UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

INFIMO - Um Toolkit para Experimentos de Intrusão de Injetores de Falhas

por

PATRÍCIA PITTHAN DE ARAÚJO BARCELOS

Tese de submetida à
avaliação, como requisito parcial
para obtenção do grau de Doutor
em Ciência da Computação

Profa. Dra. Taisy Silva Weber
Orientadora

Porto Alegre, dezembro de 2001.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Barcelos, Patrícia Pitthan de Araújo

INFIMO – Um Toolkit para Experimentos de Intrusão de Injetores de Falhas / por Patrícia Pitthan de Araújo Barcelos. – Porto Alegre: PPGC da UFRGS, 2001.
161 p.:il.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2001. Orientadora: Weber, Taisy Silva.

1. Tolerância a Falhas. 2. Injeção de Falhas. 3. Tempo Real. 4. Protocolos. 5. Intrusão I. Weber, Taisy Silva. II. Título.

Dedico este trabalho àqueles que perdi durante esta trajetória:
meus avós, meu filho e meu melhor amigo.

Agradecimentos

Não poderia deixar de iniciar os agradecimentos fazendo menção à dedicatória. Agradeço aos que dedico: meu avô, que sempre apostou em mim; minha avó que vibrava a cada conquista; meu filho, por quem eu abriria mão de tudo isso; e meu melhor amigo, companheiro das melhores e piores horas. Obrigada por vocês terem estado comigo pelo tempo que nos foi permitido.

Sem palavras para agradecer as duas pessoas mais importantes da minha vida: Iára Pitthan e Eduardo Barcelos. À Iára, minha mãe, que derruba o ditado popular: "mãe é tudo igual, só muda de endereço". Isso até pode ser verdade para as outras mães, mas a minha é diferente, supera qualquer expectativa. Ao Eduardo, meu marido, que acredita mais em mim do que eu mesma sou capaz, que sempre me garante que o esforço será recompensado.

Meu muito obrigada a Taisy Silva Weber, minha orientadora, amiga, "segunda mãe", psicóloga, ... sempre contribuindo com seus sábios conselhos intelectuais e pessoais.

Ao pessoal do Grupo de Tolerância a Falhas da UFRGS, dentre os quais destaco meu bolsista Roberto Jung Drebes.

Aos professores Maria Lúcia Lisbôa, Ingrid Pôrto e Rômulo Oliveira, cujo apoio me foi de grande importância.

Às amigas Andréa Charão, Adriana Beiler e Márcia Pasin, pela força e incentivo.

Aos amigos Antônio Marinho Pilla Barcellos, Raul Ceretta Nunes e João Carlos Cunha (da Universidade de Coimbra – Portugal), pelo tempo dispendido na discussão de assuntos relacionados ao meu trabalho.

Aos colegas Maurício Pilla, Hércules Prado e Ronaldo Mello.

Ao pessoal dos bastidores do PGCC da UFRGS: Eliane, Ida, Henrique, Elisiane, Margareth, Lourdinha, Maria do Carmo, dentre outros.

Um agradecimento especial ao meu grande amigo Luiz Otávio Soares.

Ao CNPq pelo $uporte financeiro, reafirmando o nosso lema de que "a pesquisa é o meio e não o fim".

Sumário

Lista de Abreviaturas

ACK	Acknowledgement
ADC	Ambiente para Avaliação e Experimentação de Protocolos de Difusão Confiável
CPU	Central Processing Unit
ComFIRM	Communication Fault Injection through OS Resources Modification
DOCTOR	IntegrateD sOftware fault injeCTiOn enviRonment
FERRARI	Fault and ERRor Automatic Real-time Injector
FIAT	Fault-Injection-Based Automated Testing
FIDe	Fault Injection via Debugging
FIESTA	Fault Injection for Embedded System Target Application
FINE	Fault Injection moNitoring Environment
FIRE	Fault Injection using a Reflexive Architecture
HetNOS	Heterogeneous Network Operating System
IF	Injeção de Falhas
INFIMO	INtrusiveless Fault Injector MOdule
INFIMO_DBG	INtrusiveless Fault Injector MOdule using DeBuG resources
INFIMO_LIB	INtrusiveless Fault Injector MOdule by LIBrary modifications
INFIMO_TAP	INtrusiveless Fault Injector MOdule TArget Protocol
JRTPLIB	Jori's RTP Library
MBTF	Mean Time Between Failure
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
PC	Personal Computer
PCB	Process Control Block
RFC	Request For Comments
RT	Real Time
RTCP	Realtime Transport Control Protocol
RTP	Realtime Transport Protocol
SO	Sistema Operacional
SWIFI	Software-Implemented Fault Injection
TCP	Transport Control Protocol

TCP/IP	Transport Control Protocol/Internet Protocol
TF	Tolerância a Falhas
UDP	User Datagram Protocol
UFRGS	Universidade Federal do Rio Grande do Sul
ULA	Unidade Lógica e Aritmética

Lista de Figuras

Lista de Tabelas

Resumo

Técnicas de tolerância a falhas visam a aumentar a dependabilidade dos sistemas nos quais são empregadas. Entretanto, há necessidade de garantir a confiança na capacidade do sistema em fornecer o serviço especificado. A validação possui como objetivo propiciar essa garantia. Uma técnica de validação bastante utilizada é a injeção de falhas, que consiste na introdução controlada de falhas no sistema para observar seu comportamento.

A técnica de injeção de falhas acelera a ocorrência de falhas em um sistema. Com isso, ao invés de esperar pela ocorrência espontânea das falhas, pode-se introduzi-las intencionalmente, controlando o tipo, a localização, o disparo e a duração das falhas. Injeção de falhas pode ser implementada por *hardware*, *software* ou simulação. Neste trabalho são enfocadas técnicas de injeção de falhas por *software*, desenvolvidas nos níveis da aplicação e do sistema operacional.

O trabalho apresenta o problema da validação, através da injeção de falhas, de um protocolo de troca de pacotes. Enfoque especial é dado ao impacto resultante da inclusão de um módulo extra no protocolo, uma vez que o mesmo apresenta restrições temporais. O trabalho investiga alternativas de implementação de injetores de falhas por *software* que minimizem este impacto. Tais alternativas referem-se a localização do injetor de falhas no sistema, a forma de ativação das atividades do injetor de falhas e a operação de injeção de falhas em si.

Um *toolkit* para experimentos de intrusão da injeção de falhas é apresentado. O alvo da injeção de falhas é um protocolo com característica tempo real. O *toolkit* desenvolvido, denominado INFIMO (***IN****trusiveless* ***F****ault* ***I****njector* ***MO****dule*), visa a analisar, de forma experimental, a intrusão do injetor de falhas sobre o protocolo alvo.

O INFIMO preocupa-se com protocolos com restrições temporais por esses constituírem um desafio sob o ponto de vista de injeção de falhas. O INFIMO suporta falhas de comunicação, as quais podem ocasionar a omissão de alguns pacotes. O INFIMO apresenta duas ferramentas de injeção de falhas: INFIMO_LIB, implementada no nível da aplicação e INFIMO_DBG implementada com auxílio de recursos do sistema operacional.

Destacam-se ainda como contribuições do INFIMO a definição e a implementação do protocolo alvo para experimentos de injeção de falhas, o protocolo INFIMO_TAP. Além disso, o INFIMO apresenta métricas para avaliação da intrusão provocada pelo injetor de falhas no protocolo alvo.

Palavras-Chave: Tolerância a Falhas, Injeção de Falhas, Tempo Real, Protocolos, Intrusão.

TITLE: “INFIMO – A TOOLKIT TO INTRUSION EXPERIMENTS OF FAULT INJECTORS”

Abstract

Fault-tolerant techniques aim to increase the dependability of the systems in which they are used. Therefore, it is necessary to guarantee confidence in system capacity to provide the specified service. The validation goal is to provide this guarantee. One frequently used validation technique is fault injection, which consists on the introduction of controlled faults in the system to observe their behavior.

The fault injection technique speeds up the occurrence of faults in a system. Herewith, instead of waiting for the voluntary occurrence of faults, we can intentionally introduce them, controlling their type, location, trigger and duration. Fault injection can be hardware implemented, software implemented or simulated. In this work, software fault injection techniques were focused and developed in both application and operating system levels.

The work presents the problem of validation, through fault injection, of a packet exchange protocol. Special attention is given to the impact of an extra module included on the protocol under test, since the protocol presents temporal constraints. The work analyzes implementation alternatives to software-based fault injectors that minimize this impact. Such alternatives concern on the fault injector location on the system, the fault injector tasks activation and the fault injection operation itself.

A toolkit for intrusion experiments of fault injection is presented. The target of the fault injection is a protocol with real time features. The developed toolkit, called INFIMO (***IN****trusiveless* ***F****ault* ***I****njector* ***MO****dule*), aims to analyzes the fault injector intrusion over the target protocol.

The INFIMO is concerned with protocols with temporal constraints because they are a challenge from the viewpoint of fault injection. INFIMO supports communication faults, when some packets can be omitted. INFIMO presents two fault injection tools: INFIMO_LIB, implemented at the application level and INFIMO_DBG, implemented with operating system resources.

We can also detach as INFIMO contributions, the definition and the implementation of the target protocol used to fault injection experiments, the INFIMO_TAP protocol. Besides this, INFIMO presents metrics to evaluate the intrusion imposed by the fault injectors on the target protocol.

Keywords: Fault Tolerance, Fault Injection, Real Time, Protocols, Intrusion.

1 Introdução

Este Capítulo é dividido em seis Seções. A Seção 1.1 apresenta o contexto no qual se insere este trabalho no âmbito de sistemas de computação. A motivação para o desenvolvimento do trabalho e seus objetivos são descritos nas Seções 1.2 e 1.3, respectivamente. A Seção 1.4 apresenta a metodologia de desenvolvimento. Os resultados alcançados são descritos na Seção 1.5. A organização deste volume é apresentada na Seção 1.6.

1.1 Contexto

Um sistema tolerante a falhas caracteriza-se pela capacidade de fornecer, de forma contínua, o serviço especificado, mesmo na ocorrência de falhas. No desenvolvimento de sistemas tolerantes a falhas, surgem problemas relacionados à validação dos mesmos, uma vez que, além das funcionalidades normais, os mecanismos de tolerância a falhas devem ser considerados [ARL90].

A validação é uma atividade que visa garantir a confiança na capacidade do sistema em fornecer o serviço especificado. Validação é o principal objetivo da injeção de falhas, a qual consiste em introduzir falhas no sistema de maneira controlada, a fim de observar o seu comportamento. Injeção de falhas permite que sejam fornecidas ao sistema tolerante a falhas as entradas para as quais eles foram construídos para tolerar: as falhas [LAP85].

Entretanto, a validação em sistemas tempo real se depara com limitações temporais inerentes a estes sistemas. Por representar uma carga extra no sistema, o injetor de falhas pode alterar o tempo de execução das tarefas comprometendo, desta forma, suas restrições temporais. Neste sentido, características como rigidez nos limites de tempo e sincronismo, peculiares aos sistemas tempo real, surgem como limitadores no uso de algumas técnicas de tolerância a falhas já amplamente utilizadas. Portanto, faz-se necessário uma análise destas características, das técnicas de tolerância a falhas utilizadas e, principalmente, das limitações impostas pelos sistemas tempo real na aplicação destas técnicas.

Sistemas tempo real diferem dos sistemas convencionais pelo fato de apresentarem restrições de tempo e tratarem, em muitos casos, com situações críticas. Assim, qualquer tipo de falha, inclusive falha temporal, pode causar conseqüências irreparáveis. Então, ao contrário da maioria dos sistemas onde há distinção entre correção e desempenho, em sistemas tempo real, tais requisitos estão fortemente relacionados [SHI94].

Este trabalho discute o problema da validação da comunicação de um protocolo com característica tempo real e apresenta abordagens para a implementação de injetores de falhas por *software*, que minimizem o problema de alteração da temporização do sistema devido a sobrecarga provocada pelo injetor (*overhead*). O trabalho propõe dois injetores de falhas construídos visando a analisar esta sobrecarga, referenciada ao longo do texto como intrusão. Além da especificação e implementação de injetores de falhas, o trabalho apresenta a definição e implementação do protocolo utilizado como alvo dos

experimentos de injeção de falhas. O trabalho descreve ainda métricas para avaliação da intrusão imposta pela presença de um injetor de falhas no protocolo alvo.

1.2 Motivação

Um sistema tempo real é um sistema cujo progresso é especificado em termos de exigências temporais ditadas pelo ambiente. Em sistemas tempo real a correção da computação é definida tanto em relação a obtenção de resultados como no tempo em que esses resultados são produzidos. Logo, esses sistemas possuem a capacidade de executar ações dentro de intervalos de tempo pré-especificados [VER2001].

Uma característica quase sempre presente em sistemas tempo real é a previsibilidade, que indica a capacidade de se prever uma violação de garantia ou uma incorreção temporal. Esta característica possibilita um tratamento de exceções que minimize as conseqüências de uma falha, desde que, obviamente, esta previsão aconteça em tempo hábil [RAM94].

Atualmente, um dos principais objetivos das pesquisas de tolerância a falhas é eficiência. No entanto, isso implica em aumento de complexidade. Como falhas de projeto podem ter conseqüências fatais durante a execução, deve-se submeter os algoritmos à verificação e/ou ao teste. Já que a verificação, a qual prova a correção, usualmente é aplicável em sistemas pequenos, o teste assume um papel imprescindível, e pode ser implementado através de ferramentas de injeção de falhas.

Os dois principais objetivos de injeção de falhas são a validação, que pode ser vista como um meio de testar os mecanismos de tolerância a falhas através da introdução de falhas, e o auxílio ao projeto, pois os resultados (negativos) da injeção de falhas podem ser usados para iniciar laços de realimentação melhorando os procedimentos de teste e os mecanismos de tolerância a falhas [ARL90].

O interesse por injeção de falhas surgiu com a implementação, por parte do grupo de tolerância a falhas da UFRGS, de mecanismos de tolerância a falhas em sistemas distribuídos convencionais: o ADC [BAR95] e o FIX [TEI95]. O ADC aborda comunicação confiável, enquanto o FIX trata de recuperação de processos.

O ADC [BAR95] roda em um ambiente simulado sob controle de um módulo central. A injeção de falhas é implementada através desse módulo, de acordo com uma probabilidade definida por um modelo de entrada. Este procedimento de injeção de falhas, classificado como injeção de falhas por simulação, corresponde a uma solução específica para a implementação em questão. Para o FIX [TEI95] foi previsto um módulo de injeção de falhas acionado diretamente a partir de chamadas de sistema do Linux. Esse módulo fornece primitivas que podem ser usadas para a construção de ferramentas de injeção de falhas em protocolos de recuperação de processos.

O interesse em injeção de falhas na comunicação em ambiente tempo real se dá pela crescente utilização de tais sistemas nas mais diversas áreas. A maioria dos injetores de falhas propostos visam introdução de falhas em sistemas convencionais, ou seja, não se preocupam com limitações temporais. Pode-se citar como exemplos de injetores de falhas: FIAT [SEG88], FERRARI [KAN95], FINE [KAO93], DEFINE [KAO94], DOCTOR [HAN95], FIRE [ROS98b], Xception [CAR98], Orchestra

[DAW96], Fiesta [KRI98], dentre outros. Recentes pesquisas apontam uma versão tempo real do injetor de falhas Xception, denominada RT-Xception [CUN2000]. Características relevantes de tais ferramentas encontram-se descritas no Anexo.

Em sistemas tempo real, onde as atividades por eles controladas devem ocorrer em instantes de tempo relativos a uma base de tempo externa ao sistema, a validação por injeção de falhas se torna mais crítica. As restrições temporais impostas pelos sistemas tempo real aliadas à sobrecarga provocada pelo injetor justificam este fato.

1.3 Objetivos

O trabalho investiga a viabilidade da validação, através de injeção de falhas por *software*, dos aspectos de tolerância a falhas de um protocolo com restrições temporais. O protocolo, referenciado ao longo do texto como protocolo alvo, implementa a troca de pacotes e apresenta como mecanismo de tolerância a falhas a detecção do *timeout*. O objetivo da injeção de falhas é validar o mecanismo de tolerância a falhas do protocolo, ou seja, o mecanismo de detecção do *timeout*.

Um *toolkit* de injeção de falhas para validação da dependabilidade de um protocolo com restrições temporais é apresentado. O *toolkit*, denominado INFIMO (***IN****trusiveless* ***F****ault* ***I****njector* ***Mo****dule*), consiste em um conjunto de ferramentas com objetivos comuns. Enfoque especial é dado ao impacto resultante da inclusão de um módulo extra no protocolo alvo, o módulo que implementa atividades de injeção de falhas. Por módulo extra entende-se uma rotina, um trecho de código, ou um programa, que implique em tempo de processamento adicional.

O *toolkit* é composto pelo protocolo alvo, chamado INFIMO_TAP (***IN****trusiveless* ***F****ault* ***I****njector* ***MO****dule* ***TA****rget* ***P****rotocol*) e dois injetores de falhas, denominados INFIMO_LIB (***IN****trusiveless* ***F****ault* ***I****njector* ***MO****dule by* ***LIB****rary modifications*) e INFIMO_DBG (***IN****trusiveless* ***F****ault* ***I****njector* ***MO****dule using* ***DeBuG*** *resources*). O INFIMO_LIB é implementado através de alterações nas funções da biblioteca sob a qual o INFIMO_TAP foi desenvolvido. Já o INFIMO_DBG utiliza os recursos de depuração do sistema operacional para efetuar a injeção de falhas. O objetivo do INFIMO *toolkit* é a condução de experimentos de injeção de falhas visando a analisar a intrusão dos injetores de falhas sobre o protocolo alvo. O objetivo das ferramentas de injeção de falhas do INFIMO é validar o mecanismo de detecção do *timeout* proposto pelo protocolo alvo.

A definição e a implementação de dois injetores de falhas visa ao estabelecimento de comparações entre os mesmos, no que se refere a cobertura de falhas, intrusão temporal e espacial, portabilidade e expansão para diferentes tipos de falhas. Além de comparações entre as próprias ferramentas de injeção de falhas do INFIMO, o trabalho estabelece comparações entre essas ferramentas e algumas das ferramentas de injeção de falhas disponíveis na literatura.

A comparação entre as ferramentas de injeção de falhas do INFIMO é feita experimentalmente. Já a comparação entre as ferramentas de injeção de falhas do INFIMO e algumas ferramentas propostas na literatura é feita com base nas publicações dessas ferramentas.

O trabalho preocupa-se ainda em analisar a viabilidade da implementação de ferramentas de injeção de falhas com mínimo impacto nas características temporais do protocolo alvo. Para tanto, definições de métricas para avaliação da intrusão são apresentadas.

1.4 Metodologia

O desenvolvimento deste trabalho foi dividido em cinco etapas, a saber: investigação de alternativas de implementação de injetores de falhas, especificação e implementação do protocolo alvo, implementação dos injetores de falhas, condução de experimentos e análise dos experimentos. Tais etapas são descritas a seguir.

(a) Investigação de alternativas de implementação de injetores de falhas

Por alternativas de implementação entendem-se as possíveis abordagens sob as quais as ferramentas de injeção de falhas podem ser desenvolvidas. Esta etapa consiste na investigação dos níveis nos quais o injetor de falhas pode estar localizado no sistema. Pode-se citar como exemplos dessas abordagens a inclusão injetor de falhas junto ao nível do sistema operacional. Para esta abordagem pode-se contar com recursos do depurador e chamadas de sistema (*system calls*). Além destas, a possibilidade de introdução do injetor de falhas no nível da aplicação também faz parte da investigação. Neste contexto, o injetor pode ser incorporado ao código da aplicação, implementado através de processos concorrentes ou através de *threads*.

A importância desta etapa está na avaliação da intrusão imposta pelas rotinas ou bibliotecas que implementam as atividades de injeção de falhas no protocolo. O objetivo da investigação é buscar alternativas que impliquem na injeção controlada de falhas visando a minimizar o impacto do injetor nas restrições temporais do protocolo alvo.

(b) Especificação e implementação do protocolo alvo

A dificuldade de dispor de códigos de protocolos com característica tempo real conduziu à implementação de um protocolo alvo para os experimentos de injeção de falhas. O objetivo do protocolo é viabilizar os experimentos e a análise das abordagens de injeção de falhas propostas. Portanto, foi definido e implementado um protocolo especificamente para suprir às necessidades das ferramentas de injeção de falhas do INFIMO *toolkit*. Para a implementação do protocolo alvo foi utilizada a biblioteca JRTPLIB (*Jori's RTP Library*) [JRT99], uma biblioteca orientada a objetos implementada sobre o protocolo RTP (*Realtime Transport Protocol*) [SCH97].

(c) Implementação dos injetores de falhas

Esta etapa consiste na implementação das ferramentas de injeção de falhas que compõem *toolkit*, de acordo com as abordagens investigadas (etapa (a)). A etapa envolve o projeto e a implementação de duas ferramentas de injeção de falhas, sendo uma delas implementada no nível da aplicação e a outra utilizando recursos do sistema operacional. O objetivo da implementação de duas ferramentas é a obtenção de uma base comparativa da intrusão provocada pelo injetor de falhas no protocolo alvo.

Para cada ferramenta existem requisitos bem específicos. A primeira ferramenta, a qual implementa injeção de falhas por alteração de bibliotecas da aplicação, exige conhecimento das funcionalidades da biblioteca JRTPLIB, utilizada como base para implementação do protocolo alvo. A segunda ferramenta, que implementa injeção de falhas através de recursos de depuração, faz uso do *ptrace( )* exigindo, portanto, familiarização com tal mecanismo.

Ainda nesta etapa são definidas as métricas utilizadas na análise das ferramentas de injeção de falhas desenvolvidas.

(d) Condução de experimentos

O objetivo desta etapa é conduzir experimentos que identifiquem ou não a viabilidade de utilização de injeção de falhas para validação de protocolos com restrições temporais. Neste sentido, os experimentos baseiam-se em exaustivas execuções do protocolo alvo. Fazem parte do conjunto de atividades dos experimentos:

- a especificação do modelo de falhas a ser validado;
- a coordenação das atividades de injeção de falhas;
- a injeção efetiva das falhas no protocolo alvo;
- a coleta de dados relacionados à condução do experimento;
- a análise dos dados coletados.

As execuções dos experimentos são conduzidas sem a presença do injetor de falhas, com a presença, porém sem a atuação do injetor de falhas, e finalmente com a presença e a atuação do injetor. Tais execuções viabilizam o estabelecimento de comparações e a análise das ferramentas.

(e) Análise dos experimentos

A etapa de análise dos experimentos é composta por um conjunto de comparações. Essas comparações não são automatizadas e dividem-se em duas categorias. Num primeiro momento são analisadas as ferramentas de injeção de falhas do INFIMO entre si. Esta análise considera a intrusão dos injetores de falhas sobre o protocolo alvo. Além disso, avalia-se a cobertura de falhas proporcionada pelo mecanismo de tolerância a falhas do protocolo alvo. Aspectos como manutenção da semântica do protocolo alvo e viabilidade da injeção de falhas para validação de outros protocolos com característica tempo real também são alvo de análise. A análise não consiste em uma atividade automatizada e baseia-se nos experimentos conduzidos na etapa anterior (etapa (d)).

A outra forma de comparação envolve tanto as ferramentas de injeção de falhas do INFIMO como algumas das ferramentas de injeção de falhas propostas na literatura. As comparações consideram a plataforma utilizada pela ferramenta e o mecanismo de injeção de falhas implementado. Adicionalmente são analisados os objetivos dos experimentos de injeção de falhas, o modelo de falhas e os resultados obtidos com os experimentos. Tais comparações tomam como base as informações disponibilizadas nas publicações das ferramentas.

1.5 Resultados Alcançados

O trabalho conclui pela viabilidade de aplicar a injeção de falhas na validação da dependabilidade de protocolos com restrições temporais. O desenvolvimento do INFIMO apresenta ainda como principais contribuições:

- INFIMO_TAP: o projeto e a implementação de um protocolo de troca de pacotes, o qual possui um mecanismo de tolerância a falhas incorporado e apresenta característica tempo real;
- INFIMO_LIB: o projeto e a implementação de um injetor de falhas através da alteração de funções da biblioteca da aplicação (biblioteca JRTPLIB);
- INFIMO_DBG: o projeto e a implementação de um injetor de falhas com auxílio de recursos de depuração do sistema operacional;
- o estabelecimento de métricas para avaliação de injetores de falhas;
- a possibilidade de análise da intrusão de injetores de falhas;
- o suporte para aplicações com restrições temporais.

Além disso, o projeto e implementação do INFIMO tem conduzido e incentivado o desenvolvimento de diversos outros trabalhos na área de injeção de falhas. O grupo de trabalho de tolerância a falhas da UFRGS tem abordado:

- injeção de falhas através de alteração de rotinas do *kernel* do sistema operacional, através da ferramenta ComFIRM [BAR99a];
- injeção de falhas com o uso de recursos de depuração, implementado pela ferramenta FiDE [GON2000];
- injeção de falhas voltada para a validação de um banco de dados [MAN2001].

Dentre as publicações geradas ao longo do desenvolvimento deste trabalho destacam-se principalmente:

BARCELOS, P. P. A.; WEBER, T. S. Validação de Protocolos de Comunicação de Grupo em Sistemas Distribuídos Tempo Real por Injeção de Falhas. XVI SBRC e I WTR (Workshop em Sistemas de Tempo Real), Rio de Janeiro – RJ, Maio, 1998.

BARCELOS, P. P. A.; WEBER, T. S. Usando recursos de escalonamento para ativação de injetores de falhas em protocolos com restrições temporais. XVII SBRC e II WTR (Workshop em Sistemas de Tempo Real), Salvador – BA, Maio, 1999.

BARCELOS, P. P. A.; LEITE, F. O., WEBER, T. S. Implementação de um Injetor de Falhas de Comunicação. VIII SCTF (Simpósio de Computadores Tolerantes a Falhas), SBC, Campinas – SP, Julho, 1999.

BARCELOS, P. P. A.; LEITE, F. O., WEBER, T. S. Building a Fault Injector to Validate Fault Tolerant Communication Protocols. PCS'99 (Parallel Computing Symposium), Ensenada, Baja California, México, Agosto, 1999.

BARCELOS, P. P. A.; WEBER, T. S. INFIMO: Um Injetor de Falhas de Comunicação para Aplicações RT-Linux. I WSL (Workshop sobre Software Livre), SBC, Porto Alegre – RS, Maio, 2000.

BARCELOS, P. P. A.; INFIMO: Projeto e Implementação de um Toolkit para Experimentação da Injeção de Falhas. WTD (Workshop de Teses e Dissertações), SBC, Florianópolis - SC, Março, 2001.

1.6 Organização do Volume

Esta tese de doutorado encontra-se dividida em dez Capítulos. O primeiro Capítulo apresenta uma introdução sobre o trabalho, na qual foram contextualizados os assuntos de relevância para o desenvolvimento da mesma. Neste Capítulo ainda é descrita a motivação para o desenvolvimento da tese, assim como os objetivos da mesma. A metodologia e os resultados alcançados também são enfocados no primeiro Capítulo.

O segundo Capítulo descreve a terminologia adotada neste trabalho no que se refere a tolerância a falhas e a sistemas tempo real. Esse Capítulo situa o escopo de aplicações do assunto desenvolvido no âmbito de tolerância a falhas.

O terceiro Capítulo introduz conceitos relacionados à validação por injeção de falhas. São apresentados os objetivos da injeção de falhas e uma arquitetura genérica para construção de ferramentas de injeção de falhas. O Capítulo descreve ainda uma classificação das técnicas de injeção de falhas, dando ênfase à injeção de falhas por *software*. Neste contexto, são apresentadas as características da injeção de falhas por *software*. A preocupação com intrusão em ambientes com característica tempo real é discutida.

O quarto Capítulo descreve as vantagens e desvantagens da injeção de falhas por *software.* O Capítulo apresenta métodos de injeção de falhas por *software* com base no tempo da injeção de falhas. O Capítulo investiga abordagens genéricas para o desenvolvimento de injetores de falhas por *software*, as quais compreendem a implementação no meta nível, no nível da aplicação e no nível do sistema operacional. As formas de implementação, características, vantagens e desvantagens de tais abordagens também são investigadas. Ao final do Capítulo são esboçados alguns comentários relacionados às abordagens investigadas.

Do quinto ao oitavo Capítulo é apresentado o INFIMO, o *toolkit* para experimentos de injeção de falhas para aplicações tempo real. A apresentação do INFIMO se dá conforme descrito a seguir.

O quinto Capítulo descreve características e idéias relevantes ao projeto do INFIMO. Inicialmente é apresentado o INFIMO *toolkit.* A seguir são descritas as principais características das ferramentas do INFIMO. Com base na arquitetura genérica para construção de injetores de falhas, é apresentada a arquitetura das ferramentas do INFIMO. É exposto o modelo de falhas suportado pelo INFIMO, assim como a

definição das métricas utilizadas para avaliação de suas ferramentas de injeção de falhas. O Capítulo exibe ainda aspectos relacionados aos experimentos conduzidos com o *toolkit* e o método de medição da intrusão temporal.

O sexto Capítulo descreve o protocolo alvo dos experimentos de injeção de falhas, o protocolo INFIMO_TAP. Nesta descrição é apresentada a biblioteca sobre a qual o protocolo foi desenvolvido. O Capítulo apresenta ainda o funcionamento do protocolo INFIMO_TAP, assim como a descrição detalhada da implementação do mesmo. Alguns experimentos realizados com o protocolo também são apresentados.

O sétimo Capítulo apresenta o injetor de falhas INFIMO_LIB. No Capítulo é descrito seu funcionamento, de acordo com as abordagens investigadas. São enfocados ainda os aspectos de implementação do injetor de falhas e os resultados de alguns experimentos realizados com o mesmo.

O oitavo Capítulo descreve o injetor de falhas INFIMO_DBG. O Capítulo apresenta seu funcionamento e explica os detalhes de sua implementação. Analogamente ao Capítulo anterior, alguns resultados experimentais do injetor de falhas são esboçados.

O nono Capítulo estabelece diversas comparações. As primeiras comparações relacionam as ferramentas de injeção de falhas do INFIMO entre si, ou seja, são comparadas as ferramentas INFIMO_LIB e INFIMO_DBG. A seguir, são estabelecidas comparações entre as ferramentas de injeção de falhas do INFIMO e algumas das ferramentas de injeção de falhas propostas na literatura. O Capítulo esboça estas comparações através de tabelas, as quais relacionam aspectos como os mecanismos de injeção de falhas, os modelos de falhas, os objetivos da validação, dentre outros.

O décimo Capítulo expõe as conclusões a que se chegou com o desenvolvimento deste trabalho. O Capítulo apresenta os aspectos positivos e negativos do INFIMO, bem como sua contribuição efetiva. Algumas otimizações para o INFIMO são apresentadas, assim como trabalhos futuros relacionados ao assunto desenvolvido nesta tese de doutorado.

O volume conta com um Anexo, o qual descreve o estado da arte em injetores de falhas. Esta descrição originou-se no estudo desenvolvido como base para o projeto do trabalho.

2 Tolerância a Falhas e Tempo Real

Este Capítulo descreve a terminologia adotada no decorrer deste trabalho. As Seções 2.1 e 2.2 enfocam aspectos de tolerância a falhas, abordando falhas e dependabilidade, respectivamente. A Seção 2.3 trata de sistemas tempo real. O objetivo do Capítulo é situar o escopo da tese de doutorado no âmbito de tolerância a falhas e tempo real.

2.1 Falhas

Os termos falha (*fault*), erro (*error*) e defeito (*failure*) têm sido usados de diferentes maneiras, em diferentes contextos, sendo que em alguns casos são intercambiáveis. Embora, na língua portuguesa, a terminologia de tolerância a falhas não tenha estabelecido definições padronizadas para os conceitos relativos a área, este trabalho segue as conceituações propostas por Laprie [LAP85], Jalote [JAL94] e Arlat [ARL90].

Uma falha é uma condição física anômala. As causas podem ser erros de projeto, tais como equívocos na especificação do sistema ou em sua implementação; problemas de fabricação; fadiga; acidentes ou deterioração; ou ainda distúrbios externos, como por exemplo condições ambientais nocivas, interferência eletromagnética, radiação iônica, entradas imprevistas ou mau uso do sistema. As falhas que resultam de erros de projeto e de fatores externos são especificamente difíceis de modelar; delas também é difícil proteger-se, devido à imprevisibilidade de ocorrência e efeitos.

Um erro é a manifestação de uma falha no sistema, no qual o estado lógico de um elemento difere do valor previsto. Uma falha existente no sistema não necessariamente resulta em um erro. O erro ocorre apenas quando a falha é sensibilizada, isto é, para um determinado estado do sistema e para um conjunto de entradas, resulta um próximo estado ou conjunto de saídas incorreto – uma ou outra combinação de entradas a partir do mesmo estado poderia não resultar em erro. Assim, uma falha é considerada latente quando ela ainda não foi sensibilizada pelo sistema.

O defeito corresponde à incapacidade de algum componente (de *software* ou *hardware*) realizar a função para a qual foi projetado. Os defeitos são causados pela existência de falhas e, consequentemente, erros no sistema.

A Figura 2.1 [LAP85] apresenta os universos nos quais estão inseridos os conceitos de falha, erro e defeito, de acordo com a descrição anterior. Para Laprie, a falha corresponde ao universo físico, enquanto a percepção da falha, representada pelo erro, corresponde ao universo da informação.

A injeção de falhas por *software* atua no universo da informação. A injeção de falhas consiste na emulação de falhas através da introdução de erros. O comportamento de injetores de falhas é observado quando da ocorrência de defeitos. Neste caso, pode-se afirmar que as técnicas de tolerância a falhas aplicadas no sistema não estão apresentado a cobertura adequada. A cobertura, conforme explicado na Seção 2.2.2, pode estar relacionada à detecção, localização, contenção e/ou à recuperação da falha.

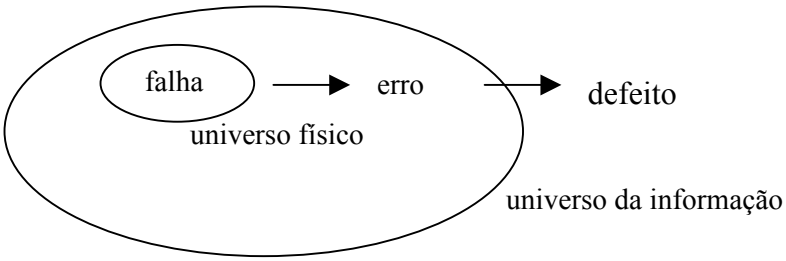

FIGURA 2.1 – Terminologia de falha, erro e defeito

Falhas de *hardware* que ocorrem durante a operação do sistema são classificadas principalmente pela duração. Falhas permanentes são causadas por defeitos de dispositivos irreversíveis em um componente devido a dano, fadiga ou manufatura imprópria. Uma vez ocorrida a falha permanente, o componente faltoso pode ser restaurado somente pela reposição ou, se possível, pelo reparo [CLA95].

Falhas transientes, por outro lado, são ocasionadas por distúrbios de ambiente tais como alterações de voltagem, interferência eletromagnética ou radiação. Esses eventos tipicamente têm uma duração curta retornando à operação normal, embora o estado do sistema possa continuar errôneo. Falhas transientes podem ser até 100 vezes mais freqüentes que falhas permanentes, dependendo do ambiente de operação do sistema. Falhas intermitentes, as quais tendem oscilar entre períodos de atividade errônea e dormência, podem permanecer durante a operação do sistema. Elas são geralmente atribuídas a erros de projeto que resultam em *hardware* instável [CLA95].

Falhas de *software* são causadas por especificação, projeto ou codificação incorreta de um programa. Embora *software* não falhe fisicamente após instalado em um computador, falhas latentes ou erros no código podem permanecer durante a operação. Isto pode ocorrer sob altas cargas de trabalho ou cargas não usuais para determinadas condições. Tais situações eventualmente conduzem o sistema a defeitos. Sendo assim, injeção de falhas por *software* é usada principalmente para teste de programas ou mecanismos de tolerância a falhas implementados por *software* [CLA95].

De acordo com Clark [CLA95], a Figura 2.2 ilustra alguns conceitos relacionados à terminologia de tolerância a falhas.

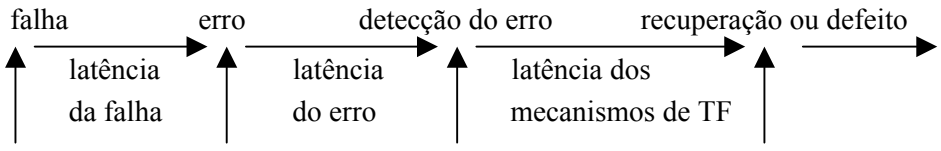

FIGURA 2.2 – Falha e conseqüências

Conforme mostra a Figura 2.2, quando uma falha causa uma alteração incorreta no estado da máquina, um erro ocorre. O tempo entre a ocorrência da falha e a primeira percepção de um erro é chamado latência da falha. Embora uma falha permaneça

localizada no código ou circuito afetado, múltiplos erros podem se originar da mesma e se propagar através do sistema. Se os mecanismos necessários estão presentes, a propagação do erro será detectada após um período de tempo denominado latência do erro. Quando os mecanismos de tolerância a falhas detectam um erro, diversas ações podem ser iniciadas para a manipulação da mesma, visando conter os erros. A recuperação ocorre se essas ações têm sucesso, caso contrário, o sistema encontra-se com defeito.

2.2 Dependabilidade

A segurança de funcionamento de um sistema, também denominada dependabilidade, é definida como a qualidade do serviço oferecido pelo mesmo conforme percebido pelos seus usuários [LAP85]. Para mensurar tal nível de segurança, empregam-se dois conceitos básicos: confiabilidade e disponibilidade.

Confiabilidade é a probabilidade de o sistema sobreviver sem falha em um intervalo de tempo, enquanto disponibilidade é a probabilidade de o sistema permanecer operacional (não falho) em um determinado instante de tempo [CLA95].

A dependabilidade de um sistema tolerante a falhas deve ser validada para garantir que sua redundância foi corretamente implementada e o sistema irá fornecer o nível desejado de serviço confiável. A introdução deliberada, ou intencional, de falhas em um sistema para determinar sua resposta visa a validar sua dependabilidade.

As Seções 2.2.1 e 2.2.2 apresentam medidas para avaliar a dependabilidade de um sistema. Enfoque especial é dado à cobertura de falhas, uma vez que corresponde a uma das métricas utilizadas pelo INFIMO.

A validação da dependabilidade de um sistema exige a utilização combinada dos seguintes métodos [LAP85]:

- prevenção de falhas ("*fault prevention*"): compreende métodos e técnicas que auxiliam na prevenção da ocorrência ou prevenção da introdução de falhas.

