Aplicando Cadeias de Markov para injeção de perdas de pacotes no sistema Android

Alexandre Felin Gindri[1], Taisy Silva Weber[1], Sérgio Luis 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

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

Abstract. *We describe the development of a communication fault injector for mobile devices that allows the emulation of packet loss following fault models suitable for wireless networks. The fault injector named Mob-FI allows the test under faults of Java applications that execute on smartphones and tablets and that are based on UDP communication. We build the fault injector modifying the code of Dalvik Virtual Machine to emulate faults according to the models of packet loss represented as Markov Chains of up to four states.*

Resumo. *Descreve-se o desenvolvimento de um injetor de falhas de comunicação para dispositivos móveis que permite a emulação de perdas de pacotes seguindo modelos de falhas adequados a redes sem fio. O injetor Mob-FI permite o teste sob falhas de aplicações alvo Java desenvolvidas para smartphones e tablets e que sejam baseadas em comunicação UDP. O injetor foi construído modificando a Dalvik Virtual Machine do sistema Android e emula falhas de acordo com os modelos de perda de pacotes em rajada representados como Cadeias de Markov de até quatro estados.*

1 Introdução

O Android [Butler 2011], criado pela Google e pela Open Handset Alliance, facilita a criação de aplicações com acesso a todos os recursos de dispositivos móveis. Graças a essa facilidade, um grande número de programas, provenientes dos mais diversos desenvolvedores, está disponível para usuários de *smartphones* e *tablets*. Muitas dessas aplicações são usadas em situações em que se esperaria a garantia de um mínimo de confiabilidade e disponibilidade. Para fornecer tal garantia, seria desejável poder testar a tolerância dessas aplicações a uma variada gama de falhas que usualmente afeta a operação de dispositivos móveis em cenários de uso real. Entre as possíveis falhas, dispositivos móveis são especialmente sensíveis a falhas de comunicação, sendo comum pacotes serem perdidos durante a transmissão de dados. A perda de pacotes em ambientes móveis segue um comportamento típico desses ambientes, mas diferente daquela que ocorre em redes cabeadas.

Um desenvolvedor para Android que decida testar a robustez de uma aplicação em um cenário de falhas de comunicação deve determinar o modelo de falhas adequado ao ambiente, depois criar uma carga de falhas que corresponda a esse modelo e então executar a aplicação com uma dada carga de trabalho aplicando a carga de falhas. O objetivo deste trabalho é desenvolver um injetor de falhas que execute no ambiente Android, permita a injeção de perdas de pacotes seguindo modelos de falhas adequados ao ambiente móvel e facilite a descrição dos cenários de falhas para experimentos de teste a serem conduzidos por um desenvolvedor. Modelos de falhas para sistemas

móveis podem ser encontrados na literatura sendo que a perda de pacotes em rajada modelada como Cadeias de Markov com dois ou quatro estados [Salsano 2009] foram os escolhidos para descrição de carga de falhas.

As aplicações do Android rodam em uma máquina virtual própria do sistema, a Dalvik Virtual Machine [Bornstein 2009], que é uma máquina otimizada para sistemas com limitações de memória, processamento e bateria. Essa máquina é baseada na máquina virtual Java, mas se diferencia quanto ao formato dos bytecodes, não oferecendo apoio ao uso de mecanismos de instrumentação comumente usados para depuração e injeção de falhas.

Este artigo explora a possibilidade de implementar injetores de falhas de comunicação para o sistema Android a partir de modificação da máquina virtual. O injetor Mob-FI (Markov Fault Injetor for Mobile Environment) foi construído modificando a classe DatagramSocket da Dalvik Virtual Machine e emula falhas de acordo com os modelos de perda de pacotes representados como Cadeias de Markov. O injetor permite o teste sob falhas de aplicações móveis desenvolvidas em Java através da emulação de perda de pacotes de acordo com modelos de falhas de comunicação apropriados para redes sem fio.

Nas próximas seções, o artigo apresenta os conceitos básicos para a compreensão da injeção de falhas em ambientes móveis, discute modelos de falhas de comunicação em rajada, apresenta o desenvolvimento do injetor Mob-FI e experimentos conduzidos para avaliação do injetor.

2 Falhas de comunicação em ambientes móveis

A operação de dispositivos em ambientes móveis é caracterizada pela computação com recursos limitados, principalmente por tamanho e consumo de energia, e pela comunicação numa rede não cabeada, com movimentação relativa entre os dispositivos.

Com a popularização de equipamentos eletrônicos que operam nas mesmas faixas de frequências, aumenta também o problema da interferência entre sinais. A alteração na distância dos dispositivos devido à sua movimentação relativa causa mudanças na propagação dos sinais, acarretando variações, por exemplo, na atenuação, nos atrasos de recepção, na reflexão, refração e difração sofridas pelo sinal durante a propagação, entre outros.

Dispositivos móveis, portanto, estão sujeitos a diversas falhas de comunicação. A perda de pacotes pode ser causada por interferência ou movimentação. A falha pode ser transitória, quando ocorre perda de um único pacote. Podem ocorrer também rajadas de perda onde algumas mensagens em uma sequência de pacotes são perdidas. O dispositivo finalmente pode perder a conexão com a rede, sendo necessária uma renegociação com a sua estação base.

Como a comunicação é essencial em grande número de aplicações móveis, uma parte importante no teste destes sistemas é conseguir injetar falhas que emulem da forma mais apropriada possível esses cenários de perda de pacotes em rajada.

