Emulando o Comportamento de TCP/IP em um Ambiente com Falhas para Teste de Aplicações de Rede

Júlio Gerchman[1], **Taisy S. Weber**[1]

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

{juliog,taisy}@inf.ufrgs.br

Abstract. *Communication faults are inevitable during the use of network applications; those applications should offer mechanisms to tolerate those faults. Fault injection is a flexible, low-cost experimental validation technique for fault tolerance mechanisms. This paper describes FIERCE, a fault injector aimed at testing Java network applications that use the TCP communication protocol. The tool injects faults through Java system class instrumentation. FIERCE emulates the errors that can affect a TCP connection, allowing the test of the application.*

Resumo. *É inevitável a ocorrência de falhas de comunicação durante a execução de aplicações de rede; a aplicação deve apresentar mecanismos para tolerá-las. Injeção de falhas é uma técnica de validação experimental de mecanismos de tolerância a falhas flexível e de baixo custo. Este trabalho apresenta FIERCE, um injetor de falhas de comunicação para teste de aplicações Java que usam o protocolo TCP. A ferramenta injeta falhas de comunicação através da instrumentação das classes de sistema da plataforma Java. FIERCE emula os erros que podem afetar uma conexão TCP, permitindo o teste da aplicação.*

1. Introdução

É irreal considerar que aplicações que usem comunicação via redes de computadores não sofram efeitos de falhas. Pacotes perdidos, congestionamento, problemas de segurança, particionamentos e erros de operação ou programação não só nas pontas como em todos os elementos que compõem essa rede tornam o ambiente altamente propenso à manifestação de defeitos. As aplicações devem considerar mecanismos de tolerância a falhas para que sua *dependabilidade* seja aceitável. Dependabilidade pode ser definida como o grau de confiança que se pode depositar em um sistema computacional e é a soma de vários atributos: confiabilidade, disponibilidade, integridade, *safety*, manutenabilidade e confidencialidade [Avizienis et al. 2004].

Esse nível de dependabilidade é assegurado através da condução de um plano de garantia de qualidade (*quality assurance*). Um bom plano assegura que o projeto é apropriado, a implementação é cuidadosa e o produto vai ao encontro aos requisitos estabelecidos. Um melhor plano inclui a análise de defeitos e uma melhora contínua no produto. A principal atividade usada é o teste de *software* [Feldman 2005].

Entre as diversas técnicas disponíveis para o teste de software, a injeção de falhas apresenta uma alta flexibilidade aliada a um baixo custo de implementação. Um protótipo

funcional do sistema é executado em um ambiente controlado onde a ocorrência de falhas é emulada enquanto que seu comportamento é monitorado e analisado. Diferentes cenários de falhas podem ser especificados, repetindo o teste o quanto necessário. Esta técnica tem como objetivos auxiliar na fase de teste de depuração do software, identificar gargalos de dependabilidade, analisar o comportamento do sistema em um cenário de falhas, determinar a cobertura dos mecanismos de detecção e recuperação de falhas e avaliar sua eficiência e desempenho [Hsueh et al. 1997].

Devido às suas características de tolerância a falhas, o protocolo mais usado para a comunicação de dados pela Internet é o TCP. Grande parte dos protocolos de mais alto nível, como HTTP e RMI, tem ele como base. Pacotes perdidos ou corrompidos são identificados e recuperados através do uso dos mecanismos de confirmação de recebimento, de números de identificação e de *checksum*. No entanto, o TCP não pode se recuperar de todas as possíveis falhas que afetam a comunicação entre processos remotos: as conexões entre processos podem ser quebradas e os processos podem também parar de responder. Para testar uma aplicação de rede baseada em TCP, esses possíveis comportamentos devem ser emulados.

Existem diversos injetores de falhas que atuam em programas que usam esse protocolo; no entanto, apresentam decisões de projeto que tornam sua utilização difícil ou pouco flexível. Como a própria implementação do protocolo TCP mascara boa parte das falhas introduzidas nas camadas mais baixas da pilha de rede, como omissão ou corrupção de pacotes, o uso de injetores que atuem nesse nível é dificultado no teste de aplicações de mais alto nível. Por outro lado, injetores especializados em protocolos de aplicação especiais, como HTTP ou SOAP, trocam o incremento na especialização por uma perda de flexibilidade.

