física e enlace) são refletidas diretamente no comportamento dos datagramas IP.

Uma das principais preocupações em relação a injetores de falhas por software se refere à intrusividade do injetor no sistema. Assim, o injetor de falhas será eficiente quando tiver o menor impacto possível no desempenho do sistema. Um dos modos de alcançar essa eficiência é estar localizado o mais próximo possível do hardware.

Uma importante preocupação dos mecanismos de injeção de falhas diz respeito a sua intrusividade. A intrusividade espacial se refere à forma onde o sistema necessita ser alterado para a inclusão do mecanismo de injeção. A intrusividade temporal está relacionada ao tempo de processamento necessário para que as rotinas de injeção de falhas sejam executadas. Como a intrusividade não pode ser evitada, é responsabilidade do injetor de falhas buscar a menor intrusividade possível.

O FIRMAMENT procura reduzir a sua intrusividade se localizando no nível do sistema operacional. Na prática, o injetor alcança a baixa intrusividade por ser um módulo disponível para o núcleo Linux. Além disso, essa abordagem permite que o injetor seja

executado com o mínimo de alterações possível no sistema original. O FIRMAMENT pode ser utilizado em várias arquiteturas de hardware distintas, já que ele utiliza apenas funções de alto nível disponíveis no núcleo Linux. São utilizados apenas os ganchos do Netfilter PRE_ROUTING e POST_ROUTING relacionados com os protocolos IPv4 e IPv6.

Quando o pacote é interceptado, a máquina virtual FIRM_VM executa micro-programas chamados de *faultlets*, que expressam os cenários de falhas utilizados nos testes. Após a execução, a máquina virtual retorna à pilha de protocolos a ação a ser realizada sobre o pacote. Entre as ações possíveis encontram-se: NF_ACCEPT (o pacote continua pelo caminho normal), NF_DROP (o pacote é descartado) e NF_QUEUE (o pacote é enviado para uma função definida pelo usuário). A ação NF_QUEUE é utilizada pelo FIRMAMENT para emular atrasos do pacote.

Para a descrição da carga de falhas a ser emulada, o FIRMAMENT utiliza aplicações chamadas de *faultlets*. Estes *faultlets* devem ser escritos utilizando a linguagem FIRMASM. Para a correta interpretação pela FIRM_VM eles devem ser traduzidos para o formato binário, utilizando um montador. A saída binária do montador pode então ser copiada para um dos arquivos virtuais de regras, localizados em */proc/net/firmament/rules*, onde serão executados pela máquina virtual quando um pacote for capturado. (ADMOB, 2010)

O FIRMAMENT já foi portado com sucesso para o sistema Android (ACKER; WEBER; CECHIN, 2010) (DOBLER; WEBER; CECHIN, 2010).

4 MODELOS DE FALHAS

Segundo Boggia, as WLAN's possuem naturalmente uma tendência de apresentar erros mais frequentemente que redes com fio (IEEE 802.3) e um dos principais desafios consiste em modelar o padrão dos erros que ocorrem em forma de rajadas (BOGGIA; CAMARDA; D'ALCONZO, 2009). Isto acontece principalmente devido aos fenômenos físicos que afetam o meio de transmissão, como atenuação do sinal e interferências externas.

Neste trabalho duas classes de modelos serão estudadas e, posteriormente, implementadas no injetor de falhas FIRMAMENT para fornecer a prova de conceito. A primeira é uma classe de perda de pacotes (*packet loss*), que é baseada no modelo de Gilbert-Elliot para canais de rádio-frequência (GILBERT, 1960) (ELLIOTT, 1963). A outra classe trata dos atrasos de entrega de pacotes, que ocorrem principalmente por colisões (RAPTIS; VITSAS; PAPARRIZOS, 2009).

4.1 Perda de pacotes

Khayam realizou testes experimentais utilizando uma rede IEEE 802.11b (KHAYAM et al., 2003). Neste estudo o autor propõe que o modelo básico de Gilbert-Elliot (GE), que consiste em uma Cadeia de Markov de dois estados, pode ser utilizado para modelar os padrões de perdas de pacotes corretamente. Os estados foram nomeados como *Good* (g) e *Bad* (b), que simbolizam, respectivamente, a ausência e a presença de falha. A tabela 4.1 mostra os resultados experimentais de Khayam com o tamanho médio das rajadas (*Mean*) e a tabela 4.2 mostra os parâmetros das Cadeias de Markov que modelam estes padrões de falhas.

Na figura 4.1, pode-se ver a cadeia de Markov do modelo GE. O estado B representa o estado de falhas (*Bad*), com a probabilidade de falhas igual a α. O estado G (*Good*) representa o estado de operação normal, com probabilidade de falhas igual a (1 - β).