2.1 Construção de injetores de falhas de comunicação

Os injetores de falhas por software [Hsueh 1997] são injetores de erros, e podem alterar o comportamento de um programa injetando erros a partir de vários recursos de

software de um sistema. Várias falhas podem ser emuladas através de erros injetados, como, por exemplo, troca de bits em memória ou registradores, mas o foco deste artigo são as falhas que afetam a troca de mensagens através de uma rede de comunicação. Exemplos de injetores que permitem emular falhas de comunicação são Orchestra [Dawson 1996], NIST Net [Carson 2003], VirtualWire [De 2003], FIRMAMENT ([Drebes 2006][Siqueira 2009]), Fiona [Jacques-Silva 2006] e FAIL-FCI [Horau 2007]. Trabalhos anteriores do grupo finalizaram o porte do injetor FIRMAMENT para o sistema Android [Acker 2010]. Entre os injetores de falhas desenvolvidos no grupo de pesquisa, FIRMAMENT atua diretamente no kernel do sistema operacional e manipula todos os pacotes IP que trafegam pela pilha de protocolos no kernel, mesmo as mensagens que não estão relacionadas à aplicação alvo, o que em algumas situações pode invalidar o teste. Fiona e Comform [Menegotto 2011] atuam a partir da máquina virtual, e são ferramentas de injeção de falhas específicas para o teste sob falhas de aplicações alvo escritas em Java.

A forma usual de injetar falhas de comunicação é interceptar pacotes enviados e recebidos pela aplicação alvo e, uma vez de posse de uma mensagem, decidir, de acordo com uma dada descrição de carga de falhas, se a mensagem vai ser descartada, atrasada ou corrompida. Para interceptar pacotes, podem ser usados alguns recursos providos pelo sistema alvo como bibliotecas de comunicação, chamadas de sistema, reflexão computacional, módulos de depuração ou filtros disponíveis no kernel do sistema operacional. Os recursos de interceptação usados para o desenvolvimento dos injetores Comform e Fiona foram Javassist e JVMTI. Javassist permite a criação de classes durante a execução e a modificação de um arquivo de classe quando a máquina virtual for carregá-la. JVMTI é usada em ferramentas de desenvolvimento e monitoração. Permite inspecionar o estado e controlar a execução de aplicações em máquinas virtuais Java. Quando os estudos para a construção do injetor Mob-FI foram iniciados, nenhuma das duas opções estava disponível para a Dalvik Virtual Machine. Assim, uma alternativa viável foi modificar o código da máquina virtual.

2.2 Modificações no código fonte do sistema

A injeção de falhas por software consiste em inserir trechos de código que emulem a ocorrência de falhas e, após, retomar a operação com a falha injetada presente no sistema. As falhas podem ser inseridas diretamente na aplicação sob teste ou em alguma camada do sistema que executa a aplicação, como por exemplo, a máquina virtual, as bibliotecas do sistema e o próprio kernel do sistema operacional.

Editar diretamente o código do sistema para inserir trechos que emulem uma falha é uma alternativa em dispositivos com processadores que não tenham hardware de depuração apropriado para permitir o uso de ferramentas de inserção de código em execução [Looker 2005]. Uma vantagem do método é não ser necessário ter disponível o código da aplicação que será testada. A alteração do código do sistema apresenta, entretanto, desvantagens. O método pode não ser válido para uma eventual certificação, já que os sistemas com e sem o injetor podem ser considerados diferentes. Apesar disso, é possível desenvolver injetores com baixa interferência no desempenho e que podem ser aplicados a qualquer programa executado pelo sistema original.

3 Modelos de falhas de comunicação

O objetivo de injetar falhas em uma aplicação alvo é verificar se os mecanismos de tolerância a falhas serão suficientes e corretos quando a aplicação estiver operando em um cenário real de ocorrência de falhas. Por essa razão, é fundamental modelar acuradamente a ocorrência de falhas para esses cenários, e assim poder definir uma carga de falhas adequada para um injetor.

Na literatura são apresentados diferentes modelos para emular falhas de comunicação em redes sem fio. Segue a descrição dos modelos para perda de pacotes mais referenciados na literatura [Hohlfeld 2008] e que foram usados como base para a definição da carga de falhas do injetor Mob-FI.

3.1 Modelo de perdas sem memória de Bernoulli

É um dos modelos mais simples. Determina que as perdas de pacotes ocorrem com uma taxa *r*. Cada pacote tem uma probabilidade fixa de ser perdido, independente de estados anteriores. Essa taxa de perda pode ser estimada a partir de uma amostra, dividindo a quantidade de mensagens perdidas pelo total de mensagens enviadas na amostra.

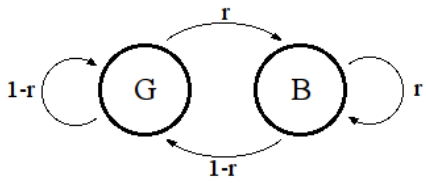

Figura 1: Cadeia de Markov representando o Modelo de Bernoulli

A Figura 1 mostra o modelo de Bernoulli representado com uma Cadeia de Markov de com estados. No estado G (“Good” ou “Gap”) as mensagens são entregues, no estado B (“Bad” ou “Burst”) as mensagens são perdidas. O parâmetro característico desse modelo é a taxa *r*, que representa a probabilidade de ir para o estado de falha e de ficar no estado de falha.

3.2 Modelos de perdas em rajada de Gilbert

O modelo de Gilbert simplificado (Figura 2) utiliza uma Cadeia de Markov semelhante à Figura 1, mas com probabilidades diferentes de transições entre estados. Na Figura 2, *p* define a probabilidade de ocorrer uma rajada de perdas, enquanto *r* afeta o tamanho da rajada. Os valores de *p* e *r* devem ser suficientemente pequenos para simular perdas corretamente [Gilbert 1960].