Para preencher o nicho existente entre as duas categorias de injetores de falhas para comunicação TCP/IP, foi desenvolvida a ferramenta FIERCE. FIERCE intercepta as chamadas de comunicação TCP/IP durante a execução de uma aplicação Java, injetando as falhas configuradas para o experimento. A ferramenta emula os erros que o protocolo TCP/IP não mascara, possibilitando o teste da aplicação através da verificação da correta manipulação das situações excepcionais que podem levar a defeitos. A injeção desses erros, de difícil reprodução sem a ferramenta, aumenta o número de possíveis casos de teste a serem realizados com o sistema alvo.

FIERCE é construído sobre uma infra-estrutura criada para a implementação de FIONA, um injetor de falhas para aplicações Java que realizam comunicação usando o protocolo UDP/IP [Gerchman et al. 2005]. FIONA realiza a instrumentação das classes de comunicação usando a biblioteca de depuração e monitoramento JVMTI, possibilitando uma injeção de falhas de forma transparente para a aplicação. Sua estrutura foi usada para a construção de FIERCE, possibilitando o reuso de código já estável e funcional.

A Seção 2 apresenta as características do método de injeção de falhas para teste de aplicações, algumas ferramentas existentes e a motivação para a criação de FIERCE. A Seção 3 descreve a ferramenta, sua arquitetura e o modelo de falhas usado. Um exemplo de uso é mostrado na Seção 4. Considerações finais são encontradas na Seção 5.

2. Ferramentas de injeção de falhas

Qualquer que seja o mecanismo de tolerância a falhas implementado em um sistema, é necessária sua validação em um ambiente controlado de testes antes que este entre em produção. Adiar a fase de testes dos mecanismos de tolerância a falhas para um período após a entrada do sistema em produção pode ser desastroso: não há garantia que este funcione de uma maneira correta. Em sistemas onde os requisitos de tolerância a falhas tenham uma maior importância, a fase de validação é essencial.

Injeção de falhas é um método de validação experimental de software que apresenta eficiência e baixo custo. Ao contrário de métodos analíticos, requer um protótipo funcional do sistema sob testes, que terá seu comportamento avaliado em um ambiente onde falhas são artificialmente introduzidas. A injeção de falhas pode ser usada para vários objetivos [Hsueh et al. 1997]: identificar gargalos de dependabilidade, analisar o comportamento do sistema na presença de falhas, determinar a cobertura dos mecanismos de detecção e recuperação de erros e avaliar a eficácia dos mecanismos de tolerância a falhas, inclusive a perda de desempenho introduzida por eles.

A injeção de falhas pode ser realizada por hardware ou por software. A primeira categoria usa equipamentos adicionais, como ponteiras ativas, para introduzir falhas diretamente no hardware do sistema alvo. A injeção por software se aproveita dos ganchos (*hooks*) providos pelo sistema operacional, bibliotecas e processadores usados. O injetor usa esses ganchos para forçar um comportamento incorreto do sistema. Os ganchos podem ser interfaces de depuração e monitoramento que permitam alterar o estado dos programas em execução. A injeção por software apresenta um grande número de vantagens como baixo custo (por não ter necessidade de aquisição de equipamentos externos), nível mais alto de abstração (aumentando a flexibilidade do método) e simplicidade de implementação.

O foco deste trabalho é a injeção de falhas de comunicação, e o método usado para tanto é a atuação nos subsistemas de troca de mensagens pela rede. A troca de mensagens por uma rede de computadores é realizada através de primitivas `send` e `receive`, oferecidas pelo módulo de rede presente no sistema (normalmente, parte do sistema operacional). A ferramenta de injeção instrumenta essas chamadas, monitorando por uma mensagem enviada ou recebida que seja de interesse para o teste em andamento. Quando uma mensagem de interesse for capturada, uma ação é realizada sobre ela, como seu descarte ou alteração, ou sobre algum elemento externo, como um contador ou temporizador.

Diversas ferramentas para injeção de falhas de comunicação foram desenvolvidas, cada uma com um objetivo e uma estratégia de implementação diferente. No entanto, nenhuma se encaixa de maneira satisfatória no objetivo do teste de aplicações Java que usem comunicação por TCP.