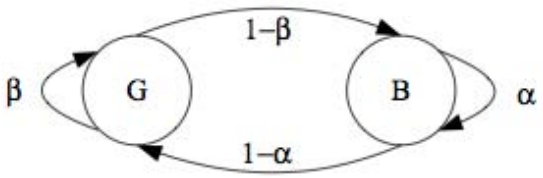

Figura 4.1: Modelo de Gilbert-Elliot (CARVALHO; ANGEJA; NAVARRO, 2005)

De acordo com o modelo, a probabilidade de ocorrência de uma rajada de *n* pacotes perdidos em sequência pode ser calculada pela fórmula:

$$P[X = n] = \alpha^{n-1} * (1 - \alpha) \tag{4.1}$$

Carvalho propõe que seja utilizada uma distribuição de probabilidade logarítmica ao invés da distribuição geométrica utilizada pelo modelo de Gilbert-Elliot. Além disso, o autor mostra que, apesar de nenhum dos modelos conseguir reproduzir fielmente os padrões de falhas, o modelo proposto produz resultados mais aproximados da realidade do que o modelo básico (GE) (CARVALHO; ANGEJA; NAVARRO, 2005).

Tabela 4.1: Estatísticas de perdas de pacotes para diferentes taxas de transmissão na rede IEEE 802.11b (KHAYAM et al., 2003)

Taxa de Transmissão	Tamanho médio da rajada	Desvio-padrão	Taxa de transferência (%)
2 Mbps	0,00029	0,02329	99,9825
5,5 Mbps	1,67930	2,94070	63,8613
11 Mbps	16,3493	12,3343	14,2361

Tabela 4.2: Probabilidades da cadeia de Markov de dois estados para diferentes taxas de transmissão na rede IEEE 802.11b (KHAYAM et al., 2003)

Taxa de transmissão	P(gg)	P(gb)	P(bg)	P(bb)	P(pacote)
2 Mbps	0,999	0,0001	0,6470	0,3529	0,0002
5,5 Mbps	0,820	0,1794	0,3163	0,6836	0,3614
11 Mbps	0,376	0,6235	0,1034	0,8966	0,8576

4.2 Atraso de entrega

Raptis propõe um modelo computacionalmente simples para o atraso de entrega de pacotes em redes IEEE 802.11 (RAPTIS; BANCHS; PAPARRIZOS, 2006). Este modelo é baseado em cadeias de Markov, assim como os modelos de perda de pacotes. Além disso, assume que o meio de transmissão é livre de erros e que os pacotes transmitidos possuem tamanho fixo.

O ponto-chave do modelo proposto por Raptis é que cada pacote possui uma probabilidade fixa de colisão p, que é independente do crescimento exponencial da janela de contenção do mecanismo CSMA/CA. Assim, os autores chegam a uma distribuição de probabilidades que serve para o cálculo de quantas colisões ocorreram antes que o pacote seja corretamente entregue, o que, pelo modelo, indica qual o atraso que afetará este pacote (RAPTIS; BANCHS; PAPARRIZOS, 2006). É importante ressaltar que, conforme foi exposto na seção 2.2.3, apesar do protocolo CSMA/CA ter o foco em evitar colisões, estas ainda acontecem.

O modelo de Raptis é baseado no modelo proposto por Wu (WU et al., 2002) e consiste de uma cadeia de Markov bi-dimensional. Uma dimensão representa o contador do *backoff* e a outra dimensão representa o estágio de *backoff*. Em uma rede IEEE 802.11, quando ocorre uma colisão o sistema espera o tempo de *backoff* aleatório entre 1 e o tamanho da janela de contenção para tentar a retransmissão e a janela de contenção é incrementada a cada colisão.

5 FAULTLETS E TESTES

Este capítulo apresenta os *faultlets*, micro-programas utilizados para a descrição de cargas de falhas no FIRMAMENT. Contém também a implementação e os testes de dois *faultlets* baseados nos modelos de falhas estudados. Os resultados parciais deste trabalho foram apresentados na Escola Regional de Redes de Computadores (MARMITT; WEBER; CECHIN, 2010).

5.1 Faultlets

Um *faultlet* é uma aplicação com alto poder de expressão que descreve uma carga de falhas (DREBES et al., 2006). No FIRMAMENT, pode ser definido um *faultlet* para cada um dos fluxos disponíveis (IPv4_IN, IPv4_OUT, IPv6_IN, IPv6_OUT). Este *faultlet* é executado cada vez que um pacote é capturado pelos ganchos do Netfilter, podendo, por exemplo, alterar a rota do pacote ou modificar o seu conteúdo.

Uma característica que contribui para o alto poder de expressão de um *faultlet* é a presença de registradores de uso geral na máquina virtual FIRM_VM que o executa. Deste modo, um *faultlet* pode possuir memória de estados anteriores, o que permite alterar o seu fluxo de execução a partir de informações dos pacotes em processamento.