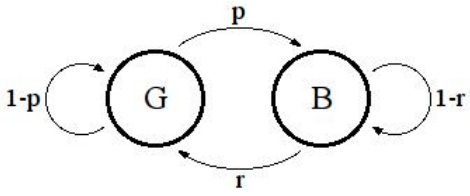

Figura 2: Cadeia de Markov representando o Modelo de Gilbert simplificado

No modelo de perdas completo de Gilbert, além de *p* e *r* é considerado também o parâmetro *h*, que é a probabilidade de não ocorrência de perdas no estado B.

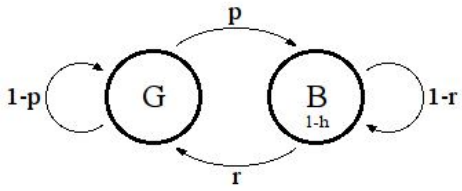

Figura 3: Cadeia de Markov que representa o Modelo de Gilbert

No estado B, Figura 3, a probabilidade de perda de mensagens é de *1 - h*. O estado B é análogo ao Modelo de Bernoulli, com a taxa de perda *r* sendo igual a *1 - h*, podendo este estado ser também modelado com uma Cadeia de Markov com dois estados.

Portanto, no Modelo Completo, a probabilidade final de perda de mensagens será uma combinação do estado do sistema (determinado por *p* e *r*) e a probabilidade *h*.

3.3 Modelo de perdas em rajada de Gilbert-Elliott

No modelo de Gilbert-Elliott [Elliott 1963] é considerada uma taxa de erro também no estado G, porém menor que no estado B. O valor *k* na Figura 4 representa a probabilidade de não ocorrência de perdas.

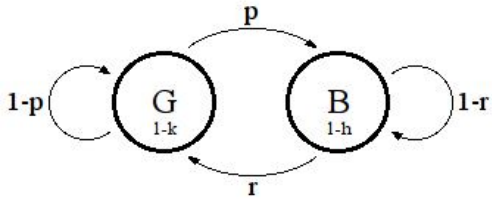

Figura 4: Modelo de Gilbert-Elliott

O modelo permite representar de forma mais satisfatória os dois estados que são característicos de redes sem fio; um estado bom (G) onde ocorrem perdas esporádicas de pacotes, e um estado ruim (B) onde ocorrem perdas em rajadas com alguns poucos pacotes que podem ser transmitidos corretamente.

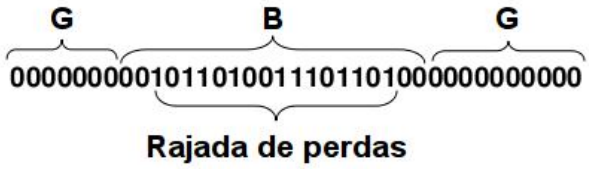

Figura 5: Fluxo de transmissão para h igual a 0,5

A Figura 5 [Salsano 2009] mostra um fluxo de transmissão, onde mensagens transmitidas são simbolizadas com 0 e mensagens perdidas com 1. Para esse fluxo a probabilidade *h* é igual à 0,5, como proposta por Gilbert.

Nota-se que, para *h* maior que 0, a duração do estado B não corresponde a largura da rajada, que é delimitada pela primeira e a última perda de mensagem. Esse comportamento é considerado um ponto fraco do modelo, uma vez que, através da manipulação dos parâmetros *p, r, k* e *h*, não é possível configurar com precisão a largura da rajada.

3.4 Modelo de perdas em rajada com Cadeia de Markov com quatro estados

Salsano (2009) apresenta um modelo que consiste em uma Cadeia de Markov com quatro estados, definida por cinco parâmetros não redundantes.

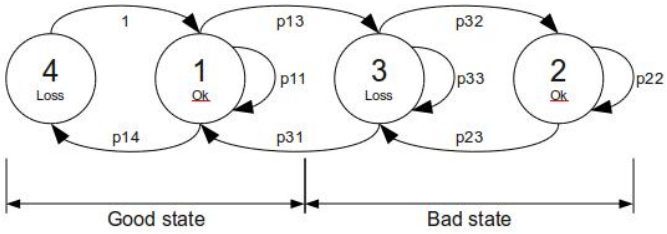

Figura 6: Cadeia de Markov de quatro estados

O modelo é semelhante ao de Gilbert-Elliott, tendo também um estado bom (na Figura 6 marcado como “Good state”) com poucas perdas e um estado ruim (“Bad state”) onde ocorrem algumas rajadas de perdas. Cada um desses estados é dividido em outros dois. Os estados 1 e 2 transmitem mensagens e, nos estados 3 e 4, as mensagens são perdidas. Os parâmetros *p13, p14, p23, p31* e *p32* são suficientes para caracterizar o modelo, sendo os parâmetros *p11, p22* e *p33* calculados como:

$$p11 = 1 - p13 - p14; \quad p22 = 1 - p23; \quad p33 = 1 - p31 - p32$$

No estado bom, existe uma pequena probabilidade *p14* de se passar do estado de transmissão 1 para o estado de perda 4. Para diminuir o número de parâmetros que caracterizam o modelo, a probabilidade de volta do estado 4 para o 1 é igual à 1, que implica em uma única perda por vez. Existe uma probabilidade *p13* de se passar para o estado ruim de rajada de perdas. Como não há transições entre os estados 1 e 2, o estado ruim inicia e finaliza no estado 3, onde efetivamente são perdidas mensagens. Assim, o ponto fraco apontado para o modelo de Gilbert-Elliott não ocorre, e o tamanho da rajada é igual à duração do estado ruim. Esta duração do estado ruim é relacionada à probabilidade *p31* de transição.