ORCHESTRA [Dawson et al. 1997] é uma ferramenta para teste de implementação de protocolos de comunicação. É focada no teste de módulos do sistema operacional, e não de aplicações e sistemas distribuídos. A ferramenta adiciona uma camada de injeção de falhas no kernel do sistema, necessitando módulos diferentes para cada protocolo sob teste. ORCHESTRA foi inicialmente desenvolvida para o *microkernel* Mach e posteriormente portada para os sistemas operacionais Linux e Solaris. No entanto, o desenvolvimento dessa ferramenta foi descontinuado e tornou-se obsoleta frente às novas

versões destes.

ComFIRM [Drebes et al. 2006] é uma ferramenta cujos objetivos são pequena intrusividade e alta flexibilidade. Em sua última versão, é compilada como um módulo para o *kernel* Linux, permitindo uma fácil ativação e desativação. ComFIRM atua nas camadas mais baixas do subsistema de troca de mensagens de rede, podendo tomar atitudes dependendo do conteúdo da mensagem, de qualquer cabeçalho associado a ela ou de agentes próprios do injetor, como temporizadores e contadores. É permitida a modificação, descarte e atraso das mensagens, ações governadas por regras alimentadas através de um formato próprio.

WS-FIT [Looker et al. 2004] é um injetor de falhas de comunicação para o teste de chamadas de *web services* que usem o protocolo SOAP (*Simple Object Access Protocol*). A ferramenta oferece uma biblioteca SOAP instrumentada que permite alterar as mensagens trocadas entre os provedores e clientes de web services, contornando as dificuldades com criptografia que injetores de nível mais baixo, como ORCHESTRA, enfrentariam.

FIONA [Gerchman et al. 2005] é uma ferramenta de injeção de falhas voltada à validação de aplicações distribuídas desenvolvidas usando a plataforma Java. Para atingir essa meta, as classes de comunicação usadas pela JVM (*Java Virtual Machine*) são instrumentadas, de forma a emular a ocorrência de erros. No entanto, a ferramenta injeta falhas apenas em aplicações que usem o protocolo UDP (*User Datagram Protocol*).

Ferramentas que atuam em níveis mais baixos das camadas de rede, como ORCHESTRA, distanciam-se do teste de aplicações ao focar no teste da implementação de protocolos no sistema operacional. A abordagem de manipulação de mensagens oferecida por ComFIRM pode dificultar o teste de aplicações, pois é necessário um conhecimento profundo de todos os protocolos envolvidos na comunicação para a montagem correta das regras. Ferramentas como WS-FIT têm o foco em protocolos ou aplicações específicas, trocando a flexibilidade pela especialização. FIONA injeta falhas apenas em comunicação via UDP, deixando de lado aplicações que usam o protocolo TCP — o tipo de aplicação que é foco deste trabalho.

O injetor de falhas FIERCE busca uma solução flexível que se encaixe no objetivo do teste de aplicações Java. A ferramenta FIONA é a que mais se aproxima deste objetivo. Assim, foi decidido usar a infra-estrutura de instrumentação e alteração de classes oferecida por ela de forma a construir um injetor que atuasse também em comunicação pelo protocolo TCP.

3. A ferramenta FIERCE

A ferramenta de injeção de falhas FIERCE (*Fault Injection Environment for Remote Communication Evaluation*) tem como objetivo o teste de aplicações de rede Java que usem comunicação via TCP. FIERCE injeta falhas através da instrumentação das classes de comunicação usadas pela plataforma Java.

Vários desafios são apresentados no desenvolvimento de um injetor para aplicações que usam TCP: além da programação em si, um modelo de falhas que represente a realidade dos cenários encontrados deve ser estabelecido.

3.1. Modelo de falhas

Segundo Schneider [Schneider 1993], um modelo de falhas é uma coleção de atributos e um conjunto de regras que governam a interação entre componentes que falharam. A adoção de um modelo de falhas permite planejar os mecanismos de tolerância, prevendo as maneiras como os erros se manifestam dentro do sistema e projetando-os para que os cenários previstos sejam cobertos.

Os modelos clássicos de falhas para sistemas distribuídos, como os encontrados em [Cristian 1991] e [Birman 1996], consideram mensagens individuais passadas entre os hosts. Esses modelos não supõem a existência de mecanismos de tolerância a falhas como os oferecidos por TCP. Eles são usados, por exemplo, na construção de uma implementação deste protocolo no subsistema de rede de um sistema operacional, no seu teste e sua validação. Injetores de falhas como ORCHESTRA usam esses modelos porque são voltados à validação dos protocolos implementados no sistema operacional e não na validação das aplicações que os usam.

