Injeção de falhas para validar aplicações em ambientes móveis

Eduardo Verruck Acker[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

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

Abstract. *It is assumed that the Android platform for smartphones will allow porting a large number of applications for these and other mobile devices. These applications, however, should be carefully tested including the test under faults. This paper discusses the strengths and difficulties of working with this new mobile platform, presents the experience of porting a communication fault injector to Android and introduces a project that aims to provide fault injection tools for testing applications in mobile environments considering the specificities of faults that can occur in these environments.*

Resumo. *Presume-se que o Android, a plataforma móvel para smartphones originalmente desenvolvido pela Google vá permitir portar um grande número de aplicações para esses e outros dispositivos móveis. Essas aplicações deverão ser cuidadosamente testadas, inclusive na ocorrência de falhas. Esse artigo discute as facilidades e dificuldades de trabalhar com esse novo ambiente, apresenta o porte de um injetor de falhas de comunicação para o Android e introduz a linha de pesquisa de um novo projeto, visando prover ferramentas de injeção de falhas para teste de aplicações em ambientes móveis, considerando as peculiaridades das falhas de comunicação que ocorrem nestes ambientes.*

1 Introdução

Para testar técnicas empregadas na detecção e correção de falhas em um sistema que deva atender a algum critério de dependabilidade, é necessária a ocorrência efetiva de falhas. Contudo, essa ocorrência é aleatória e incontrolável. Além disso, a ocorrência real de falhas apresenta taxas relativamente baixas e, portanto, inadequadas para o tempo restrito de aplicação dos testes. Como é inviável ficar esperando que uma falha específica ocorra naturalmente, devem ser usadas técnicas que emulem falhas de forma controlável e com taxas adequadas. Uma solução é injeção de falhas. Essa abordagem de validação está bem consolidada na literatura [Arlat 1990]. Existem diversos injetores de falhas reportados [Hsueh 1997], mas muitos apresentam restrições quanto à disponibilidade e ao sistema alvo ao qual o injetor pode ser aplicado.

Sistemas computacionais móveis estão se popularizando rapidamente. O ambiente móvel traz novos e interessantes desafios. As técnicas de tolerância a falhas baseadas em replicação de componentes não podem, muitas vezes, ser diretamente empregadas devido à limitação de recursos do ambiente, como restrições de energia, volume e peso. Ambientes móveis são mais sujeitos a interferências ambientais, o que torna os dispositivos móveis mais susceptíveis a erros. Tais erros devem ser detectados e corrigidos com o mínimo de percepção do usuário [Krishna 1993] e usando o mínimo

de recursos. Técnicas de tolerância a falhas vêm sendo adaptadas ou desenvolvidas para lidar com essas restrições e suas implementações precisam, portanto, ser cuidadosamente testadas. Aplicações que as empregam precisam ser validadas na presença de falhas.

O artigo trata da validação de aplicações para ambientes móveis através do uso de injeção de falhas. O sistema alvo são aplicações desenvolvidas para o Android. O Android é um ambiente aberto desenvolvido para dispositivos móveis, baseado no kernel Linux versão 2.6, e que permite a criação de aplicações por qualquer programador. Prevê-se que o ambiente será incorporado em um grande número de aparelhos e uma quantidade enorme de aplicações estará disponível em pouco tempo. Muitas dessas aplicações, provavelmente, serão oferecidas prometendo garantir requisitos relacionados à dependabilidade [Avizienis 2004] tais como confiabilidade, segurança funcional e disponibilidade.

A linha de investigação do projeto apresentado ao final do artigo visa propor uma ferramenta de injeção de falhas como parte dos recursos necessários a validação, certificação ou benchmark de dependabilidade das aplicações móveis com requisitos de dependabilidade. O projeto sendo proposto tem por objetivo pesquisar modelos de falhas que capturem o comportamento das falhas em ambientes móveis, desenvolver ferramentas de injeção de falhas para emular as falhas do modelo e aplicar essas ferramentas em benchmarks de dependabilidade de aplicações móveis.

Esse artigo mostra como foi realizado o porte de um injetor de falhas de comunicação para o ambiente Android, o primeiro passo já realizado do projeto proposto. Na seção 2 são apresentados os conceitos de injeção de falhas. Na seção 3 discutem-se os sistemas móveis e o Android. Na seção 4 são apresentadas as ferramentas utilizadas para o desenvolvimento do projeto e na seção 5 a sua integração, comentando as dificuldades encontradas nessa etapa. Na seção 6 são apresentados os testes realizados para comprovar a validade do porte realizado. Na seção 7 discute-se a continuidade do trabalho e por fim, na seção 8 encontram-se as conclusões finais do artigo.

