MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-9591-PRE/5219

TESTES DE TOLERÂNCIA A FALHAS EM SISTEMAS DE COMUNICAÇÃO

Eliane Martins*
Maria de Fátima Mattiello-Francisco
Ana Maria Ambrosio
Marcos Renato R. Araújo*

*Instituto de Computação - UNICAMP

Trabalho Apresentado no VII Simpósio de Computadores Tolerantes a Falhas – Campina Grande, Paraíba – Brasil - 01-04 julho 1997.

INPE
São José dos Campos
2003

Testes de Tolerância a Falhas em Sistemas de Comunicação

Eliane Martins
Marcos Renato R Araujo

Instituto de Computação - UNICAMP
Cid. Univ.Zeferino Vaz - CP 6176
13083-970 Campinas - SP

{eliane,maraujo}@dcc.unicamp.br

Ana Maria Ambrósio
Keila Silveira Corrêa
Maria de Fatima Mattiello Francisco
DSS-INPE
CP 515
12227-010 S.José dos Campos SP

{ana,keila,fatima}@dss.inpe.br

Resumo

Protocolos tolerantes a falhas devem fornecer seus serviços corretamente, mesmo quando alguns nós de um sistema distribuído apresentam falhas. A tais protocolos não basta a aplicação dos testes de conformidade e de interoperabilidade, é preciso exercitar os mecanismos que o fazem tolerar as falhas. Este artigo apresenta uma ferramenta para teste de protocolos cujo objetivo principal é mostrar a aplicabilidade da técnica de injeção de falhas para validação dos mecanismos de tolerância a falhas em sistemas de comunicação. Esta ferramenta, denominada Sistema de Suporte à Execução de testes (SSE), implementa a arquitetura ferry-clip, utilizada para testes de conformidade, acrescentando-lhe mecanismos de injeção de falhas. O SSE é um sistema de teste flexível e portátil projetado para suportar vários tipos de testes. Além da arquitetura, algumas soluções da implementação também são apresentadas.

Abstract

Fault-tolerant protocols must provide correct services to faultless nodes of a distributed system. To these protocols it isn't enough the conformance and interoperability testing, it is needed to run the mechanisms that tolerate the faults. This paper presents a tool for protocol testing aimed to show the applicability of the ferry-clip concept for fault tolerance testing. This tool, named System for Testing Execution Support - SSE, implements the ferry-clip architecture, which has been augmented with the fault injection technique. The resultant architecture, proposed here, combines the ferry-clip approach for conformance testing with fault-injection for fault-tolerance validation. The SSE is a flexible and portable system for supporting several types of test. Some solutions in implementing it are also presented.

Palavras-chave: testes, tolerância à falhas, protocolos, injeção de falhas, ferry-clip

1. Introdução

Um aspecto importante no desenvolvimento de sistemas de alta qualidade é a sua validação. Sistemas que devem fornecer seus serviços apesar da ocorrência de falhas, devem garantir que uma certa confiança possa, justificavelmente, ser obtida do sistema em desenvolvimento [Laprie92]. Para merecer a qualificação de sistema com segurança no funcionamento (*dependable system*), um sistema deve ser submetido a uma validação intensiva. Uma fonte de evidência para a confiança no software é o teste [Littlewood92].

No que concerne a sistemas de comunicação, a ISO - International Standardization Organization - propõe pelo menos quatro tipos de teste a serem aplicados [Rayner87]: conformidade, interoperabilidade, performance e robustez.

Além dos tipos de testes a serem aplicados, a ISO em conjunto com a CCITT (atual ITU) definiram uma metodologia e o contexto para realização de testes de conformidade dando origem a norma ISO9646- "Conformance Testing Methodology and Framework" ou ITU-T X.290. Este documento descreve, entre outros, as arquiteturas de teste de protocolos.

Uma variação da arquitetura de teste de conformidade proposta pela ISO é a arquitetura ferry-clip, a qual permite que testes de conformidade de uma dada implementação sejam realizados sem necessidade do sistema de teste estar fisicamente junto à implementação em teste.

Os testes de conformidade, entretanto, não são suficientes para validar completamente protocolos tolerantes a falhas, dado o limitado número de modelos de faltas que eles oferecem [Bosik91, Bochmann94]. Por esta razão, em adição aos testes acima citados, são recomendados testes de tolerância a falhas.

Uma técnica que vem sendo reconhecida e aceita como um método valioso para avaliação de mecanismos de tolerância a falhas é a *injeção de falhas*. Esta técnica permite acelerar a ocorrência de falhas e testar os mecanismos de tolerância a elas, de forma controlada.

Neste trabalho é apresentada uma arquitetura para teste de protocolo que combina a arquitetura ferry-clip, que cobre os testes de conformidade de protocolos do padrão ISO, com componentes injetores de falhas. Esta aquitetura é implementada no escopo do sistema chamado Sistema de Suporte à Execução de Testes - SSE.

Este artigo está organizado de forma que a seção II traz uma rápida definição dos tipos de testes recomendados e as arquiteturas de teste de protocolos, incluindo uma introdução à tecnologia de injeção de falhas. A arquitetura do sistema de teste com injeção de falhas da ferramenta SSE é apresentada na seção III. A seção IV procura esclarecer detalhes da implementação do SSE no contexto de sua primeira aplicação, uma implementação do protocolo X.25. Para esta aplicação, parte do SSE foi desenvolvida em UNIX e parte em Windows. Alguns trabalhos correlatos a este projeto são, ilustrativamente comparados na seção V e uma conclusão encerra o artigo.

II. Testes de sistemas de comunicação

II.1. Tipos de testes

A seguir é apresentada uma breve definição dos vários tipos de teste considerados para validação de protocolos.

Testes de conformidade: são aqueles que determinam se uma dada implementação de protocolo está conforme à sua especificação, verificando-se o seu comportamento externo.

Testes de interoperabilidade: neste tipo de teste verifica-se o comportamento fim-a-fim da implementação do protocolo garantindo que diferentes implementações de um mesmo protocolo comunicam-se corretamente.