Esses modelos clássicos de falhas não são apropriados à proposta de FIERCE. Parte das falhas consideradas podem se traduzir em erros gerados pelo protocolo TCP: por exemplo, a interrupção da comunicação entre dois hosts acabará quebrando a conexão e gerando notificações dessa condição para os dois processos envolvidos. Porém, algumas variáveis complicam a geração de planos de teste: neste mesmo exemplo, o tempo entre a interrupção da comunicação e a notificação para os processos é dependente de implementação, variando entre os sistemas operacionais. Além disso, existem falhas que não são consideradas nos modelos clássicos. Se um dos processos envolvidos na conexão for encerrado mas o sistema operacional (e seu módulo TCP correspondente) continuarem funcionais, mensagens de término de conexão serão trocadas entre os hosts. Este cenário ocorre quando há falhas no processo e não na rede de comunicação usada e por isso não fazem parte dos modelos clássicos. No entanto, é possível em uma conexão TCP e deve ser levado em conta em seus testes.

Neves e Fuchs [Neves and Fuchs 1997], ao desenvolver detectores de falhas, propuseram um modelo que melhor se encaixa na proposta de FIERCE. Sua proposta era usar apenas os códigos de erro retornados pela biblioteca de sockets do sistema operacional e o conhecimento prévio dos estados que os módulos TCP podem assumir para desenvolver módulos de detecção de falhas de comunicação para aplicações distribuídas tolerantes à falhas. Os autores consideraram quatro tipos de falhas que resultam no término de um processo mas com diferentes comportamentos observados na interface de sockets: término (*kill*), colapso (*crash*), reinicialização (*reboot*) e colapso com reinicialização (*crash and boot*).

Término de processo A falha faz com que a execução do processo comunicante se encerre sem afetar o funcionamento do resto do sistema. Exemplos dessa falha são uma execução de instrução ilegal ou uma falha de segmentação (*segfault*).

Colapso A máquina onde o processo comunicante estava executando colapsa (ocorre um *crash*), deixando de executar qualquer operação. Exemplos dessa falha são falhas permanentes em componentes importantes da máquina ou seu colapso (como um *kernel panic*) sem sua subseqüente reinicialização por um longo período de tempo.

Reinicialização A máquina onde o processo estava executando realiza uma rotina de reinicialização (*reboot*), onde o sistema operacional fecha todas as conexões e

desliga propriamente a máquina, que é religada em seguida. Uma máquina deve ser reinicializada por diversos motivos como, por exemplo, a atualização de um sistema chave (como o *kernel*).

Colapso com reinicialização A máquina colapsa e é reinicializada. Exemplos de falhas são as mesmas da falha de colapso; a diferença é a reinicialização da máquina em um curto tempo após congelar.

O desenvolvimento da ferramenta FIERCE foi baseado neste modelo de falhas.

3.2. Processo de falhas

Para o desenvolvimento de um injetor, é necessário compreender qual o comportamento de um sistema quando uma falha ocorre para que possa ser emulado pela ferramenta. Para isso, o código-fonte das classes de comunicação da plataforma Java foi analisado de forma a documentar as interações entre os diferentes componentes, os efeitos das falhas nestes, os códigos de erro retornados e as exceções lançadas. Essa análise foi facilitada com a introdução da licença JRL (*Java Research License*) pela Sun, que permite não só o compartilhamento do código-fonte como também sua modificação para fins de pesquisa. Além das classes Java, estão cobertas por essa licença as funções nativas, que realizam a interface entre a aplicação em execução na máquina virtual com o sistema operacional.

Através da análise dos diversos componentes envolvidos na comunicação TCP (classes Java, funções nativas e chamadas de sistema), foi traçado o comportamento do sistema no caso de cada uma das falhas descritas no modelo. O resultado deste serviu para a programação do injetor, buscando emular esse comportamento de forma igual à realidade.

Encerramento de processo

No protocolo TCP, as duas direções de envio de mensagens (a de envio, "*send-half*", e a de recepção, "*receive-half*") são consideradas independentes e são encerradas em momentos diferentes. Cada direção é conhecida também como uma "metade" da conexão.