2 Injeção de falhas

Injetores de falhas podem ser construídos em hardware ou software. Os injetores de falhas por hardware adicionam, ao sistema alvo sob teste, componentes físicos que servem para emular e monitorar falhas. Nesse método, o injetor e o monitor não usam recursos do sistema sob teste, não afetando assim o seu desempenho. Contudo, há um custo adicional significativo devido aos componentes extras e restrições à quantidade de pontos de inserção e monitoração de falhas. A injeção via software é mais fácil de implementar e é mais flexível do que por hardware, mas apresenta desvantagens como uma possível interferência na carga de trabalho do sistema sob teste e a limitação da injeção apenas a locais acessíveis ao software [Hsueh 1997]. Mas se localizado no kernel do sistema operacional, um injetor de falhas por software apresenta uma grande abrangência de classes de falhas. Ele pode injetar falhas em todos os níveis de abstração superiores àquelas em que se situa e ainda pode operar com baixíssima interferência nesses níveis, aproximando-se das vantagens de um injetor de falhas por hardware, mas sem o ônus do alto custo de implementação.

O foco deste artigo são as falhas que afetam dispositivos móveis. Além de falhas internas ao dispositivo, que comprometem a operação correta de seus componentes de hardware e software, uma importante classe de falhas são as de comunicação. Cristian (1991) definiu um modelo de falhas que se tornou comum para sistemas distribuídos. Esse modelo compreende falhas de colapso, omissão, temporização e bizantina e se aplica a nodos ou mensagens no sistema. O modelo captura o comportamento usual de uma rede de comunicação onde mensagens podem ser perdidas (omissão), chegar atrasadas (temporização) ou ter seu conteúdo alterado (falha bizantina); nodos podem parar (colapso de nodo); links podem ser rompidos (colapso de link). Esse modelo simplificado permite construir injetores de falhas eficientes para o teste do comportamento sob falhas de sistemas distribuídos e para validar protocolos de comunicação. Exemplos de injetores que permitem emular falhas de acordo com o modelo são ORCHESTRA [Dawson 1996], NIST Net [Carson 2003], VirtualWire [De 2003], FIRMAMENT ([Drebes 2005][Siqueira, 2009]), FIONA [Jacques-Silva 2006] e FAIL-FCI [Horau 2007].

Atualmente, o projeto concentra-se em falhas de comunicação e injetores de falhas por software. A forma usual de injetar falhas de comunicação é interceptar mensagens enviadas e recebidas pela aplicação sob teste 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. A aplicação sob teste pode ser um aplicativo de alto nível ou até a implementação de um protocolo de comunicação no nível do kernel. Para interceptar mensagens, são usados ganchos providos pelo sistema alvo como, por exemplo, bibliotecas de comunicação, chamadas de sistema, reflexão computacional, recursos de depuração ou filtros disponíveis no kernel do sistema.

Uma questão em aberto é se o modelo de falhas baseado em Cristian, e amplamente usado para redes nomádicas e sistemas distribuídos, é adequado a sistemas móveis. Vale notar que o modelo de Cristian não inclui, apesar de não excluir explicitamente, atenuação de sinal e variações crescentes e decrescentes no atraso ou perda de mensagens, o que pode corresponder a dispositivos afastando-se ou aproximando-se da base. Também não inclui particionamento de rede, o que é raro ocorrer em redes nomádicas, mas um evento comum em ambientes móveis [Oliveira 2009].

3 Plataformas móveis

Até pouco tempo atrás, toda a mobilidade ocorria apenas através de dispositivos como notebooks ou PDAs. Hoje, smartphones oferecem todos os recursos de um computador convencional [Oliver 2009] como sistema operacional, uma grande variedade de aplicativos, acesso a Internet, todos associados à comodidade de um telefone celular. Breve veremos crescer a diversidade de dispositivos móveis, como tablets e e-readers, de diversos fabricantes.

Várias plataformas móveis [Oliver 2009] são disponíveis para smartphones. A tabela 1 [Admob] mostra a participação no mercado mundial e no continente americano de cada um deles no último trimestre de 2009. Dados comparáveis para os primeiros três meses de 2010 ainda não estavam disponíveis quando o artigo foi escrito. Entretanto, durante o ano de 2009, a participação mundial do ambiente Android subiu de 1%, no primeiro trimestre, para 16%, no último trimestre. Permanecendo a tendência,

é possível já tenha ultrapassado o sistema Symbian da Nokia em participação no mercado mundial.

Outros tipos de dispositivos móveis também podem se beneficiar da disponibilidade dessas plataformas, principalmente se forem ambientes abertos.

Tabela 1: Participação no mercado por plataforma móvel para smartphones, Q4 2009

Sistema Operacional	Mercado Mundial (%)	América do Norte (%)	América Latina (%)
iPhone OS	51	54	56
Symbian	21	-	28
Android	16	27	1
RIM OS	6	10	8
Windows Mobile OS	3	3	6
Outros	3	6	3

Fonte: ADMOB, dezembro 2009

Apesar do mercado de aplicativos para plataformas móveis tender a um forte crescimento, com a consequente necessidade de teste sob falhas dessas aplicações, quase não foram encontradas ferramentas de injeção de falhas utilizadas exclusivamente em ambientes móveis. Uma exceção é a ferramenta mCrash [Ribeiro 2008]. Desenvolvida para o sistema operacional Windows Mobile 5, o mCrash permite testes automáticos para classes, métodos, parâmetros e objetos do framework .NET. Dinamicamente, mCrash gera scripts de teste, compila-os para o .NET, e invoca o processo de teste. Visando estritamente a validação de aplicações Java, parece possível uma adaptação de mCrash para o Android. No projeto, entretanto, pretende-se conduzir investigações com uma gama mais variada de aplicações, incluindo aplicativos de alto nível e recursos no nível de kernel.