- tolerância a falhas ("*fault tolerance*"): corresponde aos métodos e técnicas que auxiliam no fornecimento de um serviço de acordo com a especificação, apesar da ocorrência de falhas.

- remoção de falhas ("*fault removal*"): consiste em métodos e técnicas que auxiliam na redução da presença, do número e da gravidade das falhas.

- previsão de falhas ("*fault forecasting*"): é composta pelos métodos e técnicas que auxiliam na estimativa do número de falhas presentes, da incidência futura das mesmas e de suas conseqüências.

Os métodos para alcançar dependabilidade anteriormente citados resultam de atividades humanas e podem ser imperfeitos. As imperfeições introduzem dependências entre os métodos e explicam porque sua utilização combinada pode conduzir a um

sistema com característica de dependabilidade. Essas dependências podem ser descritas da seguinte forma: apesar da prevenção de falhas através de regras de construção e metodologias de projeto, falhas são geradas. Tal fato justifica a necessidade da remoção de falhas [FUC96].

Remoção de falhas é por si imperfeita, assim como são imperfeitos os componentes de *hardware* e *software*, daí a importância da previsão de falhas. As falhas que permanecem nos sistemas operacionais exigem tolerância a falhas, a qual é baseada em regras de construção. Portanto, a remoção de falhas e a previsão de falhas são exigidas para implementação de mecanismos de tolerância a falhas. Este processo recursivo pode ser estendido às ferramentas e tecnologias utilizadas no projeto e implementação de sistemas de computação para assim apresentar dependabilidade [FUC96].

Existem diversas abordagens para avaliação da dependabilidade de um sistema. Testes ao longo da vida do sistema envolvem a monitoração do comportamento do mesmo até a ocorrência de um erro e o registro das causas que conduziram a este erro. O tempo necessário para se obter um número de falhas estatisticamente significante torna o teste impraticável para a maioria dos sistemas de computação [CLA95].

A abordagem de modelagem analítica, a qual necessita de informação extraída de análise experimental, é tipicamente utilizada para prognosticar dependabilidade. Entretanto, é uma abordagem difícil de ser desenvolvida uma vez que, na prática, os modelos podem se tornar muito complexos e sua simplificação pode reduzir a utilidade de seus resultados [CLA95].

A injeção de falhas tem demostrado ser uma técnica poderosa que permite a avaliação de um protótipo do sistema sob falhas. Com injeção de falhas é possível medir a eficiência das capacidades de detecção e correção de falhas do sistema.

Este trabalho concentra sua atenção na técnica de validação, por injeção de falhas, em ambientes com restrições temporais. Injeção de falhas consiste na introdução intencional de falhas no sistema de maneira controlada, a fim de observar seu comportamento. Para efeitos de simplificação e padronização com as demais pesquisas da área, este trabalho adota a terminologia de "injeção de falhas", embora, de acordo com a conceituação apresentada na Seção anterior, o termo mais adequado seja "injeção de erros".

A validação em sistemas tempo real é um desafio uma vez que as exigências de desempenho e confiabilidade devem ser simultaneamente consideradas. Além disso, tanto a arquitetura de *software* como a arquitetura de *hardware* apresentam alto grau de complexidade.

2.2.1 Avaliação da Dependabilidade

Existem diversas medidas para avaliação da dependabilidade de um sistema. A descrição apresentada neste trabalho, baseada em Pradhan [PRA96], cita as seguintes medidas: taxa de falhas, MTTF, MTTR, MTBF e cobertura de falhas. As quatro primeiras medidas são sucintamente apresentadas nesta Seção. Por ser de interesse ao

trabalho, a medida de cobertura de falhas é tratada em uma Seção separada (Seção 2.2.2).

Taxa de falhas

Corresponde ao número esperado de falhas de um componente ou sistema por um determinado período de tempo. Esta medida pode ser usada para comparar tais sistemas ou componentes.

MTTF (Mean Time To Failure)

Especifica a qualidade de um sistema. O MTTF é o tempo esperado que um sistema irá operar antes que a primeira falha ocorra.

MTTR (Mean Time To Repair)

É o tempo médio exigido para reparar um sistema. É extremamente difícil de estimar e é geralmente determinado de forma experimental através da injeção de um conjunto de falhas, uma por vez, em um sistema e posterior coleta do tempo exigido para o reparo do sistema em cada caso. A média desses tempos consiste no MTTR.

MTBF (Mean Time Between Failure)

Corresponde ao tempo médio entre falhas de um sistema. Esta medida é frequentemente confundida com o MTTF, o qual especifica o tempo médio até a primeira falha. O MTBF é calculado pelo tempo médio entre falhas, incluindo qualquer tempo para reparar o sistema e retorná-lo ao estado operacional.

2.2.2 Cobertura de Falhas

De acordo com Pradhan [PRA96], há diversos tipos de cobertura de falhas, dependendo se o projeto está relacionado com detecção, localização, medição ou recuperação da falha. Além disso, há duas definições principais de cobertura de falhas. A primeira definição afirma que cobertura é uma medida da habilidade do sistema em realizar a detecção, localização, contenção e/ou recuperação da falha. Sendo assim, cobertura de detecção da falha é uma medida da habilidade do sistema em detectar falhas. Por exemplo, uma exigência do sistema pode ser que uma determinada parte de todas as falhas sejam detectadas. A cobertura de detecção da falha é a medida da capacidade do sistema obedecer tal exigência.

Cobertura de localização da falha é uma medida da habilidade do sistema em localizar falhas. Outra vez é muito comum exigir que um sistema localize falhas dentro de módulos facilmente substituíveis. A cobertura de localização da falha é uma medida do sucesso com o qual esta localização é realizada. De forma análoga, cobertura de contenção da falha é uma medida da habilidade do sistema em conter falhas, enquanto cobertura de recuperação da falha mede a habilidade do sistema em recuperar-se das falhas e manter um estado operacional. Uma alta cobertura de recuperação de falha exige alta cobertura de detecção, localização e contenção.

A segunda definição de cobertura de falhas corresponde a uma definição matemática. Esta definição diz que a cobertura de falhas é uma probabilidade condicional que, dada a existência de uma falha, o sistema se recupera. O problema fundamental com cobertura de falhas é a dificuldade de cálculo da mesma. Provavelmente a abordagem mais comum para estimar cobertura de falhas é desenvolver uma lista de todas as falhas que podem ocorrer em um sistema e formar, a partir desta lista, uma lista de falhas que podem ser detectadas, uma lista de falhas que podem ser localizadas, uma lista de falhas que podem ser contidas e uma lista de falhas das quais o sistema pode recuperar-se. O fator de cobertura de detecção da falha, por exemplo, é então calculado como a simples porção de falhas que podem ser detectadas, ou seja, o número de falhas detectadas dividido pelo número total de falhas. Os demais fatores de cobertura de falhas podem ser obtidos de forma similar.

Segundo Siewiorek [SIE98], o conceito de cobertura oferece a visão da confiabilidade exigida quando se discutem técnicas de detecção. Siewiorek usa duas medidas de cobertura. A primeira medida, denominada cobertura geral, é mais qualitativa. Usualmente, cobertura geral especifica as classes de falhas as quais são detectáveis e podem incluir percentuais de detecção de falha para diferentes modelos de falhas.

A segunda medida de cobertura, chamada cobertura explícita, é a probabilidade de que uma falha (qualquer falha) seja detectada. Esta cobertura pode ser determinada a partir das especificações de cobertura geral através do uso de médias das coberturas para todas as possíveis classes de falhas, comparadas com a probabilidade da ocorrência de cada classe de falha. Assim, esta segunda cobertura é mais difícil de ser obtida, já que as probabilidades relacionadas são dependentes de implementação e realmente podem não ser conhecidas. Em muitas situações, suposições de simplificação são usadas para os possíveis modos de falha e probabilidades.

O mecanismo de tolerância a falhas a ser validado pelas ferramentas de injeção de falhas do INFIMO é o mecanismo de detecção do *timeout*. Assim, a cobertura enfocada pelo INFIMO consiste na cobertura de detecção que, conforme Pradhan [PRA96] é uma medida da habilidade do sistema em detectar falhas.

2.3 Tempo Real

Segundo Veríssimo [VER2001], um sistema tempo real é um sistema cujo progresso é especificado em termos de exigências temporais ditadas pelo ambiente. Em sistemas tempo real a correção da computação é definida tanto em relação à obtenção de resultados como no tempo em que esss resultados são produzidos. Assim, esses sistemas possuem a capacidade de executar ações dentro de intervalos de tempo pré especificados.

Sistemas tempo real caracterizam-se pela necessidade fundamental de manter um sincronismo constante com o processo, isto é, o sistema deve atuar de acordo com a dinâmica de estados do processo. O relógio que controla as ações do sistema tempo real não é o relógio interno do computador, mas sim o relógio que controla a dinâmica dos estados do processo. Este processo, por sua vez, normalmente não possui a capacidade

de desfazer determinadas ações, justificando a preocupação com o sincronismo [MAL94].

Para controlar o comportamento de um sistema deve-se agir sobre o mesmo em momentos determinados do tempo. Desta forma, é possível realizar uma análise dos efeitos produzidos pela ação tomada. Esta situação ilustra o funcionamento de um sistema tempo real, cuja definição informal pressupõe um sistema que muda seu estado em função do tempo, o tempo real.

Uma das maiores dificuldades encontradas no desenvolvimento de sistemas tempo real é que eles, normalmente, são caros, pois exigem um profundo conhecimento sobre a aplicação onde atuam. Além disso, são difíceis de testar, uma vez que devem ser verificados diretamente ou por meio de simulação. Em ambos os casos, pequenas alterações no sistema acarretam em novas rodadas de testes.

Sistemas tempo real diferem de sistemas convencionais por apresentarem restrições temporais e tratarem, na maioria das aplicações, com situações críticas. Em tais aplicações, a ocorrência de qualquer tipo de falha, inclusive falha de temporização, pode causar conseqüências catastróficas. Portanto, ao contrário dos sistemas onde há uma separação entre correção e desempenho, em sistemas tempo real, correção e desempenho estão fortemente relacionados [STA88].

Em geral, sistemas tempo real são descritos através de um modelo de execução baseado em tarefas. Uma tarefa está associada à unidade de concorrência em um sistema. Tarefas recebem dados de entrada, executam um algoritmo e geram saídas. A tarefa está logicamente correta se gerar uma saída correta em função dos dados de entrada. Além da correção lógica, sistemas tempo real exigem correção temporal. Uma tarefa está temporalmente correta se gerar a saída dentro de um prazo satisfatório. Uma tarefa está correta se estiver lógica e temporalmente correta, ou seja, se gerar uma saída correta dentro de um prazo satisfatório [RAM94].

As definições de “saída correta” e “prazo satisfatório” dependem, na maioria das vezes, da aplicação. Prazo satisfatório é especificado através de limites temporais (*deadlines*), os quais correspondem ao momento máximo para a conclusão da tarefa. A princípio, toda tarefa deve ser concluída antes de seu limite temporal (*deadline*) [SHI94].

Sistemas tempo real caracterizam-se pela previsibilidade, ou seja, a capacidade de se prever violação de garantia ou incorreção temporal. Previsibilidade pode estar associada a uma antecipação determinística ou probabilística para o comportamento temporal. Em uma antecipação determinística pode-se garantir que todos os limites temporais (*deadlines*) serão cumpridos, enquanto que em uma antecipação probabilística é estabelecida uma probabilidade de o limite temporal (*deadline*) ser cumprido [KOP94].

A obtenção de garantia de correção temporal e previsibilidade exige diversos cuidados na especificação e configuração do suporte de comunicação. A configuração do suporte requer uma escolha criteriosa dos parâmetros do sistema de comunicação, que envolve a determinação de protocolos e serviços de comunicação. Além disso, a configuração do suporte deve considerar a carga do sistema de comunicação e a arquitetura de comunicação adotada para o sistema [FON97]. Este trabalho concentra sua atenção na carga imposta a um sistema com característica tempo real quando da

utilização de uma técnica de validação de mecanismos de tolerância a falhas de um sistema. Esta carga é medida em termos de tempo de execução das atividades.

2.3.1 Comunicação

No contexto de sistemas distribuídos, enfocando ou não tempo real, um dos tópicos mais importantes a serem abordados é a comunicação. O fato de se ter um sistema fracamente acoplado, característica conceitual de sistemas distribuídos, faz com que a troca de mensagens torne-se o único meio de interação entre os componentes do sistema. Com isso surgem problemas quanto à confiabilidade e também quanto às exigências temporais, contrapondo-se às vantagens desse tipo de sistema. Assim, uma grande variedade de técnicas cuja preocupação é aumentar a confiabilidade e a disponibilidade de um sistema distribuído visa principalmente o nível de comunicação.

O projeto de sistemas tempo real deve incluir protocolos de comunicação com inúmeras propriedades. Por exemplo, algumas aplicações exigem comportamento determinístico por parte dos componentes de comunicação. Para alcançar o determinismo, protocolos devem possuir atrasos de acesso aos canais e atrasos de comunicação limitados. Esses limites apresentam diferentes semânticas, as quais estão diretamente relacionadas ao protocolo utilizado. Outro aspecto a ser considerado envolve o grau máximo de falha e retransmissão na determinação da garantia do tempo de entrega das mensagens. Esta garantia está vinculada ao modelo de falhas e às suposições de retransmissão [FON97].

A comunicação em um sistema tempo real, assim como em qualquer sistema distribuído, exige a presença de dois elementos principais: uma rede e um protocolo. No caso de comunicação tempo real, tanto a rede como o protocolo devem ser de tempo real, ou seja, devem obedecer limitações temporais. O protocolo controla o acesso ao meio físico, define o tráfego da rede e todas as interações entre os nodos componentes da rede. Ambos os elementos devem apresentar comportamento confiável, estar disponíveis para o ambiente no qual atuam e cumprir os prazos de tempo previamente estabelecidos para os mesmos. Segundo Veríssimo [VER93], sob o ponto de vista de comunicação, um sistema tempo real deve apresentar os seguintes atributos funcionais:

- atrasos conhecidos e limitados na entrega das mensagens;
- comportamento determinístico na presença de distúrbios (sobrecargas e/ou falhas);
- possibilidade de estabelecimento de diferentes prioridades referentes às mensagens;
- conectividade, uma vez que problemas de comunicação não devem interferir no mundo real.

É importante salientar, no entanto, que a rigidez no cumprimento dos atributos acima relacionados está diretamente ligada ao tipo de sistema tempo real que se está trabalhando. À medida que se migra dos sistemas críticos, onde a obediência a esses atributos é fundamental, para sistemas brandos, no entanto, fica claro que se começa a relaxar tais atributos. Assim, apesar da necessidade de observação desses limites, é

importante lembrar que a rigidez dos mesmos é peculiar aos sistemas críticos, e que possibilidades de comportamento mais relaxado existem quando se trata de sistemas tempo real mais brandos (*soft real time* e *best-effort*).

Com relação à comunicação, cabe ressaltar a dificuldade em se encontrar um único protocolo ou mecanismo de comunicação que corresponda a uma solução ótima, uma vez que o desempenho do protocolo está diretamente vinculado ao ambiente no qual o mesmo está sendo aplicado. Isto ocorre porque os protocolos são o meio de obtenção de comunicação confiável e contínua entre os diversos processos de um sistema tempo real. Como cada aplicação possui características específicas, cabe ao protocolo adequar-se a elas, atendendo seus requisitos [KOP94].

A característica de especificidade das aplicações mencionada anteriormente torna o desenvolvimento de um mecanismo de validação peculiar ao protocolo ou à classe de protocolos para os quais a aplicação se refere. Desta maneira, a construção de um injetor de falhas recai no problema clássico de generalização: um injetor de falhas específico para uma aplicação corresponde a uma solução particular e, na maioria das vezes, dificilmente portável para outras aplicações e/ou sistemas. Por outro lado, um injetor de falhas genérico, ou seja, desenvolvido de forma a abranger características de diversas aplicações, pode não retratar exatamente cada aplicação em questão. O que se propõe neste trabalho é o desenvolvimento de ferramentas de injeção de falhas específicas para as classes de protocolos que se adaptem tanto à plataforma de sistema como à semântica de falhas para as quais as ferramentas estão voltadas.

3 Injeção de Falhas

O uso de técnicas de tolerância a falhas em sistemas com característica tempo real exige que as mesmas sejam validadas. Na validação por injeção de falhas são testados os mecanismos de tolerância a falhas através da introdução de falhas de forma controlada, permitindo a aceleração da ocorrência das falhas e possibilitando a observação do comportamento do sistema frente aos estímulos provocados pelas mesmas [CHI89].

Entretanto, essa forma de validação em sistemas com característica tempo real se depara com limitações temporais inerentes a estes sistemas. Por representar uma carga extra no sistema, o injetor de falhas pode alterar o tempo de execução das tarefas comprometendo, desta forma, suas limitações temporais. Esta execução de carga extra é referenciada na literatura como intrusão [CUN2000].

Este Capítulo apresenta inicialmente os objetivos da injeção de falhas (Seção 3.1) e os conjuntos FARM (Seção 3.2). Na Seção 3.3 é descrita a classificação da injeção de falhas. A Seção 3.4 mostra uma arquitetura genérica para construção de um ambiente de injeção de falhas. Na Seção 3.5 são descritas características da injeção de falhas implementada por *software*. O problema de intrusão de injetores de falhas é tratado na Seção 3.6. A Seção enfoca principalmente a intrusão de injetores de falhas por *software*. O estado da arte da pesquisa na área de injeção de falhas é apresentado no Anexo.

3.1 Objetivos da Injeção de Falhas

Para testar um sistema, deve-se forçar o mesmo a determinados estados para garantir que caminhos de execução específicos sejam seguidos. Por outro lado, pode-se forçar o sistema a um comportamento que, sob condições normais, não aconteceria. Estas são as principais motivações para o uso de técnicas de validação [DAW94].

O objetivo da validação é garantir a confiança na capacidade do sistema em fornecer o serviço especificado, ou seja, fazer com que o sistema apresente a caraterística de dependabilidade [LAP85].

A validação por injeção de falhas age tanto sobre a remoção de falhas como sobre a previsão de falhas. Para a previsão de falhas avalia-se, além da confiabilidade, a disponibilidade do sistema sob validação [FUC96]. Aspectos de injeção de falhas para previsão e remoção de falhas são enfocados na Seção 3.5.1.

Injeção de falhas é uma abordagem atrativa para validação experimental da dependabilidade de sistemas tolerantes a falhas, pois oferece as medidas necessárias para um estudo detalhado da interação entre falha, erro, defeito e mecanismos de manipulação de falhas [CHI89].

Com injeção de falhas, as falhas podem ser injetadas durante a execução da aplicação alvo e o injetor fornece dados sobre o comportamento da aplicação em presença das mesmas. A análise desses dados indica se a aplicação atende à especificação, ou seja, se realmente mascara ou recupera-se das falhas a que se propôs na fase de projeto [HSU97]. Neste sentido, pode-se afirmar que a técnica de injeção

acelera a ocorrência das falhas em um determinado sistema. Com isso, ao invés de esperar pela ocorrência espontânea das falhas, pode-se introduzi-las intencionalmente, controlando, por exemplo, o tipo, a localização e a duração das falhas.

3.2 Conjuntos FARM

Segundo Arlat [ARL90], a injeção de falhas pode ser explicada através dos conjuntos FARM, conforme mostra a Figura 3.1. Para Arlat, injeção de falhas é baseada no projeto e implementação de uma seqüência de testes que podem ser caracterizados por um domínio de entrada e um domínio de saída. O domínio de entrada corresponde a um conjunto F de *Falhas* injetadas e um conjunto A que especifica os dados utilizados para a *Ativação* das falhas injetadas. O domínio de saída corresponde a um conjunto R de *Resultados* que são coletados para caracterizar o comportamento da aplicação alvo em presença das falhas e um conjunto M de *Medidas* que são derivadas da análise e do processamento dos conjuntos FAR.

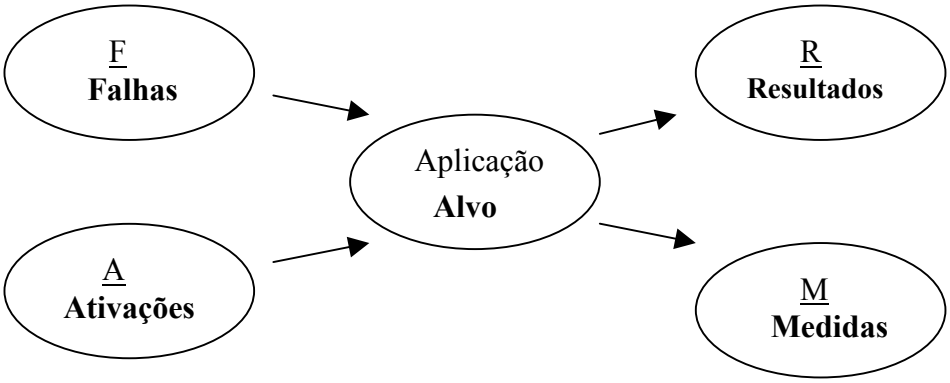

FIGURA 3.1 – Conjuntos FARM

Na Tabela 3.1 são comparadas as influências dos objetivos da remoção de falhas e da previsão de falhas nos atributos de injeção de falhas (conjuntos FARM) [FUC96].

De acordo com a Tabela 3.1, o domínio de entrada da injeção de falhas compreende as falhas (F) e os dados de teste que oferecem a ativação (A) das falhas injetadas. No caso de previsão de falhas, os padrões de entrada são selecionados com o objetivo de simular as conseqüências da ativação de falhas reais. As ativações das falhas são escolhidas para representar o ambiente operacional do sistema testado. Usualmente um grande número de falhas é injetado para garantir um bom nível de confiança nas medidas obtidas.

Visando a remoção de falhas, o principal objetivo é gerar erros que garantam a ativação dos mecanismos e algoritmos de tolerância a falhas e minimizar o número de experimentos redundantes. Portanto, o conjunto F é geralmente composto por um número pequeno de falhas específicas derivadas da especificação do projeto. Uma exigência é que os experimentos sejam reprodutíveis para que possam ser capazes de verificar os benefícios das possíveis alterações de projeto.

TABELA 3.1 – Impacto dos objetivos de validação nos conjuntos FARM

	Remoção de Falhas	*Previsão de Falhas*
F	Pequeno número de falhas específicas deduzidas das suposições de falhas feitas na fase de projeto.	Grande número de falhas estatisticamente distribuídas entre as possíveis falhas.
A	Dados de teste favorecendo a ativação das falhas injetadas e a propagação de erros nos mecanismos de tolerância a falhas.	Padrões de entrada simulando o perfil de utilização operacional.
R	Ocorrências de evento e/ou medidas de temporização (+ dados de diagnóstico).	Ocorrências de evento e medidas de temporização associadas.
M	Estado binário caracterizado pela verificação de predicados (eventos ou medidas de temporização).	Estatísticas do estado do sistema caracterizadas pelas combinações de predicados e tempo.

O domínio de saída da injeção de falhas compreende os resultados (R) dos experimentos e as medidas (M) derivadas desses resultados. No caso de remoção de falhas, o conjunto M denota uma decisão binária, ou seja, se o erro é detectado por um determinado mecanismo ou não, o que pode ser diretamente deduzido do conjunto R.

No caso de previsão de falhas, as medidas correspondem a ocorrência de estados de sistema, a duração que este sistema permanece neste estado e a freqüência/latência de alterações de estado. As medidas com relação a previsão de falhas consistem de medidas estatísticas caracterizando a dependabilidade do sistema sob teste. Exemplos de tais medidas são cobertura média e distribuições de tempo dos mecanismos de tolerância a falhas do sistema, as quais podem ser computadas do conjunto R de todos os experimentos de injeção de falhas.

A Figura 3.2 [VAR95] mostra as transições de estado que descrevem o comportamento esperado de um sistema em resposta à injeção de uma falha. Uma falha injetada pode ou não ser ativada. Quando ativada, a falha pode ser mascarada, detectada ou não detectada. Falhas mascaradas pelos mecanismos de tolerância a falhas do sistema alcançam o estado bom, enquanto falhas não detectadas alcançam o estado ruim. Falhas detectadas podem ser recuperadas ou não. Quando recuperadas, as falhas atingem o estado bom, caso contrário atingem o estado ruim.

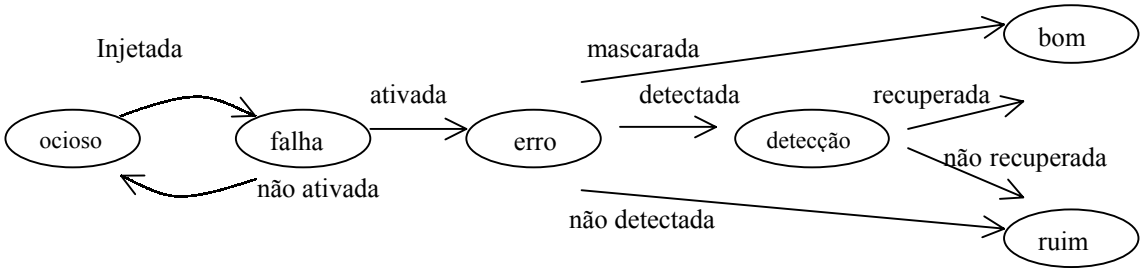

FIGURA 3.2 – Grafo da falha

A seguir são descritas as definições de cada ação de falha, de acordo a Figura 3.2 [VAR95]:

- Não ativada: ao ser injetada a falha, nenhuma ativação é observada em um determinado tempo de observação (volta para o estado ocioso).
- Ativada: a ativação da falha é observada. O tempo transcorrido entre a injeção e a ativação é registrado como *dormência*.
- Mascarada: apesar da ativação da falha, o sistema reage como se não houvesse erro (transição direta para o estado bom).
- Detectada: após a ativação da falha, os mecanismos de detecção de erros detectam um erro como conseqüência da falha injetada. O tempo decorrido entre a ativação e a detecção é registrado como *latência*.
- Não detectada: após a ativação da falha, o sistema exibe algum mal funcionamento sem ter detectado qualquer erro (transição direta para o estado ruim).
- Recuperada: se as ações de recuperação, realizadas após a detecção da falha, tiverem sucesso então a transição para o estado bom ocorre.
- Não recuperada: quando as ações de recuperação não têm sucesso dentro do tempo de observação é diagnosticada uma falha do sistema (transição para o estado ruim).

3.3 Classificação da Injeção de Falhas

Técnicas de injeção de falhas têm sido amplamente utilizadas para avaliar as características de dependabilidade dos sistemas ou simplesmente para validar determinados mecanismos de manipulação de falhas. A injeção de falhas pode ser implementada por simulação ou em sistemas reais, podendo esta última ser classificada em injeção de falhas em *hardware* e injeção de falhas por *software*. As Seções seguintes apresentam esta classificação de injeção de falhas. Descrições de ferramentas de injeção de falhas que exemplificam a classificação apresentada são exibidas no Anexo.

3.3.1 Injeção de Falhas por Simulação

Na injeção de falhas aplicada a modelos de simulação, falhas/erros são introduzidos em um modelo do sistema. Uma vantagem desta abordagem é poder ser aplicada durante a fase de desenvolvimento, o que facilita a detecção precoce de falhas de projeto.

Nesta abordagem, falhas são injetadas em um modelo de simulação da aplicação alvo, permitindo o controle da temporização, do tipo de falha e do componente afetado no modelo com maior ou menor acurácia, dependendo do nível de abstração do simulador [CUN2000].

A habilidade para controlar e monitorar falhas injetadas, a rapidez no acesso aos resultados, a facilidade na sincronização das falhas com o estado do sistema e a

habilidade para reproduzir os testes tornam a injeção de falhas aplicada a modelos de simulação uma alternativa bastante atrativa para validação [CHO92].

As principais desvantagens, entretanto, correspondem ao alto custo e à dificuldade na transposição dos resultados da avaliação para o mundo real [HSU97]. Sistemas reais usualmente não se comportam sob falhas de forma idêntica a seus modelos [CLA95].

Injeção de falhas aplicada a modelos de simulação foi previamente utilizada na implementação de técnicas de tolerância a falhas em sistemas distribuídos convencionais. Uma ferramenta para avaliação de protocolos de difusão confiável, denominada ADC [BAR95] [BAR96], foi desenvolvida para tanto. Uma característica observada com este trabalho consiste na complexidade da determinação o comportamento sob falhas das técnicas implementadas.

O ADC [BAR95] [BAR96] foi implementado em Unix (ambiente *SunOS*) e possui como suporte o sistema operacional heterogêneo HetNOS [BAC94]. O ADC roda em um ambiente simulado sob controle de um módulo central. A injeção de falhas é implementada através desse módulo, de acordo com uma probabilidade definida por um modelo de entrada, o qual denota o cenário de falhas a ser utilizado nos experimentos. O alvo da validação é um protocolo de difusão confiável cujo objetivo é obter concordância entre todos os membros do sistema livres de falha. O procedimento de injeção de falhas do ADC corresponde a uma solução específica para a implementação em questão.

Outros exemplos de ferramentas de injeção de falhas por simulação são Focus [CHO92], React [CLA93] e Mefisto [JEN94].

3.3.2 Injeção de Falhas em Hardware

Uma abordagem bastante comum de injeção de falhas consiste na injeção de falhas físicas no *hardware* do sistema real. Diversos métodos têm sido utilizados, tais como injeção de falhas a nível de pinos, através de distúrbios externos, *built-in*, dentre outros [KAR94]. Esses métodos, cuja classificação engloba os métodos com contato e sem contato, apresentam a vantagem de gerar falhas de *hardware* reais, o que os tornam mais próximos de um modelo de falhas real.

Entretanto, as abordagens de injeção de falhas em *hardware* exigem *hardware* especial. Em alguns casos, como por exemplo injeção de falhas a nível de pinos, a alta complexidade e a alta velocidade dos processadores disponíveis atualmente tornam o projeto deste *hardware* muito difícil, ou até mesmo impossível. O principal problema não está na injeção de falhas em si, mas está relacionado às dificuldades de controle e observação dos efeitos das falhas nos processadores [CUN2000].

Outra desvantagem da utilização de injeção de falhas por *hardware* é a dificuldade de sincronizar a introdução das falhas com o estado do sistema. Além disso, há exigência de *hardware* dedicado e corre-se o risco de danificar o próprio sistema.

Mesmo a detecção das falhas ativadas é muito complexa. Por exemplo, a injeção de falhas nos pinos do processador exige a utilização de *hardware* de monitoração para verificar se as falhas injetadas produzem erros internos de processador [CUN2000].

Experimentos constatam que a injeção destes tipos de falhas implica latência de detecção muito grande, o que torna a percepção da falha extremamente demorada. Tal situação se aplica quando a ativação da falha depende da utilização da posição de memória ou do registrador no qual a mesma foi inserida.

3.3.3 Injeção de Falhas por Software

Injeção de falhas por *software* emula falhas de *hardware* através de *software* corrompendo o estado do programa em execução. Deste modo, o objetivo da injeção de falhas por *software* é modificar o estado do *hardware*/*software*, que está sob controle do *software*, levando o sistema a se comportar como se falhas de *hardware* estivessem presentes [KAN95].

Injeção de falhas por *software* consiste, usualmente, na interrupção da execução da aplicação e execução do código do injetor de falhas. O injetor de falhas emula falhas pela inserção de erros em diferentes partes do sistema, como por exemplo alteração do conteúdo de registradores e posição de memória, alteração da área de código e *flags* [KAN95].

Conforme explicado na Seção 2.1, apesar da nomenclatura referenciar a injeção de falhas, sabe-se que a emulação de falhas de *hardware* se dá através da inserção de erros. Isto porque, de acordo com a Figura 2.1, a falha está compreendida no universo físico, enquanto o erro localiza-se no universo da informação. Logo, não se pretende atuar fisicamente nos componentes de *hardware* do sistema, e sim na emulação de falhas desses componentes através da inserção de erros.

Sob o ponto de vista de implementação de injetores de falhas por *software*, o injetor pode ser um processo, um conjunto de rotinas ou um objeto, que emula falhas de *hardware* corrompendo o estado do sistema em execução. O objetivo de injeção de falhas por *software* é modificar o estado do sistema, levando-o a se comportar como se falhas de *hardware* estivessem presentes. O injetor interrompe ou altera a execução do sistema e executa seu próprio código, emulando falhas pela inserção de erros no sistema.

Em sistemas com característica tempo real, onde as aplicações por eles controladas devem ocorrer em instantes de tempo relativos a uma base de tempo externa ao sistema, a validação por injeção de falhas se torna mais crítica. As restrições temporais impostas pelos sistemas aliadas a sobrecarga provocada pelo injetor de falhas justificam este fato. Neste sentido, a execução do injetor de falhas pode interferir nas características de temporização do sistema, prejudicando o teste em funções de tempo críticas [KAR94]. Este aspecto é enfatizado neste trabalho.

3.4 Arquitetura de um Ambiente de Injeção de Falhas

A Figura 3.3 [HSU97] exibe uma arquitetura genérica para o projeto de um ambiente de injeção de falhas. Na Figura, Hsueh [HSU97] apresenta um ambiente de injeção de falhas que consiste basicamente de: sistema alvo, injetor de falhas, biblioteca

de falhas, gerador de carga de trabalho, biblioteca de carga de trabalho, controlador, monitor, coletor de dados e analisador de dados.

De acordo com a Figura 3.3 [HSU97], existe um *injetor de falhas*, o qual emula a presença de falhas no *sistema alvo*. Essas falhas são armazenadas na *biblioteca de falhas*. O ambiente conta com um *monitor*, que observa o comportamento do sistema alvo e dispara a coleta de dados sempre que necessário. O *gerador de carga de trabalho* é responsável pelo fornecimento de comandos, os quais devem ser executados pelo sistema alvo, conforme consta na *biblioteca de carga de trabalho*. O *coletor de dados* recolhe os dados sobre o sistema. Estes dados podem ser analisados pelo *analisador de dados*. A coordenação de todo o ambiente é executada pelo *controlador*.

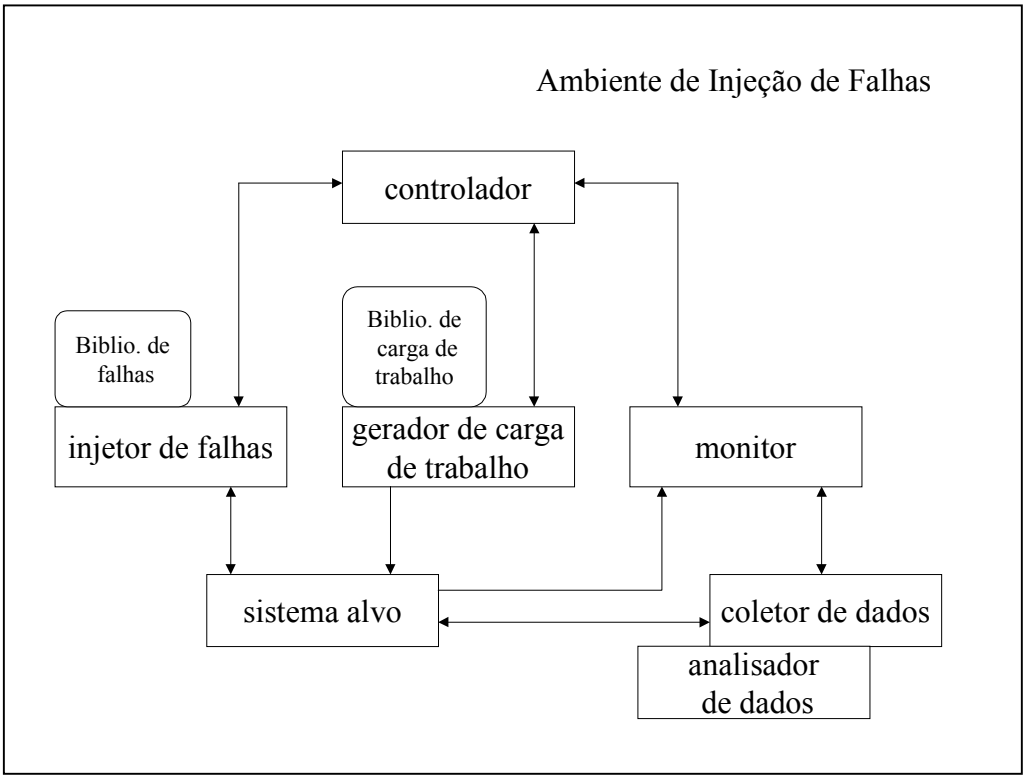

FIGURA 3.3 – Arquitetura de um Ambiente de Injeção de Falhas

Segundo Hsueh [HUS97], o controlador é um programa que pode executar no sistema alvo ou em uma máquina distinta. O injetor de falhas, que pode ser implementado por hardware ou por software, pode suportar diferentes tipos, localizações e tempos de falhas.

3.5 Características da Injeção de Falhas por *Software*

A técnica de injeção de falhas por *software* pode ser caracterizada de acordo com o exposto na Tabela 3.2 [FUC96]. Os itens a seguir (3.5.1 a 3.5.5) apresentam os principais aspectos que envolvem a injeção de falhas, de acordo com a tabela.

TABELA 3.2 – Características das técnicas SWIFI

Objetivos da injeção de falhas	Previsão de falhas Remoção de falhas
Tempo da injeção de falhas	Durante tempo de compilação Durante tempo de execução
Falhas	Nível de abstração Tipo Localização Disparo Duração
Multiplicidade	Número de falhas injetadas
Carga de trabalho	Aplicações reais Benchmarks selecionados Programas sintéticos

3.5.1 Objetivos da Injeção de Falhas Implementada por Software

De acordo com a Tabela 3.2, as técnicas de injeção de falhas implementadas por *software* podem representar dois objetivos distintos: previsão de falhas e remoção de falhas. Injeção de falhas para previsão de falhas auxilia na avaliação da eficiência dos mecanismos e algoritmos de tolerância a falhas durante a operação dos mesmos. Portanto, um modelo de falhas que simula um cenário de falhas realístico é selecionado. Em tais experimentos, um grande número de falhas estatisticamente distribuídas entre as possíveis falhas são injetadas visando a obtenção de resultados quantitativos com relação à cobertura e temporização.

De acordo com Fuchs [FUC96], no caso de remoção de falhas, o objetivo é descobrir falhas de implementação e projeto nos mecanismos e algoritmos de tolerância a falhas, o que exige um modelo de falhas significativamente diferente. Em tais experimentos um pequeno número de falhas cuidadosamente escolhidas são injetadas para revelar deficiências potenciais. A quantidade e a acurácia dos dados coletados por experimento é significativamente mais alta do que àquela para experimentos cujo objetivo é previsão de falhas.