Para sinalizar o encerramento da metade de envio de mensagens, o módulo TCP local envia para o correspondente remoto uma mensagem com o bit `FIN` ligado. O envio desta mensagem avisa que nenhuma mensagem adicional será enviada a partir daquele instante. O módulo remoto, ao receber a mensagem `FIN`, encerra sua metade de recepção e confirma a operação com uma mensagem de *acknowledge* (`ACK`). Caso o módulo remoto também decida encerrar a sua metade de envio, o mesmo procedimento é realizado.

Quando um processo é encerrado, o sistema operacional solicita ao módulo TCP o encerramento de todas as conexões associadas a essa instância. Mensagens `FIN`, de encerramento, são enviadas para os módulos TCP remotos, e as portas usadas pelo processo encerrado são liberadas. Caso uma mensagem do TCP remoto chegue no TCP local, endereçada a essa conexão já encerrada, uma mensagem com o bit `RST` (*reset*) será enviada de volta. A mensagem *reset* indica que o endereço destino não deveria receber essa mensagem; o TCP remoto, ao receber essa mensagem, encerra completamente a conexão.

A seqüência de mensagens trocadas pelos módulos TCP gera diferentes códigos de erro e sinais para as aplicações envolvidas. Esses erros são passados para o ambiente

Java de diferentes formas. Se o host remoto sinaliza o encerramento a conexão com o envio de mensagem `FIN`, ela é levada para o estado "close_wait". Neste estado, leituras associadas ao socket irão retornar os dados que já se encontram na fila de recebimento, enviados pelo processo remoto antes da falha ocorrer. Quando a fila esvaziar, um valor especial é retornado, indicando final dos dados. No caso de Java, chamadas ao método `read` da classe `SocketInputStream`, responsável pelo recebimento de dados, retornarão o valor `-1`.

Uma tentativa de envio de dados pelo socket causará o recebimento de uma mensagem com o bit `RST`, indicando que o processo remoto teve seu socket fechado. Os dados enviados na primeira tentativa são perdidos; as demais tentativas provocarão o envio de um sinal "*Broken pipe*" pelo sistema operacional. No caso de Java, chamadas ao método `write` da classe `SocketOutputStream`, responsável pelo envio de dados, causarão o lançamento de uma exceção `SocketException` contendo a mensagem "*Broken pipe*".

Colapso

Quando uma falha de colapso ocorre, nenhuma mensagem é trocada entre o módulo TCP local e o módulo TCP que estava em uso na máquina onde esta ocorreu: nenhum aviso é enviado para as máquinas que tinham conexões estabelecidas com esta, ao contrário do encerramento de processo, onde são trocadas mensagens `FIN` e `RST`.

Enquanto não for realizada uma tentativa de envio de dados para o host remoto, não existe o conhecimento da falha de colapso. Assim, operações de leitura associadas ao socket retornarão os dados que estiverem na fila de recebimento. Quando esta esvaziar, a *thread* pode ser bloqueada indefinidamente, esperando por dados que nunca serão enviados.

A falha de colapso é detectada quando o host tenta um envio de dados. O módulo TCP local irá esperar uma mensagem de confirmação (uma mensagem com o bit `ACK` ligado) por um certo tempo; este tempo é dependente de implementação. Se nenhuma confirmação for recebida após esse tempo, é suposto que houve falha de colapso e a conexão é encerrada. Após o *timeout* da conexão, as operações de leitura retornarão o código de erro "*Connection timed out*" e as de escrita farão com que o sistema operacional lance o sinal "*Broken pipe*".

No caso de Java, após a *timeout*, a primeira chamada à operação `read` causará o lançamento de uma exceção `SocketException` com a mensagem "*Connection timed out*"; as demais retornarão o valor especial `-1`, indicando final de conexão (ou seja, não há mais dados a receber). As chamadas à operação `write` causarão o lançamento de uma exceção `SocketException` com a mensagem "*Broken pipe*".

Reinicialização, colapso com reinicialização

A falha de reinicialização ocorre quando um sistema é intencionalmente reinicializado pelo seu operador. A reinicialização de um sistema é prática comum quando este apresenta problemas; pode ser considerada uma combinação das duas falhas apresentadas anteriormente. É possível dividir a reinicialização em três fases distintas: encerramento

de todos os processos da máquina, desligamento do sistema e reinicialização do sistema operacional.