3.1 Android

A Open Handset Alliance (OHA) é um grupo de mais de 40 empresas de áreas como operadores de telefonia, software e semicondutores. Liderada pela Google, a OHA visa criar padrões para a indústria da telefonia móvel. O primeiro passo nessa direção é o Android. Lançado oficialmente pela OHA em outubro de 2008, o Android é um ambiente de execução de aplicações para dispositivos móveis baseado no Linux [Chang 2010]. Aplicações podem ser desenvolvidas por terceiros, são facilmente integradas ao ambiente e têm acesso aos mesmos recursos que as aplicações originais dos fabricantes. Para facilitar a criação dessas aplicações, está disponível um kit de desenvolvimento de software (SDK). Entre os diversos componentes de desenvolvimento e depuração do SDK, pode-se destacar o emulador do Android e o conjunto de ferramentas para desenvolvimento integrado com o ambiente de desenvolvimento Eclipse [Eclipse].

O código fonte do Android [Android-source] é aberto e disponível sob licença GPL. O site de desenvolvimento para o Android [Dev-android] contém diversas informações para uso da plataforma de desenvolvimento [Android-git], incluindo as bibliotecas específicas para o sistema, o compilador adequado e os headers do sistema operacional.

3.2 Arquitetura do Android

A arquitetura do ambiente Android é mostrada na figura 1 [Chang 2010]. As suas camadas são:

- Aplicações escritas em Java (como cliente de e-mail, programa de SMS, calendário, mapas, contatos, entre outros).
- Framework de aplicação. Entre os serviços disponibilizados estão: provedores de conteúdo, que compartilham os dados entre aplicativos; gerente de notificação, que disponibiliza diferentes tipos de alertas para as aplicações; e gerente de atividades, que gerencia do ciclo de vida das aplicações.
- Bibliotecas básicas acessíveis através do framework de aplicação, como uma versão compacta da *libc* para sistemas embarcados, bibliotecas de mídia, gerente de navegação (acesso ao display), SQLite (base de dados relacional).
- Android *runtime*. Cada aplicação do Android roda o seu próprio processo, com uma instância própria na máquina virtual Dalvik. A Dalvik requer pouca memória e permite que múltiplas instâncias de sua máquina virtual executem ao mesmo tempo. As aplicações são compiladas em Java e traduzidas para o formato .dex por uma ferramenta do SDK.
- Linux Kernel, baseado no kernel versão 2.6. Usa os serviços de gerência de memória e processos, pilha de rede e controladores de dispositivos. Além disso, o kernel atua como uma camada de abstração entre o hardware e o resto do sistema.

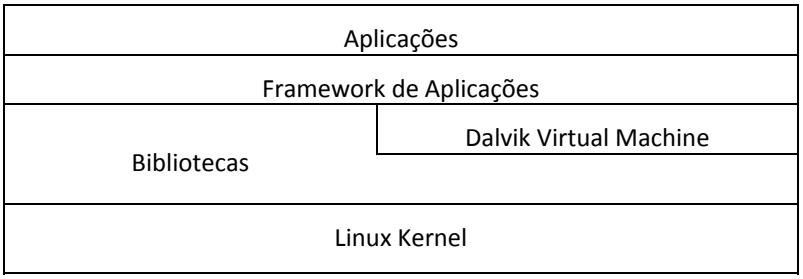

Figura 1: Arquitetura do ambiente Android

Um detalhe a ser ressaltado é que o ambiente Android não oferece a biblioteca *libc* completa, somente uma versão compacta da mesma, chamada Bionic. Esse problema é o maior empecilho para o reaproveitamento de códigos desenvolvidos para outros sistemas Linux. As razões para utilizar uma nova biblioteca em detrimento a *libc* são devidas principalmente: ao uso da licença BSD (se fosse empregada a *libc* original, a licença GPL contaminaria o espaço de usuário); à necessidade de uma biblioteca com tamanho reduzido que deve ser carregada em cada processo (a Bionic usa somente em torno de 200KB); e à inclusão de funções específicas para a melhor o desempenho do Android.

3.3 Sistema de desenvolvimento e emulação para o Android

O Kit de Desenvolvimento de Software (SDK) foi lançado junto com o Android. A versão 1.6, usada no projeto, é composta por: Android Virtual Devices (AVDs), Emulador, o Android Debug Bridge (ADB), entre outras.

