Injeção de falhas em protocolos tolerantes a falhas utilizando a arquitetura Ferry clip

Marcos Renato Rodrigues Araujo
maraujo@dcc.unicamp.br

Eliane Martins
eliane@dcc.unicamp.br

Instituto de Computação – Universidade Estadual de Campinas
Caixa Postal 6065 – 13081.970 Campinas/SP

Resumo

Na validação de protocolos tolerantes a falhas são necessários testes que validem seus mecanismos de tolerância a falhas. Para isto um dos métodos utilizados é a injeção de falhas via software, dada sua simplicidade e baixo custo de desenvolvimento. Estes testes são chamados de testes de tolerância à falhas neste contexto.

Este artigo descreve uma ferramenta que está sendo desenvolvida para permitir realizar testes de tolerância à falhas de implementações de protocolos de comunicação. Esta ferramenta utiliza uma variação da arquitetura Ferry clip, modificada com o propósito de permitir a realização de testes de tolerância à falhas.

Abstract

For the validation of fault-tolerant protocols, the validation of their fault-tolerance mechanisms is needed. One method used with this aim is software implemented fault injection, since it is simple to use and has low development cost. In our context, this kind of testing is called fault-tolerance testing.

This paper describes a tool that is being developed to support fault-tolerance tests for communications protocol implementations. This tool extends the Ferry clip architecture to support fault-tolerance testing.

1 Introdução

Durante o desenvolvimento de um sistema podemos enumerar três estágios: projeto, implementação e validação. A fase de *projeto* consiste em estruturar o protocolo através dos requisitos, onde estão descritas todas as regras que o mesmo deve obedecer. A *implementação* consiste em traduzir o projeto elaborado na fase de projeto em termos computacionais, adaptando-se o mesmo às condições do ambiente de desenvolvimento. A *validação* consiste em assegurar que a implementação do protocolo esteja de acordo com a especificação do mesmo. A arquitetura *Ferry clip* [CLP+89] foi desenvolvida com a finalidade de dar suporte à validação de protocolos.

No desenvolvimento de um protocolo Tolerante a Falhas, deseja-se validar também os mecanismos de tolerância a falhas desenvolvidos. Uma das maneiras de se validar estes mecanismos é através da injeção de falhas, que consiste em inserir de maneira controlada falhas durante a execução do protocolo, observando o comportamento do mesmo quando submetido a elas. Injetores de falhas por software emulam falhas de hardware sem a necessidade de se desenvolver um hardware adicional, com conseqüente redução nos custos de desenvolvimento.

Este resumo descreve brevemente uma ferramenta atualmente em desenvolvimento que acrescenta à arquitetura Ferry clip um injetor de falhas via software. Esta arquitetura foi escolhida devido à sua portabilidade: sua estrutura altamente modular permite que ela possa ser usada em diferentes plataformas requerendo para isso poucas modificações.