Na primeira fase, o sistema operacional encerra todos os processos que estejam em execução; o comportamento dos processos é o mesmo percebido em uma falha de encerramento de processo. Na segunda fase, o sistema foi desligado e está sendo religado: não está em execução ainda qualquer código de comunicação pela rede, e nenhum processo comunica-se pela rede. Esta fase tem o mesmo comportamento de uma falha de colapso. Numa terceira fase, a máquina já terminou seu ciclo de reinicialização, com os *drivers* de comunicação e módulo TCP carregados em memória; no entanto, a aplicação não foi reinicializada. As máquinas que tentam se comunicar percebem o comportamento de uma falha de encerramento de processo: o sistema operacional está no ar, mas a aplicação, não.

A falha de colapso com reinicialização ocorre quando a máquina pára de responder mas é reinicializada em um curto período de tempo. Ocorre quando um operador da máquina percebe seu colapso e, em um tempo hábil, a reinicializa. O processo desta falha é similar ao da falha de reinicialização, com exceção do número de fases: a primeira fase, onde o sistema operacional encerra todos os processos, não existe nesse caso.

3.3. Arquitetura da ferramenta

A idéia central de FIERCE é a instrumentação das classes de comunicação da JRE (*Java Runtime Environment*) para que falhas descritas em um cenário de falhas sejam emuladas. Para isso, a mesma infra-estrutura criada para o injetor FIONA é usada: as classes originais são substituídas por versões próprias da ferramenta no momento de sua carga, de forma transparente para a aplicação em execução. A arquitetura de FIONA é descrita com maiores detalhes em [Gerchman et al. 2005].

Para realizar essa substituição é usado JVMTI (*Java Virtual Machine Tool Interface*), uma interface nativa de depuração e monitoramento da máquina virtual Java e de aplicações que executem sobre ela. A idéia central de JVMTI é a criação de um componente nativo de software, o *agente*, compilado como uma biblioteca. Esse agente é notificado da ocorrência de eventos de seu interesse, que são registrados em sua inicialização. O principal evento de interesse, no caso de FIERCE, é *Class File Load Hook*, que permite alterar ou substituir completamente um arquivo de classe no momento em que é carregado do disco.

Quatro classes de comunicação da JRE são instrumentadas por FIERCE. As classes `Socket` e `ServerSocket` são responsáveis pelo estabelecimento de uma conexão TCP entre dois processos. O envio e o recebimento de dados são gerenciados pelas classes `SocketInputStream` e `SocketOutputStream`. A instrumentação existente nessas classes injeta as falhas descritas em arquivos de configuração, usando a mesma infra-estrutura e classes de apoio providas por `FIONA`.

Nas duas primeiras classes, os métodos alterados são `connect` e `accept`. O primeiro, da classe `Socket`, é responsável pela abertura ativa de uma conexão. O segundo método, da classe `ServerSocket`, bloqueia a thread esperando por uma requisição de abertura de conexão enviada por outro host. Nas classes `stream`, os métodos alterados são `read` e `socketWrite`, responsáveis pelo recebimento e envio de dados. Os quatro métodos descritos são os únicos pontos de entrada para os métodos nativos que efetivamente chamam as primitivas do sistema operacional.

A instrumentação existente nessas quatro classes de comunicação atua de acordo com uma configuração armazenada na classe auxiliar `BancoFalhas`, pertencente ao injetor. `BancoFalhas` é a classe responsável pela leitura do arquivo de configuração que descreve o cenário de falhas a ser injetado e pelo armazenamento dessa descrição durante o experimento. O arquivo de configuração do cenário de falhas é especificado no disparo da máquina virtual Java através de um parâmetro passado para o agente JVMTI.

4. Usando FIERCE para o teste de uma aplicação baseada em *web services*

Para ilustrar o uso da ferramenta FIERCE foi injetada uma falha de encerramento de processo em uma cliente de *web service*. Em um servidor web Apache Tomcat versão 5.5.12 foi instalada o web service WidgetPrice, parte do conjunto de exemplos oferecidos com a biblioteca Apache Axis 1.2.1. A biblioteca Axis é uma implementação de SOAP para Java, compreendendo utilitários para construção, gerência e invocação de web services.