Um dos principais componente do SDK é o emulador. Todo o desenvolvimento de aplicativos para o Android pode ser feito para rodar nesse emulador, que executa em computadores convencionais. O emulador do Android executa uma máquina virtual chamada Goldfish que opera sobre instruções ARM926T e que disponibiliza ganchos para entrada e saída de dados. Nela há arquivos específicos para a exibição na tela, o teclado do dispositivo é emulado pelo teclado do computador host, e a tela *touch screen* é emulado pelo mouse. Essas interfaces são usadas somente pelo emulador, não sendo compiladas quando o código deverá rodar em um dispositivo físico. Note-se que se pode fazer o download dos códigos fontes do kernel, para o emulador ou para os dispositivos, no mesmo link da plataforma de desenvolvimento. A versão usada para este trabalho foi a 2.6.29.

O emulador provê os componentes típicos de um aparelho celular, como botões de navegação, um teclado, e até mesmo uma tela que pode ser clicada com o mouse. No entanto, há algumas limitações, como não ter suporte a conexões USB, à câmera, a fones de ouvido, ao estado de carga da bateria, a Bluetooth e, é claro, a realização de chamadas telefônicas. As chamadas podem ser simuladas através da geração de um evento específico no console do emulador. O emulador funciona como um processo normal da máquina de desenvolvimento e comunica-se através de um roteador virtual, o qual não tem acesso direto à rede. Assim sendo, ele está sujeito às mesmas limitações que a sua rede impõe aos outros processos. O roteador suporta todos os tráfegos TCP e UDP do emulador. Contudo, outros protocolos, como o ICMP, ainda não são suportados. Atualmente, a falta de suporte do emulador se estende para mensagens multicast e IGMP.

3.4 ADB - Android Debug Bridge

O Android Debug Bridge permite a manipulação de um emulador ou um aparelho físico que esteja conectado à máquina de desenvolvimento. Ele é um programa cliente/-servidor baseado em três componentes: cliente, servidor e Daemon. O cliente e o servidor são executados na máquina de desenvolvimento. O servidor é responsável pela intermediação dos comandos originados do cliente e enviados ao Daemon. O Daemon é executado no emulador ou no aparelho móvel como um processo em background.

Ao se inicializar um cliente ADB, será feita a busca por um servidor operacional. Não encontrando um, o cliente vai disparar uma instância de servidor. Quando rodando, o servidor utiliza a porta TCP 5037 da máquina de desenvolvimento para receber comandos dos clientes. Todos os clientes enviam suas mensagens para essa mesma porta. Ao inicializar-se um servidor, são varridas todas as portas pares entre 5554 e 5584, buscando por um emulador ou dispositivo físico conectado. Naquela porta onde for encontrado um Daemon ADB, o servidor irá conectar-se. Observe que, quando for executado um emulador ou conectado um dispositivo físico à máquina de desenvolvimento, serão ocupadas duas portas em sequência: a porta par para o emulador ou dispositivo e a porta ímpar para o seu ADB.

4 Injetor de falhas para o Android

Para efetuar as primeiras avaliações da viabilidade de desenvolver um injetor de falhas em um dispositivo móvel, decidiu-se adaptar um injetor já existente ao novo ambiente. Dessa forma, as dificuldades do trabalho seriam restritas àquelas oferecidas pelo ambiente móvel e não pelo injetor. Adicionalmente, seria possível identificar quais

decisões de projeto usadas para implementar o injetor poderiam ser inconvenientes e, então, possibilitar uma especificação mais adequada de um novo injetor.

Entre os injetores disponíveis no grupo de pesquisa, um deles opera no kernel e vários outros operam com recursos de interceptação em Java, usando reflexão computacional ou recursos de depuração da máquina virtual. Esses últimos injetores são específicos para um determinado grupo de protocolos (TCP, UDP ou RMI isoladamente ou com os três protocolos simultaneamente), podem ser estendidos para novos protocolos de mais alto nível, mas visam exclusivamente o teste de aplicações escritas em Java. Como foram todos desenvolvidos também em Java, eles não são muito úteis para avaliar as peculiaridades das camadas de mais baixo nível do ambiente Android. O injetor ideal, neste caso, é o injetor que opera no kernel.

4.1 Injetor base: FIRMAMENT

A comunicação na Internet utiliza o modelo TCP/IP para troca de mensagens. Em consequência disso, as alterações feitas em pacotes IP irão se refletir em todos os outros protocolos encapsulados nele. Devido a esse uso tão comum da pilha TCP/IP, o protocolo IP está implementado no kernel do Linux e, dessa forma, atuando a partir do nível IP no kernel é possível injetar falhas em qualquer protocolo que rode sobre o IP.

Infelizmente, atuando a partir do kernel perdem-se muitas das vantagens associadas à injeção de falhas nos níveis de abstração mais altos [Menegotto 2007]. Para poder aproveitar tais vantagens, o porte e adaptação desses injetores de nível mais alto estão previstos para ser efetuados posteriormente no projeto, mas apenas quando o modelo de falhas para ambientes móveis tiver sido estabelecido e as adaptações necessárias realizadas.