Testes de performance: o objetivo destes testes é verificar se a implementação obedece aos requisitos de desempenho previamente estabelecidos.

Testes de robustez: com estes testes pretende-se observar a implementação sob situações de erros que não foram consideradas na especificação do protocolo correspondente.

Testes de tolerância a falhas: o objetivo aqui é validar a capacidade de tolerância a falhas de sistemas de comunicação, utilizando-se a técnica de injeção de falhas.

A injeção de falhas pode ser aplicada de diversas formas, dependendo do nível de abstração usado para representar o sistema alvo (modelo, protótipo ou sistema final) e do tipo de falhas injetadas. Neste artigo não será feita uma apresentação detalhada da técnica de injeção de falhas, a qual pode ser vista em [Martins 92,93ab] e [Arlat 90]. Neste estudo a forma de injeção empregada é a **injeção (de falhas) por software**, que permite injetar tanto erros de software induzidos por falhas de hardware quanto falhas de software (representando falhas que ocorrem durante o desenvolvimento do software). Apenas a injeção de conseqüências de falhas de hardware é levada em conta aqui. Os tipos de falhas mais comumente considerados neste caso são [Segall 88ab, Kanawati 92, Rosenberg 93, Kao93]: falhas de *memória*, falhas de processador, falhas de barramento, falhas de comunicação.

Serão consideradas neste estudo somente as falhas de comunicação, pois elas permitem emular vários modelos de falhas de sistemas distribuídos, que afetem tanto nós quanto o meio de comunicação, como mostram alguns estudos (c.f. [Dawson94]). Além disso, para injetar falhas de comunicação não é necessário instrumentar a implementação em teste, o que seria o caso para outros modelos de falhas. Desse modo, os testes da tolerância a falhas podem ser aplicados mesmo quando o código fonte não está disponível.

O tipo de falha indica a forma da alteração a ser injetada. Além do tipo, outros atributos que caracterizam as falhas são: *opções de injeção* - mensagens podem ser perdidas, alteradas, entregues com atraso ou duplicadas; *direção* - indica se as mensagens afetadas são somente as mensagens recebidas, transmitidas ou ambas; *localização* - indica as posições de memória ou as partes da mensagem onde serão injetadas as falhas; *multiplicidade* - representa o número de bits ou bytes que devem ser alterados; taxa de injeção: corresponde à freqüência em que uma dada falha é injetada. As falhas podem ser permanentes, transientes ou intermitentes, neste caso, o intervalo entre ocorrências das falhas pode ser determinístico (pré-fixados) ou aleatório (seguir uma distribuição, por exemplo, exponencial). Falhas podem ser injetadas também, de acordo com o estado do sistema alvo: neste caso pode-se definir uma taxa de falhas para cada estado do sistema (como é mostrado em [Goswami91]).

II.2. Arquiteturas para teste de protocolos e o Conceito de ferry-clips

A norma ISO9646 foi concebida com o intuito de generalizar resultados de testes de protocolos e evitar repetições de teste de conformidade. A arquitetura conceitual de teste de conformidade proposta nesta norma é apresentada na figura II.1. Nela a implementação a ser testada, a IUT - Implementation Under Test, é considerada uma "caixa preta", observada apenas através das suas interfaces de serviço, superior e

interior. Um testador é associado a cada uma destas interfaces: o testador superior (UT- Upper Tester) e o testador inferior (LT- Lower Tester).

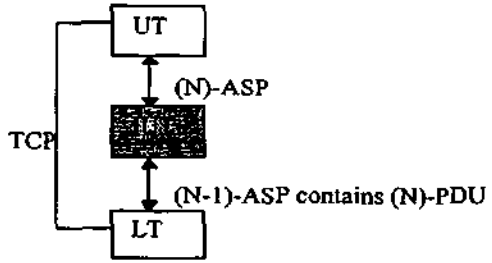

Figura II.1 Arquitetura conceitual para teste de conformidade.

O testador superior (UT) usa os serviços oferecidos pela IUT na sua interface superior para envio e recepção de primitivas de serviço abstrato (ASP- Abstract Service Primitives). O testador inferior (LT) controla e observa as interações da IUT, no que se refere a sua comunicação com o nível inferior. Assim, este testador está relacionado com o (n-1)-ASP, usado pela IUT para a transferência das unidades de dados de protocolo (PDUs- Protocol Data Units). TCP (Test Coordination Procedure) é um procedimento de coordenação de testes responsável por manter os testadores em sincronia e gerenciar a troca de dados entre eles.

A fim de aumentar a portabilidade e o poder dos sistemas de teste Zeng [Zeng86] introduziu o conceito de *ferry* ou transportador. O *ferry* consta de um mecanismo para transportar exclusivamente dados de teste entre o sistema de teste e o sistema em teste. Os testadores UT e LT, descritos acima, vão residir na mesma máquina facilitando o sincronismo entre eles e reduzindo as interferências na IUT para aplicação dos testes. O sistema em teste (contendo a IUT) fica isolado do sistema de teste e ambos se comunicam via um canal especial para transportar as informações de teste. Dois componentes implementam a comunicação por este canal, o AF- Active Ferry residindo no sistema de teste e o PF- Passive Ferry residindo no sistema em teste. Esta arquitetura foi escolhida para esse trabalho pela facilidade para a sincronização entre os testadores e pela sua flexibilidade, permitindo a realização de diferentes tipos de teste, como mostraram diversos trabalhos [Zeng88, Chanson90, Chanson92]. Sua descrição é apresentada no item a seguir.

III. Projeto de um sistema de testes baseado no conceito de ferry-clip

O Sistema de Suporte à Execução de testes (SSE) consta de uma ferramenta para controlar e monitorar a execução de testes de sistemas de comunicação. Seu objetivo é dar suporte aos testes de tolerância à falhas além de outros tipos de testes de protocolos de comunicação.