5.2 Principais instruções utilizadas nos *faultlets*

O FIRMAMENT disponibiliza um conjunto de instruções que formam a linguagem utilizada para a descrição de *faultlets*. Na tabela 5.1 se encontram as principais instruções utilizadas nos *faultlets* implementados. A lista completa das 31 instruções disponíveis no FIRMAMENT pode ser encontrada na especificação original (DREBES et al., 2006).

5.3 Sistema utilizado nos testes

Como o FIRMAMENT é um módulo disponível para o núcleo Linux, que já foi portado para o núcleo utilizado pelo sistema Android, e os testes serão executados apenas com o objetivo de verificar se os *faultlets* implementados possuem a funcionalidade esperada, qualquer sistema que possua o núcleo Linux e que execute o FIRMAMENT corretamente poderia ser utilizado para a execução dos testes. Deste modo, os testes serão executados em máquinas virtuais com distribuições Linux para Desktops. Essa estratégia facilitará a configuração e execução dos testes, sem alterar os resultados dos mesmos.

O sistema base é composto de um Intel Core2Duo 2.26 GHz e 2 GB de RAM, executando o sistema operacional Mac OS X 10.6. As máquinas virtuais foram executa-

Tabela 5.1: Instruções para descrição de *faultlets* no FIRMAMENT

Instrução	Propósito
READB Ry Rx	Copia em *Rx* o valor do byte *Ry* do pacote capturado
SET y Rx	Ajusta o valor de *Rx* em *y*
ADD Ry Rx	Soma *Rx* com *Ry* e copia o resultado em *Rx*
SUB Ry Rx	Subtrai *Ry* de *Rx* (*Rx - Ry*) e copia o resultado em *Rx*
MUL Ry Rx	Multiplica *Rx* com *Ry* e copia o resultado em *Rx*
ACP	Aceita o pacote e termina a execução do *faultlet*
DRP	Rejeita o pacote (descarta) e termina a execução do *faultlet*
DLY Ry	Atrasa o pacote em *Ry* milisegundos e termina a execução do *faultlet*
JMP x	Desvio incondicional para o rótulo *x*
JMPN Ry x	Desvio para o rótulo *x*, se *Ry* for negativo
JMPZ Ry x	Desvio para o rótulo *x*, se *Ry* for igual a zero
RND Ry Rx	Sorteio de um número pseudo-aleatório contido no intervalo (-Ry, Ry). O resultado é armazenado em *Rx*.

das utilizando-se o programa Sun VirtualBox 3.1.8. A primeira máquina virtual consiste de uma instalação Fedora 10 sobre um núcleo Linux versão 2.6.27. A segunda máquina virtual utiliza o sistema operacional Fedora 12 sobre um núcleo Linux versão 2.6.32. Em ambas as máquinas virtuais as versões do FIRMAMENT utilizadas possuíam as modificações necessárias para sua compilação ter sucesso.

Em ambos os experimentos, os *faultlets* foram carregados no fluxo IPv4_IN da máquina de destino. Essa escolha foi feita devido à características da linguagem utilizada nos scripts desenvolvidos para emular a carga de trabalho (Python). Quando o *faultlet* era carregado no fluxo de saída da máquina de origem e o pacote era descartado, o interpretador Python finalizava a execução do script acusando falha no envio do pacote.

5.4 Faultlet de perda de pacotes

Este *faultlet* emula o modelo básico de Gilbert-Elliot. Este modelo consiste em uma cadeia de Markov com dois estados (normal [G, *Good*] e falha [B, *Bad*]). No estado normal o pacote é encaminhado normalmente e no estado de falha o pacote é descartado. O código descrito utilizando a linguagem de montagem do FIRMAMENT se encontra abaixo, na seção 5.4.1. Uma característica importante de ressaltar é que o registrador R9 é utilizado neste *faultlet* para armazenar qual foi o estado final da execução anterior.