O injetor FIRMAMENT [Drebes 2006] opera como um módulo no kernel e foi desenvolvido para a avaliação de protocolos de comunicação com o mínimo de interferência possível nas aplicações sob teste e para o máximo de abrangência em relação aos protocolos nos quais injetar falhas. Para tanto, é usada uma interface de programação do kernel, o Netfilter [Russel 2002], que fornece ganchos para a pilha de protocolos TCP/IP. Somente alguns dos ganchos disponíveis no Netfilter são necessários pelo injetor. De forma semelhante, não foram necessárias todas as funções disponibilizadas pelo Netfilter, tendo sido usadas apenas aquelas necessárias para realizar injeção de falhas.

Para descrever cargas de falha, o FIRMAMENT introduziu o conceito do *faultlet*. Um *faultlet* pode especificar atraso, duplicação ou descarte de um pacote, entre outras ações possíveis, sobre um dado pacote selecionado. O *faultlet* pode também especificar condições de seleção dos pacotes. Essas características permitem a criação de estruturas complexas para a injeção de falhas. Os *faultlets* podem ser configurados de forma independente para os fluxos de entrada e saída do protocolo IP. A máquina virtual FIRMVM tem a função de executar o *faultlet*, o qual é associado ao pacote e às variáveis de estado. Ela trabalha sobre as entradas e saídas de mensagens dos protocolos IPv4 e IPv6, sendo esses os pontos de injeção de falhas. A cada fluxo está associado um conjunto de variáveis de estado que é formado por 16 registradores de 32 bits. Essa máquina virtual é a que foi portada para o ambiente Android.

5 Sistema de desenvolvimento

Para que fosse possível instalar o injetor de falhas no Android, foram usadas as ferramentas descritas anteriormente. Além disso, foram seguidos os procedimentos documentados para a correta instalação dos módulos do injetor no ambiente Android. Um ponto a favor do Android é a disponibilidade de informação para desenvolvimento no ambiente. As dificuldades iniciais decorrentes do desconhecimento de um novo sistema foram sanadas com o estudo dos manuais de desenvolvimento. Quando esses não eram suficientes, as listas de discussão [Android-kernel] possibilitaram superar as dificuldades.

Uma das dificuldades enfrentadas foi o aprendizado da compilação cruzada (*cross compilation*) e das diferenças entre as formas de geração de código para o emulador ou para um dispositivo físico.

5.1 Porte do injetor através do sistema de desenvolvimento

O ambiente de desenvolvimento utilizado foi um computador com processador Intel E7400, sistema operacional Windows (host) e uma máquina virtual VirtualBox v2.2.2 executando Ubunto versão 8.04 e kernel 2.6.24. Dessa forma, o desenvolvimento é feito em um desktop, os fontes devem ser compilados no desktop tendo como alvo o Android, e o código-objeto deve ser adequadamente transferido para o Android (ou para o simulador). Os procedimentos, em termos gerais, são os seguintes:

- Obter os fontes da plataforma de desenvolvimento, do kernel (Goldfish) e o SDK do Android;
- Cross-compilar os fontes do FIRMAMENT, para que possam executar no Android;
- Recompilar o kernel do Android. Esse procedimento é necessário, pois o kernel do Android não tem, por padrão, o Netfilter, que é essencial para a operação do injetor;
- Criar um AVD (Android Virtual Device), para possibilitar executar o kernel e o injetor em um emulador de um dispositivo real, capaz de rodar o Android.

Uma vez obtidos todos os arquivos necessários para operação do injetor no emulador do Android, deve-se seguir os seguintes procedimentos:

- Iniciar, no desktop, uma instância do emulador, indicando o AVD criado e o kernel compilado;
- Iniciar, em outro terminal, o ADB (Android Debug Bridge), de maneira a poder interoperar com o emulador (ou o equipamento que roda o Android);
- Efetuar a transferência dos arquivos desejados para o emulador (ou equipamento) usando o comando *push* do ADB.

O detalhamento desses procedimentos pode ser encontrado no mesmo site onde estão o código fonte e as ferramentas de desenvolvimento [Dev-android].

5.2 Aplicações Java versus Aplicações “C”

O Android é baseado no kernel 2.6 do Linux. Ele substitui alguns módulos (a *libc*, por exemplo), mas o comportamento e a operação geral são semelhantes. Essa informação não é importante para quem desenvolve em Java (usando o Eclipse, por exemplo).

Entretanto, ela abre uma porta para aqueles que desenvolvem em ambiente Linux, pois, idealmente, deveria ser possível portar as aplicações que rodam em um Linux com kernel 2.6 para o Android. Infelizmente, como já foi dito, nem sempre isso é possível. E um dos motivos é o uso da biblioteca Bionic em lugar da *libc*. De qualquer forma, mesmo com essa limitação, o porte das aplicações escritas em "C" para o Android é menos oneroso do que um novo desenvolvimento, mesmo que sejam necessários alguns ajustes. No caso do FIRMAMENT, cujo código está escrito em "C" e roda sob o kernel 2.6, o porte aconteceu sem a necessidade de ajustes significativos.