Alguns desafios estabelecidos para o desenvolvimento do SSE foram: i) definir uma arquitetura de teste que fosse o mais "facilmente" possível adaptável aos vários tipos de testes: tolerância à falhas, conformidade, interoperabilidade, robustez e performance; ii) possibilitar o controle e a observação dos testes durante a execução dos mesmos com o menor nível de interferência na execução do sistema alvo; iii) poder

inserir, e desta forma acelerar, a ocorrência de falhas no sistema em teste de forma controlada e reproduzível.

Nesta seção é apresentada a arquitetura do SSE e subseqüentemente a descrição de cada um de seus componentes. A arquitetura adotada é baseada no conceito de ferry-clip (item II.2) pela facilidade que esta oferece para a realização de diferentes tipos de teste. Mais precisamente, foi utilizada uma variante das arquiteturas usadas em [Zeng88], [Chanson90] e [Chanson92], acrescentando-se a elas os componentes responsáveis pela injeção de falhas, como é mostrado na figura III.1 [Ambrosio96]. Para simplificar a figura, somente os principais componentes foram representados.

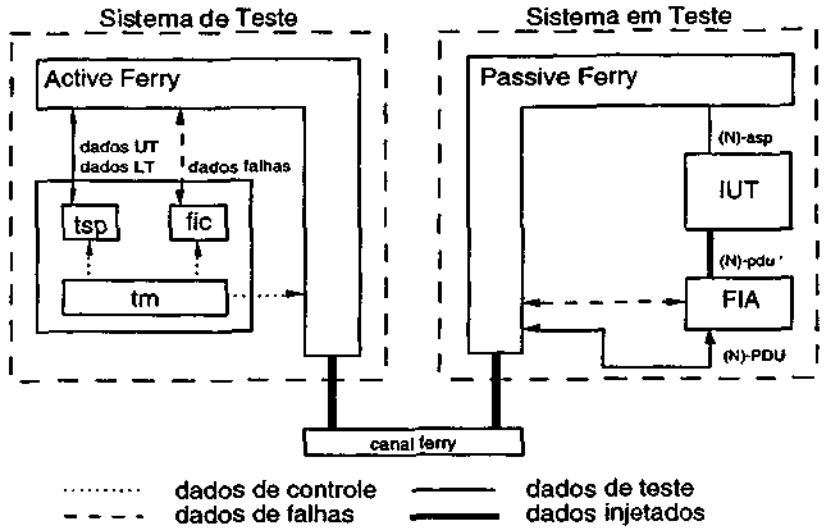

Fig. III.1. Arquitetura ferry-clip para injeção de falhas.

Descrição dos componentes

AF - Active Ferry

A divisão funcional deste componente segue a proposta de [Chanson92], na qual AF é composto de 3 módulos que se utilizam e oferecem primitivas de serviço: FT-ASP, FM-ASP, FD-ASP, FTMP-ASP, conforme mostra a figura III.2. O FD-Service transfere, através da unidade FY-PDU, os dados de teste e os dados sobre as falhas a serem injetadas.

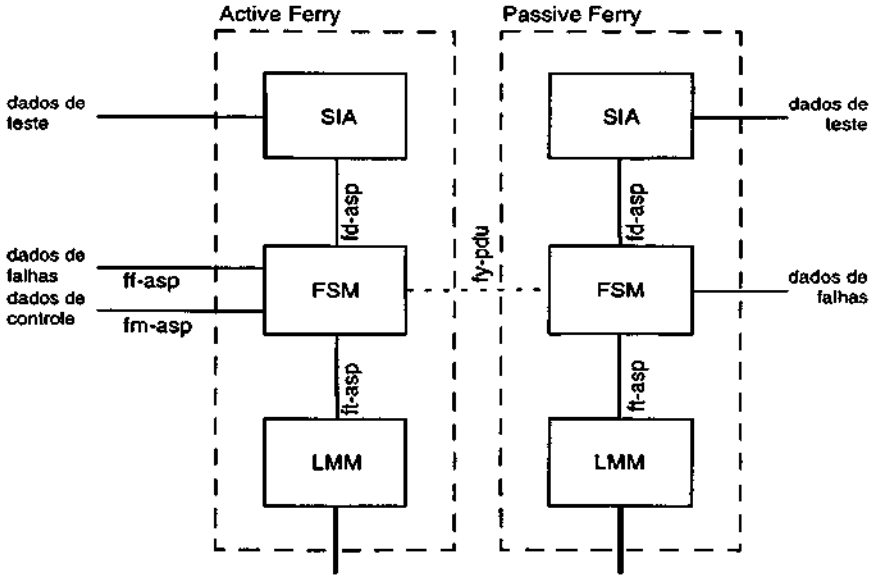

Fig. III.2. Estrutura dos componentes Active Ferry (AF) e Passive Ferry (PF)

AF-SIA (Active Ferry - Service Interface Adapter) - converte os dados de teste de um formato interno ao Sistema de Teste na representação específica da IUT. Desta forma, o TSP tem uma interface padronizada, independente da IUT em teste.. O AF-SIA é dependente da IUT, tendo que ser alterado para o teste de diferentes implementações ou de diferentes camadas de uma mesma implementação em teste.

AF-FSM (Active Ferry - Finite State Machine) - implementa os serviços do protocolo do *ferry* (FP). O serviço de transferência de dados, FD-Service, é acionado através das primitivas: FD-ASP. Os dados passados através dessas primitivas são inseridos em FY-PDU para serem transferidos ao PF. Os FY-PDUs são enviados através do canal do ferry, o qual utiliza um protocolo denominado FTMP (Ferry Transfer Medium Protocol); os serviços do FTMP são acionados através de primitivas FT-ASP. Os serviços de controle da conexão, fornecidos por este componente, podem ser acionados através das primitivas FM-ASP. A máquina de estados que ele implementa sofreu pequenas modificações para atender a um novo canal de comunicação virtual entre os componentes que cuidam da injeção das falhas: FIC e FIA, descritos à frente.