3.5.2 Tempo da Injeção de Falhas

No contexto de ativação da injeção de falhas por *software*, Hsueh [HSU97] distingue duas categorias de injetores de falhas por *software*: durante tempo de compilação e durante tempo de execução (*runtime*). A Tabela 3.2 também segue esta distinção. Cunha [CUN2000] refere-se à ativação da injeção de falhas durante tempo de compilação como injeção de falhas *off-line*.

A Seção 4.2 apresenta os métodos de injeção de falhas, os quais podem ser ativados durante tempo de compilação e durante tempo de execução.

a) Durante tempo de compilação

Com injeção de falhas durante tempo de compilação as instruções da aplicação alvo são substituídas por instruções alternativas, as quais são responsáveis pela injeção de falhas. As modificações são realizadas estaticamente, ou seja, antes do injetor ser carregado no sistema e executado. Com isso, não há interferência na carga de execução, uma vez que o erro somente torna-se ativo quando as instruções ou os dados alterados são alcançados [HSU97].

Esta categoria de ativação apresenta mínima intrusão, já que não é necessário controlar os experimentos durante sua execução e nenhuma instrumentação é introduzida na aplicação alvo. Contudo, apresenta baixa controlabilidade, uma vez que se torna difícil descobrir se uma falha foi ativada. Além disso, a monitoração dos efeitos da falha também é uma tarefa complexa [ROS98b].

De acordo com Cunha, injeção de falhas durante tempo de compilação, por caracteristicamente apresentar mínima intrusão, corresponde à abordagem preferida para validação em sistemas com restrições temporais [CUN2000].

Injeção de falhas durante tempo de compilação apresenta diversas vantagens, quando comparada a injeção de falhas durante tempo de execução, dentre as quais destacam-se [FUC96]:

- não exige *software* adicional para a injeção de falhas na aplicação alvo, o que reduz a intrusão desta abordagem;
- devido à pequena quantidade de *software* dependente do sistema, técnicas baseadas em ativação durante tempo de compilação são facilmente portáveis para outros ambientes;
- oferece alto grau de reprodutibilidade, porque a maioria dos dados necessários para injeção da falha está disponível na máquina hospedeira. Contudo, cuidado deve ser tomado para oferecer um conjunto de dados de entrada adequado e para estabelecer um comportamento temporal previsível na aplicação alvo.

b) Durante tempo de execução

Injeção de falhas durante tempo de execução (*runtime*) possui a habilidade de injetar uma vasta gama de falhas. São necessários dois elementos: um *software* adicional para injeção de falhas e algum mecanismo para disparar o injetor de falhas, ou seja, a aplicação deve ser preparada para experimentos de injeção de falhas [ROS98b]. A atividade de injeção de falhas consiste na execução do código do injetor, que emula os efeitos do erro desejado, concorrentemente à aplicação alvo. O momento em que a corrupção é realizada pode ser disparado por um evento externo, como um *timeout*; por interrupções, como busca de instruções ou acesso a dados; ou pela inserção de instruções de injeção no caminho normal de execução da aplicação alvo. Tais alternativas correspondem a métodos de injeção de falhas, os quais são apresentados na Seção 4.2.

Injeção de falhas durante tempo de execução apresenta grande facilidade de controle e monitoração, mas aumenta a intrusão, a menos que o *injetor* e o *monitor*

usem o mesmo dispositivo de *hardware* ou características especiais do sistema alvo, tais como depurador tempo real. Além disso, a presença de *software* adicional implica na execução de código extra, o que conduz inevitavelmente a sobrecarga de tempo. Esta sobrecarga pode ser reduzida se o código de injeção for preparado em uma fase *off-line* e a corrupção realizada em tempo de execução.

A capacidade de disparo com base no tempo é extremamente importante para a obtenção de resultados estatísticos significativos, os quais podem refletir falhas de *hardware* randômicas. Com disparo baseado no tempo, o injetor não tem idéia sobre a instrução onde a falha será injetada. Isto significa, para diversos modelos de falhas (por exemplo, injeção na ULA ou barramento), que a instrução atual precisa ser decodificada *online* [CUN2000].

Segundo Cunha, estes fatos elegem injeção de falhas em tempo de execução como a melhor técnica de injeção de falhas para sistemas com característica tempo real, juntamente com a capacidade de oferecer um disparo temporal. Contudo, esta técnica apresenta maior sobrecarga de tempo no que se refere à execução do código de injeção de falhas [CUN2000]. Esta sobrecarga, referenciada como intrusão, é particularmente indesejável sempre que sistemas com restrições temporais precisam ser avaliados. Este problema é investigado na Seção 3.6.

3.5.3 Falhas

Segundo a Tabela 3.2, falhas a serem injetadas podem ser classificadas de acordo com diversos parâmetros. Primeiramente deve-se especificar exatamente o que se pretende com um determinado modelo de falhas. Isto se consegue através da definição do nível de abstração, do tipo, da localização, do disparo e da duração das falhas, conforme descrito a seguir [FUC96].

a) nível de abstração

Injetores de falhas implementados por *software* são capazes de injetar falhas em diferentes níveis de abstração, os quais incluem desde linguagens de alto nível até o nível de código de máquina. Quando se projeta experimentos de injeção de falhas, a seleção de um nível apropriado para a injeção de falhas é um pré-requisito para o alcance dos objetivos dos experimentos. Se, por exemplo, o objetivo é encontrar falhas de implementação ou projeto nos mecanismos de tolerância a falhas de um sistema, então a técnica de injeção de falhas operando no nível de linguagem de programação de alto nível será apropriada. Para a emulação de falhas de *hardware* em memória, por outro lado, uma abordagem de baixo nível no nível de código de máquina é a escolha mais adequada.

b) tipo

O tipo de falha depende do nível de abstração no qual a injeção de falhas é realizada. Falhas no baixo nível incluem, por exemplo, *single bit-flips* em memória, transferência de *byte* corrompido em um barramento, etc. Falhas no nível de linguagem

(de alto nível) pode incluir, por exemplo, uma constante ou variável incorreta, a omissão de uma declaração, uma condição incompleta, etc.

A implementação de uma técnica de injeção de falhas para corromper um segmento de código ou dados de um programa é complexa e a injeção de falhas pode ser realizada em tempo de compilação ou em tempo de execução. Injeção de falhas no barramento e CPU são mais difíceis de serem implementadas. Isto porque rotinas adicionais de manipulação de interrupções e injeção de falhas devem ser implementadas na aplicação alvo. Entretanto, em um sistema de comunicação em camadas torna-se mais fácil, com privilégios apropriados, modificar ou suprimir mensagens. Dependendo do tipo de falha a ser inserido, por exemplo falha de comunicação, deve-se associar uma classe de falhas ao mesmo. Exemplos de classes de falhas são [CRI91]: *crash*, omissão, temporização, arbitrárias, etc.

c) localização

Diversas unidades de *hardware* são acessíveis por *software* habilitando a injeção ou a emulação de falhas em várias localizações, tais como memória, barramentos, processadores, *timers*, *clocks* ou interfaces de comunicação. Contudo, há localizações na maioria dos sistemas que não estão acessíveis por *software*, como por exemplo, verificação e geração de paridade, o que constitui a maior restrição de abordagens de injeção de falhas baseadas em *software*.

d) disparo

O disparo da falha em uma atividade de injeção de falhas está associado à ocorrência de um evento. Este evento pode estar relacionado a uma condição de tempo ou espaço, sendo denominados disparos temporais ou espaciais, respectivamente. Disparos temporais dependem da evolução de um *timer*, enquanto disparos espaciais estão geralmente condicionados ao alcance de um endereço ou estado.

e) duração

A maioria das abordagens de injeção de falhas por *software* é capaz de injetar ou emular tanto falhas temporárias como falhas permanentes. Falhas temporárias englobam falhas intermitentes e falhas transientes. Com base nas ferramentas de injeção de falhas analisadas (ver Anexo), grande parte das falhas de *software* é de natureza transiente, portanto muitos dos estudos de injeção de falhas tem dado enfoque à injeção de falhas transientes.

3.5.4 Multiplicidade

O penúltimo aspecto enfocado na Tabela 3.2 é a multiplicidade, a qual refere-se ao número de falhas injetadas por rodada de experimento. Comparada a outras técnicas, como bombardeamento por íons pesados (*heavy-ion*) e radiação, abordagens implementadas em *software* apresentam menor complexidade para controlar diversas falhas injetadas simultaneamente.

3.5.5 Carga de Trabalho

A carga de trabalho consiste no último aspecto exposto na Tabela 3.2. A importância de cargas de trabalho torna-se óbvia uma vez que diversos experimentos de injeção de falhas têm mostrado que a dependabilidade é altamente dependente da carga de trabalho. Injeção de falhas pode auxiliar em diferentes níveis do sistema sob teste indo desde tarefas da aplicação no nível mais alto até o *kernel* do sistema operacional.

A seleção de uma carga de trabalho depende dos objetivos dos experimentos de injeção de falhas. Se o objetivo é investigar o impacto das falhas em aplicações dedicadas, então aplicações reais rodando na fase operacional do sistema devem ser usadas para injeção de falhas. Por outro lado, se o objetivo é uma avaliação geral do impacto das falhas no comportamento da carga de trabalho, então um conjunto de benchmarks representativo deve ser usado. Ainda, se o objetivo é alcançar uma alta cobertura com relação aos locais afetados, tipo de falhas, etc, então uma carga de trabalho sintética especialmente projetada para este propósito pode ser usada.

3.6 Intrusão

A validação em sistemas com característica tempo real, se depara com limitações temporais inerentes a estes sistemas. Por representar uma carga extra no sistema, o injetor de falhas pode alterar o tempo de execução das tarefas comprometendo, desta forma, suas restrições temporais. Assim, durante o desenvolvimento de um injetor, cuidados especiais devem ser tomados para determinar a carga que esse representa ao sistema e minimizar seus efeitos sobre as medidas de confiabilidade e cobertura de falhas.

O comprometimento dos limites temporais (*deadlines*), causado pela sobrecarga em termos de tempo de processamento, é comumente referenciado na literatura como efeito de sonda ("*probe effect*") [CUN2000]. Na maioria dos modelos de falhas, há necessidade de executar um código adicional no mesmo processador que executa a aplicação alvo. Como conseqüência disto, pode-se chegar a situações onde alguns limites de tempo são perdidos devido ao tempo de execução extra por parte das atividades de injeção de falhas.

A interferência do injetor de falhas na aplicação alvo, também chamada de perturbação ou intrusão, pode ser observada de forma temporal e espacial. Na intrusão temporal tem-se caracteristicamente o tempo de execução aumentado em virtude do acréscimo das atividades de injeção de falhas, as quais devem ser executadas juntamente com a aplicação alvo. Já na intrusão espacial, pode-se ter necessidade de modificar o código da aplicação alvo, o que também consiste em comportamento intrusivo. O relacionamento entre ambos os tipos de intrusão explica-se pelo fato de que se existe código adicional (intrusão espacial) para ser executado, possivelmente haverá aumento no tempo de execução (intrusão temporal).

Com o intuito de analisar o impacto do injetor de falhas na aplicação alvo, pode-se resumir a atividade de injeção de falhas em três principais fases [CUN2000]: disparo da falha, injeção propriamente dita e coleta de dados. Para o disparo da falha, algumas ferramentas de injeção de falhas utilizam-se de monitoração do progresso da aplicação

alvo para efetuar a seleção do momento da injeção da falha. Um recurso frequentemente utilizado é a execução do programa em modo *trace* (Seção 4.2.2). Procedimentos de monitoração introduzem sobrecarga. Por outro lado, o uso de recursos de *hardware* como *breakpoints* e *timers* para disparar a injeção não representam sobrecarga, uma vez que os mesmos podem ser setados *off-line*. Neste caso, a sobrecarga da atividade de injeção de falhas resulta de duas fases: a injeção e a coleta de dados.

Na injeção de falhas propriamente dita, um erro que emula a ocorrência da falha desejada é introduzido no sistema. Durante esta fase, diversos eventos podem introduzir atrasos na aplicação alvo, dependendo do modelo de falhas a ser implementado. Algumas localizações de falhas exigem que a instrução atual seja decodificada para selecionar a maneira própria de emular os efeitos dos erros. Se a localização da falha requer um determinado tipo de instrução, é necessária a execução, em único passo, da aplicação alvo até que a instrução seja encontrada. Um exemplo disso é uma falha em unidade de inteiro, a qual exige uma instrução alvo que faça uso desta unidade. O alvo da corrupção (registrador/memória) e possível recuperação é outra fonte de sobrecarga.

Na fase de coleta de dados, uma descrição do estado do sistema no momento da injeção é armazenada para futura análise. Nesta fase, a sobrecarga resulta do tempo gasto com os dados relacionados à atividade de injeção da falha, tal como a tarefa que estava executando quando a falha foi injetada, o conteúdo dos registradores, etc, seguido pelo armazenamento dos dados em um local seguro (memória estável). Além disso, a coleta de dados pode exigir que o programa seja interrompido na instrução seguinte à injeção. Isto para que as informações salvas incluam as principais conseqüências da injeção (por exemplo, o valor do novo contador de programa, ou o que foi lido após o acesso de memória ter sido corrompido). Esta situação exige um passo adicional, sobrecarregando ainda mais o sistema.

Conforme descrito acima, as três fases do procedimento de injeção de falhas introduzem sobrecarga nos processos do sistema sob validação. Portanto, em sistemas com restrições temporais, deve-se contar com alternativas que visem a minimização da sobrecarga e portanto, a diminuição da intrusão temporal sobre a aplicação alvo.

A intrusão temporal é medida com base no *tempo de execução* da aplicação alvo. Sendo assim, a análise da intrusão temporal implica também na análise da intrusão espacial, já que a intrusão espacial pode induzir um aumento no tempo de execução da aplicação em virtude da necessidade de execução de código adicional.

4 Implementação de Injetores de Falhas por *Software*

Injetores de falhas por *software* não podem ser implementados sem alterar a carga do sistema e sem influir em maior ou menor grau nas características temporais do sistema. O comportamento do injetor de falhas é altamente dependente do nível de abstração em que o mesmo se encontra. O nível de abstração relaciona-se à camada na arquitetura do sistema sob a qual está implementado o injetor de falhas. Pode-se estabelecer basicamente três níveis de abstração possíveis para implementação de injetores de falhas [BAR99b]: no meta nível, no nível da aplicação e no nível do sistema operacional.

Diversas abordagens para injeção de falhas foram investigadas com o objetivo de avaliar as possibilidades de implementação das ferramentas de injeção de falhas do INFIMO. A Seção 4.1 apresenta vantagens e desvantagens da implementação de injeção de *software*. Na Seção 4.2 são descritos alguns dos métodos de injeção de falhas por *software*. Estes métodos baseiam-se no *tempo da injeção de falhas* [HSU97], apresentado na Seção 3.5.2. De acordo com esses métodos de injeção de falhas, as Seções 4.3 a 4.5 investigam abordagens de implementação de injetores de falhas considerando os seguintes níveis de abstração: meta nível, nível de aplicação e nível do sistema operacional. A Seção 4.6 discute as abordagens apresentadas com base nos objetivos de implementação do INFIMO.

4.1 Vantagens e Desvantagens de Injeção de Falhas por *Software*

As principais vantagens da implementação de injetores de falhas por *software*, quando comparada com a implementação de injetores de falhas por *hardware* são [FUC96]:

- Injeção de falhas implementada em *software* permite que falhas sejam injetadas na localização e no tempo controlados por *software*, sem necessidade de suporte externo de *hardware*. Diversas funcionalidades de *hardware* são visíveis através de *software*. Logo, uma variedade de falhas de *hardware* nas unidades funcionais individuais do sistema pode ser emulada.
- A preocupação da validação do sistema, e, portanto, da injeção de falhas, não é a saída individual, mas o comportamento do sistema perante às falhas. Desse modo, não é necessário ter, por exemplo, capacidade de injeção de falhas no nível de saída, mas ser capaz de emular o comportamento das falhas nos altos níveis de abstração.
- Injeção de falhas baseada em *software* apresenta menor custo do que implementação de injeção de falhas por *hardware* porque não exige suporte de *hardware* adicional para monitoração. Com a omissão de equipamentos de *hardware* sofisticados, não é exigido nenhum profissional especializado para conduzir os experimentos.
- Devido ao constante aumento de complexidade dos *chips*, o acesso físico torna-se um problema para abordagens de injeção de falhas implementadas

em *hardware*. Outro impasse ao alcance físico que surge da tecnologia embutida é o tipo de *socket* para circuitos integrados, o que pode restringir o acesso aos pinos.

- Algumas das ferramentas propostas na literatura têm sido projetadas e implementadas com exigências de alta portabilidade para outras plataformas e sistemas alvo e fácil expansibilidade para diferentes modelos de falhas. As características de portabilidade e expansibilidade de técnicas de injeção de falhas implementadas por *hardware* dependem da técnica individual e podem exigir considerável esforço para adaptar a técnica a nova aplicação alvo, como por exemplo no caso de radiação iônica.
- Comparada com injeção de falhas por *hardware* é mais fácil conduzir injeção de falhas implementada em *software* em diferentes cargas de trabalho, como aplicações de usuário, sistema operacional, ou ambos.
- O risco de permanentemente danificar o *hardware* do sistema sob teste é quase inexistente para abordagens de injeção de falhas implementadas em *software*.
- É mais fácil repetir um grande número de experimentos com injeção de falhas implementadas em *software*, o que conduz à coleta de uma quantidade de dados suficiente para obtenção de medidas estatísticas.
- As técnicas de injeção de falhas implementadas por *software* apresentam baixa complexidade de desenvolvimento em comparação às técnicas implementadas por *hardware*.

Embora a abordagem de injeção de falhas implementada em *software* seja flexível e apresente uma série de vantagens, há que se considerar também algumas desvantagens e restrições, tais como [FUC96]:

- A abordagem de injeção de falhas implementada em *software* não pode injetar falhas em locais não acessíveis por *software*. A acessibilidade da injeção de falhas implementada em *software* é principalmente restringida pela falta de uma interface de *software* apropriada em contraste à maioria das abordagens implementadas por *hardware*. Exemplos disso são os problemas com modelagem de falhas de microcódigo, falhas de retardo e alguns modelos de falhas intermitentes.
- A granularidade da temporização da injeção de falhas implementada em *software* depende do nível no qual a injeção de falhas é realizada e pode ser pior do que em abordagens de injeção de falhas implementada em *hardware*.
- A instrumentação de *software* necessária para realizar a injeção de falhas pode perturbar a carga de trabalho em execução na aplicação alvo e alterar a estrutura do *software* original.
- Considerável esforço é aplicado no desenvolvimento de modelos de falhas apropriados para emular as conseqüências de falhas de *hardware* através de

software. Contudo, nenhuma generalidade adotada para modelos de propagação de falhas e erros foi desenvolvida.

Enquanto as duas primeiras restrições não podem ser transpostas somente por meio de *software*, as últimas duas restrições podem ser relaxadas através do projeto e implementação cuidadosos ou do estabelecimento de modelos de falha mais acurados e validados.

4.2 Métodos de Injeção de Falhas por *Software*

Conforme apresentado na Seção 3.5.2, os métodos de injeção de falhas são classificados de acordo com o *tempo da injeção de falhas* [HSU97]. A injeção de falhas pode ocorrer em tempo de compilação ou em tempo de execução.

4.2.1 Injeção de Falhas em Tempo de Compilação

O método de injeção de falhas em tempo de compilação visa a *alteração de código*, a qual deve ser realizada, obviamente, em tempo de compilação. Neste método, o código original da aplicação alvo é modificado antes da mesma ser compilada. O método injeta falhas no código fonte ou assembler da aplicação para emular os efeitos das falhas. O código modificado consiste em alterações nas instruções da aplicação alvo, causando a injeção. A falha torna-se ativa quando as instruções alteradas são executadas [HSU97].

Este método de injeção de falhas não exige *software* adicional durante tempo de execução, tal como um monitor. Com isso, não há problema de intrusão temporal, já que não há interferência na execução da aplicação alvo. O método é voltado para a injeção de falhas permanentes. Problemas surgem no controle e observação da injeção da falha.

4.2.2 Injeção de Falhas em Tempo de Execução

Em injeção de falhas em tempo de execução é necessário código adicional para injetar falhas e monitorar seus efeitos. Além disso, algum mecanismo para disparar a injeção é exigido.

Hsueh [HSU97] descreve métodos de injeção de falhas baseados em rotinas de interrupção (*timeout* ou exceção de *hardware/trap* de *software*) e inserção de código. Existem ainda os métodos de injeção de falhas através de um processo especial [HAN95], injeção de falhas em modo *trace* [KRI98] e injeção de falhas por reflexão computacional [ROS98b].

Injeção de falhas por rotinas de interrupção

Os métodos de injeção de falhas por *rotinas de interrupção* utilizam *timeout* ou exceção de *hardware/trap* de *software*. Ambos os métodos apresentam baixa intrusão na aplicação alvo e são de difícil implementação e portabilidade.

O método por *timeout* é o mais simples. Existe um *timer* que expira em um tempo pré-determinado, disparando a injeção. O *timeout* gera uma interrupção que ativa a injeção da falha. O *timer* pode ser implementado por *hardware* ou *software*. Este método não exige modificação na aplicação alvo. Um *timer* deve ser adicionado ao vetor de manipulação de interrupções do sistema. A injeção da falha não é baseada em um evento ou estado. A falha é injetada em função do tempo. Assim, pode-se afirmar que a falha sempre é injetada. Entretanto, os efeitos das falhas não são previsíveis, uma vez que não se pode determinar os pontos de injeção.

O método de injeção através de exceção de *hardware* ou *trap* de *software* transfere o controle para o injetor de falhas. Diferentemente do método por *timeout*, a exceção pode injetar a falha somente se algum evento ou condição ocorrer. A transferência de controle da aplicação alvo para o injetor de falhas é realizada por meio do vetor de manipulação de interrupções do sistema. O método apresenta alta controlabilidade, ao contrário do anterior, pois a injeção da falha é baseada na ocorrência de um evento ou condição.

Injeção de falhas por inserção de código

O método de injeção de falhas baseado em *inserção de código* adiciona instruções na aplicação alvo, as quais permitem a injeção da falha. Este método é semelhante ao método de *alteração de código* (Seção 4.2.1), embora com inserção de código a modificação seja feita antes da compilação da aplicação. A injeção da falha é realizada durante tempo de execução. O injetor de falhas pode fazer parte da aplicação, estar inserido no sistema operacional, ou ser implementado como uma camada extra entre ambos. Este método possibilita controle e monitoração, embora apresente alta intrusão na aplicação alvo. As Seções 4.4.1 e 4.4.4 discutem formas de implementação de injetores usando o método de injeção de falhas por inserção de código no nível da aplicação. A Seção 4.5.1 apresenta a implementação do método de injeção de falhas por inserção de código no nível do sistema operacional.

Injeção de falhas através de processo especial

No método de injeção de falhas através de um *processo especial* o injetor de falhas corresponde a um processo, o qual possui privilégios de acesso ao espaço de memória da aplicação alvo. Através deste processo especial, é possível injetar falhas e observar seus efeitos. O método apresenta baixa intrusão na aplicação alvo, porém é de difícil controlabilidade. A Seção 4.4.2 discute a implementação do método de injeção de falhas através de processo especial.

Injeção de falhas por reflexão computacional

No método de injeção de falhas por *reflexão computacional* existem dois níveis: o meta nível e o nível funcional. O nível funcional implementa os objetos da aplicação alvo enquanto o meta nível permite a injeção de falhas através da manipulação e observação das estruturas de dados e ações do nível funcional. Este método possui alta controlabilidade e pode exigir disponibilidade do código fonte. A Seção 4.3 apresenta características da implementação do injetor utilizando o método de injeção de falhas por reflexão computacional.

Injeção de falhas em modo *trace*

Com o método de injeção de falhas em *modo trace*, a aplicação é monitorada por uma rotina que executa em modo supervisor. Neste sentido é possível acessar a memória e os registradores da aplicação, injetando a falha desejada. Este método permite fácil controle e observação, embora apresenta interferêcia na velocidade de execução da aplicação alvo. A Seção 4.5.3 discute a implementação do método de injeção de falhas em modo *trace*.

4.3 Implementação do Injetor de Falhas no Meta Nível

Pode-se usar programação reflexiva para a implementação do injetor de falhas [ROS98a]. Reflexão computacional é uma maneira fácil de separar funcionalidades de implementação da aplicação alvo. A reflexão introduz um novo modelo de arquitetura, no qual existem dois níveis: o meta nível e o nível funcional.

O nível funcional pode ser usado para implementar os objetos da aplicação alvo enquanto o meta nível permite que programadores observem e manipulem estruturas de dados e/ou ações realizadas no nível funcional e assim, pode ser usado para implementar injeção de falhas.

Uma vantagem desta abordagem é a transparência, uma vez que para o procedimento de injeção não há interrupção explícita do código da aplicação ou execução da aplicação em modo *trace*, como ocorre usualmente em mecanismos de injeção de falhas por *software*. Outra vantagem é o reuso, já que os objetos responsáveis tanto pela injeção de falhas como pelo monitoramento podem ser incorporados a outras aplicações. Além disso, injeção de falhas no meta nível não necessita da execução do programa em modo privilegiado, tampouco é preciso *hardware* dedicado. Entretanto, nesta abordagem existe interação com a execução do programa e pode haver exigência de disponibilidade do código da aplicação.

Injetores de falhas no meta nível se prestam bem a sistemas orientados a objetos e podem ser usados, por exemplo, para validar protocolos de comunicação em sistemas distribuídos. Os meta objetos do meta nível podem interceptar a comunicação entre objetos distribuídos e dessa forma selecionar e manipular as mensagens desejadas durante a injeção de falhas. No entanto, o uso de orientação a objetos e reflexão computacional em sistemas com característica tempo real, sujeitos a falhas, precisa de

uma investigação mais aprofundada e foge ao escopo original desse trabalho. Este trabalho restringe-se à exploração das abordagens de implementação de injetores de falhas nos níveis da aplicação e do sistema operacional.

4.4 Injetor de Falhas no Nível da Aplicação

A inserção do injetor de falhas no nível da aplicação visa a injeção de falhas na própria aplicação sob teste. Neste caso, a aplicação é um protocolo para alcançar tolerância a falhas em sistemas distribuídos ou é um programa mais complexo que possui ou utiliza mecanismos de tolerância a falhas. Para simplificar, no texto ambos serão designados como protocolo alvo.

Conforme será apresentado a seguir, a abordagem pode ser implementada de várias formas: o protocolo e o injetor formam um único processo; o injetor de falhas e o protocolo correspondem a dois processos distintos interativos; o injetor e o protocolo são *threads* de um mesmo processo; ou o injetor faz parte de um *software* utilizado pelo protocolo.

4.4.1 Implementação por Rotinas

Considerando o protocolo a ser validado e o injetor de falhas um único processo, há necessidade de contar com o código fonte do protocolo disponível para alteração. Isto porque se o injetor de falhas irá fazer parte do protocolo durante a execução do teste, são necessárias alterações no código fonte do protocolo para que o mesmo possa atuar em conjunto com o injetor.

O protocolo alvo deve incluir, em seu código, chamadas às rotinas do injetor de falhas. A cada chamada ao injetor, o protocolo desvia sua operação normal, executa as ações do injetor e então retorna a sua execução. O posicionamento correto das chamadas no código fonte do protocolo exige uma análise cuidadosa do código e uma boa documentação do protocolo alvo.

Esta abordagem de implementação restringe a utilização do injetor para o protocolo alvo que sofreu alteração no seu código. A portabilidade para outros protocolos fica comprometida. A validação de diferentes protocolos deve ser criteriosa. O código do protocolo deve ser minuciosamente analisado para a determinação dos melhores pontos para posicionamento das chamadas às atividades de injeção de falhas.

Outra desvantagem corresponde a integridade do protocolo. Deve-se garantir que a atividade de validação não irá comprometer a execução do protocolo em si. O injetor roda no mesmo espaço de endereçamento do protocolo e possui os mesmos privilégios. Assim, o injetor tem acesso a todos os recursos do protocolo e pode, inadvertidamente, alterar o comportamento do protocolo, mascarando as medidas de confiabilidade que podem ser obtidas pelo teste.

Essa abordagem, entretanto, pode ser bastante útil quando o injetor é desenvolvido simultaneamente com a implementação do protocolo. Nesse caso o

desenvolvedor tem perfeito domínio sobre o código e pode tirar proveito da ausência de proteção do espaço de endereçamento do protocolo em relação ao injetor.

4.4.2 Implementação por Processos Concorrentes

A situação em que o protocolo alvo e o injetor de falhas encontram-se no mesmo nível (aplicação), porém correspondem a dois processos distintos, apresenta algumas vantagens em relação à abordagem anterior. Como o injetor de falhas não faz parte do protocolo alvo, não há necessidade de disponibilidade do código fonte do protocolo.

A forma pela qual os dois processos irão interagir é através do sistema operacional, entretanto, não se faz necessária a intervenção de um processo no código fonte do outro. Neste caso, o processo correspondente ao protocolo alvo e o processo injetor executam concorrentemente. Somente haverá interação entre os dois processos na injeção de uma falha e na posterior coleta de dados do experimento. Porém, tais procedimentos não implicam em alterações no código dos processos.

Para interagir com o protocolo sem modificar o código do mesmo, o processo injetor de falhas deve receber privilégios especiais para alterar determinadas áreas de dados do processo protocolo (por exemplo para suprimir pacotes, alterar a ordem ou o próprio pacote). Para tanto, o injetor deve saber determinar as áreas alocadas ao protocolo a cada momento.

Determinar as áreas e obter privilégios de acesso a outros processos não é uma tarefa trivial para processos no nível da aplicação e está fortemente relacionada aos recursos que um determinado sistema operacional pode oferecer aos processos. Se, entretanto, for prevista no desenvolvimento do protocolo a validação por injeção de falhas, as áreas alocadas ao protocolo podem ser declaradas como áreas de acesso comum a mais de um processo, facilitando a obtenção dos privilégios necessários por parte do injetor.

Outra vantagem em relação à abordagem anterior se refere à portabilidade. Sendo o protocolo alvo um processo em separado do injetor de falhas é possível, com pouca ou nenhuma alteração, que se possa utilizar o mesmo injetor de falhas para validar outros protocolos com características semelhantes.

A desvantagem com relação à integridade da aplicação, apresentada pela abordagem anterior, é consideravelmente menor nesta abordagem, mas não totalmente ausente. Isto se deve ao fato de o injetor de falhas não interferir diretamente no código do protocolo, ou seja, a execução do injetor de falhas e do protocolo alvo em dois processos distintos evita a interferência de um no código do outro. A única interação que ocorre nesta abordagem se refere à injeção das falhas, por parte do injetor, na área do protocolo durante sua execução. Não existe alteração manual do código fonte e nem interferência dinâmica na área de código durante a execução, isso se o esquema de proteção do sistema operacional for adequado.

Entretanto, nessa abordagem o custo de chaveamento de contexto para a execução alternada dos dois processos é considerável. Se o objetivo é minimizar a carga do injetor no sistema e evitar comprometer o seu comportamento temporal, o uso dessa

solução deve considerar cuidadosamente os tempos envolvidos no chaveamento de contexto.

4.4.3 Implementação por Threads

Uma terceira abordagem, que combina características das duas anteriores, é implementar o protocolo e o injetor como *threads* de um mesmo processo. *Threads* são processos leves que compartilham o mesmo espaço de endereçamento, porém não compartilham os mesmos registradores [SIL94]. Essa abordagem permite, como a primeira (Seção 4.4.1), acesso ao espaço de endereçamento de memória do protocolo alvo, sem necessidade de privilégios especiais, e evita o custo do chaveamento de contexto na troca de processos da segunda abordagem (Seção 4.4.2). A intrusão do injetor no código fonte do protocolo alvo é pequena, se restringindo à criação das *threads*.

Para tirar o máximo proveito da implementação dessa abordagem, entretanto, o sistema operacional deve suportar a execução de *threads*. Assim, quando a *thread* do protocolo abandona a CPU por alguma razão, a *thread* do injetor pode executar e liberar a CPU antes do protocolo voltar a execução. O impacto sobre a temporização do sistema nesse caso é baixa, considerando que o injetor executa apenas umas poucas e rápidas atividades cada vez que assume a CPU. Se *threads* são suportadas apenas pela linguagem de programação, entretanto, quando o protocolo abandona a CPU leva junto o injetor, e a vantagem da redução do tempo de execução se anula.

4.4.4 Implementação por Bibliotecas

A abordagem de injeção de falhas através do *software* utilizado pelo protocolo prevê a injeção das falhas de forma indireta. O *software* pode consistir de funções, classes, rotinas ou bibliotecas, as quais dão suporte ao protocolo alvo. O contexto desta implementação é similar à implementação através de chamadas de sistema (*system calls*), descrita na Seção 4.5.1. As implementações diferem no nível de atuação. Enquanto a implementação por chamadas de sistema é realizada no nível do sistema operacional, a implementação por bibliotecas é feita totalmente no nível da aplicação.

Ao invés de diretamente alterar o código do protocolo alvo, as alterações são realizadas nas bibliotecas utilizadas pelo mesmo. Para tanto, faz-se necessário ter conhecimento de quais bibliotecas o protocolo faz uso para que as atividades de injeção de falhas possam ter efeito. O conhecimento das bibliotecas a serem alteradas depende do modelo de falhas a ser validado.

A abordagem não exige disponibilidade do código fonte do protocolo, embora seja necessário acessar o código fonte das bibliotecas que a implementam. Esta situação se aplica para bibliotecas que trabalham sob o contexto de carga dinâmica, onde não há necessidade de recompilação a cada alteração. Com isso, esta abordagem de implementação de injeção de falhas pode ser utilizada para validar diferentes protocolos, desde que façam uso do mesmo conjunto de bibliotecas. A abordagem não apresenta intrusão espacial, uma vez que não há interferência no código fonte do

protocolo. Entretanto, a introdução do código do injetor pode vir a modificar o contexto da biblioteca. Isto pode ocasionar problemas caso outros processos estejam usando as mesmas bibliotecas alteradas. Desta forma, o injetor pode acabar afetando programas de controle do ambiente, gerando assim resultados difíceis de serem analisados.

4.5 Injetor de Falhas no Nível do Sistema Operacional

Outra possibilidade é a introdução do injetor de falhas, ou parte dele, no nível do sistema operacional. Neste caso o alvo da injeção pode ser tanto um protocolo de tolerância a falhas de um nível superior, como protocolos e recursos de detecção e recuperação de erros do próprio sistema operacional.

Para as abordagens de implementação nesse nível, alguns autores incluem uma nova camada no sistema, a camada de injeção de falhas [DAW96]. A localização exata dessa camada depende da estrutura do sistema operacional: se este é organizado em camadas ou monolítico, e também do tipo de falhas que se deseja injetar. O comportamento do injetor de falhas como uma camada extra entre a aplicação e o sistema operacional, como uma camada extra entre subcamadas dentro do sistema operacional, ou como um recurso extra do sistema operacional é basicamente o mesmo.

O injetor de falhas poderá atuar, por exemplo, através de bibliotecas do sistema, através de chamadas de sistema (*system calls*) ou através da manipulação de outros recursos executáveis do sistema (por exemplo rotinas de tratamento de interrupção e exceções, *drivers* de dispositivos). Cada vez que o protocolo alvo utilizar recursos do sistema operacional, o injetor de falhas poderá atuar alterando as ações de acordo com o modelo de falhas a ser validado. Esta implementação, da mesma forma que a anterior, também pode vir a afetar outros processos que não o alvo.

4.5.1 Injeção de Falhas Através de Chamadas de Sistema

Injetores de falhas implementados através de chamadas de sistema (*system calls*) implicam na alteração das rotinas do sistema operacional em benefício do injetor de falhas. Para tanto é necessário que se tenha acesso ao código das rotinas do sistema operacional. Na injeção de falhas de comunicação pode-se, por exemplo, fazer uso de chamadas de sistema que implementam o envio e a recepção de pacotes para validar o modelo de falhas de *crash* ou omissão. A chamada de sistema que implementa a entrega de pacotes possibilita, através da manipulação de filas de entrega de mensagens, a validação do modelo de falhas de temporização.

Esta abordagem possui a vantagem de permitir que o protocolo alvo execute sem qualquer interferência do injetor de falhas. Esta característica aumenta o reuso do injetor de falhas. A portabilidade da ferramenta está vinculada à portabilidade do sistema operacional usado como plataforma. Entretanto, por encontrar-se no nível do sistema operacional e utilizar-se dele para atuar, o injetor de falhas pode interferir em sua integridade. Além disso, há necessidade de alterações do próprio sistema operacional, o qual deve apresentar flexibilidade.

4.5.2 Injeção de Falhas Usando Recursos do Escalonador do Sistema

Esta abordagem utiliza recursos do escalonador do sistema operacional de forma a priorizar as atividades do injetor de falhas em relação ao protocolo alvo, mas tentando não interferir nas restrições temporais do protocolo. É necessário que se tenha acesso ao sistema operacional para que seja possível alterar as prioridades dos processos em execução. Estes processos correspondem ao protocolo alvo e ao injetor de falhas.

O escalonador é utilizado para priorizar o injetor. Os processos do injetor que injetam falhas, uma vez escalonados e de posse da CPU, atuam de forma convencional selecionando e manipulando mensagens. O escalonamento tanto do injetor de falhas como do protocolo alvo é feito de forma independente.

4.5.3 Injeção de Falhas Usando Recursos do Depurador do Sistema

Através do depurador um processo (injetor de falhas) pode controlar a execução de outro processo (protocolo alvo). Assim, pode-se ler e modificar as variáveis locais e globais e contador de programa do processo depurado, o processo protocolo alvo.

O depurador permite que um processo injete falhas em outro. O processo que implementa o injetor de falhas cria um processo filho, o qual executa o código referente ao protocolo alvo da validação. O comportamento do processo protocolo alvo pode ser observado e modificado pelo injetor de falhas através dos recursos do depurador.

A portabilidade desta abordagem de implementação de injeção de falhas está vinculada à portabilidade do sistema operacional. É possível a utilização do injetor para a validação de outros protocolos, desde que haja compatibilidade entre os sistemas operacionais (do protocolo e do injetor). A abordagem garante ainda a integridade tanto da semântica do protocolo como do sistema operacional utilizado como suporte.

4.6 Discussão

Esta Seção discute aspectos das abordagens de implementação de injetores de falhas descritas anteriormente com vistas aos objetivos de implementação do INFIMO.

A implementação por processos concorrentes onde o protocolo alvo corresponde a um processo e o INFIMO corresponde a outro processo apresenta vantagens e desvantagens. Esta abordagem apresenta independência entre os processos, uma vez que somente há interação entre os mesmos na injeção de uma falha e na coleta dos dados do experimento. Esta interação é realizada por meio do sistema operacional, através de memória compartilhada. Tanto a injeção da falha, como a coleta de dados não implicam alterações de código dos processos.