5.3 Compilando o Kernel e transferindo arquivos para o Android

Nas primeiras tentativas de compilação do kernel, seguiu-se a documentação [Motz]. Entretanto, devido ao fato dessa documentação estar desatualizada, os compiladores e bibliotecas não são os mais indicados. Atualmente, é recomendada a utilização do kernel, compilador e bibliotecas específicas do Android [Android-git].

Após compilar o kernel e o FIRMAMENT, os arquivos objeto foram transferidos para seus devidos lugares, no sistema de arquivos do Android, e o emulador foi iniciado, tendo suas funções básicas operado corretamente.

6 Experimentos com o injetor de falhas no Android

São apresentados os experimentos realizados para comprovar se o porte foi efetivo e se o comportamento do injetor não sofreu alterações quando executado no novo ambiente. Esse foi o primeiro passo visando introduzir, no futuro, as adaptações necessárias no novo injetor, de maneira a incorporar um modelo de falhas adequado a ambientes móveis. O ambiente de testes é um computador com processador Intel E7400, sistema operacional Windows como hospedeiro de uma máquina virtual VirtualBox V2.2.2 executando Ubuntu versão 8.04 kernel 2.6.24.

O primeiro experimento injetava falhas em pacotes TCP e foi realizado para verificar se o FIRMAMENT e o Netfilter estavam minimamente operacionais. Os dois últimos experimentos foram realizados injetando-se falhas em mensagens UDP. Em todos os experimentos, o servidor foi executado no emulador e o cliente na máquina hospedeira do desenvolvimento. Como o servidor envia mensagens diretamente para um IP e porta específicos, caso se desejasse inverter a situação, com o cliente no emulador, seria necessário fazer o redirecionamento de portas da máquina de desenvolvimento para a porta em que o cliente estaria sendo executado no Android.

Convém observar que nem todas as funcionalidades do injetor foram convenientemente testadas no novo ambiente. É possível que algumas surpresas apareçam no decorrer de novos testes, mas os experimentos que apresentamos neste artigo são uma amostra significativa da compatibilidade do Linux do Android com os sistemas convencionais.

6.1 Operação básica do injetor no ambiente Android

Visando verificar se o FIRMAMENT e o Netfilter estavam operacionais após serem portados para o ambiente Android, para o primeiro experimento construiu-se um *faultlet* que selecionava todas as mensagens TCP e injetava falhas de descarte em todas as mensagens selecionadas. O resultado foi que o emulador travou totalmente, não sendo mais possível qualquer tipo de interação entre o ADB e o emulador.

Depois de alguma investigação, identificou-se que toda a comunicação entre ADB e emulador é feita através de mensagens TCP, apesar do ADB e o emulador estarem rodando na mesma máquina física. O mesmo ocorreria caso fosse tentada a comunicação com um dispositivo físico, Com isso pode-se identificar que toda a comunicação entre o ADB (rodando em uma máquina de desenvolvimento) e o Android (rodando como emulador ou em um dispositivo físico) utiliza uma única pilha TCP, de maneira a reduzir ao máximo o uso dos recursos dos dispositivos. Portanto, no nível do kernel, não existe discriminação entre mensagens de aplicação (navegador, por exemplo) e de sistema, como são as mensagens de controle e depuração trocadas com o ADB.

Portanto, numa segunda rodada deste experimento, para verificar se o injetor estava operando corretamente, o *faultlet* inicial foi alterado de maneira a descartar as mensagens dirigidas a todos os endereço IP, exceto aquele onde estava rodando o ADB. O resultado foi que a comunicação entre o emulador do Android e o ADB permaneceu operacional, mas não era possível navegar na internet usando o navegador do Android, como esperado. O experimento mostrou que o porte foi realizado com sucesso, mas os experimentos devem ser conduzidos com cuidado e os *faultlets* devem levar em consideração o ambiente de emulação e suas particularidades.

6.2 Descarte de Pacotes

Esse experimento mostra a capacidade do injetor em descartar pacotes UDP, quando operando no emulador. O injetor executa um *faultlet* que primeiro identifica o tipo de pacote que se deseja capturar, no caso, um pacote UDP. Em seguida é verificado o endereço de destino do pacote, pois foi decidido carregar o *faultlet* no fluxo de saída *ipv4_out*. Caso fosse colocado no fluxo de entrada, deveria se avaliar o remetente do pacote. Por último, para emular o descarte estatístico de pacotes, é feita a escolha dos pacotes a serem descartados com um fator aleatório e uniforme de 10% dos pacotes enviados.

A comunicação foi estabelecida através de um sistema cliente/servidor. O servidor foi acionado no emulador Android e enviou as mensagens para o cliente que estava na máquina hospedeira Windows. Para monitoração das mensagens perdidas, todas as mensagens enviadas continham uma numeração sequencial, lida pelo cliente. Por sua vez, o cliente também fez uma contagem de mensagens recebidas. O servidor foi programado para o envio de 10000 mensagens, com um tempo de 100ms entre cada uma, o que correspondeu à carga de trabalho do experimento.