A probabilidade *p32* indica a frequência de transmissões durante a rajada de perdas e a probabilidade *p23* a quantidade de transmissões consecutivas dentro da rajada de perdas.

O injetor Mob-FI permite selecionar qualquer um dos cinco modelos apresentados para modelar perda de pacotes. O injetor opera com uma Cadeia de Markov com quatro estados, onde dois são normais e dois são de perda. Os modelos com dois estados são mapeados para os quatro estados manipulando os parâmetros de probabilidade fornecidos pelo usuário na descrição da carga de falhas.

4 Injetor de falhas para o Android

A proposta deste trabalho é injetar falhas de comunicação UDP em aplicações alvo que executam no sistema Android. Como não é possível utilizar ferramentas de intercepção de mensagens disponíveis para outras plataformas, como as Javassist ou JVMTI que já foram usadas por outros injetores de comunicação [Menegotto 2011], [Jacques-Silva 2006], decidimos explorar a solução de modificação do código da máquina virtual.

O injetor Mob-FI, no estágio atual, injeta falhas em mensagens transmitidas e recebidas no protocolo UDP, que é um protocolo de comunicação da pilha TCP-IP muito popular, simples, eficiente, mas não confiável.

4.1 Implementação do injetor Mob-FI

A comunicação UDP em sistemas baseados em Java é realizada pela classe java.net.DatagramSocket. O arquivo DatagramSocket.java contém todos os atributos e métodos da classe. Na implementação do injetor de falhas UDP, foram adicionados atributos e modificados os construtores e os métodos *receive()* e *send()*, que são os métodos para receber e enviar pacotes. O injetor implementa uma cadeia de Markov com quatro estados, onde dois são de transmissão e recepção normal e dois são de perda.

A classe DatagramSocket possui cinco construtores, nos quais foi adicionada ao final uma chamada à função de inicialização do injetor. Nessa função é feita a leitura de um arquivo de configuração e são aplicadas as configurações para o DatagramSocket recém-criado. Caso o arquivo de configuração não esteja completo ou tenha alguma incoerência, o injetor não é habilitado.

O método *receive()* consiste, basicamente, em um laço, onde o método bloqueia até a chegada de um pacote válido para o socket. Quando o pacote for recebido, suas informações são colocadas em uma estrutura do tipo DatagramPacket, que é passada como parâmetro ao método. Para emular a perda de pacotes na recepção, foi adicionado um laço que engloba o laço original da função, onde o estado corrente é verificado e atualizado para o próximo estado. Caso se esteja em um estado de perda, ocorre uma nova iteração, esperando o próximo pacote; caso contrário, a condição de saída do laço do injetor vai ser alcançada e a chamada do método *receive()* é finalizada.

No método *send()*, o pacote passado como argumento é verificado e, caso o pacote seja não válido, é gerada uma exceção correspondente e o pacote não é enviado. Se nenhuma exceção for gerada, o pacote válido é enviado. No método *send()* modificado, os pacotes válidos são enviados apenas se o injetor estiver no estado sem perda de pacotes.

Informações mais detalhadas sobre a construção do injetor Mob-FI podem ser encontradas na documentação sobre a ferramenta [Gindri 2011].

4.2 Execução da carga de falhas

O injetor Mob-FI implementa com uma cadeia de Markov com quatro estados genéricos, de forma a permitir o mapeamento dos cinco modelos apresentados. Os estados são numerados de zero a três, sendo os estados “0” e “1” sem perda de pacotes, e os estados “2” e “3” com perda de pacotes (Figura 7 (a)).

Dentre os atributos da classe DatagramSocket, foi adicionada uma matriz de valores em ponto flutuante de dimensão quatro por quatro, que contém as probabilidades de transição acumuladas, sendo o primeiro índice o estado atual e o segundo índice o próximo estado. A Figura 7 ilustra a cadeia de Markov com todas as transições entre estados e a matriz de probabilidades acumuladas, onde cada linha representa as transições com o mesmo estado de origem e a coluna o estado destino.

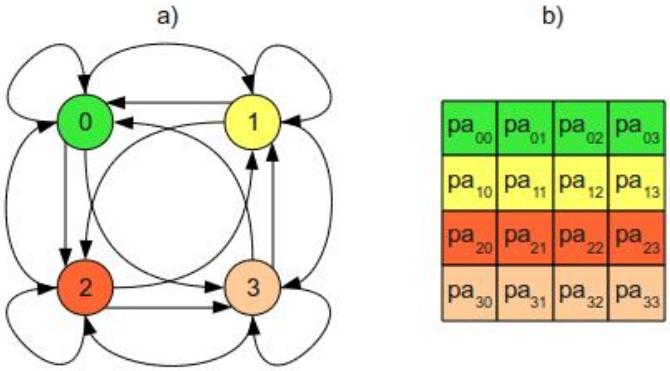

Figura 7: a) representação gráfica. b) matriz de transições.

As probabilidades de transição devem estar contidas no intervalo [0,1], sendo que a soma de todas as probabilidades deve resultar em 1. As probabilidades de transição de cada estado são acumuladas para que um valor escolhido dentro do intervalo [0,1] represente somente uma das transições.