5.4.1 Código do Faultlet

Código 5.1: Faultlet que descreve o modelo de Gilbert-Elliot

```
1  SET 9 R0          ;Filtra pacotes do tipo especificado
2  READB R0 R1
3  SET 17 R0         ;17 = UDP, 6 = TCP, 1 = ICMP
4  SUB R0 R1
5  JMPZ R1 UDP       ;Se for do tipo, segue adiante
6  ACP               ;senao, aceita o pacote
```

```
7  UDP:
8          SET 16 R0          ;Filtra pacotes pelo endereco IP destino
9          READW R0 R1
10         SET 0xc0a80002 R0 ;testa se o IP = 192.168.0.2 em hexadecimal
11         SUB R0 R1
12         JMPZ R1 IP         ;Se for o IP correto, segue adiante
13         ACP                ;senao, aceita o pacote
14 IP:
15         JMPZ R9 G_TESTE ;Teste do estado atual 0 = Good, 1 = Bad
16 B_TESTE:
17         SET 5000 R2        ;Gera um numero aleatorio entre -5000 e 5000
18         RND R2 R3
19         SET 4000 R2        ;muda o intervalo => -1000 a 9000
20         ADD R2 R3
21         JMPN R3 B_B_TRAN;se negativo (10% de chance), executa uma
22         SET 0 R9           ;transicao B_B, senao altera o estado e aceita
23         ACP                ;o pacote
24
25 B_B_TRAN:
26         DRP                ;Continua no estado B e rejeita o pacote
27
28 G_TESTE:
29         SET 5000 R2        ;Gera um numero aleatorio entre -5000 e 5000
30         RND R2 R3
31         SET 4500 R2        ;Muda o intervalo => -500 a 9500
32         ADD R2 R3
33         JMPN R3 G_B_TRAN;Se negativo (5%) executa uma transicao G_B
34         SET 0 R9           ;senao, mantem o estado e aceita o pacote
35         ACP
36 G_B_TRAN:
37         SET 1 R9           ;Muda o estado para B e rejeita o pacote
38         DRP
```

5.4.2 Detalhamento da execução do teste

Os experimentos do *faultlet* foram executados utilizando-se uma aplicação cliente-servidor sobre UDP. Tanto o cliente quanto o servidor foram programados em Python. Cada pacote enviado pelo servidor continha um número de sequência em sua área de dados. Conforme o cliente recebia os pacotes, as rajadas de perdas eram calculadas pela diferença entre o número de sequência atual e o número de sequência do último pacote recebido. Apesar do tráfego de rede não estar sujeito a interferências pela utilização de uma rede virtual, o servidor fechava o socket após a transmissão de cada pacote e aguardava 10 ms antes de iniciar a transmissão do próximo pacote.

Tabela 5.2: Probabilidades de transição entre os estados

	G	B
G	0,95 (β)	0,05
B	0,90	0,10 (α)

Inicialmente os valores probabilísticos das transições foram arbitrários e mais altos que os esperados para uma rede 802.11 em uma situação normal de operação. Isto foi feito para facilitar a verificação do funcionamento dos modelos e do injetor de falhas nos núcleos escolhidos. No *faultlet*, foram utilizados os valores de transição contidos na

tabela 5.2. Quatro transmissões foram executadas e em cada transmissão foram enviados 100.000 pacotes.

5.4.3 Resultado do teste

A tabela 5.3 mostra uma comparação entre as probabilidades de ocorrência calculadas pela fórmula (4.1) e os resultados das transmissões. A distribuição dos pacotes ocorreu conforme a distribuição teórica esperada. A figura 5.1 compara a quantidade de rajadas ocorridas no experimento com a previsão teórica. A partir destes resultados, pode-se concluir que o *faultlet* implementado tem a funcionalidade esperada.

No entanto, para que este *faultlet* seja utilizado para validar um software é necessário calcular empiricamente, por meio de simulações, os parâmetros α e β que caracterizam a situação em que o dispositivo móvel será utilizado. Dispositivos operando em ambientes vulneráveis a uma quantidade maior de interferências e/ou operando com taxas de transmissão maiores possuem uma maior probabilidade de um pacote ser perdido e um maior tamanho médio de rajada de perdas.

Tabela 5.3: Comparação entre a distribuição esperada de rajadas e os resultados do experimento com o *faultlet*

Tamanho da rajada (pacotes)	Probabilidade teórica	Probabilidade calculada sobre a média de rajadas	Primeiro experimento (pacotes)	Segundo experimento (pacotes)	Terceiro experimento (pacotes)	Quarto experimento (pacotes)
1	0,9	0,9011	4245	4259	4247	4247
2	0,09	0,0874	429	407	417	395
3	0,009	0,01039	52	38	57	49
4	0,0009	0,001007	4	4	6	5
5	0,00009	0,00005301	0	1	0	0
6	0,000009	0	0	0	0	0
7	0,0000009	0,00005301	0	0	1	0
Total Pacotes	--	5242,5	5275	5208	5283	5204
Total Rajadas	--	4715,75	4730	4709	4728	4696

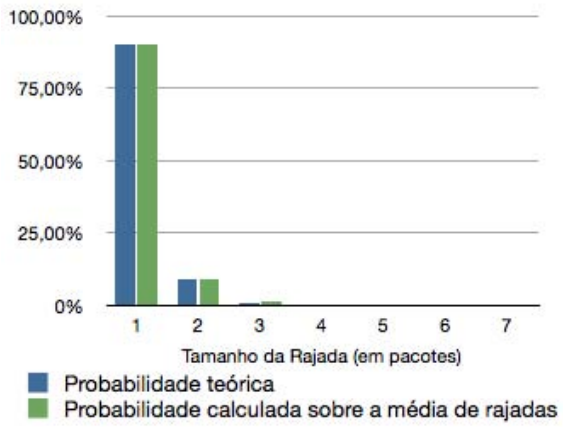

Figura 5.1: Comparação entre a média dos resultados e a previsão teórica do *faultlet* de perdas