A tabela 2 mostra o resultado médio das rodadas com e sem a atuação do injetor FIRMAMENT portado para o Android. Observa-se que a quantidade média de pacotes descartados foi um pouco superior aos 10% previstos, mas isso se deve principalmente ao procedimento de randomização usado para alcançar uma distribuição uniforme de probabilidade de perda de mensagem descrita no *faultlet*.

Tabela 2: Resultado do descarte dos pacotes

Rodada	Pacotes recebidos (%)	Tempo para envio (s)
Sem ação do injetor	100	1020
Com ação do injetor	89,63	1022

As rodadas foram executadas um número significativo de vezes de maneira a obterem-se valores minimamente confiáveis para verificar se o injetor operando no emulador Android conseguia efetivamente descartar mensagens, o que foi comprovado pelo experimento. Não era objetivo do experimento a obtenção de medidas precisas para uma avaliação de desempenho.

6.3 Atraso de Pacotes

Esse experimento simula o atraso de pacotes enviados pela rede. É muito comum a ocorrência de atrasos nas transmissões de dados, seja por causa de tráfego, ou mesmo devido ao atraso no processamento do pacote recebido. O experimento injeta um atraso variável de 12 ±5ms em cada mensagem selecionada. O valor do atraso foi escolhido unicamente para facilitar as medidas, não correspondendo a um cenário de falhas representativo de ambientes móveis. No estágio atual do projeto esse cenário de falhas representativo ainda não é conhecido.

No experimento, o envio e recebimento dos pacotes foram feitos com um sistema do tipo cliente/servidor. O servidor foi executado no emulador, enquanto o cliente foi executado na máquina hospedeira Windows. O servidor envia uma mensagem para o cliente e aguarda a resposta. O cliente recebe uma mensagem e envia uma resposta ao servidor. No total, o servidor envia 10000 mensagens, o que corresponde à carga de trabalho do experimento.

No *faultlet* usado para descrever a carga de falhas foi especificado um filtro do tipo de pacote que sofrerá a ação do injetor, no caso pacotes UDP. Em seguida selecionam-se somente as mensagens destinadas ao IP 155.19.72.189. Caso ambas as condições sejam verdadeiras, então o pacote é atrasado. Após ser confirmado o tipo de pacote e o endereço do destinatário, é feito um sorteio com números de -5 a +5. O resultado é somado ao valor constante 12 que fica armazenado em um registrador da máquina virtual do injetor. O valor desse registrador será usado como o atraso para o pacote em milissegundos.

A tabela 3 mostra os tempos de execução de rodadas sem e com a ação do injetor. Neste último têm-se o tempo de execução inserindo-se um atraso variável de 12±5ms. Os resultados permitem concluir que o injetor portado para o Android estava efetivamente promovendo o atraso variável no envio de mensagens.

Tabela 3: Resultado do atraso das mensagens

Rodada	Tempo para envio (s) de 10000 mensagens
Sem ação do injetor	1024
Com ação do injetor (atraso variável)	1127

Assim como no primeiro experimento, as rodadas foram executadas um número de vezes adequado apenas para se obter valores minimamente confiáveis. Não era objetivo do experimento a obtenção de medidas precisas de tempo, apenas verificar que o injetor estava efetivamente injetando atrasos variáveis nas mensagens, como seria esperado se o porte tivesse sido bem sucedido.

7 Trabalhos Futuros

Na continuidade do trabalho desenvolvido pretende-se atuar em duas áreas. A primeira é na direção do estudo do comportamento sob falhas das aplicações que visam fornecer dependabilidade para os dispositivos móveis e a segunda é definir modelos de falha de mobilidade, que serão usados para projetar um novo sistema de injeção de falhas.

O estudo experimental da dependabilidade provida pelas aplicações será feito através de uma série de experimentos de injeção de falhas e monitoração de comportamento, formando um Benchmark de Dependabilidade. Com isso, os usuários e desenvolvedores poderão avaliar a dependabilidade das aplicações, identificar problemas de implementação e comparar diferentes soluções para o mesmo problema, possibilitando que o produto final seja de melhor qualidade. Espera-se que, como resultado desse desenvolvimento, obtenha-se um benchmark que possa ser utilizado pelos desenvolvedores para alcançar aplicações mais robustas, mesmo sem conhecer os aspectos específicos do comportamento das falhas.

No outro ramo de desenvolvimento, o objetivo é modelar de forma mais precisa as falhas que ocorrem nos sistemas móveis. Os modelos resultantes desses estudos deverão ser incorporados aos injetores de falhas de maneira a possibilitar uma reprodução mais fiel dos cenários de falha reais. Acredita-se que os modelos de falha dos sistemas móveis sejam diferentes daqueles encontrados nos sistemas nomádicos. Entretanto, não se pode descartar a possibilidade de emular as falhas típicas dos sistemas móveis através de falhas encontradas nos sistemas nomádicos. A verificação dessa possibilidade assim como a necessidade de construção de um novo injetor de falhas é o resultado esperado nessa linha de investigação.