O web service de exemplo WidgetPrice mantém uma lista de itens que fariam parte de um estoque de uma empresa e seus respectivos preços. Ele oferece dois métodos: `setWidgetPrice`, que registra um novo item e seu respectivo preço ou altera um item já existente, e `listWidgetTypes`, que recupera a lista completa dos itens e seus preços.

O alvo do teste é um pequeno aplicativo cliente que se conecta ao servidor e invoca o web service. Dois botões são oferecidos; cada um deles invoca um dos métodos. O primeiro, "*Set Widget Price*", permite ao usuário cadastrar um item e seu preço. O segundo, "*List Widget Prices*", lista os preços na área de mensagens da janela. A Figura 1 mostra duas capturas de tela, durante uma operação de cadastro e após a listagem dos preços. As capturas desta figura foram realizadas sem nenhuma falha injetada.

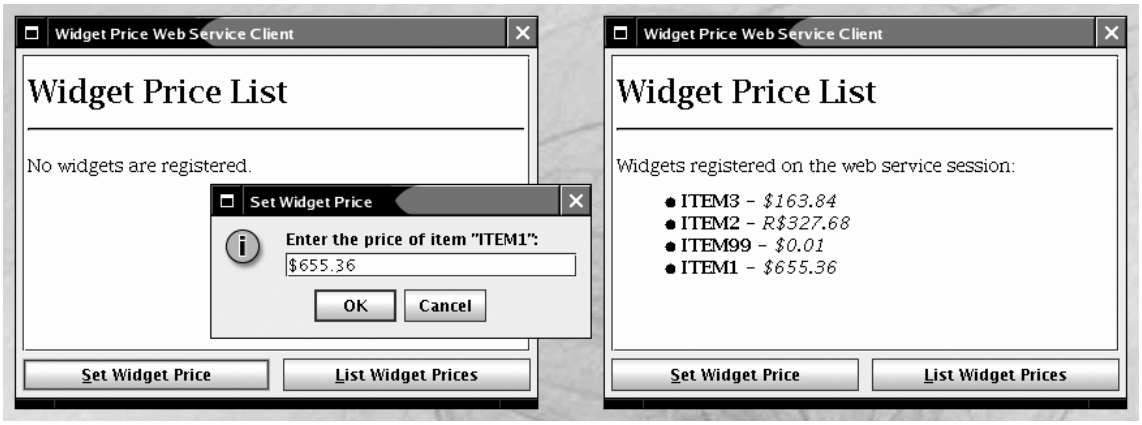

Figura 1. Exemplo de operação normal do cliente de web service

Para o teste do aplicativo, FIERCE foi configurado para injetar uma falha de encerramento de processo após 60 segundos a partir do início do experimento. Durante o teste, o injetor emularia uma situação na qual o processo do servidor web, onde o web service se localiza, seria encerrado. Nessa situação a máquina remota continuaria em funcionamento, mas sem nenhum serviço ligado à porta 80. Qualquer tentativa de conexão a essa porta retornaria uma mensagem `RST`, indicando sua recusa.

O aplicativo, então, foi iniciado com o injetor de falhas ativado. O arquivo de configuração usado é mostrado na Figura 2 e contém a descrição de uma falha de encerramento de processo. A primeira parte da descrição especifica o tipo de falha a ser injetado; no caso, `TcpKillFault`. A segunda e terceira partes especificam as condições de início da falha: neste exemplo, ela será ativada por tempo (tipo "`t`"), após 60000 milissegundos a partir do início da execução. É possível também ativar uma após um certo número de bytes trafegados na conexão, especificando o tipo "`b`". As últimas duas partes especificam qual host e qual porta sofrerão a emulação da falha.

```
TcpKillFault:t:60000:localhost:80
```

Figura 2. Exemplo de configuração para falha de encerramento de processo

O resultado da injeção de uma falha de encerramento de processo pode ser conferido na Figura 3. Antes da falha ser ativada, alguns itens foram registrados no servidor. Passados 60 segundos do início da execução da aplicação, FIERCE ativa a falha configurada. O cliente não consegue mais se conectar ao servidor web; o host remoto responderia com uma recusa de conexão. Qualquer tentativa de conexão com o método `connect` da classe `Socket` causa uma exceção `ConnectException` com a mensagem "*Connection refused*".