2 O que é *Ferry clip*?

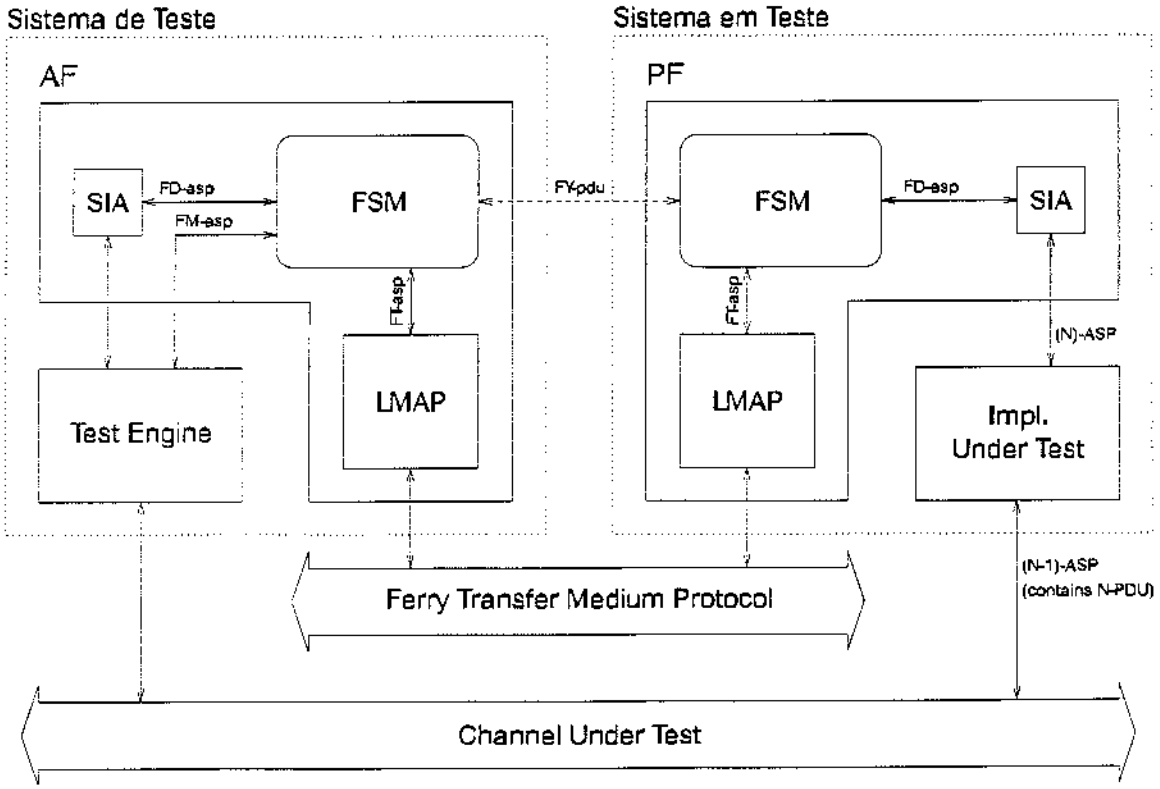

Figura 1 – Arquitetura Ferry clip

O conceito de Ferry clip [CLP+89] foi desenvolvido com o objetivo de dar suporte aos métodos de teste definidos pela ISO [ISO]. Basicamente consiste em transportar os dados de teste de maneira transparente do Sistema de Teste (responsável pela coordenação dos testes) ao Sistema em Teste (responsável pela execução dos mesmos), permitindo que ambos os testadores residam

junto ao Sistema de Teste, simplificando a sincronização entre UT e LT, minimizando assim a quantidade de software que reside junto ao Sistema em Teste.

Um sistema Ferry clip (figura 1 [CVD92]) consiste de dois componentes: o Active Ferry clip (AF), residente no Sistema de Teste, e o Passive Ferry clip (PF), residente no Sistema em Teste. Ambos os Ferry's trocam dados através de um protocolo de comunicação denominado Ferry Control Protocol (FCP), que utiliza os serviços de um canal de comunicação confiável denominado Ferry Transfer Medium Protocol (FTMP).

O Test Engine agrupa as funções do Testador Superior e o Testador Inferior, sendo também responsável pelo envio das instruções à FSM do AF tal como estabelecimento de conexão com PF, desconexão e outros dados de controle. As informações circulam entre AF e PF através de ASP's (Abstract Service Primitives): FD-asp para transporte de dados entre Test Engine e IUT, FM-asp para envio de comandos às FSM's e FT-asp, para o transporte entre AF e PF. Ambas as FSM's trocam dados através de FY-PDU's (Ferry Protocol Data Units), encapsuladas em FT-asp's. Cada Ferry possui três sub-módulos: FSM (*Finite State Machine*), que reúne as funções de gerenciamento interno de cada Ferry; SIA (*Service Interface Adapter*), encarregado de interagir com a IUT e transportar através do Ferry, de maneira transparente, sua interface ao Test Engine; e LMAP (*Lower Mapping Module*), responsável pelo transporte de dados entre os Ferry's. É esta modularidade que facilita a portabilidade entre sistemas distintos

3 Injeção de falhas

A injeção de falhas é um método de validação de protocolos que possuem métodos de tolerância e recuperação de falhas. Durante a validação são introduzidas falhas no sistema com o objetivo de estimular estes mecanismos. Os principais objetivos são [SW97]:

- Validação dos procedimentos de verificação;
- Validação dos mecanismos de tolerância a falhas, permitindo:
  1. previsão de falhas;
  2. eliminação de falhas;

A injeção de falhas pode ser feita basicamente de três maneiras: *injeção física* (via hardware), *simulação lógica* e *injeção lógica* (via software).