Além da matriz de probabilidades, foi adicionado um atributo inteiro como identificador do estado atual. Também foram adicionados mais dois atributos, do tipo booleano, para indicar se a injeção de falhas será habilitada para o método *send()* e para o *receive()* respectivamente. Apesar do injetor poder ser habilitado simultaneamente para os dois métodos, é utilizada a mesma cadeia de Markov para ambos.

No construtor do DatagramSocket, os atributos que foram adicionados são inicializados a partir de arquivos de configuração. Dentro dos métodos *send()* e *receive()*, se o estado corrente for menor que "2", o pacote é enviado / recebido, caso contrário será perdido. Para calcular o próximo estado, é sorteado um valor randômico no intervalo [0,1), e este valor é comparado com as probabilidades de transição acumuladas na matriz.

4.3 Ajustes do *timeout* para o método *receive*

O *receive()* é, por padrão, um método bloqueante, que espera até ser recebido um novo pacote. Para modificar esse comportamento é possível atribuir um valor de *timeout*, ou seja, um tempo máximo de espera pela recepção de um pacote antes de ser gerada uma exceção.

Originalmente, o controle desse *timeout* é realizado em nível mais baixo que a implementação do DatagramSocket da máquina virtual, causando o reinício da contagem de tempo a cada novo pacote recebido. Isso acontece sempre que uma mensagem for recebida, mesmo que o injetor esteja emulando a perda desse pacote. Entretanto, esse comportamento não corresponde ao que ocorreria no caso de uma perda real de mensagens, pois deveria ser gerada uma exceção de *timeout*. Essa diferença de comportamento é especialmente importante caso o injetor simule uma perda total de conexão, pois isso impediria que a aplicação fosse informada adequadamente dessa perda de conexão. A exceção de *timeout* não seria gerada mesmo que ocorresse uma perda total de conexão emulada, onde todos os pacotes são perdidos.

Para solucionar esse problema foi realizado um ajuste do *timeout*. A classe DatagramSocket modificada mantém o valor de tempo original do *timeout* que foi informado pela aplicação, e este é configurado em mais baixo nível no início de cada

chamada do método *receive()*. Também no início da chamada do *receive()*, é assumido um valor de referência de tempo: cada vez que um pacote é descartado, o tempo transcorrido desde a referência é decrementado do *timeout* original. Caso esse decremento seja maior ou igual ao *timeout* original, é gerada a exceção de *timeout* para a aplicação. Desta forma a falha injetada vai apresentar para a aplicação um comportamento equivalente à falha real.

4.4 Descrição da carga de falhas

Na inicialização, o primeiro arquivo a ser lido é o de configuração inicial, comum a todos os modelos. Através de parâmetros, o arquivo descreve a carga de falhas a ser injetada pelo injetor durante um experimento.

A descrição da carga de falha é composta por quatro números inteiros. Os dois primeiros valores habilitam, caso diferentes de "0", a injeção de falha no envio e recepção, respectivamente. O terceiro valor indica qual modelo será usado, com valores válidos de "0" até "5" representando, respectivamente, os modelos de Bernoulli, Gilbert Simplificado, Gilbert Completo, Gilbert-Elliott, Cadeia de Markov de quatro estados e uma Cadeia de Markov com quatro estados genérica, onde todas as probabilidades de transição podem ser definidas. O quarto valor indica o estado inicial, cujo significado depende do modelo.

5 Avaliação do injetor de falhas para o Android

Para avaliação do injetor de falhas de comunicação Mob-FI foi utilizada uma aplicação simples do tipo Cliente-Servidor como alvo. O Servidor é executado no emulador do Android, e o Cliente roda na máquina virtual no mesmo computador. Para as aplicações Cliente e Servidor poderem se comunicar, é necessário escolher a porta que será utilizada pelo socket UDP e habilitá-la no emulador. Após o emulador iniciar, ele deve ser acessado por telnet no endereço local, na porta do emulador (a porta padrão do emulador é a 5554) e habilitar a porta com o comando de redirecionamento. Neste caso foi utilizada a 3333 para o socket do Servidor. Além de habilitar a porta no emulador, também deve ser habilitada a permissão de acesso a Internet no arquivo AndroidManifest.xml do projeto do Servidor.

O Cliente gera pacotes com um número de sequência e envia para o Servidor. A aplicação envia os pacotes com um tempo de espera suficiente para serem processados pelo Servidor. O tempo foi de 10 milissegundos, e foram enviados 10.000 pacotes.

O Servidor é dividido em duas classes: a aplicação Android principal e uma *thread* de recepção de pacotes. A *thread* de recepção inicia criando o socket UDP e a variável para receber os pacotes. Após, entra em laço, interrompido apenas quando a aplicação for encerrada pelo usuário. Para o teste de avaliação, além de mostrar os resultados no console, o Servidor gera um arquivo, onde cada linha contém o número de pacotes recebidos a cada intervalo de cinquenta pacotes. Para a aplicação conseguir gravar no cartão de memória, foi necessário, além da permissão de acesso a Internet, dar permissão de escrita no seu arquivo AndroidManifest.xml, dentro do projeto da aplicação.

Os testes de avaliação conduzidos comprovaram a capacidade do injetor Mob-FI de emular falhas de acordo com os modelos de perda de mensagens com dois e quatro estados.

5.1 Avaliação do injetor emulando falhas segundo o modelo de Bernoulli

Neste teste, o injetor Mob-FI foi configurado para injetar as falhas na recepção de pacotes de acordo com o modelo para Bernoulli, com a taxa de erro *r* igual a 0,30 (equivalente a 30% de probabilidade de perda de pacote).