Entretanto, esta implementação apresenta dois problemas. O primeiro é a exigência de que o INFIMO conheça, a cada momento, as áreas alocadas ao processo protocolo correspondentes à manipulação de mensagens. Esta exigência está vinculada ao modelo de falhas do INFIMO, o qual visa a validar falhas de comunicação. Este problema é facilmente resolvido se, no desenvolvimento do protocolo alvo, for prevista

a sua validação por injeção de falhas. Neste caso, basta que as áreas de manipulação de mensagens sejam declaradas como áreas de acesso comum a mais de um processo.

O segundo problema se refere ao chaveamento de contexto para a execução alternada dos dois processos. Como o objetivo do INFIMO é diminuir a intrusão no protocolo alvo, deve-se considerar tanto o custo como a carga imposta pelos procedimentos de chaveamento de contexto.

A utilização isolada do recurso de *threads* para a implementação do INFIMO, apesar de apresentar vantagens em relação à implementação anterior, possui limitações. Com *threads* problemas referentes ao chaveamento de contexto não existem. Além disso, o injetor de falhas pode utilizar-se da área comum para atuar sobre o protocolo alvo. Entretanto, tal implementação implica necessariamente injeção de falhas probabilística, o que torna complexo o seu controle.

Outra solução isolada prevê o uso do escalonador com uma função adicional: a função de injeção de falhas. Esta implementação além de tornar o escalonador lento e mascarado, não corresponde ao que se pode chamar de solução elegante para validação por injeção de falhas.

Implementações de injetores de falhas que apresentam bons resultados são através de chamadas de sistema (*system calls*) e bibliotecas da aplicação. Embora em níveis distintos, ambas possuem funcionamento similar. Na implementação através de chamadas de sistema, a cada chamada à rotina do sistema operacional o injetor de falhas pode atuar. Com bibliotecas da aplicação, cada acesso às mesmas permite a injeção de falhas. Tais implementações apresentam como vantagem a portabilidade para validação de protocolos que se utilizem das chamadas de sistema ou bibliotecas que implementam a injeção de falhas. Além disso, não há exigência de disponibilidade do código fonte do protocolo alvo. Como desvantagem destaca-se a possibilidade de mascaramento do comportamento das chamadas de sistema ou bibliotecas, uma vez que as mesmas sofrem alterações para viabilizar a injeção das falhas. Adicionalmente, tais alterações implicam em intrusão, a qual pode se apresentar de forma espacial ou temporal. A intrusão espacial pode ocorrer quando o contexto da chamada de sistema ou biblioteca é modificado, enquanto a intrusão temporal origina-se no tempo de processamento extra exigido pelas atividades de injeção de falhas. A intrusão espacial pode implicar em aumento no tempo de execução do protocolo, o que corresponde a intrusão temporal.

Um recurso comumente utilizado na implementação de injetores de falhas através de chamadas de sistema é sua associação com o depurador do sistema operacional. A injeção das falhas é realizada por intermédio do depurador, o qual monitora a execução do protocolo alvo. Apesar de caracteristicamente interferir na temporização do sistema, o uso de depuradores pode ser otimizado utilizando-se a abordagem de execução até um determinado ponto do protocolo, em substituição à execução passo-a-passo.

5 INFIMO

Este Capítulo apresenta o INFIMO. Na Seção 5.1 são descritas as características e objetivos do INFIMO *Toolkit*. A Seção 5.2 apresenta as ferramentas que compõem o *toolkit*: INFIMO_TAP, INFIMO_LIB e INFIMO_DBG. Cada uma dessas ferramentas encontra-se descrita detalhadamente nos Capítulos 6, 7 e 8. Dando continuidade, a Seção 5.3 apresenta a arquitetura das ferramentas de injeção de falhas do INFIMO, de acordo com a arquitetura exibida na Seção 3.4. A Seção 5.4 descreve o modelo de falhas suportado pelo INFIMO. As métricas utilizadas nos experimentos do INFIMO são descritas na Seção 5.5. A Seção 5.6 apresenta o método de medição da intrusão temporal imposta pelo injetor de falhas no protocolo alvo. A Seção 5.7 exibe a forma como os experimentos são conduzidos.

5.1 INFIMO *Toolkit*

O INFIMO (***In****trusiveless* ***F****ault* ***I****njector* ***MO****dule*) é um *toolkit* para experimentos de injeção de falhas, o qual visa a validação da dependabilidade de protocolos com restrições temporais. Um *toolkit* consiste em um conjunto de ferramentas voltadas para um objetivo comum. No caso do INFIMO, o objetivo é a condução de experimentos de injeção de falhas visando a analisar a intrusão dos injetores de falhas sobre o protocolo alvo.

O alvo da validação por injeção de falhas do INFIMO pode ser um protocolo tolerante a falhas ou mecanismos de tolerância a falhas implementados sobre protocolos com restrições temporais. Protocolos de sincronização de *clocks*, protocolos de consenso e protocolos de *membership* correspondem a exemplos de protocolos construídos sobre protocolos de comunicação que incorporam mecanismos de tolerância a falhas, os quais devem ser validados.

Nos Capítulos 3 e 4, seguindo a nomenclatura da maioria dos autores da área, adotou-se a expressão aplicação alvo para referenciar o destino da injeção de falhas. A partir deste Capítulo a aplicação alvo será referenciada como *protocolo alvo*, já que o INFIMO utiliza um protocolo coordenado de troca de mensagens como alvo da injeção de falhas.

O principal objetivo da implementação do INFIMO é analisar a intrusão imposta pelas atividades de injeção de falhas no comportamento temporal do protocolo alvo. Esta intrusão é medida, de forma experimental, através do *tempo de execução do protocolo*. Dentre as medidas convencionais de tolerância a falhas, tais como cobertura e latência, o INFIMO enfoca a cobertura, já que a latência é desprezível considerando o modelo de falhas de comunicação. Com isso, pode-se avaliar o comportamento do protocolo alvo, ou seja, pode-se verificar se o mecanismo de tolerância a falhas atende ou não às limitações temporais (*deadlines*), apesar de falhas.

Além disso, busca-se saber, através desta tese de doutorado, até que ponto pode-se utilizar injeção de falhas para validação de protocolos com restrições temporais. Por outro lado, como o injetor de falhas somente é usado na fase de testes, deve-se garantir

que a sua retirada não irá alterar o comportamento do protocolo alvo. Neste sentido, analisa-se a intrusão temporal e espacial do injetor de falhas sobre o protocolo alvo.

A primeira idéia de implementação do INFIMO previa o desenvolvimento e implementação de uma única abordagem de injeção de falhas visando análise da intrusão do injetor de falhas nas características temporais do protocolo alvo. A dificuldade em comparar os resultados do INFIMO no que se refere à intrusão imposta foi o ponto decisivo na redefinição de sua implementação. Poderia se pensar em comparar os resultados do INFIMO com os resultados apresentados por ferramentas de injeção de falhas com propósitos semelhantes, contudo a interpretação destes resultados corresponderia a uma tarefa de alta complexidade. Isto porque, as comparações teriam que considerar o objetivo da validação, as características do protocolo alvo, a forma de implementação do injetor, as classes de falhas suportadas pelo mesmo, e assim por diante. A implementação de duas abordagens para o mesmo escopo de comparações, embora mais trabalhosa, pareceu ser uma alternativa mais atraente sob o ponto de vista de contribuição efetiva.

Otimizações na idéia inicial conduziram à implementação do INFIMO como um *toolkit* de injeção de falhas, composto por ferramentas que atuam tanto no nível da aplicação como no nível do sistema operacional e que aproveitam as facilidades oferecidas pelos níveis nos quais se encontram. Além disso, o INFIMO *toolkit* conta com um protocolo coordenado de troca de pacotes, o qual corresponde ao protocolo alvo da validação por injeção de falhas.

Cada ferramenta de injeção de falhas no INFIMO apresenta características peculiares e, portanto, foi implementada com base nestas características. Para fins de organização do texto, aspectos referentes a cada ferramenta do INFIMO encontram-se descritos separadamente nos Capítulos 6, 7 e 8. O Capítulo 9 estabelece dois tipos de comparação: compara as ferramentas de injeção de falhas do INFIMO entre si e compara as ferramentas de injeção de falhas do INFIMO com algumas das ferramentas de injeção de falhas propostas na literatura.

5.2 Ferramentas do INFIMO

Através da investigação das abordagens de implementação de injetores de falhas descritas nas Seções 4.3 a 4.6, chegou-se a conclusão que a adoção de uma solução isolada não seria suficiente para analisar a intrusão provocada pelo injetor de falhas. Com isso, decidiu-se pela implementação de um *toolkit* para experimentos de injeção de falhas. O objetivo do *toolkit* é observar, de modo experimental, o comportamento intrusivo da injeção de falhas por *software* quando submetida a abordagens de implementação distintas.

O trabalho apresentado nesta tese compreende duas maneiras de validar um protocolo de troca de pacotes com característica tempo real através da injeção de falhas, visando analisar a interferência das atividades de injeção de falhas no comportamento do protocolo alvo. A opção por mais de uma alternativa de implementação de injetor de falhas visa tanto a definição de métricas como o estabelecimento de comparações entre os injetores.

Com base na análise de soluções isoladas, decidiu-se pela implementação de soluções híbridas para o desenvolvimento das ferramentas de injeção de falhas. Tais soluções combinam a utilização dos recursos do depurador com o uso de *threads*, chamadas de sistema (*system calls*) e bibliotecas da aplicação. O *toolkit* é totalmente implementado em plataforma Linux [BEC98] e é formado basicamente por três componentes: um protocolo alvo (INFIMO_TAP) e duas ferramentas de injeção de falhas (INFIMO_LIB e INFIMO_DBG).

O protocolo, denominado INFIMO_TAP (***In****trusiveless* ***F****ault* ***I****njector* ***MO****dule* – ***TA****rget* ***P****rotocol*), é o alvo da validação por injeção de falhas. O INFIMO_TAP, descrito no Capítulo 6, consiste em um protocolo coordenado de troca de pacotes, o qual possui restrições temporais e apresenta tolerância a falhas. Para sua implementação, faz-se uso das funções da biblioteca JRTPLIB [JRT99], a qual encontra-se implementada sobre o protocolo RTP (*Realtime Transport Protocol*) [SCH97]. O mecanismo de tolerância a falhas implementado pelo INFIMO_TAP consiste na detecção do *timeout* associado aos pacotes. O objetivo das ferramentas de injeção de falhas do INFIMO é a validação, através da injeção deliberada de falhas, deste mecanismo.

Neste contexto, cabe ressaltar três objetivos: o objetivo do INFIMO, o objetivo do protocolo alvo e o objetivo das ferramentas de injeção de falhas do INFIMO. O INFIMO visa a analisar a intrusão imposta pelos injetores de falhas INFIMO_LIB e INFIMO_DBG sobre o protocolo alvo INFIMO_TAP. O protocolo INFIMO_TAP viabiliza a troca de pacotes entre processos participantes, controlando, através de mecanismo de detecção do *timeout* a recepção destes pacotes. As ferramentas de injeção de falhas INFIMO_LIB e INFIMO_DBG têm como objetivo validar o mecanismo de detecção do *timeout* implementado pelo protocolo alvo.

A primeira ferramenta de injeção de falhas, chamada INFIMO_LIB (***In****trusiveless* ***F****ault* ***I****njector* ***MO****dule* – *by* ***LIB****rary modifications*), descrita detalhadamente no Capítulo 7, é implementada no nível de aplicação e faz uso da possibilidade de alteração nas funções da biblioteca sobre a qual o protocolo alvo foi construído, ou seja, a biblioteca JRTPLIB [JRT99]. As alterações realizadas nas bibliotecas fazem com que as mesmas injetem falhas de acordo com o especificado no experimento de validação. O desenvolvimento desta ferramenta conta com a disponibilidade do código fonte do protocolo alvo, embora seja somente necessário o conhecimento de quais funções de comunicação são chamadas pelo mesmo. Esta ferramenta é específica para protocolos implementados sobre a biblioteca JRTPLIB, cujas características sejam semelhantes às do protocolo alvo.

A segunda ferramenta de injeção de falhas, denominada INFIMO_DBG (***In****trusiveless* ***F****ault* ***I****njector* ***MO****dule* – *using* ***DeBuG*** *resBources*), é implementada com auxílio dos recursos do sistema operacional. A ferramenta de injeção de falhas INFIMO_DBG, descrita no Capítulo 8, baseia-se na utilização dos recursos de depuração apresentados pelo sistema operacional. Nesta ferramenta ocorre um desvio das operações normais do protocolo alvo, o INFIMO_TAP, para as atividades de injeção de falhas. A ferramenta utiliza o *ptrace( )* do Unix para realizar as atividades de injeção de falhas. Estas atividades são responsáveis pela parada, omissão ou atraso no envio de pacotes por parte de um determinado processo do sistema. Para o desenvolvimento do INFIMO_DBG não há necessidade de disponibilidade do código fonte do protocolo alvo. Esta exigência teria sido facilmente atendida, uma vez que o

protocolo INFIMO_TAP foi desenvolvido em paralelo com as ferramentas de injeção de falhas. A ferramenta INFIMO_DBG pode ser utilizada para validação de diferentes protocolos com suporte ao *ptrace( )* do Unix.

5.3 Arquitetura do INFIMO

A Figura 5.1 apresenta a arquitetura das ferramentas INFIMO_LIB e INFIMO_DBG, ferramentas de injeção de falhas do INFIMO. Esta arquitetura foi construída com base na arquitetura genérica para ambiente de injeção de falhas, descrita na Seção 3.4.

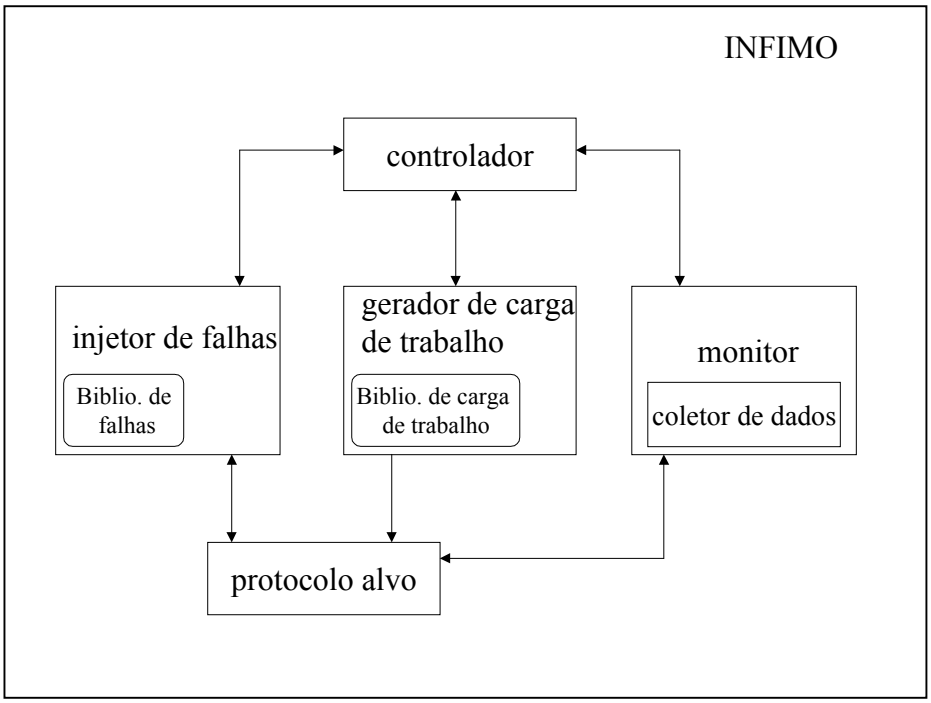

FIGURA 5.1 – Arquitetura de Injeção de Falhas do INFIMO

As ferramentas do INFIMO implementam os componentes da arquitetura da Figura 5.1 através de procedimentos incorporados aos códigos das mesmas. O protocolo INFIMO_TAP oferece procedimentos que realizam as funções do gerador de carga de trabalho e da biblioteca de carga de trabalho, conforme explicado na Seção 6.2.

Os injetores INFIMO_LIB e INFIMO_DBG, conforme descrito nas Seções 7.2 e 8.3, possuem procedimentos que executam as funções dos seguintes componentes da arquitetura: *injetor de falhas*, *biblioteca de falhas*, *controlador*, *monitor* e *coletor de dados*.

Na Figura 5.1, o *protocolo alvo* corresponde a um protocolo coordenado de troca de pacotes. Este protocolo apresenta restrições temporais e possui um mecanismo de tolerância a falhas implementado, o mecanismo de detecção de *timeout*.

Em ambas as ferramentas de injeção de falhas, a *biblioteca de falhas* está incorporada ao *injetor de falhas*. Esta biblioteca possibilita a especificação do modelo de falhas, composto pelas informações referentes ao tipo, localização, duração e disparo das falhas.

Para a ferramenta INFIMO_LIB, o *injetor de falhas* é implementado através de alterações nas funções da biblioteca JRTPLIB, implementada sobre o protocolo RTP. As alterações realizadas nas bibliotecas visam a injeção de falhas de acordo com a especificação da biblioteca de falhas.

Na ferramenta INFIMO_DBG, o *injetor de falhas* faz uso do *ptrace( )* para injetar as falhas no protocolo alvo. O *ptrace( )* permite que o injetor de falhas controle a execução do protocolo alvo e injete falhas de conforme a especificação da biblioteca de falhas.

O *gerador de carga de trabalho* compreende procedimentos que estabelecem o número de processos participantes da comunicação e o número de pacotes envolvidos. Para tanto, o gerador de carga de trabalho conta com o auxílio da *biblioteca de carga de trabalho*.

O *controlador* consiste em um procedimento que dispara as atividades de injeção de falhas. Na ferramenta INFIMO_LIB o disparo é feito a cada acesso à biblioteca alterada, conforme o modelo de falhas a ser validado. Para a ferramenta INFIMO_DBG o disparo é realizado a cada chamada ao *ptrace( )*, também de acordo com a especificação do modelo de falhas a ser validado.

O *monitor* é um procedimento que observa o andamento do experimento, coleta dados sobre o mesmo e salva-os em arquivos de *log*. O *coletor de dados* está incorporado ao monitor.

A arquitetura genérica apresentada na Seção 3.4 exibe ainda um *analisador de dados*. As ferramentas do INFIMO não possuem o analisador de dados, uma vez que o procedimento de análise é realizado manualmente.

Em ambas as implementações, as ferramentas de injeção de falhas são processos criados pelo sistema operacional através da linha de comando. O processo que implementa o injetor de falhas INFIMO_LIB incorpora em seu código o código do protocolo alvo INFIMO_TAP. O processo que implementa o injetor INFIMO_DBG cria um processo filho, o processo a ser monitorado. Este processo executa o código do protocolo alvo INFIMO_TAP. A descrição do funcionamento das ferramentas INFIMO_LIB e INFIMO_DBG é apresentada nas Seções 7.2 e 8.3, respectivamente. Detalhes de implementação do INFIMO_LIB e do INFIMO_DBG são apresentados nas Seções 7.3 e 8.4, respectivamente.

5.4 Modelo de Falhas

A utilização de qualquer técnica de tolerância a falhas está vinculada à determinação de um modelo de falhas. Um modelo de falhas em um protocolo tolerante a falhas refere-se aos cenários de falhas os quais o protocolo se propõe tolerar. Portanto, considerando a técnica de injeção de falhas, é de extrema importância a identificação do modelo de falhas tolerado para que o procedimento de validação tenha sentido.

O ideal seria que o modelo de falhas não estivesse comprometido com um determinado protocolo alvo. Há duas possibilidades relacionadas a este aspecto. A primeira refere-se ao fato de que, uma vez comprometido com um protocolo, o modelo de falhas torna-se extremamente específico. Desta forma, qualquer alteração no comportamento do protocolo alvo irá implicar na reavaliação do modelo de falhas. Por outro lado, a segunda possibilidade prevê o estabelecimento de um modelo de falhas genérico. Esta situação apresenta o inconveniente de qualquer método genérico, ou seja, não retratar exatamente o comportamento do protocolo.

No contexto deste trabalho, o escopo de falhas a ser utilizado pelo INFIMO compreende as falhas de comunicação simples no nível da aplicação. Por falhas simples entende-se a não ocorrência simultânea de mais de uma falha. Para o INFIMO falhas de comunicação consistem em falhas de processo e falhas de *link*. O INFIMO concentra sua atenção nas falhas de envio de pacotes por parte de um processo ou *link*.

A definição completa do modelo de falhas implementado pelo INFIMO consiste nas seguintes informações: tipo de falha (argumento *tipof*), localização da falha, disparo (argumento *disp*) da falha e duração da falha (argumentos *repet e dispp*). O projetista, ao disparar um experimento via linha de comando, deve informar o modelo de falhas a ser validado, através dos argumentos *tipof*, *disp*, *dispp* e *repet*. A localização da falha, que consiste na determinação do processo alvo da injeção de falhas, é especificada estaticamente. Estas informações são passadas como argumentos dos injetores de falhas, através da linha de comando. As Seções seguintes detalham estas informações.

5.4.1 Tipo de Falha

O tipo de falha (argumento *tipof*) corresponde a informação referente ao que corromper e como corromper. O INFIMO enfoca falhas de comunicação que podem corromper processos ou *links*.

A forma pela qual um processo (ou *link*) pode ser corrompido compreende a classe de falhas a ser validada. Para o INFIMO, o tipo de falha engloba falhas de comunicação. Estas, por sua vez, podem ocorrer de acordo com as classes *crash*, e omissão [CRI91]:

- falha de *crash* de processo: ocorre uma parada prematura do processo no sistema. A partir da falha, o processo não realiza qualquer função. Antes da ocorrência da falha o processo apresentava comportamento correto.
- falha por omissão de envio: um processo falha por intermitentemente omitir o envio de pacotes.

As ferramentas de injeção de falhas do INFIMO possuem ainda suporte para implementação de falhas de temporização, onde um processo falha pela violação do limite de tempo exigido para a execução de uma determinada função. Falhas de corrupção de valor estão fora do escopo de falhas do INFIMO.

A tolerância às classes de falhas descritas anteriormente dá condições para que o injetor de falhas determine medidas de tolerância a falhas para o protocolo alvo. Neste

sentido, de acordo com o mecanismo de tolerância a falhas implementado, o protocolo pode somente detectar a ocorrência da falha; detectar e recuperar-se da falha ou detectar e mascarar a falha. No caso do protocolo INFIMO_TAP, somente é possível validar o procedimento de detecção da ocorrência da falha através do mecanismo de *timeout*. No decorrer deste trabalho será usada a denominação tipo de falha para referenciar falhas de comunicação, de acordo com as classes acima descritas.

5.4.2 Localização da Falha

A localização da falha consiste na determinação do alvo da falha, que pode ser um dos processos do sistema ou um dos *links* de comunicação. Falhas de *link* são modeladas como falhas de processo. Do ponto de vista externo não é possível distinguir entre um processo que não enviou de um *link* que não transmitiu. Por questões de simplificação, o texto somente irá referenciar os processos como alvo das falhas injetadas. A definição do processo alvo da injeção de falhas pode ser determinística ou probabilística.

A localização que especifica qual processo deve ser submetido à ação da falha corresponde à definição determinística do alvo da injeção de falhas. Por outro lado, pode-se usar uma função randômica ou de probabilidade para a escolha do alvo da injeção de falhas, o que corresponde à definição probabilística. O INFIMO utiliza localização determinística do alvo da injeção de falhas, cuja definição é realizada estaticamente.

5.4.3 Disparo da Falha

O disparo ou a ativação da falha (argumento *disp*) deve estar relacionado a uma condição, a qual pode ser definida em termos espaciais ou temporais. Uma falha disparada espacialmente torna-se ativa quando o protocolo alcança determinado endereço ou estado. Por exemplo, uma falha pode ser ativada após o quinto pacote ter sido enviado.

Uma falha disparada temporalmente deve ser ativada após um determinado período de tempo. Uma vantagem do disparo temporal da falha é a certeza de que a falha será ativada, já que sua ativação não depende de nenhum estado ou evento. Neste caso, utiliza-se o *clock* interno do processador no qual o INFIMO está executando. O *clock* pode ser usado, por exemplo, para ativar uma falha um intervalo de tempo após o início da execução do protocolo.

De acordo com a arquitetura da Figura 5.1, o *controlador* é o procedimento responsável pelo disparo das atividades de injeção de falhas.

5.4.4 Duração da Falha

O injetor deve possibilitar a definição da duração da falha (argumento *repet*). Neste sentido, o INFIMO visa a implementação de falhas transientes e falhas intermitentes. A implementação de uma falha transiente é bastante simples, uma vez que a falha deve ser injetada somente uma vez. No entanto, a implementação de uma falha intermitente deve ser feita de forma criteriosa, já que a falha deve ser injetada repetidamente de acordo com uma distribuição de probabilidade [HAN95]. Para as classes de falhas apresentadas anteriormente (Seção 5.4.1), cabe ressaltar que, exceto a falha de *crash*, a qual é caracteristicamente transiente, as demais podem ser implementadas de forma transiente ou intermitente.

Se a injeção da falha deve ser repetida, *repet* recebe o atributo "intermitente" (FTINTER). Neste caso, deve-se especificar a forma de repetição. A repetição da injeção da falha pode ser especificada de contadores (*dispp*): contador de repetições ou contador de tempo. Para a injeção de falhas intermitentes através de um contador de repetições a falha injetada deve ser repetida a cada *dispp* pacotes. Já para a injeção por meio de contador de tempo, a falha injetada atinge um pacote a cada *dispp* unidades de tempo.

A implementação de falhas permanentes, cuja característica é a repetição da falha a cada operação do processo ou *link*, deve ser criteriosamente analisada no contexto de carga. A ativação contínua de uma falha ocasiona sobrecarga no sistema, o que pode acarretar problemas no cumprimento das limitações temporais impostas pelo protocolo. Por esse motivo, as ferramentas do INFIMO restringem-se à implementação de falhas transientes e intermitentes.

5.5 Definição de Métricas

Segundo Cunha [CUN2000], para aplicar injeção de falhas por *software* em protocolos com característica tempo real deve-se ser capaz de garantir que:

- as limitações temporais do protocolo (*deadlines*) serão preservadas, o que é considerado mais importante que garantir baixo tempo de resposta ou alto *throughput*;
- o escopo de aplicações tempo real onde o injetor possa ser usado não fique restrito.

Cunha afirma ainda que somente de forma experimental pode-se considerar que uma ferramenta de injeção de falhas por *software* preenche os dois requisitos anteriores [CUN2000]. Tal afirmação conduziu ao desenvolvimento de experimentos com as ferramentas do INFIMO. Para tanto, nesta Seção são descritas as métricas utilizadas nos experimentos com as ferramentas de injeção de falhas do INFIMO.

O INFIMO preocupa-se em medir a cobertura e a intrusão provocada por suas ferramentas de injeção de falhas. A cobertura de falhas refere-se à relação entre o número de falhas injetadas e o número de falhas percebidas pelo mecanismo de detecção de *timeout* do protocolo alvo. A intrusão é analisada sob dois pontos de vista:

temporal e espacial. A intrusão temporal considera a carga imposta pelas atividades de injeção de falhas no protocolo alvo e os efeitos desta carga. A métrica usada para a carga é o *tempo de execução do protocolo*. A intrusão espacial avalia o comportamento intrusivo do injetor de falhas no código do protocolo alvo.

5.5.1 Cobertura de Falhas

Para medir cobertura de falhas é estabelecida a relação entre o número total de falhas injetadas e o número de falhas detectadas pelo mecanismo de tolerância a falhas do protocolo alvo. No protocolo INFIMO_TAP, a detecção da falha é caracterizada através do mecanismo de *timeout*. Baseado na comparação destes valores pode-se analisar a cobertura de falhas obtida no experimento.

Os experimentos conduzidos consideram a relação entre a cobertura e o número de falhas e a cobertura e o número de processos, conforme descrito a seguir.

Cobertura x número de falhas

A relação entre a cobertura de falhas e o número de falhas injetadas no protocolo deve ser avaliada mantendo o número de processos constante. A partir daí deve-se aumentar progressivamente o número de falhas injetadas por rodada de execução. Neste sentido, pode-se avaliar o limite máximo de falhas suportado pelo mecanismo de detecção de falhas do protocolo alvo.

Cobertura x número de processos

Mede-se a cobertura de falhas em função do número de processos mantendo-se o número de falhas injetadas constante. A cada rodada aumenta-se o número de processos participantes do protocolo. Com isso, aumenta-se também o número de pacotes presentes no protocolo. A idéia é analisar o comportamento do protocolo para um determinado número de falhas injetadas. Assim, pode-se estabelecer o limite máximo de processos suportado pelo mesmo.

5.5.2 Intrusão

Conforme descrito na Seção 3.6, a intrusão pode apresentar-se de modo temporal e espacial. Na intrusão temporal as atividades de injeção de falhas implicam acréscimo no *tempo de execução do protocolo alvo*. Além das atividades normais do protocolo, as atividades de injeção de falhas devem ser executadas no mesmo processador, o que gera um aumento no consumo de tempo e recurso do protocolo.

A intrusão espacial refere-se à inserção de código adicional no protocolo alvo ou alteração de código existente. Em ambos os casos pode haver comprometimento do contexto do protocolo. A intrusão espacial está relacionada com a intrusão temporal, já que uma vez inserido código na aplicação, o mesmo deve ser executado, consumindo, portanto, tempo de processamento.

A seguir é mostrado como o INFIMO conduz a análise da intrusão de suas ferramentas de injeção de falhas, sob os pontos de vista temporal e espacial, respectivamente.

Intrusão temporal

O fator *tempo de execução do protocolo* visa analisar quantitativamente a intrusão temporal, ou seja, a carga imposta pelas atividades de injeção de falhas no protocolo alvo e como esta carga afeta a temporização do protocolo. A carga do protocolo é especificada pelo número de processos participantes do protocolo e pelo número de pacotes trocados entre estes processos. Ao ser submetido à injeção de falhas, o protocolo deve considerar ainda a carga referente às atividades de injeção de falhas, as quais também consomem tempo de processamento. Neste contexto, deve-se medir o tempo gasto executando o protocolo com e sem a interação do injetor de falhas, considerando os mesmos dados de entrada. Essa medição de tempo considera a execução de todo o ambiente de injeção de falhas, composto pelos componentes exibidos na Figura 3.3. Neste sentido, o tempo de execução engloba tanto as atividades de injeção de falhas como as atividades de controle do ambiente.

Intrusão espacial

Ao contrário da intrusão temporal, a intrusão espacial não pode ser medida quantitativamente. Portanto, para avaliar a intrusão espacial do injetor de falhas sobre o protocolo alvo deve-se recorrer à forma de implementação do mesmo. Neste sentido, deve-se verificar se o contexto do protocolo foi modificado após a introdução do injetor de falhas, ou seja, após a inserção ou alteração do código no protocolo alvo.

O objetivo desta avaliação é verificar se a intrusão espacial do injetor de falhas não está comprometendo a semântica do protocolo. A idéia é tornar o mais transparente possível a passagem de um injetor de falhas no protocolo alvo.

5.6 Método de Medição da Intrusão Temporal

A intrusão temporal do injetor de falhas sobre o protocolo alvo é obtida por meio do *tempo de execução do protocolo*. Este tempo é medido através do relógio local do processador no qual o protocolo está executando. Para cada experimento toma-se o valor do relógio antes do início da execução do protocolo e após o término da mesma.

Para se analisar a carga do sistema em termos de *tempo de execução do protocolo*, deve-se executar, para os mesmos dados de entrada (número de processos e modelo de falhas, quando for o caso), a seguinte seqüência de experimentos:

1. execução somente do protocolo alvo
2. execução do protocolo + injetor inativo
3. execução do protocolo + injetor ativo

Num primeiro experimento deve-se executar somente o protocolo alvo, ou seja, apenas a troca de pacotes e o controle do recebimento do ACK são considerados. Neste caso não deve haver injeção de falhas. O objetivo deste experimento é analisar a intrusão imposta pelo protocolo em si, através da medição do *tempo de execução do protocolo*.

Num segundo experimento executa-se o protocolo alvo com a presença, porém sem a atuação do injetor de falhas. Este experimento visa avaliar, quando em comparação com o primeiro, a intrusão imposta pelo injetor com máscara de injeção nula. Isto significa realizar todas as atividades normais do injetor, exceto que no final nada é modificado no protocolo alvo, já que nada foi especificado na máscara de injeção.

Num terceiro experimento deve-se considerar a execução do protocolo alvo juntamente com o injetor de falhas em plena atividade, ou seja, realizando as atividades de injeção de falhas no protocolo alvo.

Para os mesmos dados de entrada, pode-se realizar experimento local e distribuído. No experimento local todos os processos participantes da comunicação executam na mesma máquina, enquanto no experimento distribuído os processos executam em máquinas distintas. Cabe ressaltar que o injetor de falhas atua localmente. Desta forma, em experimentos distribuídos apenas os processos que se encontram na mesma máquina do injetor podem ser alvo de falhas.

A execução de experimento local é justificada pela necessidade de obtenção de dados que reportem uma execução distribuída na qual diversos processos executam na mesma máquina. O experimento local em si não faz sentido, uma vez que o mesmo não exige um protocolo de comunicação.

5.7 Condução de Experimentos

Diversos experimentos foram conduzidos com o *toolkit*. O objetivo desses experimentos é ilustrar, sob a forma de tabelas e gráficos, como a dependabilidade do mecanismo de detecção de *timeout* pode ser medida com o INFIMO.

Os experimentos apresentados no decorrer deste trabalho, assim como a implementação do *toolkit*, foram realizados em computadores PC Pentium MMX, com processador 233Mhz, 64MB de memória RAM, conectados via Ethernet 10Mbps, rodando Linux 2.2.14.

Para cada experimento, deve ser selecionado o modelo de falhas a ser injetado. Um experimento deve ser executado diversas vezes para que se possa obter dados estatísticos. Neste sentido, um experimento é composto por um conjunto de rodadas. Esse conjunto de rodadas deve ser executado com os mesmos dados de entrada. Com isso, obtêm-se médias baseadas em execuções exaustivas dos mesmos dados de entrada. A definição de exaustão na literatura é um tanto imprecisa. Para alguns experimentos 20 rodadas de execução corresponde a um experimento exaustivo, enquanto para outros são necessárias 1000 rodadas. Por exemplo, a ferramenta DOCTOR [HAN95] executa 20 rodadas para experimentos de cobertura e 1000 rodadas para experimentos de latência.

Já a ferramenta FINE [KAO93], cujo objetivo é analisar a latência, executa 50 rodadas de experimento.

Os experimentos descritos neste trabalho consideram um mínimo de 50 rodadas de execução. Caso os valores obtidos com as 50 rodadas não se apresentem estáveis, são executadas mais algumas rodadas. Este critério de busca por estabilidade nos valores dos experimentos segue a proposta da ferramenta de injeção de falhas FERRARI [KAN95].

Para a realização dos experimentos considera-se que os relógios locais das máquinas nas quais os processos participantes do protocolo executam estão sincronizados ou que as diferenças entre os mesmos são irrelevantes. A preocupação com sincronização de relógios foge ao escopo deste trabalho.

Durante os experimentos, as ferramentas de injeção de falhas coletam informações relativas ao estado dos processos participantes do protocolo e as armazenam em arquivos de *log*.

Os experimentos do INFIMO possuem três fases principais: definição do experimento, execução do experimento e análise de resultados do experimento.

a) *definição do experimento*

Um novo experimento pode ser definido do estado inicial (instante zero) ou pelo reuso de dados de entrada de um experimento anterior. Esta última situação é utilizada para aplicar os mesmos dados em diferentes cargas de trabalho, para efeitos de comparação.

TABELA 5.1 – Definição do experimento

Argumento	**Descrição**
total	número total de processos participantes do protocolo
numf	número de falhas a serem injetadas
tipof	tipo de falha a ser injetada
disp	condição de disparo da falha
repet	duração da falha
dispp	momento de repetição

A definição de um novo experimento de injeção de falhas está vinculada a especificação dos dados de entrada descritos na Tabela 5.1.

A Tabela 5.1 apresenta os argumentos que devem ser passados pelo projetista, através da linha de comando, para a condução dos experimentos. Estas informações correspondem ao modelo de falhas (*tipof*, *disp*, *dispp* e *repet*), número total de processos participantes do protocolo (*total*) e número de falhas a serem injetadas para o experimento (*numf*). A definição do experimento é realizada pelo *gerador de carga de trabalho*, que incorpora a *biblioteca de carga de trabalho*, e pelo *injetor de falhas*, através da *biblioteca de falhas*. No INFIMO_LIB e no INFIMO_DBG, a *biblioteca de falhas*, ao receber estas informações realiza o *parsing* e ativa o *injetor de falhas*.

b) *execução do experimento*

Seguindo a arquitetura da Figura 5.1, a execução do experimento é uma atividade que envolve todos os componentes, a saber: o *protocolo alvo*, o *gerador de carga de trabalho*, a *biblioteca de carga de trabalho*, o *injetor de falhas*, a *biblioteca de falhas*, o *controlador*, o *monitor* e o *coletor de dados*. Detalhes de funcionamento das ferramentas de injeção de falhas INFIMO_LIB e INFIMO_DBG, bem como do protocolo alvo INFIMO_TAP são apresentados nas Seções 7.2, 8.3 e 6.2. A implementação das ferramentas de injeção de falhas INFIMO_LIB e INFIMO_DBG, assim como do protocolo INFIMO_TAP encontra-se descrita nas Seções 7.3, 8.4 e 6.3.

Na ferramenta INFIMO_LIB, as atividades de injeção de falhas são implementadas pelas funções da biblioteca JRTPLIB. A injeção da falha se dá pela execução, por parte do *protocolo alvo*, destas funções. Na ferramenta INFIMO_DBG, a injeção da falha é ativada através de chamadas ao *ptrace( )*.

Em ambas as ferramentas, o disparo das atividades de injeção de falhas é função do *controlador*. Cabe ao *monitor* observar a condução do experimento. Além disso, o *monitor* coleta dados sobre o experimento e salva-os em arquivos de *log*, já que o *coletor de dados* é parte integrante do monitor.

Conforme mencionado no início desta Seção, são consideradas, no mínimo, 50 rodadas de execução de cada experimento.

c) *análise do experimento*

Após o término do experimento de injeção de falhas, os arquivos de *log* estão disponíveis para análise. A análise de dados não é automatizada e deve, portanto, ser realizada manualmente pelo projetista do experimento.

Através da comparação entre o modelo de falhas introduzido no experimento e os arquivos de *log* gerados durante o mesmo, consegue-se obter dados para análise. Tais dados são suficientes para que o projetista investigue o comportamento do protocolo mediante o modelo de falhas validado.