AF-LMM (Active Ferry - Lower Mapping Module) - este módulo mapeia as primitivas de serviço de transferência do ferry (FT-ASP) em primitivas específicas do FTMP. Dessa forma, este é o único módulo do componente AF a ser alterado no caso de mudanças no FTMP.

PF - Passive Ferry

O componente PF contém os mesmos módulos que o Active Ferry, a saber:

PF-SIA (Passive Ferry - Service Interface Adapter) - implementa a interface com a IUT. É o componente a ser alterado em caso de mudança na IUT

PF-FSM (Passive Ferry - Finite State Machine) - implementa os serviços do protocolo do ferry (FP), no ambiente da IUT.

PF-LMM (Passive Ferry - Lower Mapping Module) - mapeia as primitivas do serviço de transferência do ferry (FT-ASP) em primitivas específicas do FTMP. Dessa forma, este é o único módulo de PF a ser alterado no caso de mudanças no FTMP.

TM - Test Manager

Este componente é responsável pelo início e término das sessões de teste. Ele lê e executa os testes de acordo com a parte descritiva de um *script* de teste, o qual contém, as configurações e inicializações necessárias à realização dos testes. Ele deve controlar o tempo de execução de cada sequência de teste, bem como, atender às interações de um usuário.

TSP - Test Suite Processor

A execução propriamente dita dos testes é realizada por este componente. Os comandos de teste são adquiridos da parte executiva do *script*. Os testadores UT e LT foram incorporados por este componente. Todas as trocas de dados efetuadas com o sistema em teste são armazenadas para análise futura.

FIC - Fault Injection Controller

Este componente controla remotamente a injeção de falhas. A descrição das falhas é obtida do *script* e passada ao componente FIA numa unidade FY-PDU usando-se os serviços de transporte do ferry. O controle da frequência de execução da falha, ou seja, a *taxa de injeção*, que pode ser baseada numa distribuição estatística no caso de falhas intermitentes e/ou transientes, é controlada por este componente. As características da falha a ser injetada, como: a localização e a multiplicidade, no caso de erro do PDU e/ou a ação seja, exclusão, retenção ou duplicação de PDU são passadas para o componente FIA. Todos os dados transmitidos e recebidos do componente FIA são devidamente armazenados por este componente.

FIA - Fault Injection Agent

Componente localizado no sistema em teste capaz de interceptar todos os PDUs que chegam e saem para/da IUT, assim, podendo injetar qualquer um dos tipos de falhas indicado pelo componente FIC. Informações como o momento da injeção e o tipo de falha injetada são reportadas de volta ao FIC para conferência e armazenamento.

A arquitetura descrita nesta seção refere-se à realização dos testes de tolerância à falhas sendo que testes de robustez também podem ser aplicados utilizando-se a mesma arquitetura. Para realização exclusiva dos testes de conformidade, os componentes de

falhas (FIC e FIA) são desativados ou excluídos da arquitetura sem prejuízo funcional. Esta arquitetura é aplicável ao teste de uma IUT isoladamente.

Os testes de interoperabilidade (c.f. II.1) visam observar a interação entre várias IUTs para aumentar a probabilidade de que elas possam trabalhar corretamente em situações reais. Neste caso, são necessárias duas ou mais IUTs ou uma IUT e uma outra implementação do mesmo protocolo. Para realização deste tipo de teste a alteração necessária no sistema de teste é a seguinte: i) o componente TSP deve coordenar mais de um testador superior (UT), sendo que haverá um UT para cada IUT, ii) deve haver um componente AF para cada IUT ligada ao sistema de teste, iii) o testador inferior (LT) é desativado ou excluído pois as IUTs se comunicarão diretamente pelo canal de teste. Esta arquitetura também pode ser usada nos testes de tolerância a falhas, quando se deseja observar o comportamento de várias IUTs em presença de falhas. Da mesma forma que o UT, o FIC é replicado para controlar a injeção de cada IUT, permitindo que sejam aplicados modelos de falhas diferentes para as diversas IUTs. É possível também que algumas IUTs não sejam submetidas à injeção; neste caso, não há FIC associado a elas, e o FIA também não é ativado no sistema em teste.

IV. Implementação do sistema

O Sistema de Suporte à Execução está sendo implementado em duas plataformas diferentes: o *Sistema de Teste* roda em uma estação Sun com sistema operacional Solaris 2.5 e o *Sistema em Teste* roda em um microcomputador tipo IBM-PC 486 com Microsoft Windows for Workgroups 3.11 com suporte à comunicação em rede.

A escolha da estação Sun Solaris para a implementação do Sistema de Teste deve-se à disponibilidade do equipamento, enquanto que a escolha do PC foi uma exigência da IUT, no caso uma implementação do protocolo X.25 desenvolvida no INPE.

Como protocolo do meio de tranferência (FTMP) foi escolhido o protocolo TCP/IP, o qual está disponível nos dois sistemas e garante a troca ordenada e sem erros dos dados na comunicação entre os sistemas. O protocolo ferry (FP) é implementado pelos componentes AF e PF.

Na implementação do AF-FSM foram utilizados *sockets* em Solaris, para criação e troca de pacotes TCP/IP através de uma rede local. Para se obter os mesmos resultados e permitir a comunicação no canal *ferry*, em PF-FSM, implementado no PC, foi utilizado um pacote de facilidades para Microsoft Windows denominado *Trumpet WinSock*.

Os programas do Sistema de Teste estão sendo implementados em linguagem C++, utilizando-se o compilador g++ da GNU e o Sistema em Teste está sendo implementado em C++ utilizando-se o Microsoft Visual C++.

No Sistema de Teste os componentes TSP, FIC e TM formam um único processo denominado TE (Test Engine). O componente AF foi dividido em dois processos diferentes: FSM-LMM, responsável pelo estabelecimento e manutenção da conexão entre AF e PF e o processo SIA que realiza a codificação/decodificação dos dados para o formato compatível com a IUT.