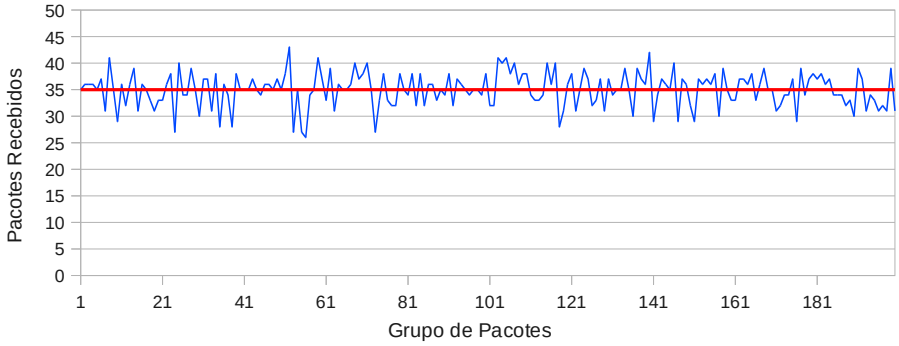

Figura 8: Perdas injetadas de acordo com o modelo de Bernoulli

Na Figura 8, é possível observar a quantidade de pacotes recebidos a cada 50 pacotes consecutivos e, na linha reta, a média de pacotes recebidos durante toda a execução.

Na recepção, os valores estão próximos da média de 35 pacotes, equivalente aos 70% esperados de acordo com a taxa de perda de 30% configurados na carga de falhas para o experimento. O Servidor recebeu 69,99% dos pacotes enviadas pelo Cliente comprovando que o injetor operou corretamente para a carga de falhas descrita.

5.2 Avaliação do injetor emulando falhas segundo os modelos de Gilbert

O injetor foi configurando para o modelo simplificado de Gilbert, com o parâmetro *p* igual a 0,001 e *r* igual a 0,005. Essa carga de falhas corresponde a uma probabilidade de 0,1% de ocorrer uma rajada de perdas de pacote, que deve finalizar com probabilidade de 0,5%.

Como pode ser visto na Figura 9, ocorrem rajadas de erro, onde nenhum pacote é recebido. As rajadas de perda próximas aos grupos de pacotes 70 e 100 não alcançaram o valor zero: são rajadas curtas, pois estão divididas em dois grupos por mensagens que puderam ser recebidas. Nesse teste de avaliação o Servidor recebeu 79,75% dos pacotes enviadas pelo Cliente.

Em um novo teste, agora operando no modelo completo de Gilbert, foi mantido o parâmetro *p*. O parâmetro *r* foi diminuído para aumentar o tamanho da rajada e foi configurada a taxa de envio em estado de erro *h* para 0,20. Com isso, em lugar de não se receber nenhum pacote durante as rajadas de perda, serão recebidos cerca de 20% dos pacotes.

Neste experimento de teste, as rajadas ocorrem de forma semelhante ao experimento anterior (seguindo o modelo simplificado de Gilbert), porém apresentam uma maior duração.

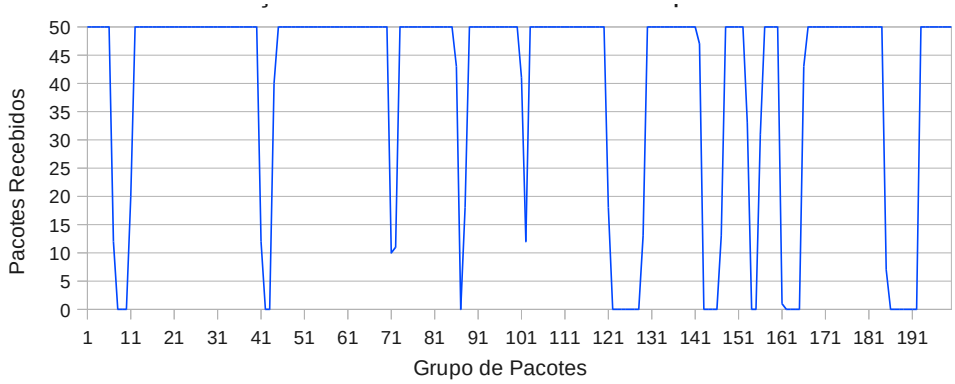

Figura 9: Injeção de perdas seguindo o modelo de Gilbert simplificado

Na figura 10, foi marcada a linha correspondente a 10 pacotes, que equivale a 20% dos 50 pacotes por grupo.

A não ser na rajada de perdas próxima ao grupo 130, que foi a mais curta da execução, as demais rajadas apresentaram um número de pacotes recebidos próximos aos 20% configurados no parâmetro *h*.

Nessa execução do experimento de teste, o Servidor recebeu 86,49% dos pacotes enviadas pelo Cliente.