A *injeção física* ou via hardware consiste em injetar falhas diretamente no hardware da implementação, tais como sinais elétricos ou distúrbios eletromagnéticos. Estas técnicas em geral possuem alto custo de implementação devido à necessidade de desenvolvimento de hardware próprio com esta finalidade, além de eventualmente causar danos físicos à implementação.

A *simulação lógica* consiste em desenvolver uma representação da IUT em software e estudar seu comportamento. Esta técnica está associada a um alto custo de desenvolvimento devido à necessidade de se representar toda a implementação em software, algo nem sempre possível, além de não haver garantias de que a simulação corresponderá a uma implementação real.

A *injeção lógica* ou *injeção via software* possui duas características básicas: tem baixo custo de implementação e não sujeita a IUT a danos físicos, e consiste em um meio-termo entre injeção via hardware e simulação de falhas: emula falhas de hardware no software da própria implementação, de modo que a IUT detecte a simulação como uma falha; deste modo não há necessidade de um hardware dedicado à injeção de falhas, nem necessidade de uma representação da IUT em software. É um método muito mais simples de ser implementado e a um custo muito mais baixo.

Sua principal desvantagem é a limitação para representar determinados tipos de falhas (controle da CPU por exemplo) além de afetar as características de temporização do sistema, limitando seu uso em sistemas de tempo real. Tais desvantagens podem ser minimizadas através da redução da quantidade de software associado à injeção de falhas. A arquitetura Ferry clip foi escolhida aqui com o objetivo de minimizar o software associado à implementação em teste.

4 A Ferramenta

Nossa ferramenta pretende dar suporte ao teste de protocolos visando um alto grau de portabilidade e minimização do Sistema em Teste através da utilização da arquitetura Ferry clip, dando suporte a testes de protocolos tolerantes à falhas, com ou sem injeção de falhas. Neste caso o requisito mínimo de nossa ferramenta é ter acesso às interfaces superior e inferior da IUT.

4.1 Características

A arquitetura Ferry clip permite um alto grau de minimização do código residente junto à implementação e um alto grau de portabilidade para a ferramenta, uma vez que para uma diferente plataforma apenas alguns módulos devem ser adaptados. No caso de testes realizados em IUT's distintas, apenas os módulos AF-SIA e PF-SIA necessitam modificação. No nosso esquema o injetor de falhas é colocado na camada abaixo da IUT.

Utiliza-se uma ferramenta para geração automática dos testes. Esta ferramenta está descrita em [SM98] e gera casos de teste determinísticos a serem usados pelo UT e o LT. A geração automática de falhas ainda não foi implementada.