5.5 Faultlet de Atraso

Este *faultlet* contém a implementação do modelo proposto por Raptis e apresentado na seção 4.2. O modelo também é composto por uma cadeia de Markov de dois estados, mas o comportamento do algoritmo difere do algoritmo utilizado no *faultlet* de perdas. Nesta implementação, o atraso do pacote atual é independente do estado final do pacote anterior. O estado inicial do algoritmo é o estado normal (G). Durante a execução do *faultlet* são executadas iterações de acordo com as probabilidades definidas no modelo (neste caso, α = 0,1 e β = 0,95). A execução termina quando o próximo estado é normal (G). Em seguida, o atraso básico (neste caso, 20 ms) é multiplicado pelo número de iterações em que o *faultlet* permaneceu no estado de falha (B). O atraso resultante é, então, aplicado ao pacote capturado.

Neste *faultlet* o modelo de Raptis apresentado na seção 4.2 foi simplificado para apenas uma dimensão. Essa simplificação assume que o tempo de *backoff* não é incrementado a cada retransmissão. Para que o *faultlet* reproduzisse fielmente o modelo, o atraso básico deveria ser incrementado a cada colisão (ocorrência do estado de falha). Além disso, tanto os valores de probabilidade de transição quando o atraso básico foram escolhidos arbitrariamente com valores altos, para que a distribuição dos resultados fosse de fácil visualização e medição.

5.5.1 Código do Faultlet

Código 5.2: Faultlet que descreve o modelo de atrasos

```
1          SET 1 R9          ;Inicializa o fator de multiplicacao do atraso
2          SET 9 R0          ;Verifica se o pacote eh do tipo UDP
3          READB R0 R1
4          SET 17 R0         ;1 = ICMP, 17 = UDP, 6 = TCP
5          SUB R0 R1
6          JMPZ R1 UDP
7          ACP
8  UDP:
9          SET 16 R0         ;Verifica o endereco IP destino do pacote
10         READW R0 R1
11         SET 0xc0a80002 R0
12         SUB R0 R1         ;Se o endereco for igual a 192.168.0.2
13         JMPZ R1 IP_G      ;0xc0a80002 em hexadecimal
14         ACP
15
16 IP_G:
17         SET 5000 R2       ;5% de chance do pacote sofrer o primeiro
18         RND R2 R3         ;atraso
19         SET 4500 R2
20         ADD R2 R3
21         JMPN R3 IP_B
22         ACP
23
24 IP_B:
25         SET 5000 R2       ;10% de chance do pacote sofrer atrasos
26         RND R2 R3         ;subsequentes
27         SET 4000 R2
28         ADD R2 R3
29         JMPN R3 ONE_MORE
30         SET 20 R0         ;multiplica o numero de atrasos pela taxa
31         MUL R0 R9         ;basica
```

```
32          DLY R9              ;atrasa o pacote e termina
33
34  ONE_MORE:
35          SET 1 R1
36          ADD R1 R9
37          JMP IP_B
```

5.5.2 Detalhamento da execução do teste

Os experimentos utilizando este *faultlet* foram realizados utilizando uma aplicação cliente-servidor sobre UDP. Tanto o cliente quanto o servidor foram programados em Python. Para a medição do tempo aproximado de atraso foi utilizada a seguinte estratégia:

- O tempo atual da máquina 1 foi obtido com a função *time.time()* da linguagem Python.
- Este tempo foi transmitido na área de dados do pacote UDP.
- A máquina 2, que estava executando o *faultlet* no fluxo IPv4_IN, recebe o pacote e envia de volta para a máquina 1.
- Um segundo script Python recebe a resposta na máquina 1 e compara o tempo existente na área de dados com o tempo atual do sistema.

Entre cada um dos envios, a máquina 1 aguardava 100 ms, para evitar que os pacotes acumulassem no buffer de entrada da máquina 2. Embora a medição executada não forneça um resultado preciso do atraso de cada pacote, a classificação dos pacotes em intervalos de atraso absorve o erro incluído nas medições.

5.5.3 Resultado do teste

A figura 5.2 apresenta a distribuição de atrasos em cada um dos experimentos e a distribuição esperada dos atrasos (calculada pela fórmula 4.1). O eixo vertical apresenta a quantidade de pacotes que sofreram um determinado atraso. O eixo horizontal contém os experimentos, a média dos resultados dos experimentos e os resultados teóricos. Cada barra vertical representa uma categoria de atraso. A semelhança entre os grupamentos dos experimentos mostra que o *faultlet* apresenta o comportamento esperado, principalmente nas quatro primeiras categorias de atraso, onde o número esperado de pacotes afetados é maior que 1.