Figura 3. Exemplo de falha de encerramento de processo

O teste mostrou que a aplicação apenas apresenta ao usuário uma mensagem técnica de erro se o processo do servidor contatado não está disponível. Um engenheiro de teste deveria verificar se esse comportamento é esperado; caso contrário, essa condição deveria ser reportada para melhoramento do sistema.

O experimentou mostrou ser possível testar a aplicação em uma situação difícil de atingir em um ambiente comum. Para realizar o mesmo teste sem o uso de FIERCE, o servidor no qual o web service é oferecido deveria ser encerrado. Este procedimento pode não ser possível, como no caso do servidor ser compartilhado por mais de um usuário ou engenheiro de teste.

5. Considerações Finais

Aplicações de rede devem prever a ocorrência de falhas de comunicação durante sua execução; os mecanismos usados para que essas falhas sejam toleradas devem ser testados antes do sistema entrar em produção. O desenvolvimento de aplicações que necessitem de um alto nível de dependabilidade torna ainda mais importante essa validação. Injeção de falhas é uma técnica de validação experimental de mecanismos de tolerância a falhas que se mostra flexível e de baixo custo.

Este trabalho apresenta FIERCE, um injetor de falhas de comunicação para teste de aplicações Java que usam o protocolo TCP. Para o desenvolvimento dessa ferramenta foi necessária a análise das classes de comunicação Java para que seu comportamento em um ambiente com falhas fosse determinado. O injetor emula esse comportamento durante a execução da aplicação, possibilitando seu teste nessas condições.

A implementação de FIERCE usa a infra-estrutura de FIONA, um injetor de falhas de comunicação em protocolo UDP para aplicações Java. A ferramenta realiza a instrumentação das classes de comunicação da JRE usando a interface de depuração e monitoramento da máquina virtual JVMTI (*Java Virtual Machine Tool Interface*). FIERCE é composto de um agente JVMTI, classes de comunicação instrumentadas e classes auxiliares para injeção de falhas. Sua ativação é completamente transparente para a aplicação sob testes.

O experimento mostrado na Seção 4 exemplifica o uso de FIERCE para o teste de aplicações. A ferramenta é flexível, possibilitando a criação de uma série de cenários de falhas.

Trabalhos futuros incluem a integração de serviços providos por FIONA, como monitoramento do experimento por *logging* e a gerência de um experimento distribuído em máquinas diferentes através da comunicação entre os injetores. Apesar disso, FIERCE já pode ser usado para o teste de aplicações que usem serviços construídos sobre o protocolo TCP, como servidores HTTP e *web services*.

Referências

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. E. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, pages 11–33.

Birman, K. P. (1996). *Building Secure and Reliable Network Applications*. Manning Publications, Co, Greenwich.

Cristian, F. (1991). Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78.

Dawson, S., Jahanian, F., and Mitton, T. (1997). Experiments on six commercial TCP implementations using a software fault injection tool. *Software – Practice and Experience*, 27(12):1385–1410.

Drebes, R. J., Jacques-Silva, G., da Trindade, J. M. F., and Weber, T. S. (2006). A kernel-based communication fault injector for dependability testing of distributed systems. *Lecture Notes in Computer Science*, 3875:177–190.

Feldman, S. (2005). Quality assurance: Much more than testing. *ACM Queue*, 3(1):26–29.

Gerchman, J., Jacques-Silva, G., Drebes, R. J., and Weber, T. S. (2005). Ambiente distribuído de injeção de falhas de comunicação para teste de aplicações Java de rede. In *Anais do 19º. Simpósio Brasileiro de Engenharia de Software*, volume 1, pages 232–246, Uberlândia, MG. Sociedade Brasileira de Computação.

Hsueh, M.-C., Tsai, T. K., and Iyer, R. K. (1997). Fault injection techniques and tools. *IEEE Computer*, 30(4):75–82.

Looker, N., Munro, M., and Xu, J. (2004). WS-FIT: A tool for dependability analysis of web services. In *28th Annual International Computer Software and Applications Conference (COMPSAC'04)*, pages 120–123.

Neves, N. and Fuchs, W. K. (1997). Fault detection using hints from the socket layer. In *Proc. of the Symposium on Reliable Distributed Systems*, pages 64–71.

Schneider, F. B. (1993). What good are models and what models are good? In Mullender, S., editor, *Distributed Systems*, pages 17–26. Addison-Wesley, Workingham, 2nd edition.