4.2 Modelo esquemático

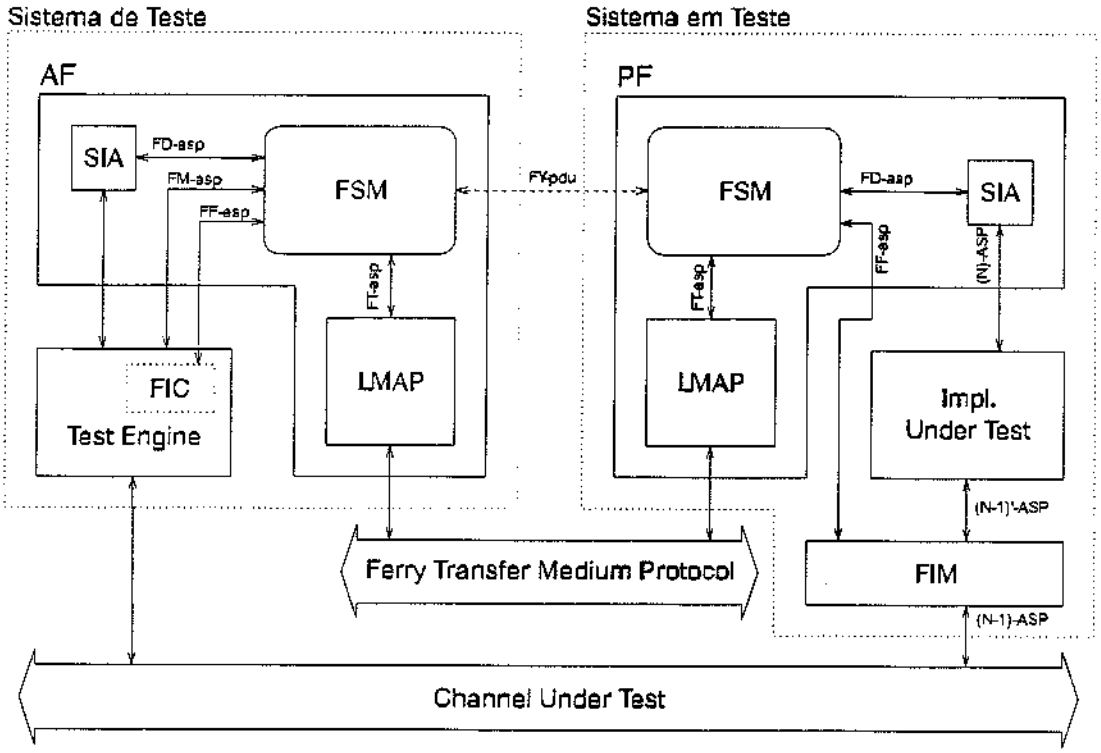

Figura 2 – Modelo esquemático da ferramenta

A Figura 2 mostra o modelo esquemático da ferramenta. Percebe-se pequenas alterações ao modelo original do Ferry clip. De fato, a idéia é ser o mais próximo desta arquitetura possível. A principal diferença esquemática está no acréscimo do FIC ao Test Engine e do FIM ao Sistema em Teste. O FIC (Fault Injection Controller) é responsável pelo gerenciamento remoto do FIM (Fault

Injection Module). O envio de dados ao FIM é feito através de FF-asp's, acrescidas ao modelo original do Ferry clip (ver Figura 1).

O FIM, como mencionado, é o módulo responsável pela interceptação e tratamento das mensagens que passam pela camada inferior da IUT. Em um caso típico, o FIC envia ao FIM as instruções sobre como ele deve proceder. A figura 3 ilustra o diagrama esquemático do FIM.