O processo AF-SIA está implementado através de 2 classes: **CConvX25**, cujos serviços montam os dados de teste no formato do protocolo X.25, e **CBuffer**, que contém a definição da estrutura de um quadro X.25.

Devido às limitações do ambiente Windows (3.11), a comunicação entre processos no Sistema em Teste é feita da seguinte forma: cada programa executa sua parte e, através de mensagens, informa ao outro que é a sua vez de executar o serviço. As mensagens trocadas entre os processos correspondem a *eventos*, que por sua vez são realizados através de *sockets*, especialmente configurados para trabalhar na mesma máquina. Para a implementação da comunicação dos processos no PC foi utilizada a classe CWinsock oferecida pelo Visual C++.

Quando se optou pelo protocolo TCP/IP, cada máquina recebeu um endereço IP através do qual é acessível na rede. Ao ser inicializado, o Sistema em Teste deixa disponível um ponto de comunicação ao qual são estabelecidas as conexões e fica inativo até que algum pedido de conexão chegue a ele. Este ponto de comunicação recebe o nome de *port*.

Quando o Sistema de Teste deseja estabelecer conexão com o Sistema em Teste, este envia um pacote através da rede contendo o endereço IP da máquina que está executando o Sistema em Teste e seu respectivo *port*. Este, por sua vez, recebe o pedido e estabelece a conexão. Estabelecida a conexão, ambos podem interagir como um único sistema.

O componente PF também foi dividido em dois processos. O primeiro processo agrupa os módulos FSM e LMM e o segundo implementa o módulo SIA. O processo SIA do PF ao receber um FY-PDU, desmonta-o, verifica sua origem e direciona-o a uma determinada porta da IUT, dependendo se for um comando do UT ou um frame do LT. Analogamente, recebe as respostas da IUT e dependendo do port, monta um FY-PDU informando a origem e o destino do dado.

O processo PF-SIA é implementado por 4 classes: **CPfsia**, **CWinsock**, **CStreamSock** e **Cpdu**. A classe **CPfsia** define a aplicação com 4 ports, sendo 3 para a comunicação com a IUT (ports 2100, 2200 e 2300), e 1 voltado à conexão com o processo FSM-LMM (port 2000). O serviço OnConnPort0() atende ao pedido de conexão por parte do processo FSM-LMM. O tratamento das informações de teste recebidas pelo PF-SIA é feito, segundo a sua ocorrência, pelos serviços OnDataPort1(), OnDataPort2(), OnDataPort3(), e OnDataPort0().

```
class CPFSia : public CformView
{
private:
 CWinSock * m_pWinSock;
 CStreamSocket * m_pStream1;  // Stream socket para envio/recepção de dados pela porta 2100
 CStreamSocket * m_pStream2;  // Stream socket para envio/recepção de dados pela porta 2200
 CStreamSocket * m_pStream3;  // Stream socket para envio/recepção de dados pela porta 2300
 CStreamSocket * m_pStream0;  // Stream socket para enviar/receber dados pela porta 2000
 CPdu *m_pPdu;
protected:
 afx_msg LONG OnDataPort1();  // trata envio/recebimento de dados pela porta 2100
 afx_msg LONG OnDataPort2();  // trata envio/recebimento de dados pela porta 2200
 afx_msg LONG OnDataPort3();  // trata envio/recebimento de dados pela porta 2300
 afx_msg LONG OnConnPort0();  // responde a pedido de conexao na porta 2000
 afx_msg LONG OnDataPort0();  // trata envio/recebimento de dados pela porta 2000
};
```

As classes **CWinsock** e **CStreamSocket** cuidam da comunicação entre as aplicações, como inicialização do socket, criação das portas, conexão com as demais aplicações, envio e recebimento de mensagens. Os serviços da classe **CPdu** extraem do socket o dado de teste recebido (CPdu::GetField()) e informam a origem/destino da informação (CPdu::GetPCO()) para a porta adequada dos processos IUT e FSM-LMM.

```
class CWinSock
{
private:
 WORD m_wVersionRequired;
 int m_nLastError;
 WSADATA m_wsaData;
public:
 int Startup(); // inicializar processo de comunicação
 int Shutdown(); // finalizar processo de comunicação
};

class CPdu
{
 char pszMessage[300];  // buffer de messagem
public:
 int GetField(); // extrair dado da msg recebida
 int GetPCO(); // informar origem/destino do dados de teste
(0 = L.T./1=U.T.)
};
```

```
class CStreamSocket : public CWnd
{
private:
 CWnd *m_pParentWnd; // janela notificação de eventos
 UINT m_uMsg;   // mensagem a ser enviada
 SOCKET m_s;   // identificador do socket
 CPtrList m_listWrite;  // dado a ser enviado
 CPtrList m_listRead;  // dado lido
public:
 int CreateSocket();          // criar socket
 int DestroySocket();         // destruir socket
 int Connect();               // pedir conexão
 int Accept();                // aceitar pedido de conexão
 int Write();                 // escrever dados no socket
 LPVOID Read();               // ler dados do socket
};
```

V. Trabalhos correlatos

Existem diferentes técnicas e estratégias de validação de sistemas distribuídos. Dentre os estudos existentes os que estão mais proximamente relacionados com o trabalho apresentado pertencem basicamente às seguintes áreas: testes de protocolos no contexto normativo (ISO9646) e testes experimentais de protocolos por injeção de falhas.

Com relação aos testes no contexto normativo, o SSE, por ser baseado na arquitetura ferry apresenta as vantagens mostradas em II.2 com relação às arquiteturas recomendadas pela norma ISO9646. A arquitetura utilizada é baseada naquela apresentada nos trabalhos de Chanson et al. [Chanson90, Chanson92], sendo portanto a principal diferença com relação a esses trabalhos o acréscimo das funcionalidades referentes à injeção de falhas.