Os dados coletados para cada experimento são descritos na Tabela 5.2. O modelo de falhas é utilizado para fins de comparação. O *tempo de execução do protocolo* viabiliza a avaliação da intrusão do injetor de falhas sobre o protocolo alvo. O *número de falhas* possibilita a comparação entre as falhas introduzidas pelo injetor de falhas e a cobertura de falhas obtida pelo mecanismo de detecção do protocolo alvo.

TABELA 5.2 – Análise do experimento

Modelo de falhas	Tipo, localização, disparo e duração da falha
Tempo de execução	Tempo de execução do protocolo
Quantidade de falhas	Erros detectados pelo mecanismo de detecção

6 Protocolo Alvo: INFIMO_TAP

A necessidade de um protocolo alvo para a realização dos experimentos de injeção de falhas conduziu à implementação de um protocolo com característica tempo real. Este protocolo denomina-se INFIMO_TAP – ***In****trusiveless* ***F****ault* ***I****njector* ***MO****dule* ***TA****rget* ***P****rotocol*. O protocolo consiste em um mecanismo coordenado de troca de pacotes e foi implementado com o auxílio da biblioteca JRTPLIB (*Jori's RTP Library*) [JRT99], a qual é executada com suporte do protocolo RTP (*Realtime Transport Protocol*) [SCH97]. O objetivo do INFIMO_TAP é permitir a comunicação entre processos participantes controlando, através de um mecanismo de tolerância a falhas, o envio e a recepção de pacotes. O INFIMO_TAP utiliza *timeout* na delimitação do tempo máximo de espera por um pacote. Neste contexto, o INFIMO_TAP implementa a detecção de falhas como mecanismo de tolerância a falhas. Portanto, esta detecção é viabilizada através do controle por *timeout*.

Este Capítulo é dividido basicamente em quatro Seções. A Seção 6.1 apresenta o protocolo RTP e a biblioteca JRTPLIB. Aspectos de implementação e utilização do RTP e da JRTPLIB são discutidos. A Seção descreve como a biblioteca JRTPLIB permite o envio e a recepção de pacotes em um sistema de comunicação.

Na Seção 6.2 é descrito o funcionamento do INFIMO_TAP, o protocolo alvo da injeção de falhas. Detalhes de implementação deste protocolo são apresentados na Seção 6.3, a qual descreve a utilização de *threads* e o mecanismo de *timeout* proposto para o protocolo. A Seção 6.4 apresenta experimentos realizados com o INFIMO_TAP.

No decorrer deste Capítulo, assim como nos Capítulos 7 e 8, trechos de código que exibem a implementação das ferramentas são apresentados.

6.1 RTP e JRTPLIB

O RTP (*Realtime Transport Protocol*) é um protocolo que trabalha no nível de transporte e, portanto, oferece funções de transporte fim-a-fim para aplicações tempo real. RTP trabalha em conjunto com um protocolo de controle, denominado RTCP (*Realtime Transport Control Protocol*) [SCH97].

O RTP difere de outros protocolos de transporte, como o TCP, uma vez que não oferece qualquer forma de confiabilidade ou controle de fluxo e congestionamento. Além disso, o RTP executa sobre o UDP e portanto, é considerado não confiável, não oferecendo garantia de entrega. O RTP segue, normalmente, as especificações contidas na RFC 1889 [SCH96].

O RTP carrega dados que possuem propriedades tempo real, enquanto o RTCP monitora a qualidade de serviço e oferece informações sobre os participantes de uma sessão.

Dentre as aplicações nas quais o protocolo RTP tem sido utilizado pode-se citar aplicações de áudio e vídeo interativos e ferramentas de diagnóstico tais como monitores de tráfego.

JRTPLIB (*Jori's RTP Library*) [JRT99] é uma biblioteca orientada a objetos para utilização do RTP. Esta biblioteca foi parcialmente desenvolvida como tese de doutorado de Jori Liesenborgs na School for Knwoledge Technology, através de uma cooperação entre o Limburgs Universitair Centrum (LUC) e a Universiteit Maastricht (UM) - Belgium/Netherlands.

A biblioteca JRTPLIB usa o protocolo RTP na arquitetura TCP/IP. A JRTPLIB utiliza UDP para encapsular o pacote RTP. As rotinas de rede são implementadas através do uso de *Berkeley sockets*, disponíveis na maioria dos sistemas baseados em Unix e em sistemas baseados em plataforma Windows, os quais usam a biblioteca *WinSock*.

Segundo consta na documentação [JRT99], a versão da biblioteca utilizada para a implementação do INFIMO_TAP foi testada em plataformas Linux e Windows (9x e NT). Também tem-se conhecimento de que a biblioteca roda nas plataformas Solaris, HP-UX, FreeBSD e VxWorks.

6.1.1 Utilização do RTP e da JRTPLIB no INFIMO_TAP

O RTP é utilizado como base para a implementação do protocolo alvo INFIMO_TAP, através da biblioteca JRTPLIB [JRT99]. Neste sentido, pode-se analisar de que forma o INFIMO_TAP utiliza-se da JRTPLIB e como esta acessa os recursos do sistema operacional para executar suas funções. A Figura 6.1 apresenta o nível de atuação do INFIMO_TAP, bem como os níveis inferiores a ele.

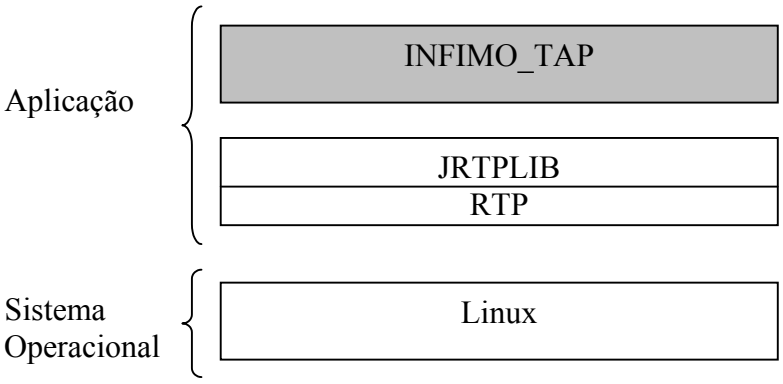

FIGURA 6.1 – INFIMO_TAP

Os procedimentos de interação entre o INFIMO_TAP, a JRTPLIB, o RTP e o sistema operacional utilizado como plataforma (Linux) estão vinculados a operações de comunicação, ou seja, ao envio e recepção de pacotes. A única interação com o nível do sistema operacional ocorre na utilização das chamadas de sistema do Unix que realizam o envio e a recepção de pacotes através de *sockets*. Cabe ressaltar que o Linux, utilizado como plataforma de implementação do INFIMO_TAP, preserva a utilização das chamadas de sistema *sendto( )* e *recvfrom( )* do Unix para o envio e a recepção de

pacotes, respectivamente. Essas chamadas de sistema são usadas com *sockets* sobre o protocolo UDP (*datagram sockets*) [BEC98].

O INFIMO_TAP utiliza-se das facilidades da biblioteca JRTPLIB no que se refere à troca de pacotes. A característica tempo real do INFIMO_TAP, implementada através da imposição de restrições temporais, não faz uso das propriedades tempo real oferecidas pelo protocolo RTP. O objetivo do INFIMO_TAP não é ser um protocolo tempo real, apenas possuir controle através de restrições temporais. Através deste controle é possível aplicar o mecanismo de detecção do *timeout*. Entretanto, a opção pela biblioteca JRTPLIB, além de facilitar na construção da comunicação propriamente dita, pode vir a facilitar o desenvolvimento de outros protocolos alvo, os quais se preocupem com a implementação de aplicações com características tempo real mais robustas.

O fluxo de execução do protocolo INFIMO_TAP é mostrado na Figura 6.2. Cada processo participante do protocolo, ao executar uma chamada a uma função de comunicação é desviado para as funções da JRTPLIB. Estas, por sua vez, utilizam o RTP para viabilizar a execução de suas operações através do uso de chamadas de sistema. A descrição do funcionamento do protocolo, bem como detalhes de implementação do mesmo são apresentados nas Seções 6.2 e 6.3, respectivamente.

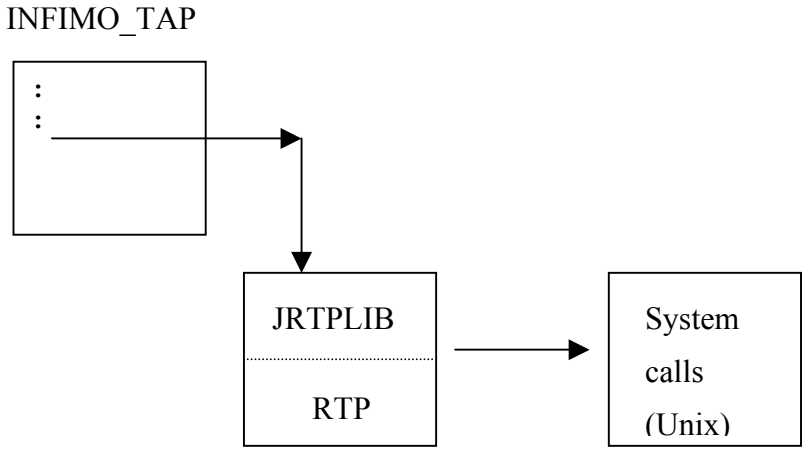

FIGURA 6.2 – Execução do INFIMO_TAP

A Tabela 6.1 resume o fluxo de chamadas do INFIMO_TAP para o envio de pacotes. De acordo com as duas primeiras colunas da Tabela 6.1, a função *enviapacote()* do INFIMO_TAP chama a função *SendPacket( )*, que chama a função *SendRTPData*, pertencentes às classes *RTPSession* e *RTPConnection*, ambas da JRTPLIB. A função *SendRTPData*, através do RTP, executa a chamada de sistema *sendto( )*. A terceira coluna identifica o nível de atuação de cada procedimento no contexto do sistema, ou seja, tanto o protocolo alvo, INFIMO_TAP, como a biblioteca JRTPLIB que usa o RTP, encontram-se no nível de aplicação.

TABELA 6.1 – INFIMO_TAP: Fluxo de chamadas para envio de pacotes

Função	Descrição	Nível de atuação
enviapacote( ) ⇓	Função do INFIMO_TAP	Aplicação
SendPacket( ) ⇓	Função da classe *RTPSession* da JRTPLIB	Aplicação
SendRTPData( ) ⇓	Função da classe *RTPConnection* da JRTPLIB	Aplicação
sendto( )	Chamada de sistema do *Unix*	Sistema operacional

A Tabela 6.2 apresenta o fluxo de chamadas do INFIMO_TAP para a recepção de pacotes. A JRTPLIB viabiliza a recepção de pacotes por parte dos processos participantes de uma sessão através das funções *PollData( )* e *ReceiveRTPData( )*, pertencentes às classes *RTPSession* e *RTPConnection*, respectivamente. A chamada de sistema *recvfrom( )*, pertencente à *socketcall*, é responsável pela execução efetiva da operação de recepção de pacotes. As funções *PollData( )* e *ReceiveRTPData( )*, por pertencerem à biblioteca JRTPLIB, encontram-se no nível da aplicação, enquanto que a chamada de sistema *recvfrom( )* encontra-se no nível do sistema operacional, conforme exibe a terceira coluna da Tabela 6.2.

TABELA 6.2 – INFIMO_TAP: Fluxo de chamadas para recepção de pacotes

Função	Descrição	Nível de atuação
PollData( ) ⇓	Função da classe *RTPSession* da JRTPLIB	Aplicação
ReceiveRTPData( ) ⇓	Função da classe *RTPConnection* da JRTPLIB	Aplicação
recvfrom( )	Chamada de sistema do *Unix*	Sistema operacional

O restante desta Seção descreve a forma pela qual o INFIMO_TAP faz uso das funções da biblioteca JRTPLIB exibidas nas Tabelas 6.1 e 6.2. O INFIMO_TAP, descrito na Seção 6.2, utiliza estas funções para viabilizar o envio e a recepção de pacotes obedecendo restrições temporais, conforme descrito nas Seções seguintes (6.1.1.1 e 6.1.1.2).

6.1.1.1 Envio de Pacotes

Para ser capaz de realizar algo útil com a biblioteca JRTPLIB, é necessário primeiramente criar uma sessão RTP. Para tanto, deve-se criar uma variável do tipo *RTPSession* e especificar um número de identificação para a porta local, conforme mostram as linhas de código da Figura 6.3.

```
RTPSession sessao;

status = sessao.Create(7000 + 2 * (rank - 1));
checkerror (status);
```

FIGURA 6.3 – Abertura de sessão RTP

Na Figura 6.3, a chamada à função *checkerror( )* possibilita a verificação de problemas com a abertura de sessão RTP.

Antes de enviar os dados, deve-se determinar para quais destinos cada pacote deve ser transmitido. Isso pode ser especificado conforme exibe o trecho de código da Figura 6.4. Para a função *AddDestination* devem ser passados como parâmetros o IP remoto e a porta destino remota.

```
status = sessao.AddDestination(destip, 7000 + 2 * (destino - 1));
checkerror (status);
```

FIGURA 6.4 – Especificação de destinos

Após especificados os destinos dos pacotes RTP, deve-se dar início ao envio de dados para todos os destinos envolvidos na comunicação. Para tanto, utiliza-se a função *SendPacket*, como mostra o código da Figura 6.5.

```
status = sessao.SendPacket(&pacote, sizeof(pacote), 0, false,10);
checkerror (status);
```

FIGURA 6.5 – Envio de pacotes

De acordo com a Figura 6.5, o primeiro parâmetro da função *SendPacket* corresponde aos dados a serem enviados, o segundo refere-se ao tamanho dos dados, o terceiro identifica o tipo de pacote, o quarto parâmetro corresponde a um *flag* de controle, enquanto o último parâmetro especifica o incremento do *timestamp*. Os três últimos parâmetros são geralmente usados com valores *default*.

Conforme exibido na Tabela 6.1, a função *SendPacket* chama a função *SendRTPData*, a qual pertence à classe *RTPConnection*. A função *SendRTPData* utiliza a chamada de sistema *sendto( )* para enviar os pacotes ao destino. O trecho de código da Figura 6.6 mostra a utilização da chamada *sendto( )*, cuja base de implementação faz uso de *sockets*.

```
sendto (sendsock, (const char *) packetbuffer, blocklen, 0, (struct
       sockaddr *) &addr, sizeof(struct sockaddr));
```

FIGURA 6.6 – Chamada de sistema *sendto( )*

Após o envio dos pacotes aos referidos destinatários, deve-se remover os destinos especificados, como mostra a linha de código da Figura 6.7.

```
sessao.ClearDestinations();
```

FIGURA 6.7 – Remoção de destinos

6.1.1.2 Recepção de Pacotes

Para a recepção de pacotes, deve-se chamar a função *PollData( )*, que pertence à classe *RTPSession*. A Figura 6.8 exibe a chamada a esta função. A função *PollData( )* verifica se há pacotes RTP e RTCP nas portas.

```
status = sessao.PollData();
checkerror (status);
```

FIGURA 6.8 – Verificação de Pacotes

A função *PollData( )* chama a função *ReceiveRTPData( )*. Esta função utiliza a chamada de sistema *recvfrom( )* para receber pacotes. A Figura 6.9 mostra a linha de código da chamada de sistema *recvfrom( )*. Esta chamada de sistema, da mesma forma que a chamada *sendto( )*, utiliza *sockets* para comunicação.

```
recvfrom (sock, (char *) packetbuffer, (int)len, 0, (struct
       sockaddr *) &addr, &frontlen);
```

FIGURA 6.9 – Chamada de sistema *recvfrom( )*

Para acessar os pacotes recebidos, deve-se interagir com os diferentes participantes da sessão e resgatar cada um dos pacotes RTP. Esta interação com os participantes se dá conforme exibe o trecho de código da Figura 6.10.

```
if (sessao.GotoFirstSourceWithData()) {
    do {
        RTPPacket *pack;
        while ((pack = sessao.GetNextPacket())!= NULL) {
              memcpy(&pacote, (*pack).GetPayLoad(), sizeof(pacote));
              delete pack;
        :
```

FIGURA 6.10 – Processamento de pacotes

De acordo com a Figura 6.10, a função *GotoFirstSourceWithData( )* seta um ponteiro para o primeiro participante origem que possui dados pendentes em sua fila. A função *GetNextPacket( )* retira um pacote da fila do atual origem. Se há dados disponíveis, a função retorna uma instância da classe *RTPPacket* contendo os dados. Caso contrário, retorna NULL. Se um pacote é retornado, ele deve ser processado. Após seu processamento, o pacotes deve ser destruído. Isto se dá através da função *delete*.

6.2 Descrição do INFIMO_TAP

O objetivo dos experimentos de injeção de falhas do INFIMO é a validação do mecanismo de detecção do *timeout* de um protocolo com restrições temporais. O protocolo, denominado INFIMO_TAP (***In****trusiveless* ***F****ault* ***I****njector* ***MO****dule* ***TA****rget* ***P****rotocol*) foi implementado sobre o protocolo RTP [SCH97], através da biblioteca JRTPLIB [JRT99].

O INFIMO_TAP é um protocolo que permite a troca de pacotes entre processos participantes e coordena a recepção dos pacotes através de *timeouts*. Cada processo controla o envio e a recepção de pacotes ao longo da rede de comunicação utilizando-se de números de seqüência e *timeouts*. *Timeout* é um dos mecanismos de detecção de falhas mais usados em sistemas síncronos e em sistemas com restrições temporais. O objetivo da validação concentra-se na detecção do *timeout* por parte dos processos participantes do protocolo.

A execução do protocolo é disparada pela linha de comando, através da qual o sistema cria os processos participantes do protocolo. O *gerador de carga de trabalho*, juntamente com a *biblioteca de carga de trabalho*, é responsável pelo controle dos processos. Faz parte deste controle a criação das *threads* e sua associação aos respectivos processos, conforme explicado mais adiante.

O INFIMO_TAP suporta um total de 50 processos participantes. Internamente, um processo do protocolo INFIMO_TAP corresponde a um conjunto de *threads*. Para cada processo existe uma *thread* principal, uma *thread* de resposta e várias *threads* de envio. Associado à *thread* principal existe ainda o *signal_handler*, uma espécie de vetor de interrupções responsável pela coordenação do *timer* (explicado na Seção 6.3.2).

Supondo uma execução com 4 processos. Primeiro é necessário o disparo, via linha de comando, de cada processo. Este disparo pode ser feito na mesma máquina ou em máquinas distintas. A Figura 6.11 exibe os 4 processos participantes do protocolo.

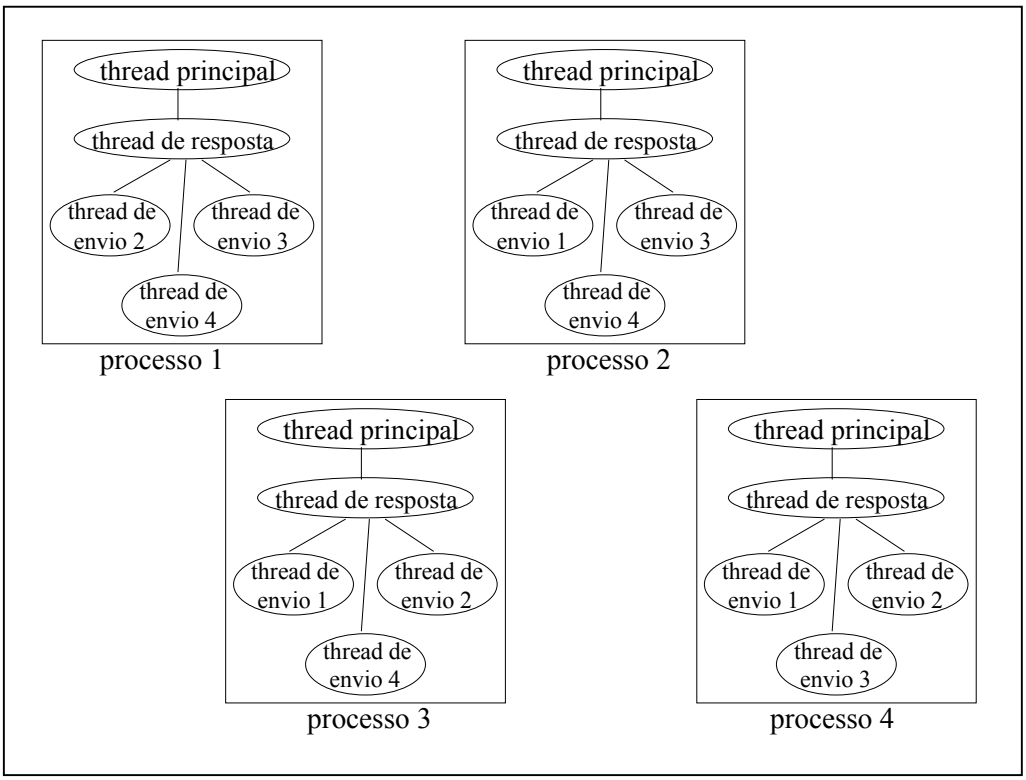

FIGURA 6.11 – INFIMO_TAP com 4 processos participantes

De acordo com a Figura 6.11, cada processo possui uma *thread* principal, uma *thread* de resposta e *n*-1 *threads* de envio, onde *n* é o número de processos participantes. Com 4 processos participantes, existem 3 *threads* de envio. Cada *thread* de envio é responsável por um destino. A criação das *threads* obedece a seguinte ordem: ao ser disparado, cada processo cria a *thread* principal, que por sua vez, cria a *thread* de resposta. Esta *thread* consiste em um *loop* que aguarda o recebimento de pacotes.

O primeiro pacote a ser recebido pelos processos participantes é um *pacote de inicialização*, cujo objetivo é permitir a criação das *threads* de envio. Este pacote de inicialização é gerado pelo processo participante com número de identificação mais alto, no caso do exemplo, o processo 4. O objetivo deste pacote é sinalizar o término da criação dos processos participantes do protocolo.

Cada *thread* de resposta, ao receber o pacote de inicialização dispara a criação das *threads* de envio. Com 4 processos participantes, o processo 4 envia o pacote de inicialização aos processos 1, 2 e 3. Através de suas *threads* de recepção, cada processo, incluindo o próprio processo 4, cria 3 *threads* de envio cada um (uma *thread* para cada destino). Após a criação das *threads* de envio, pode-se dar início à comunicação propriamente dita. As funções de cada *thread* e a forma de implementação das mesmas são descritas na Seção 6.3.1.

Os processos participantes do protocolo devem gerar um número randômico, o qual indica o número de pacotes que cada processo deve enviar para os demais. Esta função é realizada pelo *gerador de carga de trabalho*, o qual interage com a *biblioteca de carga de trabalho*. A escolha do destino de cada pacote é feita aleatoriamente pelos processos. O destino dos pacotes é irrelevante por dois motivos. Primeiro, em ambas as ferramentas INFIMO_LIB e INFIMO_DBG, a injeção de falhas ocorre no envio dos pacotes. Segundo, o controle da detecção do *timeout* é realizado na origem.

Após o início da comunicação, cada processo envia seus pacotes, os quais são formados pelos campos apresentados na Tabela 6.3. Os pacotes do INFIMO_TAP possuem 65 bytes. O RTP adiciona ainda seu cabeçalho, composto por 12 bytes.

TABELA 6.3 – Formato dos Pacotes

Campo do pacote	**Descrição**
origem	processo origem do pacote
destino	processo destino do pacote
R	número de pacotes a serem enviados
id	identificação do pacote
n_seq	número de seqüência do pacote, ($1 \geq n\_seq \geq R$)
valor_qualquer	conteúdo do pacote

A Figura 6.12 ilustra a seqüência de envio de um pacote A do processo 2 para o processo 4. O envio se dá através de uma *thread* de envio do processo origem (processo 2), enquanto a recepção é feita pela *thread* de resposta do processo destino (processo 4). A *thread* de envio do processo 2, responsável pelo destino 4 (pacotes destinados ao processo 4) é a *thread* de envio 4.

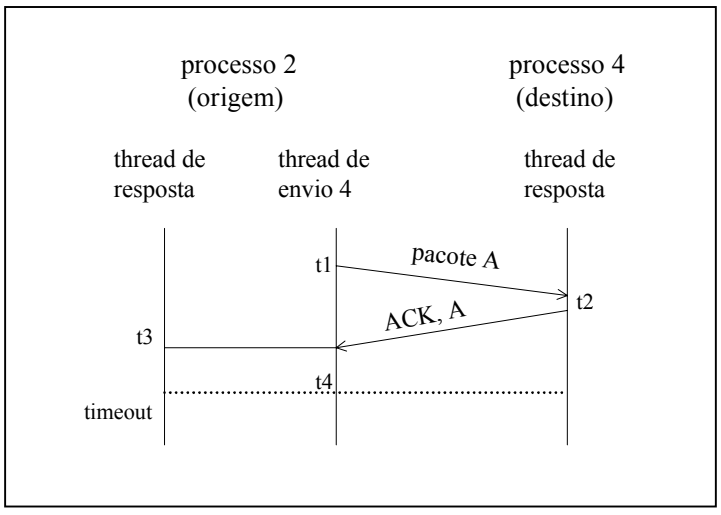

FIGURA 6.12 – INFIMO_TAP: Envio de pacote

Conforme a Figura 6.12, no tempo t_1, a *thread* de envio 4 do processo 2 envia o pacote A. Ao enviar o pacote, a *thread* de envio 4 do processo 2 adiciona um *timer* ao pacote e coloca este *timer* em uma fila de *timers*, denominada *timers queue*. No tempo t_2, a *thread* de resposta do processo 4, ao receber o pacote, armazena-o e envia um ACK ao processo 2. O ACK do pacote A é recebido pela *thread* de resposta do processo 2 no tempo t_3. Neste momento a *thread* de resposta do processo 2 remove o *timer* do pacote A da fila de *timers*. Toda a seqüência de envio de pacote e recebimento do ACK foi realizada antes do *timeout* expirar (tempo t_4).

A Figura 6.13 exibe a situação em que o *timeout* expira. No tempo t_1, a *thread* de envio 4 do processo 2 envia o pacote A. A *thread* de resposta do processo 4 recebe o pacote A no tempo t_2. O pacote é armazenado e a *thread* de resposta do processo 4 envia um ACK ao processo 2. No tempo t_3 ocorre o *timeout*. Neste momento, o *timer* do pacote A é removido da fila de *timers* e o pacote é reenviado (com bit de retransmissão ligado). O *timer* do pacote reenviado é adicionado no fim da fila de *timers*. No tempo t_4 a *thread* de resposta do processo 2 recebe do processo 4 o ACK referente ao pacote A. Este ACK é descartado, uma vez que seu *timer* foi removido da fila de *timers* devido ao *timeout*.

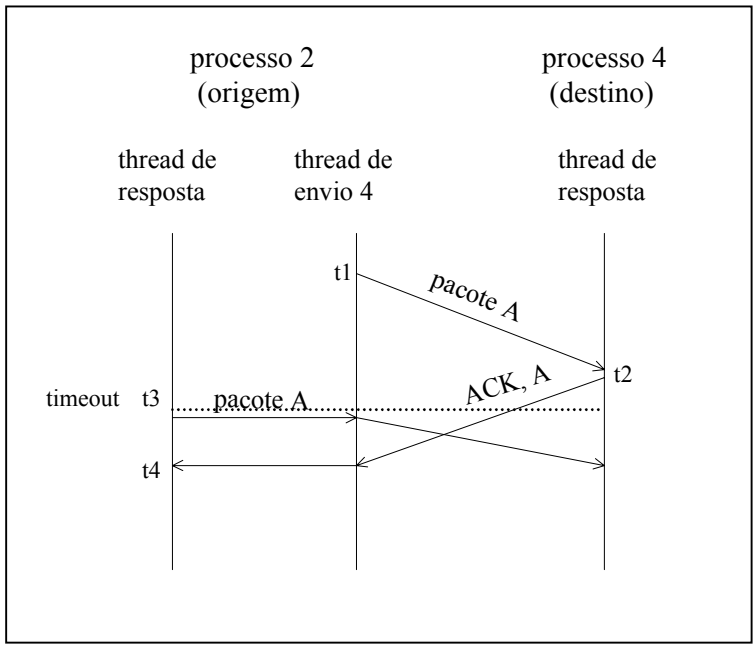

FIGURA 6.13 – INFIMO_TAP: Envio de pacote duplicado

Com a retransmissão, o processo 4 recebe o pacote A duplicado. O protocolo não faz nenhum tratamento para pacotes duplicados. O bit de retransmissão sinaliza para as ferramentas de injeção de falhas que o pacote não está suscetível a injeção de falhas.

Os procedimentos de manipulação de *timers* e reenvio de pacote são realizados por um conjunto de funções do INFIMO_TAP (alarme.cpp). Estas funções gerenciam os *timers* com base no relógio local da máquina em que se encontra o processo.

Aspectos de implementação do controle do *timeout* através dos *timers* são apresentados na Seção 6.3.2.

Existem basicamente quatro tipos de pacotes: envio, ACK, controle e reenvio. Pacotes de envio, ACK e reenvio são gerados pelo INFIMO_TAP, enquanto pacotes de controle pertencem ao RTP. Dentre os quatro tipos de pacotes, os únicos suscetíveis a injeção de falhas são os pacotes de envio. O INFIMO_TAP possui ainda o pacote de inicialização, responsável pelo disparo da criação das *threads* de envio de cada processo participante do protocolo. Este pacote não está sujeito a injeção de falha.

As ações de cada processo participante do protocolo são armazenadas em arquivos de *log*, os quais, desta forma, apresentam um histórico dos processos ao longo de cada experimento. O objetivo deste histórico é possibilitar a análise referente ao modelo de falhas pretendido, o qual possui os argumentos utilizados na condução do experimento, e o modelo de falhas executado no protocolo.

6.3 Implementação do INFIMO_TAP

Conforme mencionado anteriormente, o protocolo INFIMO_TAP foi implementado utilizando as bibliotecas da JRTPLIB como base para manipulação das caraterísticas de comunicação entre os diversos processos componentes do sistema. O INFIMO_TAP faz uso de *threads* para a implementação das operações de envio e recepção de pacotes. Além disso, o INFIMO_TAP controla o *timeout* através de um conjunto de funções de alarme (alarme.cpp). As Seções seguintes descrevem detalhadamente o uso das *threads* e o controle do *timeout* na implementação do protocolo alvo. Trechos de código da implementação do INFIMO_TAP são exibidos.

6.3.1 Threads

As funções de envio e resposta de pacotes executadas pelo INFIMO_TAP são coordenadas por um conjunto de *threads*. Conforme mencionado na Seção 6.2, existe uma *thread* de envio para cada destino no conjunto de processos participantes do protocolo. Desta forma, cada *thread* controla o envio de pacotes para cada processo destino. As *threads* de envio executam as tarefas de setar o *timer* associado ao pacote e enviar o pacote. Uma única *thread* de resposta realiza a manipulação dos pacotes de cada processo participante do protocolo.

Conforme explicado na Seção 6.2, cada processo do protocolo é composto por um conjunto de *threads*. Para cada processo participante do protocolo existe uma *thread* principal, uma *thread* de resposta e n-1 *threads* de envio, onde n é o número de processos participantes do protocolo. A *thread* principal corresponde à função principal do protocolo (*main( )*). Esta *thread* é responsável por toda a inicialização e controle do protocolo, que consiste principalmente:

- na verificação dos argumentos passados via linha de comando ao ser disparado o protocolo;

- na manipulação dos arquivos de *log* dos processos participantes do protocolo;
- na inicialização das máquinas (IP + nome de máquina);
- na manipulação das funções de alarme, responsáveis por inicializar, verificar e remover *timers*;
- na abertura da sessão RTP;
- na criação da *thread* de resposta.

A *thread* de resposta é responsável pela recepção dos pacotes, manipulação dos *timers* associados aos pacotes, envio de ACKs e reenvio de pacotes. A *thread* de envio tem como função enviar os pacotes e associar os respectivos *timers*. A descrição do funcionamento das *threads* de resposta e envio é apresentada a seguir.

Por questões de implementação, a ferramenta INFIMO_DBG agrega as funções da *thread* de envio à *thread* principal. Esta alteração é necessária para que o *ptrace( )* possa manipular diretamente as operações de envio de pacotes. A *thread* principal é vista pelo *ptrace( )* como um processo. Sendo assim, o *ptrace( )* pode atuar sobre o mesmo. O INFIMO_DBG faz uso desse recurso para a injetar falhas.

Thread de Resposta

A *thread* de resposta é criada pela *thread* principal. A primeira função da *thread* de resposta é disparar a criação das *threads* de envio. Este disparo é realizado a partir do recebimento de um pacote de inicialização do processo participante do protocolo cujo número de identificação é o mais alto (processo *n*). Com base neste número a *thread* de resposta cria *n*-1 *threads* de envio.

```
if (pacote.ack == 0) {
         if ((!iniciado) && (pacote.source == total)) {
           fprintf(logfile, "Recebeu sinal pra comecar de %d\n", total);
           emissoras = (pthread_t *) malloc((total-1) * sizeof(pthread_t));
           gettimeofday(&tv, NULL);
           fprintf(logfile, "Inicio %ld.%ld\n", tv.tv_sec, tv.tv_usec);
           for (i = 0; i < total; i++) {
             if (i != (rank-1)) {
             pthread_create(&emissoras[i], NULL, testaf, (void *) (i+1));
             }
           }
           iniciado = 1;
          }
          fprintf(stderr, "Enviando ACK\n");
          enviapacote(1, pacote.dest, pacote.source, pacote.numseq,
                      pacote.total, 0, "Eu sou um ACK");
        }
        else {
          fprintf(logfile, "Recebeu ACK %2d-%2d\n", pacote.source,
                  pacote.numseq);
          removetimer(pacote.numseq, pacote.source);
         }
```

FIGURA 6.14 – *Thread* de resposta

A Figura 6.14 mostra o trecho de código que implementa a *thread* de resposta. A *thread* de resposta fica recebendo pacotes e verificando se estes são pacotes de envio ou pacotes ACK. Se é um pacote de envio, a *thread* responde com um *ACK* através da função *enviapacote( )*. Se é um pacote ACK, a *thread* remove o *timer* associado a ele, através da função *removetimer( )*. O item seguinte, que descreve a implementação da *thread* de envio, apresenta a função *enviapacote( )*. A função *removetimer( )* é descrita na Seção 6.3.2.

Thread de Envio

A *thread* de envio transmite um determinado número de pacotes e adiciona os *timers* referentes aos mesmos na fila de *timers*. Conforme mencionado na Seção 6.2, o número de pacotes é gerada por uma função randômica e o destino dos mesmos é irrelevante. O trecho de código que exibe o funcionamento da *thread* de envio é apresentado na Figura 6.15.

```
for (i = 0; i < numpac; i++) {
    numseq = i + 1;
    adicionatimer(numseq, destino, total, payload);
    fprintf(logfile, "Enviando pacote %2d-%2d\n", destino, numseq);
    enviapacote(0, source, destino, numseq, numpac, 0, payload);
  }
```

FIGURA 6.15 – *Thread* de envio de pacotes

De acordo com a Figura 6.15, a *thread* de envio chama duas funções: *adicionatimer( )* e *enviapacote( )*. A função *adicionatimer( )* é responsável pelo controle do *timeout*, juntamente com a função *removetimer( )*, cuja chamada é feita pela *thread* de resposta. Estas funções, relacionadas à utilização do *timer*, encontram-se descritas na Seção 6.3.2.

A função *enviapacote( )* prepara o pacote e o destino do mesmo e chama a função *SendPacket( )* da classe JRTPLIB para enviá-lo. Após o envio, o destino é removido. Estes procedimentos são apresentados na Figura 6.16, embora tenham sido explicados detalhadamente na Seção 6.1.1.1.

```
status = sessao.AddDestination(destip, 7000 + 2 * (destino - 1));
  checkerror(status);
  status = sessao.SendPacket(&pacote, sizeof(pacote), 0, false, 10);
  checkerror(status);
  sessao.ClearDestinations();
```

FIGURA 6.16 – Trecho de código da função *enviapacote( )*

6.3.2 Timeout

O controle de entrega dos pacotes entre os processos participantes do protocolo alvo se dá através de *timers*. Por se tratar de um protocolo que utiliza como mecanismo de tolerância a falhas a detecção do *timeout*, a manipulação de *timers* consiste em uma tarefa muito importante.

O Unix permite a criação de até três *timers*, através da chamada à função *setitimer( )*. Entretanto, somente um desses *timers*, o ITIMER_REAL decrementa em tempo real. O INFIMO_TAP utiliza o ITIMER_REAL para controlar o *timeout*. Esse *timer*, ao expirar, envia um SIGALARM ao processo que o inicializou [TAN97]. O ITIMER_REAL foi utilizado pelo INFIMO_TAP para emular os outros *timers*, um para cada pacote. Neste sentido foi criada a fila de *timers*, denominada *timers queue*. A fila de *timers* é verificada por *polling*, ou seja, periodicamente é verificado se um *timer* expirou. Esse período é a menor resolução do *timer*. Com isso, se dois ou mais pacotes são enviados com um intervalo de tempo menor que esse período, seus *timeouts* provavelmente irão expirar na mesma verificação da fila de *timers* e os pacotes devem ser reenviados.

Os procedimentos de manipulação dos *timers* no INFIMO_TAP são implementados por um conjunto de funções as quais inicializam, adicionam, verificam e removem *timers*. Estas funções utilizam uma estrutura de dados, denominada *timer*, a qual é exibida na Figura 6.17. A estrutura *timer* tem como principal objetivo permitir a reconstrução do pacote caso seu *timeout* venha a expirar. Através desta reconstrução é possível o reenvio do pacote.

```
struct timer {
  char numseq;
  char dest;
  char total;
  char payload[59];

  long faltau;
  long adicionau;
  struct timer *next;
};

struct timer *timers;
```

FIGURA 6.17 – Estrutura *timer*

Alguns dos campos apresentados na Figura 6.17 correspondem aos campos do pacote próprios do RTP. São eles: *numseq*, *dest* e *payload*, representando o número de seqüência do pacote, seu destinatário e conteúdo, respectivamente. O campo *total* identifica o número total de *host*s presentes no experimento.

Faltau e *adicionau* são campos da estrutura e indicam a quantidade de tempo (em milisegundos) que falta para o *timeout* expirar e o momento em que o pacote foi adicionado, respectivamente. Este último campo é necessário para, ao adicionar o próximo *timer*, colocar em seu campo *faltau* somente a quantidade de *usec* que falta

quando disparar o *timer* anterior. Um *timer* é adicionado quando um pacote é enviado. O campo *next* é um ponteiro para o próximo *timer*.

O controle de *timeout* é feito através da fila de *timers*, representada na Figura 6.17 pela estrutura *timer*. Nesta fila, o primeiro elemento é o *timer* que vai disparar antes, ou seja, quando seu campo *faltau* chegar a zero. O tempo que falta para o segundo disparar é a soma do *faltau* do primeiro e do segundo, e assim sucessivamente.