Tabela 5.4: Resultados dos experimentos com o *faultlet* de atrasos

Tempo (ms)	Exp 1	Exp 2	Exp 3	Exp 4	Média	Pacotes / Total	Pacotes / Atrasados
< 15	18982	19029	18995	19005	19002,75	95,014%	-------
15-35	924	872	885	898	894,75	4,474%	89,722%
35-55	86	91	109	89	93,75	0,469%	9,401%
55-75	8	8	10	7	8,25	0,041%	0,827%
75-95	0	0	0	1	0,25	0,001%	0,025%
95-115	0	0	0	0	0	0,000%	0,000%
115-130	0	0	1	0	0,25	0,001%	0,025%
> 130	0	0	0	0	0	0,000%	0,000%
Total Atrasados	1018	971	1005	995	997,25	4,986%	
Total	20000	20000	20000	20000	20000		

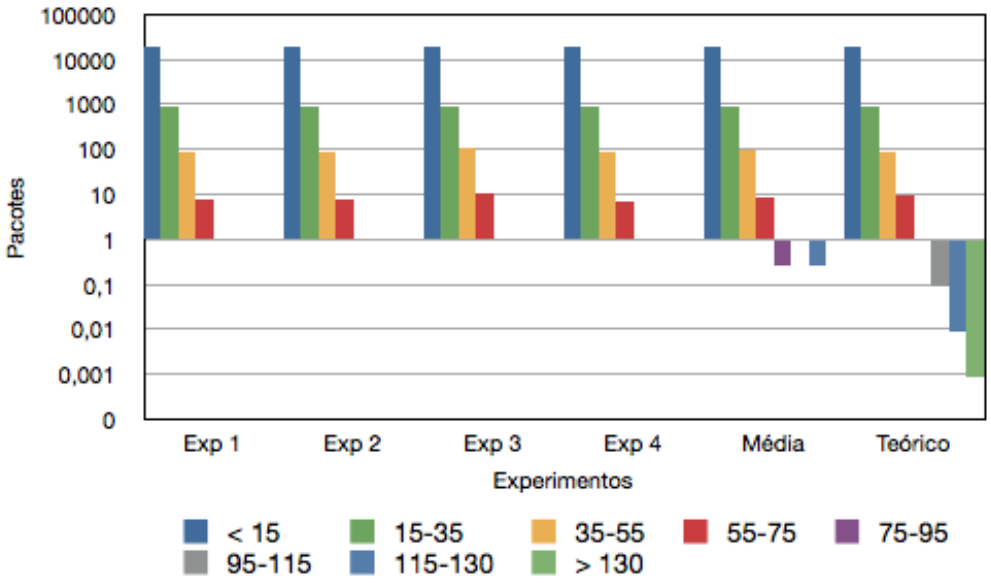

Figura 5.2: Comparação entre as distribuições de atraso nos experimentos executados e a previsão teórica (escala logarítimica)

A figura 5.3 apresenta a distribuição apenas dos pacotes atrasados e compara os resultado práticos e teóricos dos experimentos. Esta figura apresenta de forma mais clara a distribuição exponencial gerada pela cadeia de Markov utilizada.

Assim como no *faultlet* de perdas, para que este *faultlet* seja utilizado na validação de softwares são necessárias simulações reais que forneçam a probabilidade esperada que uma colisão ocorra e qual o tempo de *backoff* esperado. Além disso, para que o modelo reproduza de forma mais precisa as situações reais, a segunda dimensão (*backoffs* com aleatórios e janela de contenção crescente) poderia ser acrescentada na cadeia de Markov.

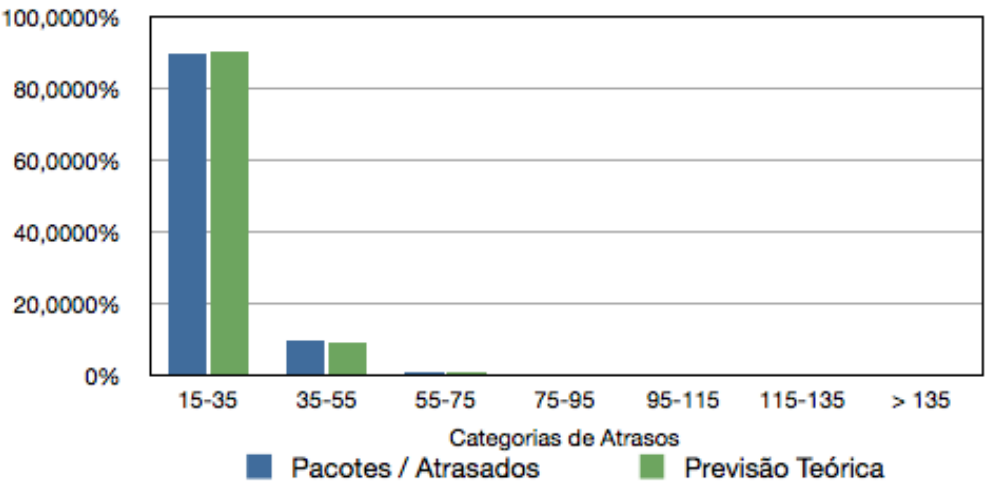

Figura 5.3: Comparação entre a média dos resultados e a previsão teórica do *faultlet* de atrasos