Com relação aos testes de protocolos por injeção de falhas o número de estudos a respeito ainda não é muito significativo, mas a área está em expansão. Em [Arlat90] e [Martins93a] por exemplo, é mostrado o uso de injeção física de falhas na validação de um protocolo de difusão atômica, parte do sistema de comunicação do projeto Delta-4 [Powell91]. Com relação ao uso de injeção por software, que é a forma de injeção enfocada neste trabalho, tem-se alguns estudos significativos. Por exemplo, [Winfrey89] apresenta um injetor onde falhas de comunicação são introduzidas alterando-se as primitivas SEND e RECEIVE usadas pela IUT para enviar e receber mensagens. O injetor trabalha com base em um *script* onde estão descritas as falhas a serem injetadas. Em [Avresky91] é mostrada a validação de um protocolo inter-réplicas (IRP), também parte do sistema de comunicação de Delta-4. Neste estudo o enfoque está no uso do modelo formal, utilizado para especificar o mecanismo de tolerância à falhas em teste, tanto para a geração das entradas de teste (falhas e dados para o IRP) quanto para a obtenção das saídas esperadas, útil para a análise dos resultados. Nestes dois trabalhos a validação tinha por objetivo a eliminação de falhas de projeto/implementação dos mecanismos de tolerância a falhas da IUT.

No caso do trabalho aqui descrito, o SSE deve dar suporte tanto para a eliminação de falhas quanto para a obtenção de medidas da eficiência dos mecanismos de tolerância a falhas. Os trabalhos que mais se aproximam do estudo aqui apresentado são os que descrevem as ferramentas EFA e SockPFI, as quais serão apresentadas a seguir.

A ferramenta EFA [Echtle92] baseia-se no uso de uma camada de injeção, inserida abaixo da IUT, a qual é responsável por realizar a injeção de falhas. Essa camada reside sempre acima da camada de enlace, e deve estar presente em cada nó da arquitetura distribuída. A ferramenta executa em processadores da família 68000, sob o sistema operacional A/ROSE, um sistema distribuído com suporte para tempo real . A similaridade com o SSE vem do fato de que é usada uma camada de injeção entre a IUT e as camadas inferiores; essa camada é implementada pelo FIA, como descrito em III. No entanto, em EFA essa camada de injeção controla quando e como injetar uma falha, o que não acontece no SSE, onde a decisão de quando injetar fica a cargo do FIC, residente no sistema de teste. Dessa forma reduz-se a complexidade de código de teste que precisa residir no sistema sob teste. Uma outra diferença com relação ao EFA é que no SSE o FIA não precisa estar presente em cada nó da arquitetura distribuída, sendo facultativo aos nós para os quais se deseja efetuar a injeção de falhas. Também no caso do SSE o FIA não precisa necessariamente residir acima da camada de enlace, como acontece com a camada de injeção do EFA.

A ferramenta SockPFI [Dawson95] também se baseia no uso de uma camada de injeção, como a EFA, só que com algumas diferenças. Em SockPFI a camada de injeção também não é fixa, como para a EFA, podendo variar conforme a IUT. Da mesma forma que para a EFA, a decisão de quando injetar também está embutida na camada de injeção; enquanto em EFA o algoritmo de controle de injeção é compilado junto com as outras funcionalidades da camada de injeção, em SockPFI o controle é feito por um *driver* que interpreta um *script*, o que a torna mais flexível para a realização de diferentes conjuntos de testes.

No caso do SSE, o FIC também é dirigido por um *script*; a diferença está em que no SSE o FIC não reside no nó sob teste. Uma outra semelhança entre o SSE e o SockPFI é que em ambos a camada de injeção não precisa residir em todos os nós da arquitetura sob teste, como na EFA. Dessa forma é possível conectar à arquitetura sob teste alguns nós onde não seja possível inserir uma camada abaixo da IUT. SockPFI foi projetada para validar sistemas de tempo real; para reduzir o impacto que a camada de injeção causa no sistema sob teste, são utilizadas facilidades oferecidas pelo sistema operacional distribuído com suporte para tempo real, RTMach.

Uma diferença importante do SSE com relação aos estudos citados anteriormente é que, conforme já foi mencionado, o SSE é uma ferramenta multi-testes, servindo tanto para realizar os testes de protocolos recomendados pela ISO9646 quanto os testes da tolerância a falhas usando injeção de falhas. Por essa razão o SSE deve dar suporte tanto aos testes visando a eliminação de falhas de projeto/implementação da IUT (ou de seus

mecanismos de tolerância a falhas), quanto para a avaliação, seja do desempenho da IUT, seja da eficiência de seus mecanismos de tolerância a falhas.

Finalmente uma outra diferença importante com relação, principalmente, as ferramentas de injeção de falhas que foram apresentadas, é que o SSE é multi-plataforma, sendo que tanto o sistema de teste quanto a parte residente no sistema sob teste podem ser executadas tanto em ambiente Unix, DOS/Windows ou qualquer outro que dê suporte para o uso de *sockets*. Dessa forma pretende-se que o SSE possa ser usado para validar uma vasta gama de sistemas, inclusive com arquiteturas heterogêneas, requerendo para isso um mínimo de alterações.

ANÁLISE PRELIMINAR

A ferramenta está atualmente em fase de implementação, através de implementação incremental. Até o momento foram realizados testes que comprovaram a possibilidade de troca de dados entre SUN Solaris e Microsoft Windows 3.11.

Os testes de comunicação consistiram de se implementar um software no Windows que ecoava de volta quaisquer caracteres que fossem enviados através de um port de comunicação especificado por ele. Este pequeno software aguardava conexões (no caso através do port 2000), mostrava o que foi recebido em uma janela e ecoava de volta a mesma mensagem de volta ao solicitante. Foi feito o mesmo esquema utilizando o mesmo programa no Solaris e utilizando um software cliente no PC para estabelecer a conexão. A troca de dados não apresentou nenhum problema em nenhum dos casos.