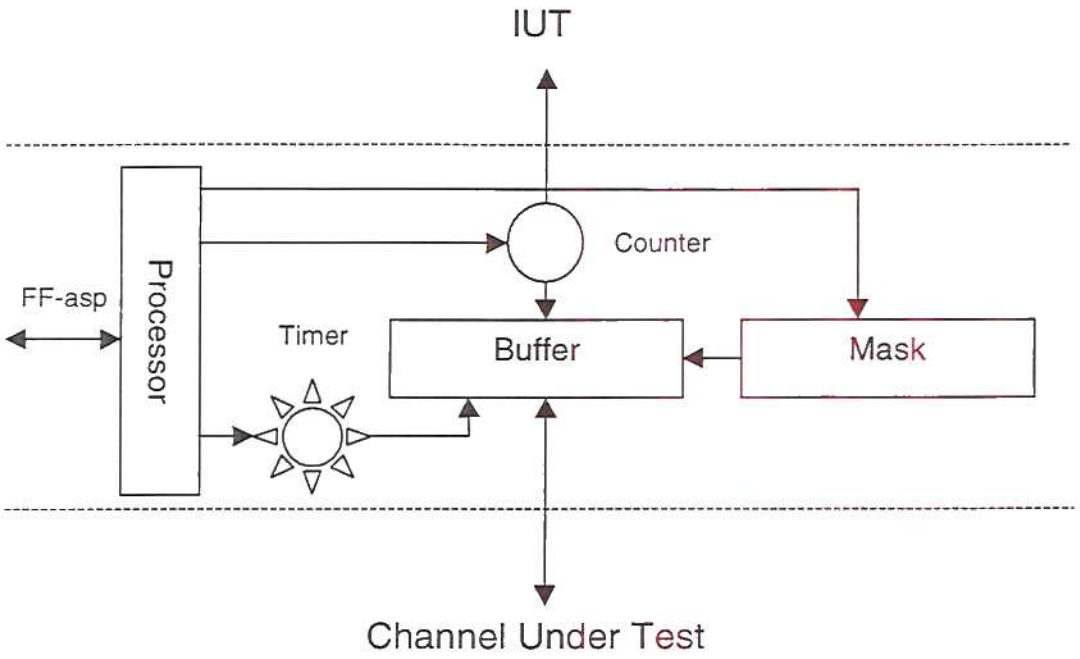

Figura 3 – Fault Injection Module

O FIM injeta falhas diretamente nas mensagens que circulam através dele, sendo responsável por três atividades: filtragem (interceptação de mensagens), tratamento (injeção da falha em si) e injeção (devolver a mensagem, alterada ou não, ao sistema). O FIM é capaz de gerar quatro tipos de falhas de comunicação:

1. *duplicação*: a mensagem é enviada várias vezes;
2. *supressão*: a mensagem é suprimida totalmente;
3. *corrupção*: a mensagem é corrompida de algum modo;
4. *temporização*: a mensagem é enviada com atraso;

O módulo *processor* é responsável pela recepção das instruções e pela sua execução sobre os outros módulos. O *buffer* armazena a última mensagem; *mask* armazena a máscara para a injeção das falhas e o *timer* armazena a informação para atraso da mensagem.

Para a duplicação de mensagens, o *buffer* armazena a mensagem e o *processor* informa quantas vezes a informação deve ser repetida antes de ser eliminada; o *timer* controla as falhas de temporização informando quanto o *buffer* deve atrasar antes de enviar o pacote (normalmente o *timer* fica em zero, enviando a mensagem imediatamente); o *mask* armazena a máscara de corrupção das mensagens. O *counter* é utilizado para contagem de mensagens que passam pelo FIM, necessário para saber qual será a mensagem na qual será aplicada a falha; neste esquema o

usuário da ferramenta informa que a mensagem na qual será injetada a falha será a n-ésima mensagem a passar pelo FIC.

Como a ferramenta pode ser utilizada para testes de conformidade seja com ou sem injeção de falhas, um esquema de script compatível deve ser considerado. Escolheu-se trabalhar com um único script responsável pela programação do FIM e pelos casos de teste. Para tanto o script é dividido em duas partes: a primeira prepara o FIM para a injeção de falhas, aguardando confirmação dele para cada instrução enviada; a segunda consiste do script com os casos de teste. Uma vez preparado, o injetor fica ativo até o fim dos casos de teste. O envio das instruções e a espera da confirmação é feita pelo FIC.

4.3 Implementação

A implementação da ferramenta está sendo feita com o objetivo inicial de se testar uma implementação do protocolo X.25 desenvolvido para plataforma PC. Para tanto foi escolhido o sistema operacional Windows95 para dar suporte ao Sistema em Teste. Visando testar a funcionalidade da ferramenta em plataformas distintas, o Sistema de Teste está sendo implementado em plataforma Solaris 2.5, ambas as partes desenvolvidas em C++ com Orientação a Objetos. Para troca de dados entre as partes no Sistema de Teste utiliza-se o esquema de pipes, sob a forma de sockets Unix. Este esquema permite que um único Test Engine "dispare" diversos processos AF simultaneamente e mantenha conexão com todos eles sem necessidade de modificações na ferramenta.