A Figura 6.18 exibe o código que implementa a função *initalarm( )*. Esta função é chamada pelos processos participantes do protocolo, através da *thread* principal. A função *initalarm( )* instala um *timer* cuja função é acordar o processo a cada 100 milisegundos. Esta função interage com a função *verificatimer( )*.

```
void initalarm(void)
{
  struct sigaction myalarm;
  struct itimerval mytimer;
  bzero(&myalarm, sizeof(myalarm));
  myalarm.sa_handler = sigalarm_handler;
  sigaction(SIGALRM, &myalarm, NULL);
  bzero(&mytimer, sizeof(mytimer));
  mytimer.it_interval.tv_sec = 0;
  mytimer.it_interval.tv_usec = ALARMINTER;

  mytimer.it_value.tv_sec = 0;
  mytimer.it_value.tv_usec = ALARMINTER;
  setitimer(ITIMER_REAL, &mytimer, NULL);
}
```

FIGURA 6.18 – Função *initalarm( )*

A verificação dos *timers* é feita por *polling*. Não há sinalização por meio de interrupção. A cada 100 milisegundos a fila de *timers* é verificada. O período de 100 milisegundos consiste na menor resolução do *timer*. Nesse caso, se dois ou mais pacotes são enviados com um intervalo de tempo menor que 100 milisegundos, os *timeouts* desses pacotes provavelmente irão expirar na mesma verificação do *timer*. Os pacotes que tiveram *timeout* expirado devem ser reenviados. O valor *default* do *timeout* do pacote é de 500 milisegundos. Logo, o *timeout* de um pacote somente expira após 5 verificações de seu *timer*.

Na função *verificatimer( )*, exibida na Figura 6.19, se o *timer* ainda não expirou, o valor do tempo que falta é decrementado. Se o *timer* expirou, o pacote é copiado, seu *timer* é removido da fila de *timers*, o pacote é reenviado e seu *timer* é adicionado no fim da fila de *timers*, já que o tempo de vida do pacote vai mudar. Os procedimentos de manipulação da estrutura *timer*, comuns a todos os pacotes de um mesmo processo, são realizados com o auxílio de semáforos.

Quando o *timer* do pacote é removido da fila de *timers*, o valor de *timers* muda, pois o *timer* que expirou era o primeiro da fila. Por isso, o próximo *timer* da fila é testado para verificar se também não expirou. Este teste é realizado até que não haja pacote com *timer* expirado. Mais de um *timer* pode expirar na mesma chamada de *verificatimer( )*. O teste garante que todos os *timers* que devem expirar o façam, não somente o primeiro.

```
void verificatimer(void)
{
  char dest, numseq, total, payload[59];
  extern char rank;
  extern FILE *logfile;
  extern pthread_mutex_t mut;
  pthread_mutex_lock(&mut);
  if (timers) {
    if (timers->faltau > 0) {
      timers->faltau -= ALARMINTER;
    }
    else {
      while (timers->faltau <= 0) {
       dest = timers->dest;
       numseq = timers->numseq;
       total = timers->total;
       strncpy(payload, timers->payload, 59);
       removetimer(timers->numseq, timers->dest);
       fprintf(logfile, "Sending packet %2d-%2d (re-send)\n", dest, numseq);
       adicionatimer(numseq, dest, total, payload);
       enviapacote(0, rank, dest, numseq, total, 0, payload);
       fflush(logfile);
      }
    }
  }
  pthread_mutex_unlock(&mut);
}
```

FIGURA 6.19 – Função *verificatimer( )*

```
void adicionatimer(char numseq, char dest, char total, char *payload)
{
  struct timeval agora;
  struct timer *novotimer, *last;

  if (!timers) {
    timers = (struct timer *) malloc(sizeof(struct timer));
    timers->numseq = numseq;
    timers->dest = dest;
    timers->total = total;
    timers->faltau = 5 * 1000000;
    gettimeofday(&agora, NULL);
    timers->adicionau = tv2long(agora);
    strncpy(timers->payload, payload, 59);
    timers->next = NULL;
  }
else {
    for (last = timers; last->next != NULL; last = last->next);
    novotimer = (struct timer *) malloc(sizeof(struct timer));
    novotimer->numseq = numseq;
    novotimer->dest = dest;
    novotimer->total = total;
    strncpy(novotimer->payload, payload, 59);
    gettimeofday(&agora, NULL);
    novotimer->faltau = tv2long(agora) - last->adicionau;
    novotimer->adicionau = tv2long(agora);
    novotimer->next = NULL;
    last->next = novotimer;
  }
```

FIGURA 6.20 – Função *adicionatimer( )*

A função *verificatimer( )*, exibida na Figura 6.19, chama as funções *adicionatimer( )* e *removetimer( )*, cujos códigos podem ser vistos nas Figuras 6.20 e 6.21, respectivamente.

A função *adicionatimer( )*, exibida na Figura 6.20, coloca o *timer* de um pacote na fila de *timers*. Se a fila está vazia, basta alocar o *timer* e colocá-lo no início. Caso a fila não esteja vazia, deve-se adicionar o *timer* no fim da fila. Neste caso, a variável *faltau* desse pacote deve apresentar a diferença de tempo desde que o último pacote foi adicionado.

A função *removetimer( )* é mostrada na Figura 6.21. Nesta função, se o *timer* a ser removido não é o primeiro da fila, significa que o envio foi concluído com sucesso. Neste caso, o *timer* está sendo removido porque seu ACK foi recebido, e não porque expirou. Sendo assim, a diferença de tempo entre o estouro do *timer* anterior e do que vai ser removido deve ser adicionada ao tempo do próximo *timer* da fila. Isto porque o tempo de disparo de um *timer* corresponde a soma dos valores da variável *faltau* de todos os *timers* antes dele.

```
void removetimer(char numseq, char dest)
{
  struct timer *die, *anterior, *mytimer;
  struct timeval tv;
  extern FILE *logfile;

  if (timers) {
      if ((timers->numseq != numseq) || (timers->dest != dest)) {
      anterior = timers;
      for (mytimer = timers;
          (mytimer->numseq != numseq) || (mytimer->dest != dest);
          anterior = mytimer, mytimer = mytimer->next);
      anterior->next = mytimer->next;
      if (mytimer->next != NULL)
       (mytimer->next)->faltau += mytimer->faltau;
      die = mytimer;
      free(die);
    }
    else {
      die = timers;
      timers = timers->next;
      if (timers) {
       timers->faltau += die->faltau;
      } else {
            gettimeofday(&tv, NULL);
            fprintf(logfile, "Termino %ld.%ld\n", tv.tv_sec, tv.tv_usec); }
      free(die);
    }
  }
}
```

FIGURA 6.21 – Função *removetimer( )*

Se o *timer* a ser removido é o primeiro da fila, o segundo *timer* recebe o tempo do primeiro. Pode acontecer de o segundo *timer* apresentar valor negativo, já que a precisão do *signal* é menor que a do relógio. Se o tempo for negativo, deve-se verificar quais dos *timers* seguintes também já expiraram. Um *timer* expirou se o valor dele somando com o anterior é menor que zero. É feita esta verificação de tempo até que o

primeiro *timer* da fila seja positivo. Este procedimento é de extrema importância para o controle de *timeout* e é realizado em conjunto com a função *verificatimer( )*.

6.4 Experimentos

Esta Seção apresenta experimentos realizados com o protocolo INFIMO_TAP. O objetivo destes experimentos é verificar a carga do protocolo, ou seja, analisar o tempo de execução do mesmo. Com base nos valores destes experimentos pode-se avaliar o quanto a presença do injetor pode vir a sobrecarregar o sistema.

Nos experimentos descritos nesta Seção somente o protocolo alvo INFIMO_TAP é executado, isto é, apenas a troca de pacotes e o controle temporal do recebimento do ACK são considerados.

As métricas utilizadas na análise experimental das ferramentas de injeção de falhas do INFIMO são cobertura de falhas e intrusão, que pode ser temporal ou espacial. Dentre estas métricas, a única utilizada no INFIMO_TAP é a intrusão temporal, já que o protocolo está livre da injeção deliberada de falhas. A intrusão temporal é dada pela carga imposta pelas operações de comunicação, medida através do *tempo de execução do protocolo*.

A execução destes experimentos permite medir a carga imposta pelo protocolo alvo em si. Para os mesmos números de processos e pacotes enviados, são realizados dois experimentos: um local e um distribuído. No experimento local todos os processos participantes do protocolo executam em uma única máquina. Já no experimento distribuído, os processos executam em máquinas diferentes.

O objetivo do experimento local é avaliar a carga de vários processos em uma única máquina. Este experimento supõe a carga imposta em uma execução distribuída onde diversos processos encontram-se numa mesma máquina.

O reuso dos dados de entrada destes experimentos para submissão do protocolo a uma execução sujeita a injeção de falhas visa a obtenção de resultados, os quais viabilizam uma análise intrusiva do injetor sobre o protocolo alvo (INFIMO_TAP). Para esta avaliação podem ser usadas as ferramentas de injeção de falhas INFIMO_LIB ou INFIMO_DBG. Os experimentos conduzidos com estas ferramentas realizam esta análise, conforme mostram as Seções 7.4 e 8.5.

TABELA 6.4 – INFIMO_TAP: Tempos de Execução (em ms)

	Pacotes	**Ambiente local**	**Ambiente distribuído**
Processo 1	27	40,689	40,174
Processo 2	30	42,601	39,050
Processo 3	24	46,589	33,107

A Tabela 6.4 exibe a média dos tempos de execução (em milisegundos) relativos aos experimentos local e distribuído, considerando os mesmos dados de entrada. Os

experimentos do INFIMO_TAP consideram a presença de 3 processos participantes do protocolo. Cada processo gera um número randômico, o qual representa o número de pacotes que cada um deve enviar. O valor definido para o *timeout* é de 500 milisegundos, ou seja, se após 500 milisegundos não chegar o ACK do referido pacote, o *timeout* expira.

De acordo com a tabela 6.4, os valores de tempo apresentados no experimento local são mais altos que os do experimento distribuído. Isto pode ser explicado porque enquanto no experimento distribuído tem-se três processos concorrentes (apesar de compartilharem o meio de comunicação), no experimento local tem-se três processos disputando o escalonamento do único processador que compartilham. Com isso, no experimento local, ao se mandar um pacote do processo 1 para o processo 2, o processo 1 deve enviar o pacote e o processo 2 somente irá receber quando for escalonado. No experimento distribuído apesar de todos os processos disputarem o acesso ao meio (barramento), o tempo de transmissão é baixo. Neste caso pode-se afirmar ainda que poucas colisões ocorreram.

7 Injetor de falhas através de bibliotecas: INFIMO_LIB

A ferramenta de injeção de falhas INFIMO_LIB (***In****trusiveless* ***F****ault* ***I****njector* ***MO****dule – by* ***LIB****rary modification*) prevê a modificação das bibliotecas de comunicação utilizadas como suporte à implementação do protocolo alvo. Logo, a introdução das falhas ocorre por meio de alterações nas bibliotecas no nível da aplicação.

Este Capítulo é dividido em quatro Seções. Na Seção 7.1 é introduzido o injetor INFIMO_LIB. A Seção 7.2 apresenta a descrição do funcionamento do injetor de falhas. A Seção 7.3 descreve aspectos de implementação do injetor de falhas, ou seja, através de que alterações nas funções da biblioteca JRTPLIB foi possível o desenvolvimento do INFIMO_LIB. A Seção 7.4 exibe alguns experimentos conduzidos com o INFIMO_LIB. Tais experimentos contemplam as métricas cobertura de falhas, intrusão temporal e intrusão espacial.

7.1 INFIMO_LIB

O protocolo alvo INFIMO_TAP foi construído sobre a biblioteca JRTPLIB [JRT99], que por sua vez é suportada pelo protocolo de transporte RTP [SCH97]. Com isso, pode-se utilizar da facilidade de alteração das funções da biblioteca JRTPLIB para emular a ocorrência de falhas no protocolo que roda sobre a mesma. Esta abordagem conta com a facilidade da disponibilidade do código fonte do protocolo alvo, embora seja somente imprescindível que se tenha acesso ao código fonte da biblioteca que implementa o protocolo. A Figura 7.1 esboça a interação entre o nível de usuário e o nível do *kernel* no que refere ao protocolo alvo, JRTPLIB, RTP e o sistema operacional.

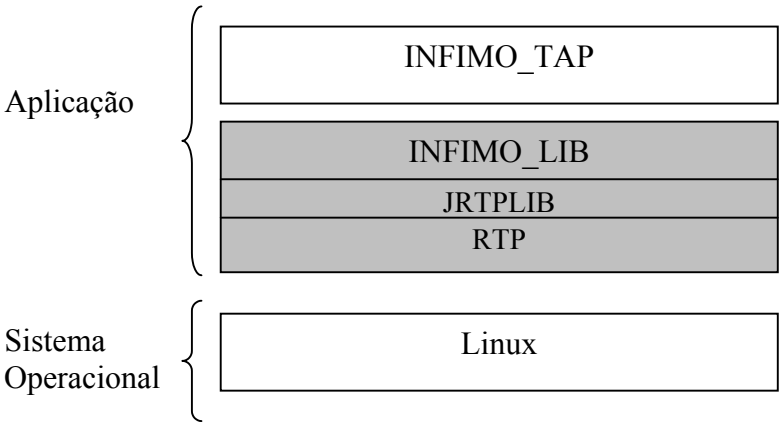

FIGURA 7.1 – INFIMO_LIB

De acordo com a Figura 7.1, tanto a ferramenta de injeção de falhas INFIMO_LIB como a biblioteca JRTPLIB e o protocolo RTP localizam-se no nível da aplicação. Com isso, o protocolo alvo, ou seja, o INFIMO_TAP, ao executar qualquer operação de comunicação, deve acessar as funções da biblioteca JRTPLIB ou as funções de injeção de falhas do INFIMO_LIB, dependendo se a rodada de experimento

prevê ou não a injeção de falhas. Rodadas de experimento que prevêem a injeção de falhas devem, obrigatoriamente, executar as funções do INFIMO_LIB. Entretanto, rodadas cuja especificação não prevê a injeção de falhas podem executar somente as funções da biblioteca JRTPLIB ou as funções do INFIMO_LIB. Esta última situação se aplica quando se deseja executar o protocolo alvo com carga de injeção nula.

O INFIMO_LIB, conforme explicado nas Seções seguintes, é composto pelo código do protocolo alvo (INFIMO_TAP) acrescido do código que implementa o injetor de falhas.

A Figura 7.2 exibe o comportamento do injetor de falhas INFIMO_LIB. Um processo participante do protocolo INFIMO_TAP, ao executar uma operação de comunicação é desviado para as funções do INFIMO_LIB. Estas funções consistem na modificação das bibliotecas originais da JRTPLIB a fim de possibilitar a injeção de falhas. Da mesma forma que as funções da biblioteca JRTPLIB, as funções do INFIMO_LIB viabilizam a comunicação por meio de chamadas de sistema.

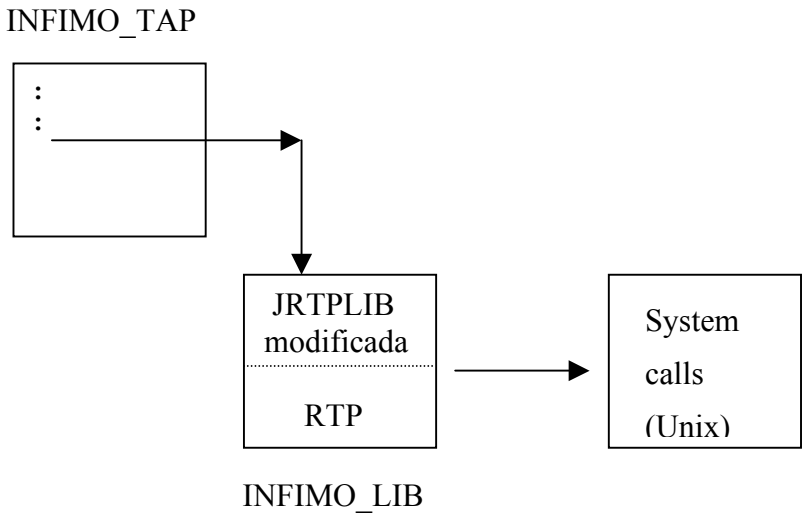

FIGURA 7.2 – Execução do INFIMO_LIB

O escopo de falhas a ser implementado pelo INFIMO consiste em falhas de comunicação. As únicas funções da JRTPLIB alteradas em benefício da injeção de falhas são as que implementam o envio de pacotes. O injetor de falhas do INFIMO_LIB somente atua na operação de envio de pacotes. A recepção está livre da injeção de falhas.

7.2 Descrição do Injetor INFIMO_LIB

O INFIMO_LIB é um injetor de falhas implementado através de alterações nas funções da biblioteca JRTPLIB sobre a qual está implementado o protocolo alvo (INFIMO_TAP). No protocolo INFIMO_TAP, conforme explicado na Seção 6.2, existe

um conjunto de processos participantes, cada um dos quais é composto por um conjunto de *threads*. No INFIMO_LIB também existe um conjunto de processos participantes, onde cada um possui um conjunto de *threads*. Entretanto, enquanto os processos do INFIMO_TAP utilizam as funções da biblioteca JRTPLIB, os processos do INFIMO_LIB fazem uso de funções modificadas da JRTPLIB, as quais são referenciadas como funções do INFIMO_LIB.

Pode-se afirmar que o INFIMO_TAP corresponde a um protocolo de troca de pacotes, enquanto o INFIMO_LIB corresponde ao INFIMO_TAP acrescido de atividades de injeção de falhas. Logo, o INFIMO_LIB mantém toda a estrutura de funcionamento do INFIMO_TAP. Esta estrutura de funcionamento segue a descrição da Seção 6.2 e refere-se a: disparo dos processos, criação das *threads* principal e de resposta, envio do pacote de inicialização, criação das *threads* de envio, geração do número de pacotes a serem enviados (número randômico), seqüência de envio e reenvio, manipulação dos *timers* e geração de *logs*.

A execução do INFIMO_LIB é disparada pela linha de comando. O disparo do INFIMO_LIB deve ser feito para cada processo participante do protocolo. Assim como no INFIMO_TAP, este procedimento é controlado pelo *gerador de carga de trabalho*, juntamente com a *biblioteca de carga de trabalho*. A Tabela 7.1 exibe os argumentos que devem ser passados ao ser disparado o INFIMO_LIB. É utilizada a passagem de argumentos padrão da linguagem C (ambiente Unix), através do apontador para um vetor de cadeias de caracteres (*argv*) que contêm os argumentos, um por cadeia [KER90].

TABELA 7.1 – Argumentos do INFIMO_LIB

argv	**argumento**	**Descrição**
0	*progr*	injetor a ser executado (INFIMO_LIB)
1	*rank*	posição do processo participante do protocolo
2	*total*	número total de processos participantes do protocolo
3	*tipof*	tipo de falha a ser injetada
4	*disp*	condição de disparo da falha
5	*dispp*	momento de disparo (no tempo ou no espaço)
6	*repet*	duração da falha
7	*numf*	número de falhas a serem injetadas

De acordo com a Tabela 7.1, devem ser informados o nome do injetor a ser executado, a posição do processo (*rank*) e o número de processos participantes do protocolo (*total*). A posição do processo é importante uma vez que o último processo disparado é o responsável pela inicialização do protocolo através do envio do pacote de inicialização. Os argumentos *tipof*, *disp*, *dispp* e *repet* compõem o modelo de falhas a ser validado pelo injetor e encontram-se descritos na Seção 7.3.1. Além disso, deve ser informado o número de falhas a serem injetadas (*numf*).

Ao ser executado, o INFIMO_LIB, através da *biblioteca de falhas*, interpreta o modelo de falhas, possibilitando assim, a execução efetiva da injeção de falhas. Os processos participantes do INFIMO_LIB executam conforme os processos participantes do INFIMO_TAP. Entretanto, de acordo com a especificação do modelo de falhas, a comunicação se dá através da execução de funções da JRTPLIB modificadas, ou seja, são executadas as funções de injeção de falhas do INFIMO_LIB. A coordenação da execução das atividades da JRTPLIB e do INFIMO_LIB é feita pelo *injetor de falhas*. O *injetor de falhas* é o componente da arquitetura do INFIMO (Seção 5.3) que interage diretamente com a *biblioteca de falhas*.

O histórico de ações de cada processo do INFIMO_LIB é armazenado em arquivos de *log* (um arquivo para cada processo). Após o término da execução do injetor, pode-se comparar os dados relativos à execução do INFIMO_LIB com o modelo de falhas introduzido. Para esta comparação são utilizados os arquivos de *log* gerados no decorrer da execução do INFIMO_LIB. De acordo com a arquitetura apresentada na Seção 5.3, estes procedimentos são realizados pelo *coletor de dados*, que atua em conjunto com o *monitor*. O *controlador* é o responsável pelo gerenciamento de todo o procedimento do INFIMO_LIB, desde a inicialização do protocolo alvo até a coleta de dados do experimento.

As métricas utilizadas nesta abordagem são a cobertura de falhas e a intrusão. Na cobertura de falhas utiliza-se como base o número de falhas injetadas e o número de falhas percebidas pelo mecanismo de tolerância a falhas do protocolo, o qual é caracterizado como um mecanismo de detecção de *timeout*. Com base na comparação destes valores, pode-se analisar a cobertura de falhas obtida no experimento.

A intrusão visa analisar, sob os pontos de vista temporal e espacial, a carga imposta pelas atividades de injeção de falhas sobre o protocolo alvo. Para intrusão temporal deve-se examinar os *tempos de execução do protocolo alvo* com e sem a atuação do injetor de falhas, considerando os mesmos dados de entrada. Intrusão espacial analisa o comprometimento do contexto do protocolo após a inserção do injetor de falhas.

Um aspecto a ser considerado na implementação desta abordagem se refere à integridade e, neste sentido, pode-se destacar duas preocupações. A primeira está relacionada à integridade do protocolo alvo, o qual encontra-se sob validação por injeção de falhas. Alterações nas funções da biblioteca utilizadas pelo protocolo alvo não devem interferir no contexto do mesmo. A segunda preocupação está voltada à manutenção da semântica das funções da biblioteca sob a qual está implementado o protocolo, ou seja, a JRTPLIB. Neste contexto, deve-se garantir que as alterações impostas pelas atividades de injeção de falhas não comprometam o comportamento das funções alteradas. A integridade do sistema operacional não consiste em uma preocupação, já que não há possibilidade de interferência sobre o mesmo.

Outro aspecto a ser considerado é a portabilidade da abordagem. A idéia é poder utilizar esta abordagem para validação de outros protocolos alvo, implementados também sobre a JRTPLIB, que apresentem características semelhantes.

Esta abordagem possui a vantagem de disponibilidade do código do protocolo alvo. Isto facilita a injeção de falhas por alteração de funções do próprio protocolo alvo. Entretanto, no caso do INFIMO_LIB, apenas é preciso alterar o código fonte das

funções da JRTPLIB. A abordagem conta ainda com a vantagem de implementação conjunta do protocolo alvo e do injetor de falhas.

Como desvantagem percebe-se a diminuição do controle da carga de trabalho, uma vez que se está implementando o injetor em um nível mais alto. Assim, a carga de trabalho é provavelmente mais alta do que se a implementação fosse desenvolvida em baixo nível, como no nível do sistema operacional.

7.3 Implementação do Injetor INFIMO_LIB

Conforme mencionado anteriormente, o INFIMO_LIB foi implementado através de alterações nas funções da biblioteca JRTPLIB para introduzir as atividades de injeção de falhas. Ao ser chamado, o INFIMO_LIB recebe as informações que compõem o modelo de falhas a ser validado no experimento, conforme descrito na Seção 7.3.1. Este modelo de falhas deve ser passado para a classe *RTPSession* da JRTPLIB, conforme apresentado na Seção 7.3.2.

7.3.1 Modelo de Falhas do INFIMO_LIB

Como pode ser visto na Seção 5.4, o modelo de falhas do INFIMO consiste nas informações a respeito do tipo, localização, disparo e duração da falha. Conforme mostra a Tabela 7.1, estas informações correspondem a argumentos passados pelo projetista do experimento de injeção de falhas através da linha de comando.

As Figuras a seguir mostram trechos de código do INFIMO_LIB referentes aos argumentos do modelo de falhas, a saber: tipo, disparo e duração da falha. Faz parte ainda do modelo de falhas a determinação da localização da falha, porém esta tem como alvo somente os processos participantes do protocolo, conforme explicado na Seção 5.4.

De acordo com a Figura 7.3, para *tipof*, que especifica o tipo de falha a ser injetada, podem ser atribuídas as classes: *crash*, omissão ou temporização, as quais são identificadas pelos valores FTCRASH, FTOMISS e FTDELAY, respectivamente. Um experimento sem falhas pode ser especificado pelo valor FTNONE.

```
if (!strcmp(argv[3], "crash")) {
        *tipof = FTCRASH;
  } else if (!strcmp(argv[3], "omissão")) {
        *tipof = FTOMISS;
  } else if (!strcmp(argv[3], "temporização")) {
        *tipof = FTDELAY;
  } else if ((!strcmp(argv[3], "semfalha")) || (!strcmp(argv[3], "unset")))
{
        *tipof = FTNONE;
  } else {
    fprintf(stderr, "Tipo de falha inválido: %s\n", argv[3]);
    exit(-1);
  }
```

FIGURA 7.3 – Argumento tipo de falha

A Figura 7.4 exibe o trecho de código que determina a condição de disparo da falha, sendo considerados os disparos temporal ou espacial. Portanto, a variável *disp* pode receber os valores FTSPACE ou FTTIME. Ambas as condições de disparo devem ser acompanhadas de um valor inteiro (*dispp*) que especifica o momento no espaço ou no tempo para o disparo da falha. Pode-se citar como exemplo a especificação do valor inteiro 3 para *dispp*, considerando o disparo FTSPACE. Neste caso, uma falha será disparada a cada 3 pacotes enviados pelo processo alvo da injeção de falhas.

```
if (!strcmp(argv[4], "espacial")) {
        *disp = FTSPACE;
        *dispp = atoi(argv[5]);
  } else if (!strcmp(argv[4], "temporal")) {
        *disp = FTTIME;
        *dispp = atoi(argv[5]);
  } else if (! (!strcmp(argv[4], "unset"))) {
        fprintf(stderr, "Condição de disparo desconhecida: %s\n", argv[4]);
  }
```

FIGURA 7.4 – Argumento disparo da falha

O argumento duração da falha é ilustrado na Figura 7.5. Uma falha pode se apresentar de forma transiente ou intermitente. A variável *repet* consiste na definição da repetição da falha, que pode assumir os valores FTTRANS ou FTINTER, para falhas transientes e intermitentes, respectivamente. Para falhas intermitentes é usado o valor do argumento *dispp* como intervalo de repetição da falha.

```
if (!strcmp(argv[6], "transiente")) {
        *repet = FTTRANS;
  } else if (!strcmp(argv[6], "intermitente")) {
        *repet = FTINTER;
  } else if (! (!strcmp(argv[6], "unset"))) {
        fprintf(stderr, "Condição de repetição desconhecida: %s\n",
argv[6]);
  }
```

FIGURA 7.5 – Argumento duração da falha

Os argumentos do modelo de falhas devem ser passados para uma função do INFIMO_LIB que gera o cenário de falhas. Esta função, denominada *SetFaultScenary( )*, foi implementada especialmente para os propósitos do INFIMO_LIB, ou seja, esta função foi acrescentada à classe *RTPSession* da JRTPLIB. A Seção 7.3.2 apresenta a classe *RTPSession*, com suas inclusões e alterações.

A linha de código que executa a chamada à função que gera o cenário de falhas é exibida na Figura 7.6. De acordo com esta chamada, são passados os argumentos tipo de falha (*tipof*), condição de disparo (*disp*), momento de disparo (*dispp*), duração da falha (*repet*) e número de falhas (*numf*). O número de falhas indica quantas falhas devem ser injetadas no experimento. Este valor é útil para o controle da quantidade de falhas a serem ativadas.

```
sessao.SetFaultScenary(tipof, disp, dispp, repet, numf);
```

FIGURA 7.6 – Chamada à função que especifica o cenário de falhas

A chamada à função *SetFaultScenary( )* é feita no corpo do programa principal do INFIMO_LIB. Esta função é descrita na Seção seguinte (Seção 7.3.2).

7.3.2 Classe RTPSession

Nesta Seção são descritas as modificações realizadas nas funções da classe *RTPSession* a fim de implementar a ferramenta de injeção de falhas INFIMO_LIB. As modificações do INFIMO_LIB consistem:

- na inclusão da função *SetFaultScenary( )* à classe *RTPSession* da JRTPLIB;
- na alteração da função *SendPacket( ),* pertencente à classe *RTPSession* da JRTPLIB;
- na inclusão da função *ShouldFail( )* à classe *RTPSession* da JRTPLIB.

Além destas modificações, foram necessários alguns ajustes, como por exemplo nos argumentos passados como parâmetros em funções da JRTPLIB relacionadas.

A Figura 7.7 exibe a função *SetFaultScenary( )*, criada pelo INFIMO_LIB para gerar o cenário de falhas a serem injetadas nos experimentos de injeção de falhas. Nesta função são recebidos os argumentos referentes ao modelo de falhas a ser validado no experimento de injeção de falhas, conforme apresentado nas Seções 5.4 e 7.3.1.

```
void RTPSession::SetFaultScenary(char ptipof, char pdisp, char pdispp, char
prepet, char pnumf)
{
      tipof = ptipof;
      disp = pdisp;
      dispp = pdispp;
      repet = prepet;
      numf = pnumf;
      fprintf(stderr, "tipo: %d disparo: %d pardisp: %d repeticao: %d
numero: %d \n", tipof, disp, dispp, repet, numf);
}
```

FIGURA 7.7 – Função *SetFaultScenary( )*

A Figura 7.8 mostra a função *SendPacket( )* modificada. Esta função é chamada pela função *enviapacote*, a qual pertence ao INFIMO_TAP, descrita na Seção 6.3.

De acordo com a Figura 7.8, com as modificações feitas, a função verifica se o pacote recebido corresponde a dados ou controle. Se o pacote é de tamanho válido e possui dados, é possível injetar falha. Para isso, basta verificar o valor de retorno da função *ShouldFail( )*, o qual identifica se o pacote deve ser omitido (para falhas de

crash ou omissão) ou atrasado (para falhas de temporização). Por definição, os pacotes de ACK, controle e reenvio não estão sujeitos a injeção de falhas.

```
int RTPSession::SendPacket(void *data,int len,unsigned char pt,bool mark, unsigned
long timestampinc)
{
        int status;
        unsigned long curtime;
        char *payload;

        if (!initialized)     return ERR_RTP_SESSIONNOTCREATED;
        if (len > 0)
        {
                if ((payload[0] == 0) && (payload[5] == 0)){
                        if (ShouldFail() != 1) {
                                if (ShouldFail() == 2) {
                                        usleep(5000000);
                                        fprintf(stderr, "dormiu!\n");
                                }
                                status=rtpconn.SendRTPData(data,len,&contribsources,pt,
                                        mark,localinf.seqnum,localinf.timestamp);
                                if (status < 0)       return status;
                        }
                        localinf.timestamp += timestampinc;
                        localinf.seqnum++;
                        localinf.octetcount += (RTPuint32)len;
                        localinf.packetcount++;
                        payload = (char *) data;
                        numpacotes++;
                        rtcpmodule.SentData();
                } else {
                        status = rtpconn.SendRTPData(data,len,&contribsources,pt,
                                mark,localinf.seqnum,localinf.timestamp);
                        if (status < 0)       return status;
                        localinf.timestamp += timestampinc;
                        localinf.seqnum++;
                        localinf.octetcount += (RTPuint32)len;
                        localinf.packetcount++;
                }
        }
        curtime = time(NULL);
        if (rtcpmodule.RTCPCheck(curtime))
        {
                status = rtcpmodule.RTCPRoutine(curtime);
                if (status < 0)       return status;
        }
        return 0;
}
```

FIGURA 7.8 – Função *SendPacket( )*

A Figura 7.9 apresenta a função *ShouldFail( )*. Esta função é responsável por verificar, através de comparações entre os argumentos do modelo de falhas, se os valores especificados correspondem a uma condição válida de falha. Neste sentido, a função consiste de um conjunto de *if's* encadeados que ao final podem retornar os seguintes valores:

- valor de retorno = 0: não deve ser inserida a falha. Esta situação ocorre quando se deseja executar o injetor de falhas com máscara de injeção nula;
- valor de retorno = 1: o pacote deve ser omitido. A omissão pode ocorrer uma única vez ou aleatoriamente. O primeiro caso retrata uma falha de *crash*, enquanto o segundo especifica falha por omissão;

- valor de retorno = 2: o pacote deve ser atrasado. Com isso, determina-se uma falha de temporização por atraso.

```
int RTPSession::ShouldFail(void)
{
       if (disp == FTSPACE) {
              if ((repet == FTTRANS) && (numpacotes == (dispp-1)))  {
                     if (tipof == FTNONE) return 0;
                     if ((tipof == FTCRASH) && (numefetfalha < numf)) {
                            fprintf(stderr, "Crash!!\n");
                            numefetfalha++; return 1;}
                     if ((tipof == FTOMISS) && (numefetfalha < numf)) {
                            fprintf(stderr, "Omitindo!! %d\n", numpacotes);
                            numefetfalha++; return 1;}
                     if ((tipof == FTDELAY) && (numefetfalha < numf)) {
                            fprintf(stderr, "Atrasando!!\n");
                            numefetfalha++; return 2;}
              }
              if ((repet == FTINTER) && ((numpacotes % dispp) == (dispp -
1))) {
              if (tipof == FTNONE) return 0;
              if ((tipof == FTOMISS) && (numefetfalha < numf)) {
                     fprintf(stderr, "Omitindo!! %d\n", numpacotes);
                     numefetfalha++; return 1;}
              if ((tipof == FTDELAY) && (numefetfalha < numf)) {
                     fprintf(stderr, "Atrasando!!\n");
                     numefetfalha++; return 2;}
              }
       }
       return 0;
}
```

FIGURA 7.9 – Função *ShouldFail( )*

Conforme exibe a Figura 7.9, a função *ShoudFail( )* testa se o disparo da falha é espacial. O suporte a modelos de falhas com disparo temporal caracteriza uma forma não determinística de injeção de falhas, o que dificulta a percepção dos efeitos da falha injetada.

7.4 Experimentos

Diversos experimentos de injeção de falhas foram realizados, usando um conjunto representativo dos modelos de falhas descritos na Seção 5.4. Os experimentos conduzidos consideram o valor *default* do *timeout*, especificado no INFIMO_TAP. Este valor é de 500 milisegundos. A Seção 7.4.1 exibe experimentos que apresentam a cobertura de falhas obtida em relação ao número de falhas e ao número de processos. A Seção 7.4.2 mostra experimentos que descrevem o comportamento intrusivo, sob o ponto de vista temporal, das atividades de injeção de falhas. A Seção 7.4.3 avalia a intrusão espacial imposta pelo injetor de falhas INFIMO_LIB.

7.4.1 Cobertura de Falhas

A cobertura de falhas é usada para quantificar a eficiência dos mecanismos de tolerância a falhas. Com cobertura de falhas busca-se descobrir as limitações dos mecanismos de tolerância a falhas. Neste sentido, ao serem injetadas as falhas, a métrica de cobertura de falhas especifica quantas delas foram percebidas pelo sistema. Nos experimentos descritos nesta Seção, o objetivo é observar as limitações do mecanismo de detecção do *timeout* implementado pelo protocolo alvo. Com isso, avalia-se se o aumento tanto no número de falhas injetadas como no número de processos participantes do protocolo interfere na cobertura de falhas obtida pelo mecanismo de detecção do *timeout*.

São apresentados dois experimentos considerando a cobertura de falhas. O primeiro refere-se à cobertura e número de falhas, enquanto o segundo relaciona cobertura e número de processos. Estes experimentos foram executados de forma distribuída e encontram-se descritos nas Seções 7.4.1.1 e 7.4.1.2. Ambos os experimentos são baseados no modelo de falhas exposto na Tabela 7.2.

TABELA 7.2 – Modelo de falhas dos experimentos de cobertura

Argumento	**Descrição**
Tipo de falha	Falhas por omissão (falhas simples)
Localização da falha	Processo
Disparo da falha	Disparo espacial (a cada 3 pacotes)
Duração da falha	Falhas intermitentes

De acordo com a Tabela 7.2, nesses experimentos somente são consideradas falhas por omissão. São consideradas falhas simples, ou seja, somente uma falha deve ser ativada por vez. Apenas um dos processos é alvo das falhas. A condição de disparo da falha é espacial, ou seja, uma determinada condição (estado ou endereço) ativa a falha. A duração das falhas é intermitente, isto é, as falhas devem ser disparadas a cada três pacotes enviados pelo processo alvo da injeção de falhas até que se alcance o número de falhas desejado (*numf*).

7.4.1.1 Cobertura x Número de Falhas

O experimento que relaciona a cobertura com o número de falhas avalia se o aumento no número de falhas injetadas interfere na cobertura de falhas obtida pelo mecanismo de detecção do *timeout*.

Considerando um número constante de processos, este experimento deve injetar um determinado número de falhas (*numf*) e verificar quantas foram detectadas.

O número de processos envolvidos é 4, onde cada processo é capaz de enviar 35 pacotes para os demais. O processo alvo da injeção de falhas é o processo 2. Para os experimentos de cobertura a identificação de qual processo é o alvo da injeção de falhas é irrelevante. A idéia é aumentar o número de falhas e observar o comportamento do

protocolo alvo. As falhas não ocorrem simultaneamente, mas em uma seqüência especificada por um determinado número de iterações (argumento *dispp*). A Tabela 7.3 mostra as médias e os percentuais de cobertura obtidos para a situação descrita anteriormente.

TABELA 7.3 – INFIMO_LIB: Cobertura x Número de Falhas

Falhas injetadas	Falhas detectadas	Cobertura
1	1	100%
2	2	100%
4	4	100%
8	8	100%
12	12	100%
15	15	100%
16	15,99	99,98%
20	18,79	93,95%
25	22,18	88,72%

Conforme pode ser observado na Tabela 7.3, o argumento *numf*, o qual identifica o número de falhas injetadas no experimento varia de 1 a 25 falhas. Com até 15 falhas injetadas o número de falhas detectadas é o mesmo que o número de falhas injetadas. A cobertura é de 100%.