Para efeito de desempenho, foi colocado no software cliente um time-stamp para que se pudesse cronometrar o tempo de ida e volta dos dados. Em ambos os casos o tempo de ida e volta da mensagem não ultrapassou 5ms (milissegundos) no pior caso, ficando em média em torno de 2ms. A rede local consistia de uma rede Ethernet englobando várias máquinas, inclusive as máquinas servidora e cliente. Estes valores são considerados apropriados para o desenvolvimento da ferramenta.

CONCLUSÕES E PERSPECTIVAS

O SSE é uma ferramenta que está prevista para fazer parte de um ambiente de teste maior denominado ATIFS - Ambiente de Teste baseado em Injeção de Falhas por Software, que vem sendo desenvolvido em cooperação entre a UNICAMP e o INPE.

A primeira aplicação do SSE está sendo preparada para a validação de uma implementação do protocolo X.25 desenvolvida pelo INPE, para ser executada em PC, cuja codificação já está concluída.

Uma segunda aplicação possível é para o teste de uma implementação do protocolo solo-bordo desenvolvido no escopo do programa de Satélites Científicos - SACI. Este protocolo é baseado nas recomendações CCSDS (Consultative Committee for Space Data Systems) para comunicação entre equipamentos que ficam em solo com equipamentos a bordo do satélite.

Com relação a arquitetura de teste proposta podemos dizer que a introdução dos componentes injetores de falhas não comprometeu os princípios da arquitetura ferry-clip de máxima independência entre os sistemas de teste e sistema em teste e mínima alteração no sistema em teste. Toda estrutura *ferry* foi aproveitada para transportar os

dados sobre as falhas ao agente injetor localizado no sistema em teste. O controle, o armazenamento das informações e dos resultados, bem como, a monitoração do processo de injeção das falhas é executado no componente Fault Injector Controller, localizado no sistema de teste.

A adoção da arquitetura ferry no Sistema de Suporte à Execução de testes permitiu que vários tipos de testes fossem realizados em diferentes IUTs com o mínimo de alteração no código do sistema de teste.

Neste artigo foi apresentada a arquitetura do SSE, demonstrando a viabilidade da implementação física da combinação dos conceitos de ferry-clip e injeção de falhas. A arquitetura de teste do SSE se utiliza de todas as vantagens do ferry-clip e ainda permite a injeção de falhas para a validação da tolerância à falhas de um protocolo. Visando generalizar a criação de objetos de sistemas de teste para ampliar e facilitar ainda mais a re-utilização dos componentes do SSE foi elaborado um modelo de objetos mais geral que a arquitetura apresentada [Martins96]. A próxima etapa deste trabalho consta da aplicação dos testes na implementação do protocolo X.25 e da avaliação dos resultados de teste, principalmente com relação às falhas injetadas.

AGRADECIMENTOS

Gostaríamos de agradecer ao CNPq, pelo financiamento da bolsa de pesquisa de Eliane Martins, bem como pela bolsa de estágio de Marcos Renato Araujo e Keila Silveira Correa que realizaram parte do trabalho de implementação do sistema de teste. E pela bolsa de especialização no Centro de Ciência Espacial da Universidade de Sussex concedida a Ana Maria Ambrosio. Ao Dr. M. Paul Gough, agradecemos a oportunidade de pesquisa na área de injeção de falhas.

REFERÊNCIAS

[Ambrosio96] A.M.Ambrosio, E.Martins. Documento de especificação dos Requisitos de Software do Subsistema de Suporte a Execução. *Relatório técnico interno - INPE* 1996.

[Arlat 90] J.Arlat, M.Aguera, L.Amat, Y.Crouzet, J.-C.Fabre, J.-C.Laprie, E.Martins, D.Powell. Fault injection for dependability validation - a methodology and some applications. *IEEE Transactions on Software Engineering*, vol. 16, fev. 1990.

[Arlat 92] J.Arlat. Fault injection for the experimental validation of fault tolerant computer systems. *Relatório interno do LAAS*, nº 92489, Outubro de 1992.

[Avresky 91] D.R.Avresky, J.Arlat, J.-C.Laprie, Y.Crouzet. Guiding the process of fault injection for testing fault tolerance. *Relatório interno LAAS nº 91-351*. Dezembro 1991.

[Bochmann94] G.V.Bochmann, A.Petrenko. Protocol testing: review of methods amd relevance for software testing. *The Intl. Symposium on Software Testing and Analysis* (ISSTA), 1994, pp. 109-124.

[Bosik91] B.Bosik, M.Uyar. Finite state machine based formal methods in protocol conformance testing: from theory to implementation, in *Computer Network and ISDN Systems*, nr. 22, 1991, pp. 7-33.

[Chanson 90] S.T.Chanson; B.P. Lee; N.J. Parakh; X.H. Zeng. Design and implementation of a ferry-clip test system. Protocol Specification, Testing and Verification IX, edited by E. Brinksma, G.Scollo, C.A.Vissers. Elseviers Science Publishers, North-Holland, 1990, pp. 101-118.

[Chanson 92] S.T.Chanson; S.T. Vuong; H. Deny. Multy-party and interoperability testing using the ferry-clip approach. Computer communications, vol 15,nr 3, april 1992, pp. 153-168 .

[Dawson94] S.Dawson, F.Jahanian. Probing and fault injection of protocol implementations. Research report from Univ. of Michigan, n. CSE-TR-217-94, 1994, 22pg.

[Dawson95] S.Dawson, F.Jahanian, T.Mitton. A software fault injection tool on Real-Time Mach. Relatório de pesquisa da Univ. de Michigan, 1995.

[Echtle 92] K.Echtle, M.Leu. The EFA fault injector for fault-tolerant distributed system testing. Anais do IEEE Workshop on Fault Tolerant Parallel and Distributed Systems, Amherst, MA, EUA, 1992.

[Goswami 91] K.K.Goswami, R.K.Iyer, "DEPEND: a simulation based environment for system level dependability analysis". Relatório da Univ. of Illinois at Urbana-Champaign n° CRHC-UIUC, 1991.