8 Conclusão

O artigo apresentou os primeiros estudos de uma nova linha que estamos investigando que inclui a validação de aplicações em ambientes móveis sujeitas a falhas de comunicação. Apresentou também os resultados da primeira tarefa deste novo projeto que foi o porte de um injetor de falhas de comunicação, que opera como um módulo no nível do kernel Linux, para o ambiente Android.

A experiência com o novo ambiente foi considerada muito satisfatória. As dificuldades encontradas foram mais devido a pouca familiaridade com o ambiente e imprecisões na documentação do que a restrições ou bugs de implementação do Android. Não é objetivo do projeto testar o ambiente Android, que julgamos ser suficientemente robusto e confiável. O objetivo é avaliar aplicações desenvolvidas para o ambiente Android. Uma grande quantidade dessas aplicações tem sido disponibilizada, tais aplicações surgem vindas das mais diversas fontes. Um benchmark que possa ajudar a avaliar a dependabilidade de aplicações para ambientes móveis é útil não apenas a desenvolvedores, mas também a usuários ou sistemas que precisam depositar confiança no funcionamento correto dessas aplicações.

9 Referências bibliográficas

Admob (2010). admob-mobile-metrics-report-dezember-09. Disponível em: < http://metrics.admob.com/ >. Acessado em: março, 2010.

Android-ADB (2009). Disponível em: < http://developer.android.com/guide/developing/tools/adb.html >. Acessado em: setembro, 2009

Android-git (2009). Disponível em: < http://android.git.kernel.org/ >. Acessado em: setembro, 2009.

Android-kernel (2009). Disponível em: < http://groups.google.com/group/android-kernel >. Acessado em: setembro, 2009.

Android-source (2009). Disponível em: < http://source.android.com/ >. Acessado em: junho, 2009

Arlat, J.; Aguera, M.; Amat, L.; Crouzet, Y.; Laprie, J.; Fabre, J.; Martins, E.; Powell, D. (1990). Fault-injection for dependability validation: a methodology and some applications. IEEE Trans. on Soft. Eng., Special Issue on Experimental Computer Science, 3, vol. 16, n.2, p. 166-82, Feb. 1990.

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

Carson, M.; Santay, D. (2003). NIST Net – A Linux-based Network Emulation Tool. ACM SIGCOMM Computer Communications Review, vol.33, pp.111-126, 2003.

Chang, G.; Tan. C.; Li, G.; Zhu, C. (2010). Developing Mobile Applications on the Android Platform. In: Mobile Multimedia Processing, LNCS 5960, Springer, pp. 264–286

Cristian, F. (1991). Understanding fault-tolerant distributed systems. Communications of the ACM, vol. 34, n.2, p. 56-78

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 (ICDCS'03), pp.214

Dev-android (2009). Disponível em: < http://developer.android.com/index.html >. Acessado em: agosto, 2009.

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.

Eclipse (2009) Disponível em: < http://www.eclipse.org/>. Acessado em: agosto, 2009.

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

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 International

Computer Software and Applications Conference – COMPSAC 2006, Chicago. IEEE Computer Society Press, 2006. v. 1. p. 421-428.

Krishna, P., Vaidya, N., Pradhan, D. (1993). Recovery in distributed mobile environments. In Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems, pp. 83-88.

Menegotto, C. C.; Vacaro, J. C.; Weber, T. S. (2007). Injeção de Falhas de Comunicação em Grids com Características de Tolerância a Falhas. In: VIII Workshop de Teste e Tolerância a Falhas, 2007, Belém. WTF 2007 - VIII Workshop de Teste e Tolerância a Falhas. v. 1. p. 71-84.

Motz (2009). Disponível em: < http://honeypod.blogspot.com/2007/12/compile-android-kernel-from-source.html >. Acessado em: agosto, 2009.

Oliveira, G. M.; Cechin, S.; Weber, T. S. (2009). Injeção Distribuída de Falhas de Comunicação com Suporte a Controle e Coordenação de Experimentos. In: Workshop de Testes e Tolerância a Falhas, João Pessoa. WTF 2009, v. 1. p. 101-114

Oliver, E. 2009. A survey of platforms for mobile networks research. *SIGMOBILE Mob. Comput. Commun. Rev.* 12, 4 (Feb. 2009), 56-63.

Ribeiro, J.C. (2009). mCrash: a Framework for the Evaluation of Mobile Devices' Trustworthiness Properties; Organization: University of Coimbra, Portugal; Supervisor: Prof. Mário Zenha-Rela; Period: 2005-2008; Presentation Date: 17th of December, 2008.

Russel, R.; Welte, H. (2002). Linux net filter hacking HOWTO. 2002. Disponível em: http://www.netfilter.org/documentation/

Siqueira, T.; Fiss, B. C.; Weber, R.; Cechin, S.; Weber, T. S. (2009). Applying FIRMAMENT to test the SCTP communication protocol under network faults. In: 10th Latin American Test Workshop. v. 1. p. 1-6