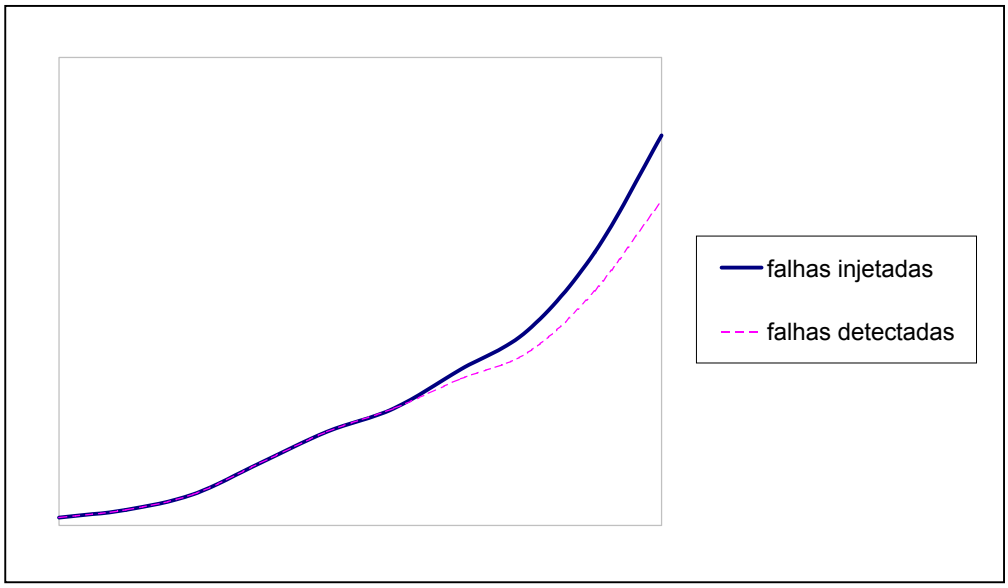

FIGURA 7.10 – INFIMO_LIB: Cobertura x Número de falhas

O experimento mostra o limite de falhas injetadas em um dos 4 processos participantes do protocolo (processo 2), supondo a situação em que cada processo envia

35 pacotes. Quando o número de falhas injetadas alcança 16, o número de falhas detectadas começa a diminuir. Esse comportamento é confirmado para 20 e 25 falhas. O gráfico da Figura 7.10 mostra esta situação.

O *timeout* possui valor *default* de 500 milisegundos. A cada envio de pacote é fixado um *timer*, cujo objetivo é controlar o *timeout* (Seção 6.3.2). É gerada uma fila de *timers* (*timers queue*) por processo participante do protocolo. Quanto mais falhas são injetadas, maior a complexidade na manipulação da fila de *timers*. A detecção da falha em um pacote A, através do estouro do *timeout* implica na retirada do *timer* do pacote A da fila, no reenvio de um novo pacote A (cópia do pacote original com bit de retransmissão ligado) e na inserção do *timer* do novo pacote no final da fila de *timers*.

Neste sentido, o experimento mostra que o protocolo alvo não tem tempo suficiente para detectar e manipular falhas acima de um determinado valor. O conhecimento das limitações deste mecanismo de detecção ajuda o projetista a estabelecer estratégias alternativas para alcançar confiabilidade no protocolo alvo.

7.4.1.2 Cobertura x Número de Processos

O experimento que relaciona a cobertura e o número de processos visa verificar a interferência do aumento do número de processos participantes do protocolo na cobertura de falhas obtida pelo mecanismo de detecção do *timeout*. Isto porque, quanto mais processos participando do protocolo, maior a carga do mesmo, já que mais pacotes estarão circulando entre os processos participantes.

Considerando um número constante de falhas injetadas, este experimento deve ser executado com um determinado número de processos, verificando quantas falhas foram detectadas.

TABELA 7.4 – INFIMO_LIB: Cobertura x Número de Processos

Processos	**Falhas detectadas**	**Cobertura**
2	6	100%
3	6	100%
4	6	100%
6	6	100%
9	6	100%
10	5,98	99,66%
15	5,90	98,33%
20	5,79	96,50%
25	5,60	93,83%

O número de falhas a serem injetadas (*numf*) é 6, onde cada falha tem como alvo o processo 2. Conforme mencionado na Seção anterior, esta informação é irrelevante para os experimentos de cobertura. Cada processo envia 15 pacotes para os demais. A

idéia é aumentar o número de processos e observar o comportamento do protocolo alvo. A ocorrência das falhas não é simultânea. A Tabela 7.4 mostra as médias e percentuais obtidos para a situação descrita anteriormente.

De acordo com a Tabela 7.4, com até 9 processos o número de falhas detectadas é o mesmo que o número de falhas injetadas, ou seja, a cobertura é de 100%. Quando o número de processos participantes do protocolo aumenta para 10, o número de falhas detectadas começa a diminuir. Este experimento mostra o limite do número de processos participantes do protocolo alvo, estando um dos processos sujeito a injeção de 6 falhas. Neste caso, o protocolo apresenta cobertura máxima com até 9 processos trocando um total de 15 pacotes cada um.

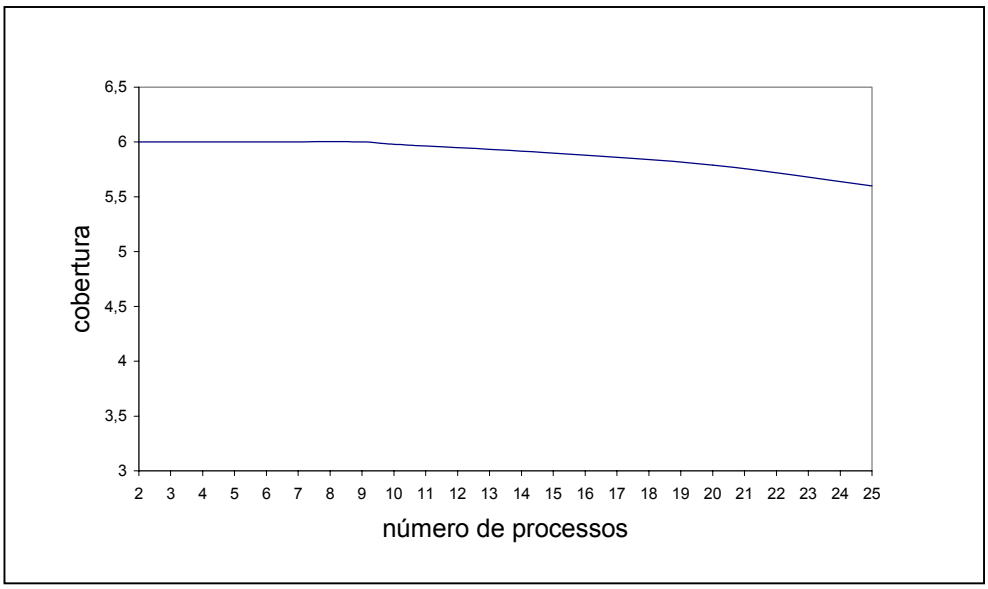

FIGURA 7.11 – INFIMO_LIB: Cobertura x Número de Processos

É necessário aumentar significativamente o número de processos no protocolo para que se obtenha uma redução também significativa na cobertura de falhas. A carga do protocolo depende do número de processos. Quanto maior o número de processos no protocolo, maior a carga do mesmo. Isto ocorre devido ao aumento no número de pacotes os quais devem ser manipulados. Assim, a probabilidade de perda da detecção do *timeout* também aumenta. Com isso, a cobertura de falhas tende a diminuir. O gráfico da Figura 7.11 ilustra esta situação.

7.4.2 Intrusão Temporal

Nesta Seção são descritos alguns experimentos que mostram os *tempos de execução* da ferramenta INFIMO_LIB, ou seja, a carga imposta pelas atividades do injetor de falhas sobre o protocolo alvo (INFIMO_TAP).

Considera-se a situação onde existem 3 processos no sistema: processos 1, 2 e 3. Cada processo gera um número randômico, o qual representa o número de pacotes a

serem enviados aos demais. Os experimentos descritos baseiam-se no modelo de falhas apresentado na Tabela 7.5.

TABELA 7.5 – INFIMO_LIB: Modelo de falhas dos experimentos de intrusão

Argumento	Descrição
Tipo de falha	Falhas por omissão (falhas simples)
Localização da falha	Processo 2
Disparo da falha	Disparo espacial (a cada 2 pacotes)
Duração da falha	Falhas intermitentes

De acordo com a Tabela 7.5 são consideradas falhas por omissão, as quais omitem o envio de alguns pacotes, mas mantêm intacto o contexto dos demais. Considera-se falhas simples, que devem ser ativadas uma a cada vez. O alvo da injeção de falhas é o processo 2. A condição de disparo das falhas é espacial, ou seja, uma falha somente é ativada se uma determinada condição for alcançada. Considera-se ainda falhas intermitentes. As falhas devem ser disparadas a cada dois pacotes enviados pelo processo alvo, até que se alcance o número total de falhas injetadas (*numf*). Cada processo deve enviar um determinado número de pacotes, a saber: processo 1 deve enviar 27 pacotes, processo 2 deve enviar 30 pacotes e processo 3 deve enviar 24 pacotes. Conforme mencionado na Seção 6.2, o destino dos pacotes é irrelevante, uma vez que a injeção da falha é realizada na operação de envio de pacotes.

Para a avaliação da intrusão da ferramenta INFIMO_LIB são descritos diversos experimentos. O primeiro experimento, descrito na Seção 7.4.2.1, é realizado localmente e visa relacionar o protocolo alvo (INFIMO_TAP) com o injetor de falhas INFIMO_LIB inativo. O segundo experimento, apresentado na Seção 7.4.2.2, considera a mesma relação, porém é executado de forma distribuída. Na Seção 7.4.2.3 é analisada a intrusão das atividades de injeção de falhas do INFIMO_LIB. A Seção mostra ainda o atraso de recuperação de uma falha com o INFIMO_LIB.

7.4.2.1 Intrusão do Injetor em Ambiente Local

O experimento avalia a intrusão através de medidas de *tempo de execução*, considerando as seguintes situações:

- execução local do protocolo alvo (INFIMO_TAP);
- execução local do injetor de falhas (INFIMO_LIB) inativo, ou seja, o injetor de falhas está presente no experimento, porém não atua nas atividades de injeção de falhas.

O objetivo deste experimento é verificar se o código do injetor INFIMO_LIB em si representa carga em relação ao tempo de execução do protocolo alvo. Cabe ressaltar que o injetor INFIMO_LIB possui o código do protocolo alvo (INFIMO_TAP) incorporado ao seu próprio código.

Para este experimento apenas interessa a carga imposta pela presença do injetor INFIMO_LIB. Logo, considera-se sua condição inativa. A carga imposta é medida através do *tempo de execução do protocolo*. A literatura refere-se comumente a estes experimentos como experimentos com máscara de injeção nula.

A Tabela 7.6 exibe a média dos valores (em milisegundos) coletados através de execuções exaustivas, supondo o modelo de falhas descrito na Tabela 7.4. Os valores de tempo consideram a média dos tempos de execução gastos para o envio de 27, 30 e 24 pacotes, por parte dos processos 1, 2 e 3, respectivamente.

TABELA 7.6 – INFIMO_LIB: Intrusão local (em ms)

	Processo 1	**Processo 2**	**Processo 3**
Apenas protocolo (INFIMO_TAP)	40,689	42,601	46,589
Protocolo e injetor (INFIMO_TAP + INFIMO_LIB)	40,671	41,960	45,886

De acordo com os valores apresentados na Tabela 7.6, pode-se constatar que a simples presença do injetor de falhas não interfere nos tempos de execução dos processos participantes do protocolo, em especial no tempo de execução do processo alvo da injeção de falhas (processo 2). Desta forma, rodadas que consideram somente a execução do protocolo alvo (INFIMO_TAP) não apresentam valores muito distintos de rodadas que contemplam a execução do protocolo com o injetor (INFIMO_LIB), estando este último sem exercer atividades de injeção de falhas.

A explicação para os valores apresentados está na forma de implementação do injetor. Estando o injetor inativo, o trecho de código a ser executado tanto pelo protocolo como pelo injetor é bastante semelhante. A análise, por parte do injetor, da condição que pode ou não conduzir à execução da função *ShouldFail( )* consiste em uma das diferenças.

A comparação entre o número de pacotes a serem enviados e o tempo gasto para o envio não apresenta relação proporcional. Por ser um ambiente local, todos os processos competem pelo mesmo processador, logo tanto a operação de envio como a operação de recepção de pacotes estão condicionadas ao escalonamento executado pelo processador compartilhado pelos processos participantes do protocolo.

7.4.2.2 Intrusão do Injetor em Ambiente Distribuído

O experimento analisa a intrusão imposta pelas ferramentas, considerando:

- a execução distribuída do protocolo alvo (INFIMO_TAP);
- a execução distribuída do injetor de falhas (INFIMO_LIB) inativo.

Da mesma forma que o anterior, este experimento apresenta a interferência do código do injetor de falhas (INFIMO_LIB) nos tempos de execução do protocolo (INFIMO_TAP), estando o primeiro inativo. A única diferença entre os dois experimentos está no tipo de ambiente. Este experimento considera um ambiente distribuído, ou seja, os três processos encontram-se em máquinas distintas.

A Tabela 7.7 mostra a média dos tempos de execução (em milisegundos) para o envio de 27, 30 e 24 pacotes por parte dos processos 1, 2 e 3, respectivamente. Os valores de tempo apresentados pelo INFIMO_TAP consideram somente a execução do protocolo alvo em um ambiente distribuído. Já os valores do INFIMO_LIB consideram a execução do código do protocolo alvo juntamente com o código do injetor de falhas, também em ambiente distribuído. Entretanto, este último executa com máscara de injeção nula (ou inativo), o que significa que todos os passos para a injeção das falhas são executados, exceto a injeção propriamente dita.

TABELA 7.7 – INFIMO_LIB: Intrusão distribuída (em ms)

	Processo 1	**Processo 2**	**Processo 3**
Apenas protocolo (INFIMO_TAP)	40,174	39,050	33,107
Protocolo e injetor (INFIMO_TAP + INFIMO_LIB)	44,256	32,608	38,029

Da mesma forma que o experimento anterior, a presença do injetor de falhas não necessariamente implica em aumento no tempo de execução. A disparidade de valores advém da característica distribuída do experimento. Com experimento distribuído tem-se três processos concorrentes em um meio de comunicação compartilhado. Portanto, apesar de todos os processos participantes do protocolo disputarem o acesso ao meio, o tempo de execução é razoavelmente pequeno. A causa disso pode ser o baixo número de colisões. Além disso, os processos não estão disputando o escalonamento do mesmo processador. Este experimento foi executado por mais de 50 rodadas, visando obtenção de uniformidade de valores, conforme explicado na Seção 5.7.

7.4.2.3 Intrusão da Injeção de Falhas em Ambiente Distribuído

Para o mesmo modelo de falhas dos experimentos anteriores, o experimento da intrusão da injeção de falhas em ambiente distribuído considera:

- a execução distribuída do INFIMO_LIB com máscara de injeção nula, ou seja, sem atividade de injeção de falhas;
- a execução distribuída do INFIMO_LIB, com aumento progressivo do número de falhas injetadas (1, 2, 4, 8 e 12 falhas).

O experimento exibe a carga imposta pelas atividades de injeção de falhas. Esta carga possui como alvo o processo 2 e é medida através dos tempos de execução.

Conforme mostra a Tabela 7.8, para os processos 1 e 3, os tempos de execução obtidos para o envio de 27 e 24 pacotes respectivamente, mantêm valores aproximados (valores em milisegundos). Entretanto, considerando o processo 2, alvo da injeção de falhas, percebe-se que os valores de tempo apresentados sofrem um elevado acréscimo em função das atividades de injeção de falhas. Com isso, pode-se constatar que para todos os experimentos, os quais introduzem de 1 a 12 falhas, o *timeout* expirou. Isto significa que, para cada falha injetada foi necessário: copiar o pacote, remover seu *timer* da fila de *timers*, reenviar o pacote (com o bit de retransmissão ligado) e adicionar o novo *timer* do pacote no final da fila de *timers*.

TABELA 7.8 – INFIMO_LIB: Intrusão com injeção de falhas (em ms)

	Processo 1	**Processo 2**	**Processo 3**
Sem falha	44,256	32,608	38,029
1 falha	41,324	666,641	38,859
2 falhas	40,698	691,139	38,482
4 falhas	40,083	674,899	39,414
8 falhas	40,057	682,603	38,586
12 falhas	40,464	703,460	38,469

A Figura 7.12 exibe o gráfico referente à Tabela 7.8. O gráfico compara a intrusão provocada pelo INFIMO_LIB em atividade. São contrastados os processos livres de falhas (processos 1 e 3) com o processo alvo da injeção de falhas (processo 2).

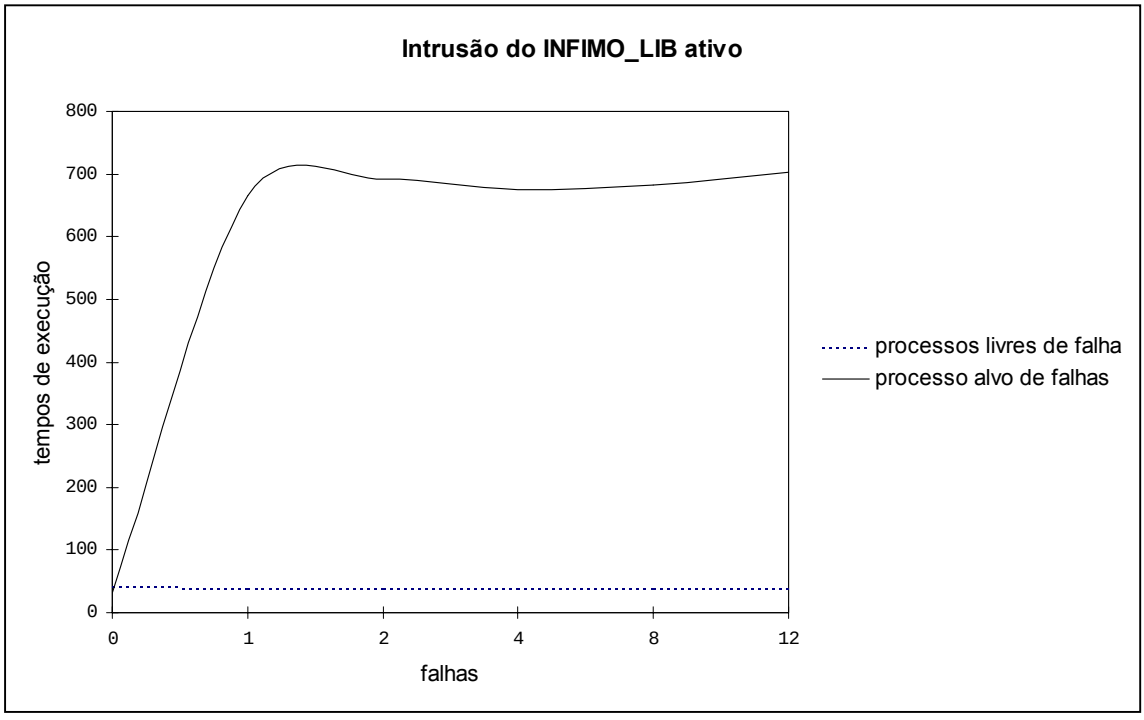

FIGURA 7.12 – Intrusão do INFIMO_LIB ativo

De acordo com a Figura 7.12, o tempo de execução do processo alvo de falhas aumenta significativamente após a inserção da primeira falha. Para as falhas sucessivas a variação no tempo de execução é pequena. O acréscimo no tempo execução deve levar em conta o valor do timeout, que é de 500 ms. Somente após esse tempo são tomadas as providências relativas à recuperação da falha.

```
1  Process 2 started
2  Listening on port 7002
3  Received signal from 3 to start
4  Begin on 978539254.014726
5  To send 11 packets to process 1
6  Sending packet  1- 1
7  Sending packet  1- 2
8  Sending packet  1- 3
9  Sending packet  1- 4
10 Sending packet  1- 5
11 Sending packet  1- 6
12 Sending packet  1- 7
13 Sending packet  1- 8
14 Sending packet  1- 9
15 Sending packet  1-10
16 Sending packet  1-11
17 To send 19 packets to process 3
18 Sending packet  3- 1
19 Sending packet  3- 2
20 Sending packet  3- 3
21 Sending packet  3- 4
22 Sending packet  3- 5
23 Sending packet  3- 6
24 Sending packet  3- 7
25 Sending packet  3- 8
26 Sending packet  3- 9
27 Sending packet  3-10
28 Sending packet  3-11
29 Sending packet  3-12
30 Sending packet  3-13
31 Sending packet  3-14
32 Sending packet  3-15
33 Sending packet  3-16
34 Sending packet  3-17
35 Sending packet  3-18
36 Sending packet  3-19
37 Received ACK  1- 1
38 Received ACK  1- 3
39 Received ACK  1- 5
40 Received ACK  1- 7
41 Received ACK  1- 9
42 Received ACK  3- 1
43 Received ACK  3- 2
44 Received ACK  3- 3
45 Received ACK  3- 4
46 Received ACK  3- 5
47 Received ACK  3- 6
48 Received ACK  3- 7
49 Received ACK  3- 8
50 Received ACK  3- 9
51 Received ACK  3-10
52 Received ACK  3-11
53 Received ACK  3-12
54 Received ACK  3-13
55 Received ACK  3-14
56 Received ACK  3-15
57 Received ACK  3-16
58 Received ACK  3-17
59 Received ACK  3-18
60 Received ACK  3-19
61 Received ACK  1-10
62 Received ACK  1-11
63 Sending packet  1- 2 (re-send)
64 Sending packet  1- 4 (re-send)
65 Sending packet  1- 6 (re-send)
66 Sending packet  1- 8 (re-send)
67 Received ACK  1- 2
68 Received ACK  1- 4
69 Received ACK  1- 6
70 Received ACK  1- 8
71 Finished on 978539254.674294
```

FIGURA 7.13 – INFIMO_LIB: Arquivo de log do processo alvo

A Figura 7.13 mostra um dos arquivos de *log* do processo alvo (processo 2). O arquivo exibe o *log* de um experimento que injeta 4 falhas por omissão no envio de pacotes do processo 2 para o processo 1. No total são enviados 30 pacotes, 11 para o processo 1 e 19 para o processo 3. Dos 11 pacotes que o processo 1 recebe, 4 sofrem falha por omissão.

Na Figura 7.13, a seqüência de números da esquerda indica as linhas do arquivo de *log*. Após receber o sinal de pronto (linha 3), o processo 2 dá início ao envio de pacotes. Na linha 4 é exibido o tempo de início desta rodada de execução. A linha 5 informa a quantidade de pacotes a serem enviados do processo 2 para o processo 1, ou seja, 11 pacotes. Das linhas 6 a 16 são enviados os 11 pacotes para o processo 1.

A linha 17 informa que serão enviados 19 pacotes do processo 2 para o processo 3. Das linhas 18 a 36 são enviados os 19 pacotes. Neste momento encerra-se o procedimento de envio de pacotes por parte do processo 2.

Das linhas 37 a 41, o processo 2 recebe do processo 1 os *ACKs* para os pacotes de números 1, 3, 5, 7 e 9. Das linhas 42 a 60, o processo 2 recebe do processo 3 os *ACKs* referentes aos pacotes de números 1 a 19. Nas linhas 61 e 62, o processo 2 recebe do processo 1 os *ACKs* que confirmam a recepção dos pacotes 10 e 11. Das linhas 63 a 66, o processo 2 reenvia para o processo 1 os pacotes 2, 4, 6 e 8, cujos *ACKs* não foram recebidos e para os quais o *timeout* expirou. Das linhas 67 a 70, o processo 2 recebe os *ACKs* dos pacotes reenviados. A linha 71 exibe o tempo final de execução desta rodada de experimento. A diferença entre os tempos inicial e final indica o tempo gasto na execução do experimento, cujo valor é 659,568 milisegundos.

Atraso de recuperação da falha

O atraso de recuperação da falha é o tempo gasto com o reenvio do pacote que sofreu falha, a recepção do ACK para este pacote e a manipulação de timers. A Tabela 7.9 exibe uma comparação entre os experimentos conduzidos. Assume-se a injeção de uma única falha por omissão em cada rodada de experimento.

TABELA 7.9 – INFIMO_LIB: Atraso de recuperação da falha (em ms)

	Timeout expirou	Fim da execução	Atraso de recuperação
Valor Mínimo	609,224	636,153	10,148
Valor Máximo	699,245	719,422	30,105
Valor Médio	646,489	666,641	20,152
Desvio padrão	40,260	39,676	7,056

Conforme a Tabela 7.9, os valores exibidos na segunda coluna especificam os tempos em que o *timeout* expirou. A terceira coluna apresenta os valores de tempo de "Fim da execução", os quais consideram o término da rodada. Após expirado o *timeout*, o processo alvo reenvia o pacote e aguarda pela confirmação de seu recebimento (*ACK*). A última coluna da tabela exibe os valores de tempo gastos tanto com o reenvio como com a espera pelo *ACK*, referenciado como atraso de recuperação da falha.

A tabela compara os valores mínimo, máximo, médio e desvio padrão para os tempos descritos anteriormente. São consideradas pelo menos 50 rodadas de experimento. Assim, de acordo com a terceira coluna da tabela, o valor mínimo para "Fim da execução" do processo alvo quando submetido a uma falha por omissão é de 636,153 milisegundos. O valor máximo é de 719,422 milisegundos. Ao longo da execução do experimento, o qual consiste em execuções exaustivas, o tempo médio de execução é de 666,641 milisegundos. Este valor também é exibido na Tabela 7.8 (terceira coluna, terceira linha), que especifica os tempos médios de execução dos processos sem a injeção de falhas e com a injeção de 1, 2, 4, 8 e 12 falhas.

Na Tabela 7.9 observa-se ainda que o valor médio do atraso de recuperação da falha é de 20,152 milisegundos (quarta coluna, quarta linha). Com base neste valor, pode-se afirmar que em média, para a injeção de uma única falha por omissão, o tempo gasto com sua recuperação é de 20,152 milisegundos.

Experimentos adicionais mostram que o valor de atraso de recuperação da falha não aumenta proporcionalmente ao número de falhas injetadas. Isso pode ser explicado porque, uma vez acessado o meio de comunicação compartilhado (barramento), os pacotes trafegam em lotes, diminuindo, portanto, o tempo de recuperação de cada falha.

7.4.3 Intrusão Espacial

A interferência do código do injetor de falhas no código do protocolo é referenciada como intrusão espacial. O objetivo de se analisar este tipo de intrusão é avaliar se a inserção do injetor de falhas não está comprometendo a semântica do protocolo alvo. Por inserção do injetor de falhas entende-se a inclusão de código que implementa as atividades de injeção de falhas ou a alteração do código do protocolo de forma a retratar as atividades de injeção de falhas. Ambos os casos implicam em intrusão espacial. Porém, o que se busca saber é se esta intrusão consiste ou não em alterações na semântica do protocolo alvo.

No caso da ferramenta de injeção de falhas INFIMO_LIB, as atividades de injeção de falhas são anexadas às funções da biblioteca JRTPLIB utilizada como base para a implementação do protocolo alvo. O código do protocolo em si não sofre nenhuma interferência. Entretanto, por estar embasado na biblioteca e utilizar-se dela para executar suas próprias funções, deve-se garantir que a semântica das funções da biblioteca não será comprometida.

Analisando o código da ferramenta INFIMO_LIB, pode-se afirmar que o impacto no código das funções da biblioteca é mínimo. São feitas alterações no código de uma função da biblioteca JRTPLIB (função *SendPacket( )*) e são incluídas duas novas funções à biblioteca: função *ShouldFail( )* e função *SetFaultScenary( )*, ambas adicionadas à classe *RTPSession*.

As funções *SetFaultScenary( )* e *ShouldFail( )* foram criadas pelo INFIMO_LIB para especificar e manipular o cenário de falhas a ser validado nos experimentos. Além disso, estas funções verificam se o cenário de falhas corresponde a uma condição válida de falha. A função *ShouldFail( )* é chamada pela função *SendPacket( )*, enquanto a função *SetFaultScenary( )* é chamada pelo INFIMO_LIB em caso de validação por injeção de falhas.

A função *SendPacket( )* é chamada pela função *enviapacote( )*, pertencente ao protocolo INFIMO_TAP. As alterações feitas na função *SendPacket( )* envolvem a verificação do tipo de pacote a ser enviado (dados ou controle). Esta verificação é necessária uma vez que, para o INFIMO_LIB, somente pacotes de dados estão sujeitos a injeção de falhas.

De acordo com a descrição acima, a intrusão espacial nas funções da biblioteca JRTPLIB é pequena. No que se refere à função incluída na biblioteca JRTPLIB (*ShouldFail( )*) pode-se afirmar que há mais impacto temporal do que espacial. Embora esta função tenha que ser inserida no contexto da biblioteca, a característica intrusiva desta inserção é mais acentuada sob o ponto de vista temporal, medida em termos de tempo de execução. As alterações realizadas na função *SendPacket( )* não chegam a comprometer a semântica do protocolo alvo.

8 Injetor de falhas através do *ptrace( )*: INFIMO_DBG

A ferramenta de injeção de falhas INFIMO_DBG (***IN****trusiveless* ***F****ault* ***I****njector* ***MO****dule – using DeBuG resources*) consiste na utilização dos recursos de depuração oferecidos pelo sistema operacional sob o qual o protocolo alvo está implementado. O INFIMO_DBG faz uso do *ptrace( )* do Unix, o qual permite que um processo (injetor de falhas) controle a execução de outro processo (processo alvo).

Este Capítulo divide-se basicamente em cinco Seções. Na Seção 8.1 é introduzido o injetor de falhas INFIMO_DBG. A Seção 8.2 descreve sucintamente o mecanismo *ptrace( )*. Na Seção 8.3 é apresentado o funcionamento do injetor de falhas INFIMO_DBG. A Seção 8.4 descreve aspectos relacionados à implementação do injetor, ou seja, de que forma o depurador do sistema operacional possibilita a injeção de falhas. A Seção 8.5 mostra alguns dos experimentos realizados com o INFIMO_DBG.

8.1 INFIMO_DBG

O controle de execução exercido pelo injetor de falhas permite a injeção das falhas através das chamadas de sistema executadas pelo protocolo alvo, o INFIMO_TAP. Na verdade o injetor de falhas injeta falhas em um dos processos participantes do protocolo alvo, o processo alvo. Embora a utilização do *ptrace( )* em depuradores e em injetores de falhas seja usualmente voltada para introdução de falhas em memória e processador, o INFIMO_DBG faz uso do *ptrace( )* para localizar e manipular chamadas de sistema que implementam operações de comunicação.

O injetor de falhas e o processo alvo correspondem a processos distintos. Com isso, é possível fazer com que o injetor de falhas coordene a execução do processo alvo. Embora se tenha acesso, para a implementação desta abordagem não há necessidade de disponibilidade do código fonte do protocolo alvo, tampouco do código fonte da biblioteca que o implementa (JRTPLIB).

A Figura 8.1 esboça a interação da ferramenta INFIMO_DBG com as demais ferramentas do INFIMO. A base do INFIMO_DBG encontra-se no nível do sistema operacional devido à utilização do *ptrace( )*.

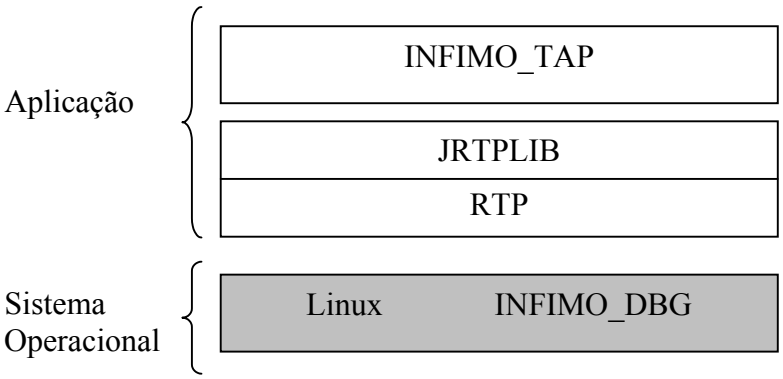

FIGURA 8.1 – INFIMO_DBG

A Figura 8.2 mostra o comportamento do injetor de falhas INFIMO_DBG. O injetor, ao executar uma chamada ao *ptrace( )*, monitora a execução de um dos processos do protocolo alvo (INFIMO_TAP). Isto permite a injeção de falhas bem como a observação do comportamento do protocolo estando um de seus processos sujeito a injeção de falhas. Portanto, sob comando do INFIMO_DBG, o processo alvo do INFIMO_TAP interage com a JRTPLIB para a execução das operações de comunicação, as quais utilizam chamadas de sistema do sistema operacional. O INFIMO_DBG atua diretamente no nível do sistema operacional, interceptando uma chamada de sistema e injetando a falha.

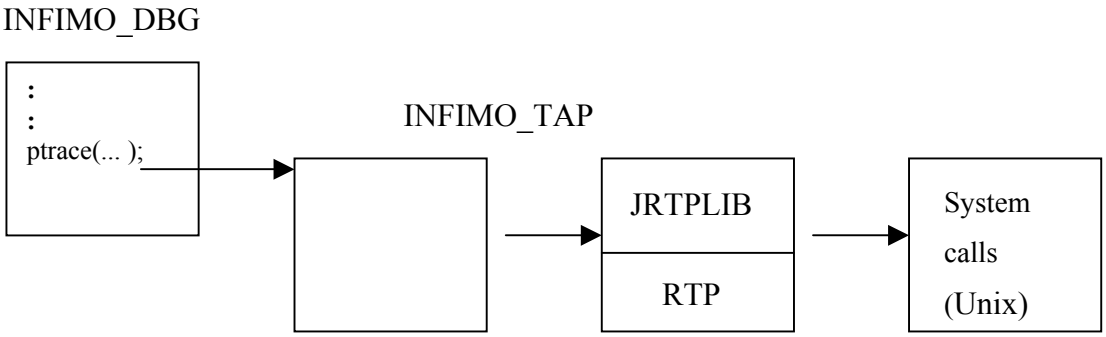

FIGURA 8.2 – Execução do INFIMO_DBG

As ferramentas de injeção de falhas do INFIMO propõem-se a validar falhas de comunicação. As chamadas de sistema as quais devem ser monitoradas e, possivelmente, manipuladas pelo INFIMO_DBG são àquelas que implementam o envio de pacotes.

8.2 *ptrace( )*

Normalmente nenhum processo Unix (ou Linux) tem acesso ao estado interno de outro processo. Os processos são isolados um do outro pelo uso de diferentes endereços físicos, embora compartilhem o mesmo processador. O acesso direto a recursos protegidos de *hardware* como placas controladoras de dispositivos, memória física e certos registradores de controle não é permitido aos processos.

A Figura 8.3 exibe as permissões de acesso relacionadas aos processos de um sistema. A Figura ilustra três níveis distintos [ROS96]: o nível de aplicação, o nível do sistema operacional e o nível do *hardware*. No nível de aplicação encontram-se dois processos, o processo injetor de falhas e o processo que executa o protocolo alvo. O único meio de interação entre os processos do nível de aplicação e o *hardware* é através do sistema operacional. Este, por sua vez, disponibiliza chamadas de sistema que possibilitam esta interação.

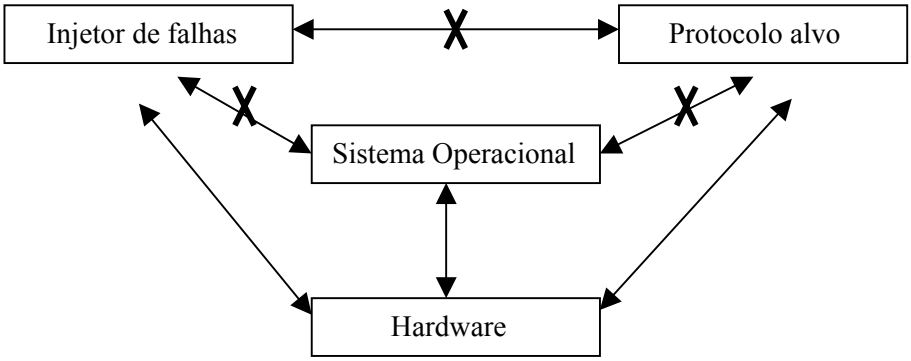

FIGURA 8.3 – Permissões de acesso dos processos

O *ptrace( )* é uma chamada de sistema oferecida pelo Unix, originalmente projetada para permitir a depuração de processos. Um depurador deve ser capaz de ler e modificar as variáveis locais e globais e o apontador de instruções (*instruction pointer*) do processo depurado. O fluxo de dados entre o depurador e o programa testado é verificado pelo *kernel*. Alterações no estado do processo depurado não podem afetar outros processos em andamento. A Figura 8.4 exibe a forma pela qual os recursos do *hardware* tornam-se acessíveis através do depurador do sistema operacional [ROS96].

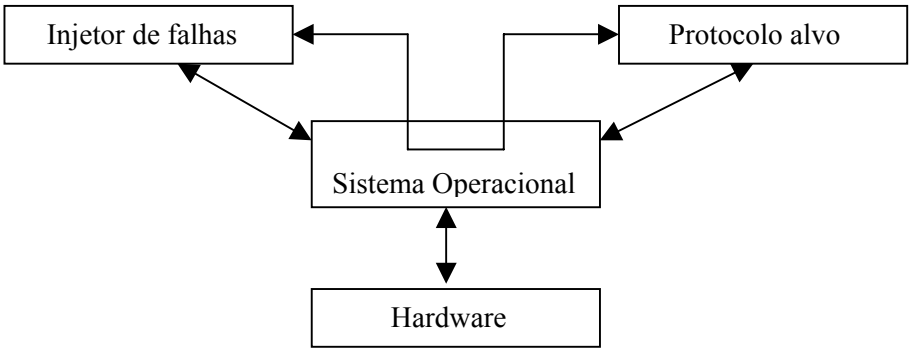

FIGURA 8.4 – Permissões de acesso via depurador

Com *ptrace( )* um processo é capaz de injetar falhas em outro processo. O processo que implementa o injetor de falhas cria um processo filho, o qual executa o programa correspondente ao protocolo alvo da injeção de falhas. O comportamento do processo filho pode ser observado e modificado através do *ptrace( )*.

ptrace( ) oferece diversos recursos de depuração de processos, como execução passo-a-passo, execução até um ponto de parada ou execução até uma determinada chamada de sistema. No que se refere à injeção de falhas, tanto a execução passo-a-passo como a execução até um ponto de parada apresentam desvantagens. Um exemplo de desvantagem é a dificuldade na definição dos pontos de injeção, uma vez que faz-se necessário um criterioso estudo do código do protocolo. Para a execução passo-a-passo,

soma-se ainda a desvantagem relativa ao atraso provocado pelo tempo de processamento exigido. Neste caso, qualquer noção de tempo é perdida.