[Kanawati 92] G.A.Kanawati, N.A.Kanawati, J.A.Abraham. FERRARI: A Tool for the Validation of System Dependability Properties. *Proc. FTCS-22*, Boston, MA, USA, 1992.

[Kao 93] W.Kao, R.K.Iyer, D.Tang. FINE: A fault injection and monitoring environment for tracing the Unix system behavior under faults. *IEEE Transactions on Software Engineering*, 19(11), 1993.

[Laprie 92] J.-C.Laprie. Sûreté de fonctionnement: concepts de base et terminologie. *Dependable Computing and Fault Tolerance.* Springer Verlag, 1992.

[Leite 87] Julius C.B.Leite, Orlando G.Loques F°. Software. *II SCTF*, cap. 4 do mini-curso intitulado: Introdução à Tolerância a Falhas, Campinas, SP, 1987.

[Littlewood 92] B.Littlewood. The Risks of Software. *Scientific American*, pp. 62-75, Nov. 1992.

[Lovric 93] T.Lovric, K.Echtle. ProFI: Processor Fault Injection for dependability validation. *IEEE Intl. Workshop on Fault and Error Injection for Dependability Validation of Computer Systems*, Gotemburgo, Suécia, 1993.

[Madeira 93] H.Madeira, F.Moreira, M.Rela, P.Furtado, J.G.Silva. Pin-level fault injection for dependability validation: some research results at the University of Coimbra. *IEEE Intnl. Workshop on Fault and Error Injection for Dependability Validation of Computer Systems*, Gotemburgo, Suécia, 1993.

[Martins 92] E.Martins. Validation de systèmes répartis par injection de fautes. Tese de doutorado, ENSAE, 1992.

[Martins 93a] E.Martins. Teste de protocolos tolerantes a falhas por injeção de falhas. *11° SBRC*, Campinas, Maio de 1993.

[Martins 93b] E.Martins. Injeção de falhas na validação experimental da tolerância a falhas. *V SCTF*, mini-curso, S.José dos Campos, Outubro de 1993.

[Martins 96] E.Martins, A.M.Ambrosio, M.R.de Araujo. A framework for developing a software-based fault injection tool for the test of communication systems. Mexico, 1996.

[Rayner87] D.Rayne. OSI conformance testing. *PComputer Network and ISDN Systems*, nr. 4, pp. 79-98, 1987.

[Rosenberg 93] H.A.Rosenberg, K.G.Shin. Software Fault Injection and its Application in Distributed Systems. *Proc. FTCS-23*, Toulouse, França, 1993.

[Segall 88a] Z.Segall, D.Vrsalovic, D.P.Siewiorek, D.Yaskin, J.Kownacki, J.Barton, R.Dancey, A.Robinson, T.Lin. FIAT-Fault Injection based Automated Testing environment. *Proc. FTCS-18*, Tokyo, Japão, jun 1988.

[Segall 88b] Z.Segall, J.Barton, D.Vrsalovic, D.P.Siewiorek, R.Dancey, A.Robinson. Fault injection based automated Testing: practice and examples. *Proc.8° Digital Avionics System Conference*, San Jose, EUA, 1988.

[Zeng86] H.X. Zeng; D. Rayner. The impact of the ferry concept on protocol testing. Protocol Specification, Testing and Verification V, edited by MDiaz. Elseviers Science Publishers, North-Holland, 1986, pp. 533-544.

[Zeng88] H.X. Zeng; Q. Li; X.F. Du; C.S. He. New advances in ferry testing approaches. Computers networks and ISDN Systems, nr. 15, 1988, pp 47-54.

[Winfrey 89] T.L.Winfrey, G.E.Kaiser. Testing reliable distributed applications through simulated events. *Proc. FTCS-19*,1989.

MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

AUTORIZAÇÃO PARA PUBLICAÇÃO

Número
EEI-162/2002

Titulo
Testes de Tolerância a Falhas em Sistemas de Comunicação

Autor
Eliane MARTINS, Maria de Fátima MATTIELLO-FRANCISCO, Ana Maria AMBROSIO
Marcos Renato R. ARAUJO, Kiila Silveira CORRÊA

Tradutor

INPE-9591-PRE/5019 **Editor**

Origem	Projeto	Série	No. de Páginas	No. de Fotos	No. de Mapas
ETE/DSS			15		

Tipo
☐ RPQ ☒ PRE ☐ NTC ☐ PRP ☐ MAN ☐ PUD ☐ TAE

Divulgação
☒ Externa ☐ Interna ☐ Reservada ☐ Lista de Distribuição Anexa

Periódico / Evento
VII Simpósio de Computadores Tolerantes a Falhas
Campina Grande - Caraíba/PB 02 a 04 Julho 1997

Convênio

Autorização Preliminar
__/__/__
Data
Edenilse F.E. Orlandi
Chefe da Divisão de Desenv. de Sistemas de Solo

Revisão Técnica
☐ Solicitada ☒ Dispensada
Recebida __/__/__ Devolvida __/__/__
Leonardo Fernando Perondi
Titular de Nível "A"
Assinatura do Revisor

Revisão de Linguagem
☐ Solicitada ☒ Dispensada
Recebida __/__/__ Devolvida __/__/__
Titular de Nível "A"
Assinatura do Revisor

Autorização Final
__/__/__
Data
Edenilse F.E. Orlandi
Chefe da Divisão de Desenv. de Sistemas de Solo

Palavras Chave
testes - tolerância a falhas - protocolos - injeção de falhas - sistemas comunicação

(continua na folha 2/2)

Autorização Para Publicação

Secretaria

__/__/__ Recebida __/__/__ Devolvida __/__/__

Data

_____ _____

Encaminhado Por Devolvido Por

Controle e Divulgação

__/__/__ Recebido Por: _____ Devolvido Para: _____

Data

Pronto Para Publicação em: __/__/__ ___/___/___

Data

No. _____ Quant._____ _____

Assinatura

Observações