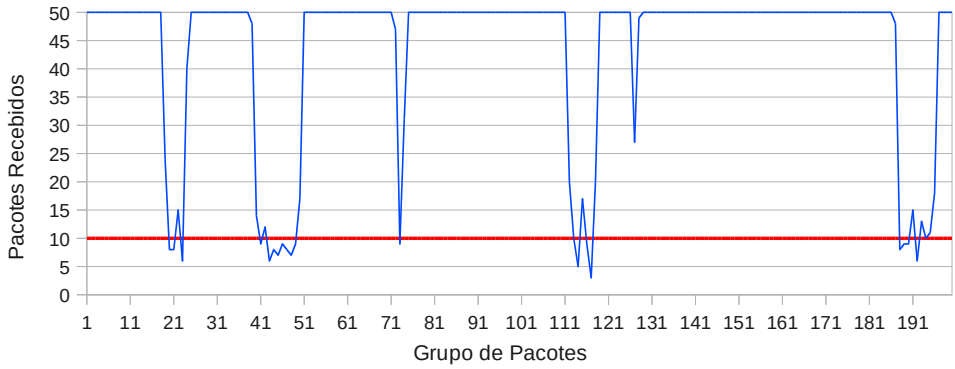

Figura 10: Injeção de perdas seguindo o modelo de Gilbert

5.3 Avaliação do injetor aplicando o modelo de Gilbert-Elliott

O teste realizado com o modelo de Gilbert-Elliott mantém o valor dos parâmetros *p* e *r* dos testes conduzido com os modelos de Gilbert mostrados anteriormente, mas o valor de *h* foi diminuído para 0,1 e o valor do parâmetro *k* foi configurado em 0,85. Com essa configuração, há uma diferença razoável entre as taxas de recepção no estado bom e no estado ruim, facilitando a diferenciação desses estados.

Foram marcadas, na Figura 11, as taxas de recepção esperadas no estado bom, linha reta superior, e do estado ruim, linha reta inferior.

O experimento mostrou que o comportamento da aplicação alvo está coerente com o esperado. Nessa execução do experimento de teste, o Servidor recebeu 70,99% dos pacotes enviadas pelo Cliente.

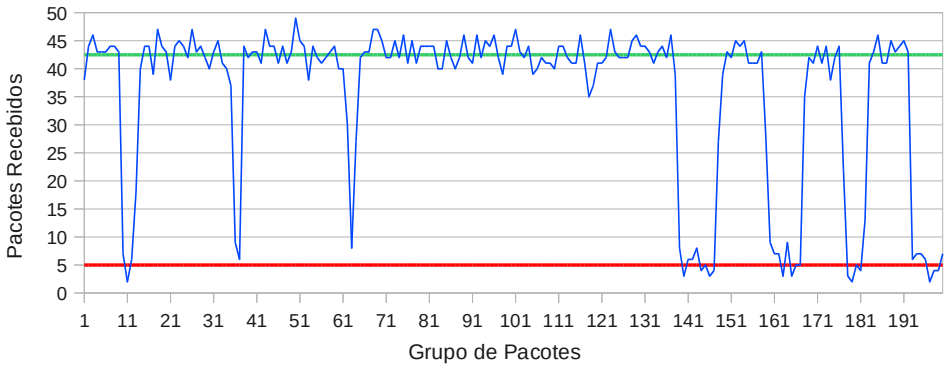

Figura 11: Injeção de perdas seguindo o modelo de Gilbert-Elliot

5.4 Avaliação do injetor aplicando o modelo com quatro estados

Para este teste de avaliação do injetor, os parâmetros do modelo com quatro estados foram configurados de forma a emular rajadas de perda de forma semelhante ao modelo de Gilbert e de Gilbert-Elliot, mas com taxa pequena de perdas no estado bom e uma taxa de transmissão no estado ruim um pouco maior, e de maior duração. Os valores atribuídos aos parâmetros foram *p13* igual a 0,001, *p14* igual a 0,005, *p23* igual a 0,25, *p31* igual a 0,002 e *p32* igual a 0,006.

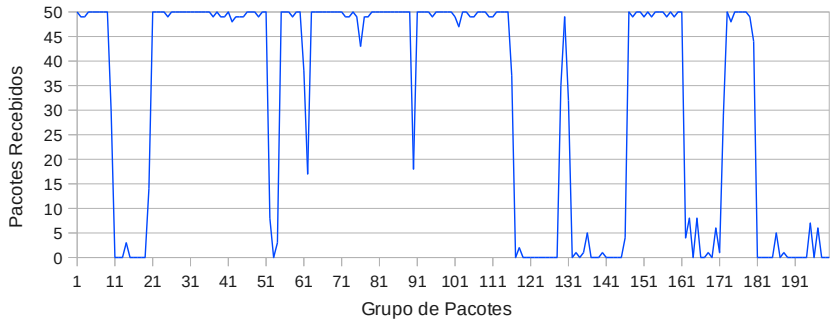

Figura 12: Injeção de perdas seguindo o modelo de quatro estados

No Figura 12 é possível observar os erros simples no estado bom, embora ocorram mais de um dentro de um mesmo grupo em alguns pontos. Também podem ser observados poucos pacotes transmitidos no estado ruim. Com essa configuração ocorreram mais perdas e as rajadas tiveram maior duração. Nessa execução o Servidor recebeu 63,76% dos pacotes enviadas pelo Cliente.

Os testes de avaliação mostraram a capacidade do injetor de emular falhas de acordo com os modelos escolhidos na descrição das cargas de falha. Desta forma, um desenvolvedor ou a equipe de teste pode escolher o modelo de falhas mais apropriado ao teste de sua aplicação móvel, aplicar o injetor Mob-FI em um experimento de teste e com isso determinar o comportamento da aplicação alvo quando sujeita a um cenário de perdas de pacotes. Os cenários emulam situações reais de perdas de pacote em redes sem fio e permitem o teste da aplicação alvo em um ambiente controlado facilitando a observação do comportamento da aplicação alvo sob falhas.

6 Conclusão

O Android, sendo um sistema de código aberto, com vasta documentação e de simples utilização, permite que até mesmo desenvolvedores menos experientes possam criar

aplicações. Somando-se isso com a facilidade de disponibilizar software na web, percebe-se uma oferta muito grande de aplicações vindas das mais diversas fontes, muitas delas desconhecidas e, portanto, pouco confiáveis. Além disso, os dispositivos móveis, como os que executam o sistema Android, estão sujeitos a operar em condições de interferência, atenuação e perda de sinais, o que se traduz em uma grande probabilidade de ocorrência de falhas de comunicação.