Tabela 5.5: Comparação dos resultados de atrasos com a previsão teórica

Tempo (ms)	Pacotes / Atrasados	Previsão Teórica
< 15	----	----
15-35	89,7220%	90,0000%
35-55	9,4010%	9,0000%
55-75	0,8270%	0,9000%
75-95	0,0250%	0,0900%
95-115	0,0000%	0,0090%
115-135	0,0250%	0,0009%
> 135	0,0000%	0,0001%

5.6 Intrusividade dos Faultlets

Com o objetivo de estimar a intrusividade temporal dos *faultlets* criados, foi executado o seguinte experimento. Com o utilitário *PING*, foram enviadas 100.000 mensagens ICMP entre as duas máquinas virtuais. A ferramenta *PING* calcula o tempo de ida-e-volta entre duas máquinas, utilizando o protocolo ICMP (*Internet Control Message Protocol*) e marcas de tempo enviadas no pacote ICMP.

Na primeira execução, nenhum *faultlet* estava carregado, e o fluxo não era interrompido. Na segunda execução, o FIRMAMENT estava carregado com o *faultlet* de perdas modificado. A modificação consistiu em alterar as instruções que descartavam os pacotes para instruções de aceitação (ACP), deste modo o injetor capturava o pacote, executava o faultlet, mas sempre permitia que o pacote chegasse ao destino. A terceira execução foi similar à segunda, mas com o *faultlet* de atrasos (com modificações similares) carregado.

Analisando os resultados médios de tempo obtidos, mostrados na tabela 5.6, podemos ver que os valores médios estão próximos. As diferenças (da ordem de dezenas de microsegundos) são, possivelmente, resultado das imprecisões de medição e do escalonamento das máquinas virtuais na máquina hospedeira.

Tabela 5.6: Medidas de tempo ida-e-volta (em ms, exceto em Taxa de Pacotes)

	Mín.	Méd.	Máx.	Desvio	Tempo	Taxa de Pacotes
Referência	0,220	0,965	34,322	0,748	120167	832,17
Faultlet de Perdas	0,153	0,954	10,138	0,713	118651	842,8
Faultlet de Atrasos	0,221	0,972	10,44	0,729	120972	826,63

6 CONCLUSÃO

A injeção de falhas permite que softwares tenham seus mecanismos de tolerância a falhas devidamente testados e validados. Entretanto, é necessário que estes mecanismos sejam testados com modelos de falhas que emulem com a maior precisão possível a carga de falhas que ocorrerá na situação real de uso deste software. Os modelos de falhas de ambientes móveis se tornaram extremamente importantes com a popularização de aparelhos dotados de um ou mais dispositivos de comunicação integrados.

Os modelos de falhas apresentados neste trabalho representaram duas situações comuns em comunicação de dados: pacotes perdidos e pacotes entregues com atraso. Apesar dos modelos serem aproximações para a carga de falhas real, possuem o equilíbrio necessário entre a precisão e o desempenho da emulação da carga de falhas. Os dois *faultlets* implementados possuem a funcionalidade esperada, baixa intrusividade e baixa complexidade computacional.

A implementação bem sucedida dos *faultlets* também comprova que o injetor de falhas utilizado (FIRMAMENT) é adequado para injeção de falhas de acordo com os modelos pesquisado, mesmo que não tenha sido inicialmente projetado para tratar com falhas específicas de redes móveis. Nenhuma alteração no código do injetor foi necessária para a implementação dos modelos e a linguagem FIRMASM permitiu que os algoritmos utilizados para implementar as cadeias de Markov fossem escritos corretamente e sem dificuldades.

Com os *faultlets* aqui descritos, será possível validar softwares que utilizem redes IEEE 802.11 para comunicações e operem em sistemas baseados no núcleo Linux. Uma sugestão para a continuidade deste trabalho é a realização de medições experimentais da distribuição de perdas e atrasos em situações reais. Com estas medições, seria possível compilar uma tabela com os parâmetros que devem ser utilizados em cada um dos *faultlets* para emular as principais situações de uso dos aparelhos que executam os softwares que serão validados.