Atualmente estão concluídas em ambiente Solaris as implementações do AF e do PF. Para efeito de validação foram feitos dois testes básicos sobre estes componentes: interconexão e troca de dados. Para o teste de interconexão simplesmente colocou-se o AF e o PF executando em máquinas distintas e, através de uma interface simples, solicitou-se uma conexão e uma desconexão. Ambos os comandos foram aceitos sem problemas.

Para o teste de troca de dados desejava-se também verificar o desempenho do canal de comunicação. Para tanto, utilizou-se o mesmo esquema de AF e PF, este executando em modo *'loopback'* (retornar qualquer dado que chega pelo mesmo caminho), em máquinas distintas e uma interface simples com o AF cuja finalidade é solicitar abertura de conexão com um determinado PF, enviar uma determinada informação 10 mil vezes, comparar o dado recebido com o que foi enviado e desconectar. O tempo de execução deste esquema foi cronometrado dez vezes, obtendo-se dez medidas de tempo distintas.

As medidas obtidas foram (em segundos, em ordem crescente): 14.201s, 14.592s, 15.245s, 15.884s, 17.294s, 17.818s, 19.674s, 19.958s, 24.167s, 24.201s, dando uma média de 18.303s. Desprezando-se o tempo gasto na conexão e na desconexão, dividiu-se o tempo gasto pelo número de mensagens trocadas (10 mil) e obteve-se o tempo médio de ida e volta de uma mensagem cujo valor é 1.83ms, resultando em 0.92ms o tempo gasto médio para uma mensagem ser enviada de uma máquina a outra. O protocolo de comunicação utilizado foi o TCP/IP, garantindo-se assim a não ocorrência de mensagens perdidas. Ambas as máquinas localizam-se na mesma rede local.

5 Conclusão

Foi descrita aqui uma nova ferramenta que está sendo implementada para o teste de protocolos e seus mecanismos de tolerância à falhas. Para tanto utiliza-se o conceito de injeção de falhas por software para validação dos mecanismos de tolerância à falhas desenvolvidos para o protocolo, e o conceito de Ferry clip como arquitetura básica para desenvolvimento da ferramenta.

Atualmente a ferramenta está em fase de implementação, estando prontos o AF e PF, ambos implementados em Solaris, e uma versão beta do Test Engine, utilizada para testar a conexão entre AF e PF. O próximo passo é portar o PF para o ambiente Windows95, concluir a implementação do FIM e concluir o Test Engine.

Apêndice: Nomenclatura utilizada

AF: Active Ferry

ASP: Abstract Service Primitives

FIC. Fault Injector Controller

FIM: Fault Injection Module

FSM: Finite State Machine

FD-asp: Ferry Data Abstract Service Primitives

FM-asp: Ferry Management Abstract Service Primitives

FF-asp: Ferry Fault Abstract Service Primitives

FT-asp: Ferry Transport Abstract Service Primitives

IUT: Implementation Under Test

LMAP: Lower-Mapping Module

PF: Passive Ferry

SIA: Service Interface Adapter

Bibliografia

[CLP+89] Chanson, S. T., Lee, B. P., Parakh, N. J., Zeng, H. X., *"Design and Implementation of a Ferry clip Test System"*, Proc. 9th IFIP Symposium on Protocol Specification Testing & Verification, Enschede, Holanda (Junho 1989)

[CVD92] Chanson, S. T., Vuong, S., Dany, H., *"Multi-party and interoperability testing using the Ferry clip approach"*, Computer Communications, vol 15, nº 3 (Abril 1992)

[ISO] ISO TC97/SC21, DIS 9646, *"OSI Conformance Testing Methodology and Framework"*, ISO, Genebra, Suíca (Novembro 1989)

[SM98] Sabião, S. B., Martins, E., *"CONDADO – uma ferramenta para geração de testes de protocolos combinando Controle e Dados"*, submetido ao 16º Simpósio Brasileiro de Redes de Computadores (Janeiro 1998)

[SW97] Sotoma, I., Weber, T. S., *"AFIDS - Arquitetura para Injeção de Falhas em Sistemas Distribuídos"*, Anais do XV Simpósio Brasileiro de Redes de Computadores, São Carlos/SP (1997)