Assim, é fundamental dispor de ferramentas de injeção de falhas, que possam ser usadas tanto pelo desenvolvedor e equipe de teste de uma aplicação, que devem verificar a tolerância de suas aplicações a variados cenários de falhas, quanto pelo usuário comum, que tem interesse em saber se o aplicativo que vai empregar será confiável no ambiente real de operação.

Neste artigo, foram relatados os resultados da pesquisa sobre as opções para injetar falhas a partir da máquina virtual do Android. As ferramentas mais comumente usadas para instrumentação nas máquinas virtuais Java convencionais não podem ser utilizadas, devido às diferenças nos bytecodes e na arquitetura da Dalvik Virtual Machine, a máquina virtual do Android. Sendo o código do Android livremente disponibilizado, foi possível explorar a solução de alterar os recursos do núcleo da máquina virtual Java.

Para desenvolver o injetor e testá-lo no emulador de dispositivos, foi necessário o estudo aprofundado de todo o processo de preparação do sistema, obtenção do código fonte, alteração do código e compilação com o SDK. Assim, foi possível criar um injetor de falhas versátil, que consegue emular adequadamente alguns dos modelos mais difundidos de falhas de comunicação do tipo perda de pacotes em rajada. Além de ser flexível, o injetor mostrou baixa interferência no desempenho, o que é importante para garantir que o sistema instrumentado apresente comportamento mais próximo possível daquele de um sistema não instrumentado.

O injetor está disponível para ser empregado em experimentos de injeção de falhas tendo como alvo qualquer aplicação desenvolvida em Java para dispositivos móveis Android, cuja comunicação seja baseada no protocolo UDP. Trabalhos futuros incluem a extensão do injetor para outros protocolos de comunicação e a validação usando o injetor em aplicações que apresentem requisitos explícitos de confiabilidade e disponibilidade.

7 Referências bibliográficas

Acker, E. V. ; Weber, T. S. ; Cechin, S. L. (2010) “Injeção de falhas para validar aplicações em ambientes móveis”. In: Workshop de Testes e Tolerância a Falhas, 11., 2010, Gramado. XI Workshop de Testes e Tolerância a Falhas. v. 1. p. 61-74.

Bornstein, Dan (2008) “Dalvik Virtual Machine Internals”. Google I/O Developer Conference, 2008 - imamu.edu.sa

Butler, M. (2011) “Android: Changing the mobile landscape”. IEEE Pervasive Computing, 2011 - computer.org

Dawson, S; Jahanian, F.; Mitton, T. (1996) “ORCHESTRA: A Probing and Fault Injection Environment for Testing Protocol Implementations”. Proceedings of IPDS’96. Urbana-Champaign, USA.

De, P.; Anindya Neogi, Tzi-cker Chiueh. (2003) “VirtualWire: A Fault Injection and Analysis Tool for Network Protocols”. 23rd IEEE International Conference on Distributed Computing Systems, pp. 214

Drebes, R. J.; Jacques-Silva, G.; Trindade, J.; Weber, T. S. (2006) “A Kernel based Communication Fault Injector for Dependability Testing of Distributed Systems.” In: First Int. Haifa Verification Conf., Springer-Verlag. v. 3875. p. 177-190.

Elliott, E.O. (1963) "Estimates of Error Rates for Codecs on Burst-Noise Channels", In: Bell System Technical Journal 42, p. 1977-1997

Gilbert, E. N. (1960) “Capacity of a Burst-Noise Channel”. In: Bell System Technical Journal, vol.39.

Gindri, A. F. (2011) “Estudo de Injeção de Falhas para a Máquina Virtual do Sistema Android”. UFRGS. lume.ufrgs.br.

Hoarau, W.; Sebastien Tixeuil, Fabien Vauchelles (2007) “FAIL-FCI: Versatile fault injection” , Future Generation Computer Systems, Volume 23, Issue 7, Pages 913-919.

Hohlfeld, O., Geib, R., Haßlinger, G. (2008) “Packet Loss in Real-Time Services: Markovian Models Generating QoE Impairments”. In: Proceedings of IWQoS, 2008. p. 239-248

Hsueh, Mei-Chen; Tsai, T. K.; Iyer, R. K. (1997) “Fault Injection Techniques and Tools”. Computer, pp. 75-82.

Jacques-Silva, G. ; Drebes, R. J. ; Gerchman, J. ; Trindade, J. ; Weber, T. S.; Jansch-Pôrto, I. (2006) “A Network-level Distributed Fault Injector for Experimental Validation of Dependable Distributed Systems.” In: 30th Annual Int. Computer Software and Applications Conf., Chicago. IEEE Computer Society Press, 2006. v. 1. p. 421-428.

Looker, N.; Munro, M.; Xu, J. (2005) “A Comparison of Network Level Fault Injection with Code Insertion”. In: Computer Software and Applications Conference, 2005, v. 1. p. 479 – 484

Menegotto, C. C.; Weber, T. S. (2011) “Communication fault injection for multi-protocol Java applications testing.” In: Latin-American Test Workshop, 12th IEEE, 2011, Porto de Galinhas. LATW 2011, 2011. v. 1. p. 1-14.

Salsano, S.; Ludovici, F.; Ordine, A. (2009). “Definition of a general and intuitive loss model for packet networks and its implementation in the Netem module in the Linux kernel”. Technical report, University of Rome “Tor Vergata”, October 2009.