REFERÊNCIAS

ACKER, E. V.; WEBER, T. S.; CECHIN, S. L. Injeção de falhas para validar aplicações em ambientes móveis. In: XI WORKSHOP DE TESTES E TOLERÂNCIA A FALHAS, Gramado, RS. **Anais...** [S.l.: s.n.], 2010. p.61–74.

ADMOB. **May-2010-AdMob-Mobile-Metrics-Highlights. http**://metrics.admob.com/. [acessado em Novembro/2010].

ANDROID-ADG. **What is android?. http**://developer.android.com/guide/index.html. [acessado em Maio/2010].

ARLAT, J. et al. Fault Injection for Dependability Validation: a methodology and some applications. **IEEE Trans. Softw. Eng.**, [S.l.], v.16, n.2, p.166–182, 1990.

AVIZIENIS, A. et al. Basic Concepts and Taxonomy of Dependable and Secure Computing. **IEEE trans. on dependable and secure computing**, [S.l.], v.1, n.1, p.11–33, 2004.

BOGGIA, G.; CAMARDA, P.; D'ALCONZO, A. Performance of Markov models for frame-level errors in IEEE 802.11 wireless LANs. **Int. J. Commun. Syst.**, Chichester, UK, v.22, n.6, p.695–718, 2009.

CARVALHO, L.; ANGEJA, J.; NAVARRO, A. A new packet loss model of the IEEE 802.11g wireless network for multimedia communications. **Consumer Electronics, IEEE Transactions on**, [S.l.], v.51, n.3, p.809 – 814, aug. 2005.

DOBLER, R. J.; WEBER, T. S.; CECHIN, S. L. Porte do Injetor de Falhas Firmament para o Ambiente Android. In: ESCOLA REGIONAL DE REDES DE COMPUTADORES, 8., Alegrete, RS. **Anais...** [S.l.: s.n.], 2010. v.1, p.9–15.

DREBES, R. J. et al. A Kernel based Communication Fault Injector for Dependability Testing of Distributed Systems. In: SPRINGER-VERLAG (Ed.). **First Int. Haifa Verification Conf.** [S.l.: s.n.], 2006. v.3875, p.177–190.

ELLIOTT, E. O. Estimates of error rates for codes on burst-noise channels. **Bell Syst. Tech. J**, [S.l.], n.42, p.1977–1997, 1963.

FOROUZAN, B. A. **Comunicação de Dados e Redes de Computadores**. 4.ed. [S.l.]: McGraw-Hill, 2008. p.29–37, 421–440, 579–596.

GILBERT, E. Capacity of a Burst-Noise Channel. **Bell Systems Technical Journal**, [S.l.], n.39, 1960.

HSUEH, M.-C.; TSAI, T.; IYER, R. Fault injection techniques and tools. **Computer**, [S.l.], v.30, n.4, p.75 –82, apr. 1997.

IDC. **Worldwide Quarterly Mobile Phone Tracker. http**://www.idc.com/. [acessado em Novembro/2010].

IEEE. IEEE Standard for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks-Specific Requirements - Part 11: wireless lan medium access control (mac) and physical layer (phy) specifications. **IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)**, [S.l.], p.C1 –1184, june 2007.

IETF. **RFC 791 - Internet Protocol**. 1981.

KHAYAM, S. A. et al. Performance analysis and modeling of errors and losses over 802.11b LANs for high-bit-rate real-time multimedia. **Signal Processing: Image Communication**, [S.l.], v.18, n.7, p.575 – 595, 2003.

MARMITT, H. F.; WEBER, T. S.; CECHIN, S. L. Modelos para injeção de falhas em ambientes móveis. In: ESCOLA REGIONAL DE REDES DE COMPUTADORES, 8., Alegrete, RS. **Anais...** [S.l.: s.n.], 2010. v.1, p.1–8.

RAPTIS, P.; BANCHS, A.; PAPARRIZOS, K. A simple and effective delay distribution analysis for IEEE 802.11. In: Personal, Indoor and Mobile Radio Communications, 2006 IEEE 17th International Symposium on. **Anais...** [S.l.: s.n.], 2006. p.1 –5.

RAPTIS, P.; VITSAS, V.; PAPARRIZOS, K. Packet delay metrics for IEEE 802.11 Distributed Coordination Function. **Mob. Netw. Appl.**, Hingham, MA, USA, v.14, n.6, p.772–781, 2009.

RUSSELL, R.; WELTE, H. **Linux netfilter Hacking HOWTO. http**://www.netfilter.org/. [acessado em Maio/2010].

WEBER, T. S. Tolerância a falhas: conceitos e exemplos. In: INTECH BRASIL (SãO PAULO), São Paulo, SP. **Anais...** [S.l.: s.n.], 2003. v.52, p.32–42.

WU, H. et al. Performance of Reliable Transport Protocol over IEEE 802.11 Wireless LAN: analysis and enhancement. In: **Anais...** [S.l.: s.n.], 2002. p.599–607.