A implementação da ferramenta de injeção de falhas INFIMO_DBG faz uso da execução do *ptrace( )* até uma determinada chamada de sistema. Assim, utiliza-se o *ptrace( )* para localizar a chamada de sistema que efetua o envio de pacotes entre os processos participantes do protocolo. Através de alterações no comportamento desta chamada pode-se, por exemplo, omitir o envio de todos os pacotes entre os processos.

8.3 Descrição do Injetor INFIMO_DBG

O injetor de falhas INFIMO_DBG adota a técnica de alteração do estado de execução do processo alvo, realizada por dois processos executando concorrentemente. Estes processos são o processo de injetor de falhas e o processo alvo, no qual são injetadas as falhas durante execução. O mecanismo de injeção é manipulado através da interceptação da comunicação entre o processo injetor de falhas e o processo alvo. A interceptação da comunicação permite que o processo injetor altere o estado de execução do processo alvo.

Enquanto no INFIMO_LIB existem somente o processo alvo e os demais processos participantes do protocolo, o INFIMO_DBG apresenta ainda o processo injetor de falhas. O processo injetor de falhas executa concorrentemente ao processo alvo e aos demais processos participantes do protocolo. O processo alvo interage com os processos participantes na troca de pacotes. O processo injetor de falhas intercepta a execução do processo alvo para injetar a falha desejada. Não há interação entre o processo injetor de falhas e os demais processos participantes do protocolo.

A inicialização do INFIMO_DBG é similar a do injetor INFIMO_LIB. A execução do INFIMO_DBG é disparada pela linha de comando. O disparo do INFIMO_DBG deve ser feito para cada processo participante do protocolo. Este procedimento é coordenado pelo *gerador de carga de trabalho*, que atua juntamente com a *biblioteca de carga de trabalho*. Os argumentos passados no disparo do INFIMO_DBG correspondem aos mesmos argumentos do INFIMO_LIB, exceto o nome do injetor a ser executado. A Tabela 8.1 exibe os argumentos do INFIMO_DBG.

TABELA 8.1 – Argumentos do INFIMO_LIB

argv	**argumento**	**Descrição**
0	*progr*	injetor a ser executado (INFIMO_DBG)
1	*rank*	posição do processo participante do protocolo
2	*total*	número total de processos participantes do protocolo
3	*tipof*	tipo de falha a ser injetada
4	*disp*	condição de disparo da falha
5	*dispp*	momento de disparo (no tempo ou no espaço)
6	*repet*	duração da falha
7	*numf*	número de falhas a serem injetadas

Os argumentos da Tabela 8.1 são passados através do *argv*, um apontador para um vetor de cadeias de caracteres que contém os argumentos [KER90]. Os argumentos são: *prog*, *rank*, *total*, *tipof*, *disp*, *dispp*, *repet* e *numf*. O argumento *prog* identifica o injetor de falhas a ser executado. *rank* especifica a posição do processo. Com base na posição do processo pode-se fixar o alvo da injeção de falhas, que geralmente corresponde ao último processo disparado. O número total de processos participantes do protocolo é definido através do argumento *total*. O modelo de falhas é especificado através do tipo de falha (*tipof*), condição de disparo da falha (*disp*), momento do disparo (*dispp*) e duração da falha (*repet*). O número de falhas a serem injetadas é definido pelo argumento *numf*.

Ao ser executado, o INFIMO_DBG, através da *biblioteca de falhas*, interpreta os argumentos do modelo de falhas. Um dos processos disparados pelo sistema assume a responsabilidade de injetar falhas, tornando-se o processo injetor de falhas. O processo injetor de falhas cria um processo filho, o processo alvo da injeção de falhas. Neste contexto, o injetor de falhas corresponde a um processo no nível de aplicação, o qual gera um processo filho, o processo alvo. Com isso, o processo injetor de falhas tem total controle sobre o alvo da injeção de falhas, podendo, portanto atuar sobre o mesmo.

A Figura 8.5 ilustra o funcionamento do INFIMO_DBG. A Figura apresenta um conjunto de processos, composto pelo processo injetor de falhas, o processo alvo da injeção de falhas e os demais processos participantes do protocolo. O processo injetor de falhas cria, através da chamada de sistema *fork( )*, o processo alvo. O processo alvo executa o mesmo código dos demais processos participantes do protocolo, ou seja, o código referente a troca de pacotes. As ações do processo alvo podem ser interceptadas pelo processo injetor de falhas. Através da interceptação o processo injetor de falhas decide, com base no modelo de falhas, sobre as atividades de injeção de falhas.

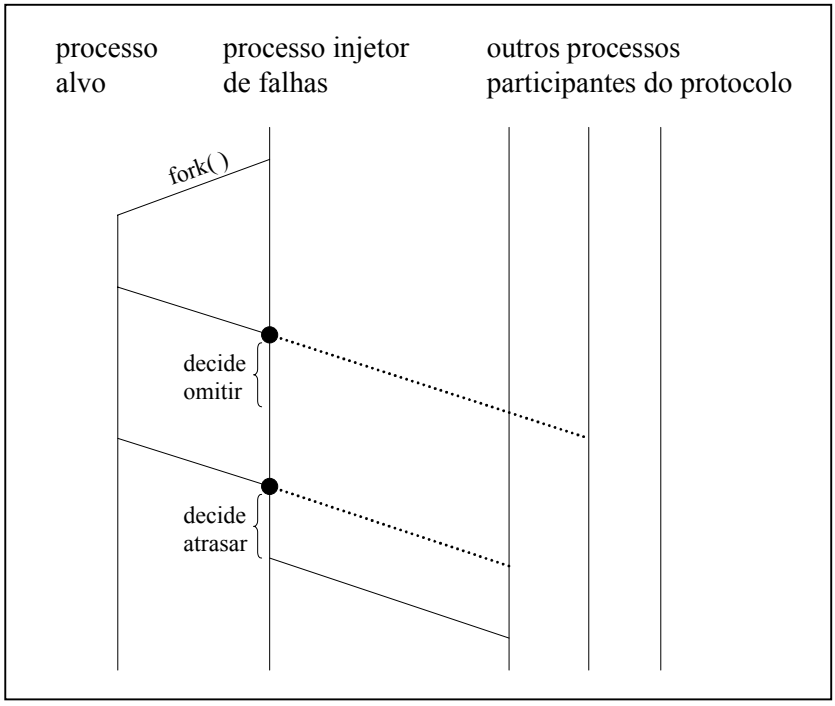

FIGURA 8.5 – Interceptação do processo alvo

Na Figura 8.5 são mostradas duas situações de intercepção do processo alvo por parte do processo injetor de falhas. Na primeira interceptação o injetor de falhas, com base no modelo de falhas, decide pela omissão do pacote enviado pelo processo alvo para um dos processos participantes do protocolo. Esta situação ilustra os tipos de falha *crash* ou omissão. A segunda interceptação o injetor supõe o atraso no envio do pacote. Esta situação mostra o tipo de falha de temporização, para o qual o INFIMO_DBG possui suporte para implementação.

Ao ser executado, o processo alvo recebe do processo injetor de falhas os argumentos referentes ao modelo de falhas, os quais possibilitam a execução efetiva da injeção de falhas. O histórico de ações de cada processo do protocolo é armazenado em arquivos de *log* (um arquivo para cada processo). Após o término da execução do INFIMO_DBG, utiliza-se os arquivos de *log* para comparar os dados relativos à execução do protocolo com o modelo de falhas introduzido. O *monitor* e o *coletor de dados* são os responsáveis por estas funções.

O INFIMO_DBG baseia-se no uso do depurador do sistema operacional através da execução do protocolo alvo da validação até a chamada de sistema *sendto( )*. Esta chamada de sistema é responsável pelo envio de pacotes entre os processos participantes do protocolo. No momento em que encontra a chamada *sendto( )*, o INFIMO_DBG pode alterar os parâmetros da mesma para injetar a falha desejada. A opção de alteração ou não dos parâmetros obedece o modelo de falhas especificado para o experimento. A coordenação das atividades do INFIMO_DBG é realizada pelo *controlador*, de acordo com a arquitetura exibida na Seção 5.3.

Para emular uma falha de *crash* no processo alvo, todos os pacotes oriundos deste processo são desviados para um destino não válido. O mesmo acontece na emulação de uma falha por omissão, entretanto, neste caso o desvio não ocorre para todos os pacotes do processo alvo, porém somente para alguns. O atraso pode ser emulado através da retenção do pacote por um determinado período de tempo. Para isso pode ser utilizado como contador de tempo a função *usleep( )* do Unix.

Para viabilizar a utilização do *ptrace( )* para a injeção de falhas o INFIMO_DBG precisou alterar a estrutura das *threads* do protcolo INFIMO_TAP. A estrutura de *threads* do INFIMO_TAP, apresentada na Seção 6.2, supõe a existência de uma *thread* principal, que cria uma *thread* de resposta, que por sua vez cria *n threads* de envio, uma para cada processo destino. A necessidade do *ptrace( )* atuar sobre o nível mais alto de threads fez com que o contexto das *threads* de envio fosse incorporado a um *loop* da *thread* principal.

Para o INFIMO_DBG cada processo participante do protocolo é composto por uma *thread* principal e uma *thread* de resposta. O processo injetor de falhas, através do *ptrace( )*, pode manipular diretamente as operações de envio de pacotes, uma vez que as mesmas fazem parte da *thread* principal. Esta *thread* é vista pelo *ptrace( )* como um processo e pode, portanto, ser monitorada.

Assim como na ferramenta INFIMO_LIB, as métricas utilizadas são a cobertura de falhas e a intrusão do injetor. Para a cobertura de falhas relaciona-se o número de falhas injetadas e o número de falhas percebidas pelo mecanismo de detecção de *timeout* do protocolo. Alterações no número de falhas injetadas e no número de processos participantes do protocolo possibilitam a realização de uma análise.

Para analisar a intrusão imposta pelo injetor de falhas INFIMO_DBG, utiliza-se o *tempo de execução do protocolo*. Desta forma, obtém-se uma avaliação da intrusão temporal do injetor de falhas. Além disso, pode-se verificar a intrusão espacial do código do injetor sobre o código do protocolo.

Dentre as vantagens da implementação desta abordagem destacam-se duas: a portabilidade e a integridade. Por ser uma ferramenta embasada no padrão Unix e utilizar recursos do próprio sistema operacional para a injeção de falhas, torna-se possível sua utilização para validação de outros protocolos, implementados ou não sobre a biblioteca JRTPLIB, porém compatíveis com o ambiente Unix.

Quanto à integridade pode-se afirmar que o contexto do protocolo alvo não sofre interferência em função da subordinação ao injetor de falhas. Além disso, a abordagem mantém a integridade do sistema operacional sobre o qual está implementado.

8.4 Implementação do Injetor INFIMO_DBG

O INFIMO_DBG foi implementado através dos recursos de depuração oferecidos pelo sistema operacional utilizado como suporte. Mais especificamente, faz-se uso do *ptrace( )* para monitoração e injeção das falhas no processo alvo. Da mesma forma que o INFIMO_LIB, o INFIMO_DBG, ao ser disparado pela linha de comando recebe como argumentos as informações que compõem o modelo de falhas.

Na Seção 8.4.1 é descrito o modelo de falhas do INFIMO_DBG, de acordo com o modelo apresentado na Seção 7.3.1. A Seção 8.4.2 apresenta a implementação do INFIMO_DBG propriamente dita, onde são descritos os detalhes da utilização do *ptrace( )* na implementação do injetor.

8.4.1 Modelo de Falhas do INFIMO_DBG

Sob o ponto de vista de implementação, o modelo de falhas do INFIMO_DBG segue a mesma estrutura do modelo de falhas do INFIMO_LIB, apresentada na Seção 7.3.1.

A implementação do modelo consiste em uma série de "if"s" encadeados, os quais especificam:

- o tipo de falha (*tipof*) a ser injetada, dentre os tipos "*crash*", "omissão", "temporização" e "sem falha";
- o disparo da falha (*disp*), que pode estar condicionada no espaço ("espacial") ou no tempo ("temporal") e deve ser acompanhada do valor condicional para o disparo;
- a duração da falha (*repet*), que pode ser "transiente" ou "intermitente";

A localização do processo alvo da falha é especificada estaticamente. A definição dos componentes do modelo de falhas encontra-se descrita na Seção 5.4.

8.4.2 O ptrace( ) no INFIMO_DBG

A inicialização do injetor INFIMO_DBG se dá com a criação dos processos participantes do protocolo. O INFIMO_DBG deve ser disparado para cada processo participante do protocolo. Este procedimento corresponde a criação dos processos participantes. A partir daí um dos processos participantes assume a função de injetor de falhas, tornando-se assim o processo injetor de falhas.

O processo injetor de falhas cria um processo filho, o qual deve executar o programa correspondente ao processo alvo da injeção de falhas. Este programa é o mesmo que os demais processos participantes executam e corresponde ao mecanismo coordenado de troca de pacotes com controle do *timeout*. O comportamento do processo filho, referenciado como processo alvo, pode ser observado e modificado, sob coordenação do processo pai, o processo injetor de falhas, através do *ptrace( )*. Assim é estabelecida a subordinação do processo alvo ao processo injetor de falhas.

A Figura 8.6 exibe a inicialização do processo alvo. O processo injetor de falhas dispara a criação do processo alvo, pela execução da chamada de sistema *fork( )*. O processo filho, uma vez criado, é submetido à execução do *ptrace( )*. Em resposta a esta função, o *kernel* seta o bit *trace* no PCB (*Process Control Block*) do processo alvo, indicando que o processo está sendo monitorado.

A seguir, o processo alvo executa a chamada de sistema *execve( )*, que carrega o programa a ser executado por ele em memória e inicia sua execução. A partir deste momento o processo alvo substitui, no seu espaço de memória, o código do processo injetor de falhas pelo código do programa que implementa a troca de pacotes. Este código corresponde ao código do INFIMO_TAP, executado pelos demais participantes do protocolo.

```
pid = fork();
if(!pid) {
   ptrace(PTRACE_TRACEME, NULL, NULL, NULL);
   execve("./main", argv, environ);
}
```

FIGURA 8.6 – Inicialização do processo alvo

As linhas de código da Figura 8.7 são executadas pelo processo injetor de falhas após o *fork( )*. Inicialmente o processo injetor chama a função *iniciaargs( )*, responsável pela especificação do modelo de falhas. A função *iniciaargs( )* realiza o *parsing* dos argumentos do modelo de falhas, apresentado na Seção 8.4.1. Em seguida, o processo injetor de falhas aguarda pela sincronização com o processo alvo. Isto ocorre porque quando o processo alvo reconhece que o bit *trace* está setado em seu PCB, ele envia um SIGTRAP para o processo injetor de falhas. Com isso, o processo injetor de falhas é acordado, uma vez que já deve ter executado a chamada de sistema *wait( )*.

```
iniciaargs (argc, argv, &tipof, &disp, &dispp, &repet, &numf);
wait (&status);

ptrace (PTRACE_SYSCALL, pid, NULL, NULL);
```

FIGURA 8.7 – Sincronização entre processos

Ainda na Figura 8.7 inicia-se a monitoração do processo alvo, que está parado desde que enviou o SIGTRAP para o processo injetor de falhas. Este manda o processo alvo executar até alcançar uma chamada de sistema ou um sinal.

A Figura 8.8 apresenta o trecho de código de atuação do *ptrace( )* em benefício da injeção de falhas. De acordo com a Figura, ao alcançar uma chamada de sistema, o processo injetor de falhas copia os valores dos registradores. A seguir, verifica se é entrada ou saída de chamada de sistema. Esta verificação é necessária uma vez que o *ptrace( )* pára na entrada e na saída das chamadas de sistema monitoradas. Para o INFIMO_DBG só interessa a parada na entrada na chamada de sistema. Sendo assim, o teste deve ser feito a cada parada em chamada de sistema. A parada na saída da chamada de sistema pode ser útil para verificação dos efeitos da injeção de uma falha. Esta situação se aplica em falhas cuja latência de detecção é grande.

```
ptrace (PTRACE_GETREGS, pid, NULL, &regs);
if (entrada)
   {
   if (callno == 102)
              {
              if (regs.ebx == 11)
                 {
                 reenvio = ptrace (PTRACE_PEEKDATA, pid, REENVIO_ADDR,NULL);
                 if (!reenvio)
                     {
                     if (shouldfail ())
                {
                        regs.ebx = 19;
                        ptrace (PTRACE_SETREGS, pid, NULL, &regs);
                    }
                     numpacotes++;
                     fprintf(stderr, " numpac: %d\n", numpacotes);
                 }
              }
   }
entrada = 0;
:
```

FIGURA 8.8 – Parada em chamada de sistema

Conforme exibe o trecho de código da Figura 8.8, após verificado que se trata de entrada de chamada de sistema, são realizadas diversas outras verificações. Primeiramente o processo injetor de falhas analisa se a chamada de sistema é uma *socketcall*, já que o objetivo da injeção são falhas de comunicação. Se for *socketcall* e se

tratar de uma chamada de sistema *sendto( )*, deve-se garantir que esta operação não seja de reenvio. O INFIMO_DBG, por definição, não atua sobre pacotes de retransmissão.

Em seguida, o processo injetor chama a função *shouldfail( )*, pertencente ao INFIMO_DBG. O objetivo da função *shouldfail( )*, mostrada na Figura 8.9, é verificar se a injeção de falhas deve ou não ser efetuada neste momento, conforme especificado no modelo de falhas.

De acordo com a Figura 8.8, caso o valor de retorno da função *shouldfail( )* indique a injeção de uma falha, o parâmetro da *socketcall* é modificado para um valor inválido para o envio de pacotes. Com isso, o pacote será suprimido do conjunto de pacotes enviado pelo processo alvo. A próxima parada do processo alvo será em uma saída de chamada de sistema. Logo, o processo injetor deve controlar para que o processo alvo pare somente na próxima entrada de chamada de sistema.

```
int shouldfail (void)
{
  if (disp == FTSPACE)
     {
     if (((repet == FTTRANS) && (numpacotes == (dispp - 1))) ||
         ((repet == FTINTER) && ((numpacotes % dispp) == (dispp - 1))))
       {
         if (tipof == FTNONE)  return 0;
         if ((tipof == FTCRASH) && (numefetfalha < numf))
            {
             fprintf (stderr, "Crash!!\n");
             numefetfalha++;
             return 1;
          }
         if ((tipof == FTOMISS) && (numefetfalha < numf))
            {
             fprintf (stderr, "Omitindo!! %d\n", numpacotes);
             numefetfalha++;
             return 1;
          }
         if ((tipof == FTDELAY) && (numefetfalha < numf))
            {
             fprintf (stderr, "Atrasando!!\n");
             usleep(500000);
             numefetfalha++;
             return 2;
          }
       }
    }
  return 0;
}
```

FIGURA 8.9 – Função *shouldfail( )*

A Figura 8.9 exibe o código da função *shouldfail( )*. Esta função foi implementada na ferramenta INFIMO_LIB com os mesmos propósitos, ou seja, verificar, através de comparações entre argumentos do modelo de falhas, se os valores especificados correspondem a uma condição válida de falha. Para o INFIMO_LIB, entretanto, o código desta função está incorporado à biblioteca JRTPLIB, descrito na

Seção 7.3.2. Atualmente a função *shouldfail( )* do INFIMO_DBG implementa falhas de *crash* e omissão e oferece suporte para falhas de temporização.

O atraso temporal para injetar uma falha usando o *ptrace( )* é aquele exigido para efetuar duas trocas de contexto. A primeira ocorre entre o processo injetor de falhas e o processo alvo, estando o processo injetor pronto para injetar uma falha. A segunda troca de contexto é executada quando o processo alvo termina normalmente ou devido a uma falha. A terminação normal do processo alvo é implementada através da chamada de sistema *exit( )*. Em caso de terminação anormal, o sistema acorda o processo injetor e passa a ele o sinal que termina o processo alvo.

8.5 Experimentos

Da mesma forma que para a ferramenta INFIMO_LIB, foram realizados diversos experimentos de injeção de falhas com o INFIMO_DBG. Os experimentos seguem a determinação do INFIMO_TAP quanto ao valor do *timeout*, que é fixado em 500 milisegundos. Na Seção 8.5.1 são apresentados experimentos que relacionam a cobertura de falhas obtida com os números de falhas e de processos. A Seção 8.5.2 exibe experimentos que descrevem o comportamento intrusivo da injeção de falhas em ambiente local e distribuído. Na Seção 8.5.3 é analisada a intrusão espacial do código do injetor de falhas INFIMO_DBG sobre o protocolo INFIMO_TAP.

8.5.1 Cobertura de Falhas

Os experimentos de cobertura de falhas das ferramentas de injeção de falhas do INFIMO visam avaliar as limitações do mecanismo de detecção do *timeout* implementado pelo protocolo INFIMO_TAP. A cobertura de falhas é analisada sob dois pontos de vista: relacionando o número de falhas e o número de processos.

O primeiro experimento refere-se à cobertura e número de falhas, enquanto o segundo relaciona cobertura e número de processos. Ambos os experimentos do INFIMO_DBG baseiam-se no mesmo modelo de falhas utilizado nos experimentos da ferramenta INFIMO_LIB. Este modelo de falhas, apresentado na Tabela 7.2, consiste basicamente na definição de falhas por omissão simples, as quais possuem como alvo um determinado processo participante do protocolo. O disparo das falhas é espacial, devendo ocorrer a cada três pacotes enviados pelo processo alvo. As falhas possuem duração intermitente, até que seja alcançado o número máximo de falhas definido para o experimento (argumento *numf*) .

8.5.1.1 Cobertura x Número de Falhas

Neste experimento analisa-se se o aumento gradativo do número de falhas interfere na cobertura de falhas fornecida pelo mecanismo de tolerância a falhas do protocolo, ou seja, o mecanismo de detecção do *timeout*.

Para um número constante de processos, injeta-se um determinado número de falhas (*numf*) e verifica-se quantas falhas foram detectadas. O experimento considera 4

processos, cada um dos quais envia 35 mensagens para os demais. O alvo da injeção de falhas é o processo 4, embora esta informação não seja relevante para os experimentos de cobertura. A Tabela 8.2 exibe as médias e os percentuais de cobertura obtidos neste experimento.

TABELA 8.2 – INFIMO_DBG: Cobertura x Número de Falhas

Falhas injetadas	Falhas detectadas	Cobertura
1	1	100%
2	2	100%
4	4	100%
8	8	100%
12	12	100%
15	15	100%
17	16,82	98,94%
20	18,23	91,15%
25	21,97	87,88%

De acordo com a Tabela 8.2, o número de falhas injetadas no experimento (argumento *numf*) varia de 1 a 25 falhas. Considerando a injeção deliberada de até 15 falhas, obtém-se cobertura de 100%, ou seja, o número de falhas detectadas é o mesmo que o número de falhas injetadas. Ao serem inseridas 17 falhas o número de falhas detectadas não acompanha o número de falhas injetadas. Isso ocorre também para 20 e 25 falhas. A Figura 8.10 exibe um gráfico que ilustra este experimento.

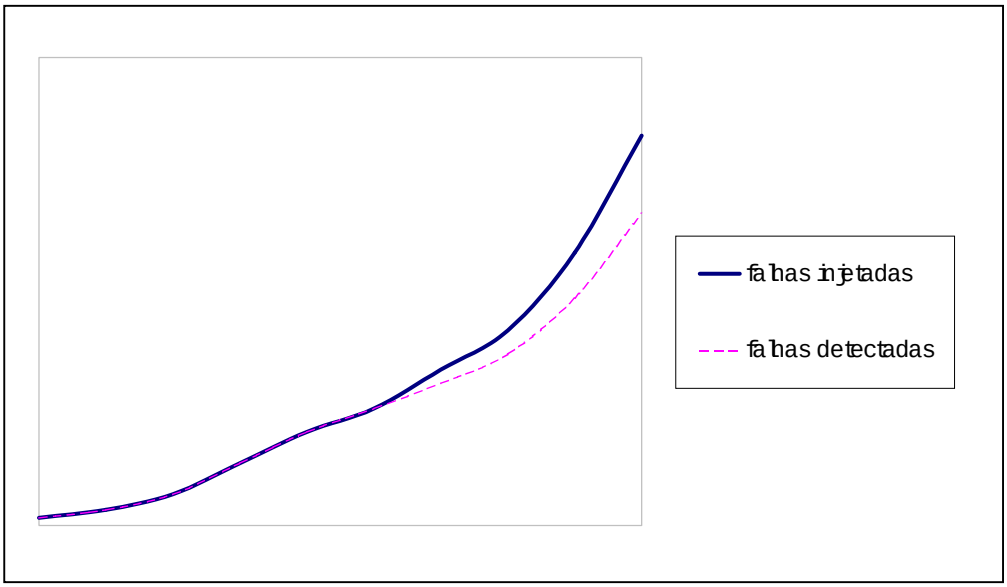

FIGURA 8.10 – INFIMO_DBG: Cobertura x Número de Falhas

A explicação para a diminuição do número de falhas detectadas está relacionada com a definição do valor do *timeout*. Por ser um valor fixo e, portanto, não ser alterado em função do aumento do número de falhas injetadas, o *timeout* expira antes de efetuar a detecção das falhas injetadas. Comportamento semelhante foi verificado nos experimentos do INFIMO_LIB (Seção 7.4.1.1). Uma comparação entre estes experimentos de cobertura pode ser observada na Seção 9.1.1.

Quanto maior o número de falhas injetadas maior a complexidade de manipulação do mecanismo de detecção do *timeout*. Este mecanismo é controlado por *timers*. Cada processo participante fixa um *timer* por pacote enviado, formando uma fila de *timers* (*timers queue*). Ao expirar o *timeout* de um pacote, deve-se copiar o pacote, retirar seu *timer* da fila de *timers*, reenviar o pacote com o bit de retransmissão ligado e adicionar o *timer* do pacote reenviado na fila de *timers*.

Os procedimentos descritos anteriormente devem ser executados a cada falha injetada. O experimento revela as limitações do mecanismo de detecção do *timeout* proposto pelo protocolo INFIMO_TAP.

8.5.1.2 Cobertura x Número de Processos

No experimento que avalia a cobertura em relação ao número de processos é possível observar se o aumento no número de processos interfere na cobertura de falhas obtida pelo mecanismo de detecção do *timeout*. Neste sentido, busca-se saber as limitações do protocolo em termos de aumento de carga.

Da mesma forma que no experimento realizado com a ferramenta INFIMO_LIB (descrito na Seção 7.4.1.2), este experimento considera a injeção de um número constante de falhas e aumenta gradativamente o número de processos participantes do protocolo. Com isso, observa-se a quantidade de falhas detectadas.

O modelo de falhas consiste na injeção de falhas por omissão, cujo alvo é um dos processos participantes do protocolo. As falhas são disparadas espacialmente a cada 3 pacotes enviados pelo processo alvo. A duração das falhas é intermitente até que se alcance o número máximo de falhas especificado para o experimento, definido pelo argumento *numf*.

Considera-se o último processo criado como alvo das 6 falhas injetadas. Assim, para execução com 2 processos, o processo 2 é o alvo da injeção de falhas; com 9 processos o processo 9 é o alvo. Entretanto, conforme mencionado na Seção anterior, esta informação é irrelevante. Os processos enviam 15 mensagens para os demais. No decorrer do experimento vai-se aumentando o número de processos, visando analisar o comportamento do protocolo. A Tabela 8.3 exibe os valores médios e os percentuais obtidos para este experimento.

Conforme exibe a Tabela 8.3, com até 9 processos obtém-se uma cobertura de 100%, ou seja, o número de falhas detectadas é o mesmo que o número de falhas injetadas. Entretanto, quando o número de processos no protocolo aumenta para 10, o número de falhas detectadas começa a diminuir. Este comportamento também pode ser observado nos experimentos com a ferramenta INFIMO_LIB (Seção 7.4.1.2).

TABELA 8.3 – INFIMO_DBG: Cobertura x Número de Processos

Processos	Falhas detectadas	Cobertura
2	6	100%
3	6	100%
4	6	100%
6	6	100%
9	6	100%
10	5,95	99,16%
15	5,83	97,16%
20	5,67	94,50%
25	5,51	91,83%

Da mesma forma que nos experimentos com a ferramenta INFIMO_LIB, a diminuição na cobertura de falhas é bastante lenta. Para que seja observada uma redução significativa na cobertura faz-se necessário um acréscimo também significativo no número de processos do protocolo. Com o aumento na carga do protocolo, a tendência é que a verificação dos *timers* fique prejudicada, o que conduz à perda da detecção do *timeout* por parte de alguns processos. O gráfico da Figura 8.11 ilustra a curva de diminuição da cobertura de falhas.

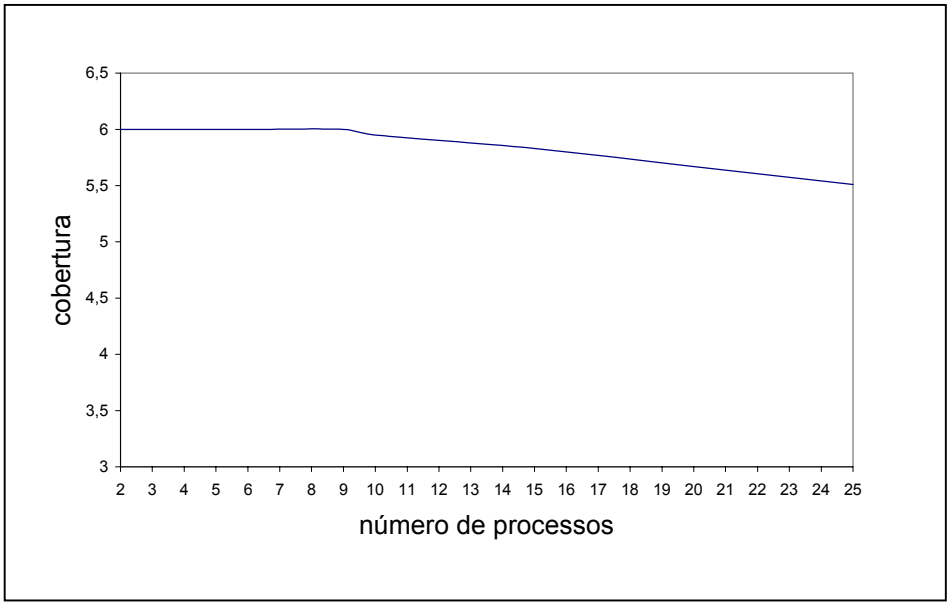

FIGURA 8.11 – INFIMO_DBG: Cobertura x Número de processos

8.5.2 Intrusão Temporal

Assim como com o injetor de falhas INFIMO_LIB, foram realizados diversos experimentos com o injetor INFIMO_DBG. Esta Seção apresenta alguns experimentos

que exibem a carga, medida através do *tempo de execução do protocolo*, imposta pela ferramenta INFIMO_DBG sobre o protocolo INFIMO_TAP.

Os experimentos consideram a existência de 3 processos no sistema, onde cada processo envia um determinado número de pacotes para os demais. Os experimentos baseiam-se no modelo de falhas apresentado na Tabela 8.4.

TABELA 8.4 – INFIMO_DBG: Modelo de falhas dos experimentos de intrusão

Argumento	Descrição
Tipo de falha	Falhas por omissão (falhas simples)
Localização da falha	Processo 3
Disparo da falha	Disparo espacial (a cada 2 pacotes)
Duração da falha	Falhas intermitentes

Conforme descreve a Tabela 8.4, os experimentos consideram falhas por omissão simples, as quais devem ser ativadas uma a cada vez. O processo alvo da injeção de falhas é o processo 3. As falhas são disparadas a cada dois pacotes enviados pelo processo 3, até que se alcance o número de falhas injetadas (*numf*), de acordo com o especificado para o experimento. O processo 1 deve enviar 27 pacotes, processo 2 deve enviar 24 pacotes e processo 3 deve enviar 30 pacotes para os demais. Como nos outros experimentos, os destinos destes pacotes não interessam.

A intrusão do INFIMO_DBG é analisada de acordo com diversos experimentos. O primeiro experimento, descrito na Seção 8.5.2.1, é executado localmente e busca relacionar o protocolo INFIMO_TAP com o injetor de falhas INFIMO_DBG inativo. O segundo experimento, apresentado na Seção 8.5.2.2, considera a mesma relação, porém é executado de forma distribuída. O experimento descrito na Seção 8.5.2.3 visa a injeção de falhas em ambiente distribuído. A Seção exibe ainda o atraso de recuperação da falha por parte do INFIMO_DBG.

Na execução do INFIMO_DBG inativo o injetor de falhas realiza todas as verificações padrão, porém não injeta falhas já que o retorno da função *shouldfail( )* indica execução com máscara de injeção nula.

8.5.2.1 Intrusão do Injetor em Ambiente Local

Com base nas medidas de tempo de execução, o experimento possibilita analisar a intrusão do injetor, considerando:

- execução local do protocolo alvo (INFIMO_TAP);
- execução local do INFIMO_DBG inativo, ou seja, o injetor executa com máscara de injeção nula.

Com este experimento pode-se avaliar o quanto o processamento do injetor de falhas INFIMO_DBG interfere no tempo de execução do protocolo INFIMO_TAP.

A Tabela 8.5 exibe a média dos valores de tempo de execução obtidos (em milisegundos), supondo o modelo de falhas descrito na Tabela 8.4.

TABELA 8.5 – INFIMO_DBG: Intrusão local (em ms)

	Processo 1	**Processo 2**	**Processo 3**
Apenas protocolo (INFIMO_TAP)	40,689	45,589	42,601
Protocolo e injetor (INFIMO_TAP + INFIMO_DBG)	57,983	56,066	55,980

Conforme mostra a Tabela 8.5, pode-se afirmar que a presença do injetor de falhas, embora sem exercer suas atividades de injeção de falhas, implica em acréscimo nos tempos de execução de todos os processos participantes do protocolo, não somente do processo alvo (processo 3). Esta afirmação baseia-se na comparação das execuções do INFIMO_DBG com execuções que consideram somente a execução do protocolo INFIMO_TAP. Neste caso pode-se afirmar que toda a máquina ficou mais lenta, uma vez que todos os processos participantes do protocolo, os quais executam na mesma máquina, tiveram seus tempos de execução aumentados. Quanto mais processos, mais chaveamento de contexto. Estando todos os processos na mesma máquina, todas as atividades são executadas por um único processador.

O INFIMO_DBG utiliza o *ptrace( )*, um mecanismo de monitoramento caracteristicamente lento. Conforme explicado na Seção 8.1, esta característica foi amenizada com a execução contínua do *ptrace( )* até uma chamada de sistema. Outras formas de utilização do *ptrace( )* envolvem a execução passo-a-passo e a execução até um ponto de parada. Ambas as formas apresentam dificuldade na definição do momento da injeção da falha. Além disso, a execução passo-a-passo implica em maior atraso no tempo de execução.

8.5.2.2 Intrusão do Injetor em Ambiente Distribuído

Este experimento avalia a intrusão imposta pelo INFIMO_DBG sobre o protocolo INFIMO_TAP, considerando:

- a execução distribuída do protocolo alvo (INFIMO_TAP);
- a execução distribuída do injetor de falhas (INFIMO_DBG) inativo.

O objetivo deste experimento é analisar a intrusão do processamento do injetor INFIMO_DBG nos tempos de execução do protocolo INFIMO_TAP, estando o injetor com máscara de injeção nula. A diferença entre este experimento e o anterior está no tipo de ambiente. Neste experimento considera-se um ambiente distribuído, onde os três processos encontram-se em máquinas distintas. Na Tabela 8.6 é apresentada a média dos tempos de execução deste experimento, em milisegundos.

TABELA 8.6 – INFIMO_DBG: Intrusão distribuída (em ms)

	Processo 1	**Processo 2**	**Processo 3**
Apenas protocolo (INFIMO_TAP)	40,174	33,107	39,050
Protocolo e injetor (INFIMO_TAP + INFIMO_DBG)	40,051	39,741	40,459

Para este experimento não se pode afirmar que a presença do injetor de falhas implica em aumento no tempo de execução. Os valores de tempo apresentados tanto pelo processo alvo, como pelos demais processos participantes do protocolo são em média razoavelmente próximos. No experimento distribuído, embora todos os processos tenham que disputar o acesso ao meio (barramento), o tempo de execução é geralmente menor que em experimento local. Isso pode ser causado pelo baixo número de colisões. Além disso, os processos executam em máquinas distintas e, portanto, não estão disputando o escalonamento do mesmo processador. O tempo de chaveamento de contexto entre o injetor de falhas e o processo alvo é mascarado pelos tempos de acesso ao meio de todos os processos participantes do protocolo.

8.5.2.3 Intrusão da Injeção de Falhas em Ambiente Distribuído

O experimento da intrusão da injeção de falhas em ambiente distribuído considera:

- a execução distribuída do INFIMO_DBG com máscara de injeção nula, ou seja, sem atividade de injeção de falhas;
- a execução distribuída do INFIMO_DBG, com aumento progressivo do número de falhas injetadas (1, 2, 4, 8 e 12 falhas).

O experimento apresenta a carga imposta pelas atividades de injeção de falhas, considerando o modelo de falhas apresentado na Tabela 8.4. Esta carga possui como alvo o processo 3 e é medida através dos tempos de execução, dados em milisegundos.

De acordo com a Tabela 8.7, os tempos de execução apresentados pelos processos 1 e 2 mantêm valores aproximados. Contudo, considerando o processo alvo da injeção de falhas (processo 3), verifica-se um aumento nos valores de tempo apresentados. Este aumento se deve às atividades de injeção de falhas. Os experimentos consideram um *timeout* com valor *default* igual a 500 milisegundos.

TABELA 8.7 – INFIMO_DBG: Intrusão com injeção de falhas (em ms)

	Processo 1	**Processo 2**	**Processo 3**
Sem falha	40,051	39,741	40,459
1 falha	40,424	39,880	647,589
2 falhas	40,233	39,418	678,759
4 falhas	40,110	39,526	679,683
8 falhas	39,638	39,722	687,102
12 falhas	39,465	38,154	688,042

No caso deste experimento, pode-se afirmar que para todas as quantidades de falhas introduzidas (1 a 12 falhas), o *timeout* expirou. Logo, para cada falha injetada foi necessário copiar o pacote, remover seu *timer* da fila de *timers*, reenviar o pacote (com o bit de retransmissão ligado) e adicionar o novo *timer* do pacote no final da fila de *timers*.

A Figura 8.12 mostra o gráfico referente à Tabela 8.7. Neste gráfico é exibida a intrusão provocada pelo injetor INFIMO_DBG em atividade. Os processos 1 e 2 são processos livres de falhas, enquanto o processo 3 é o processo alvo da injeção de falhas. De acordo com o gráfico, o tempo de execução do processo alvo de falhas aumenta significativamente logo após a inserção da primeira falha. O aumento progressivo no número de falhas introduzidas implica em mínima variação.

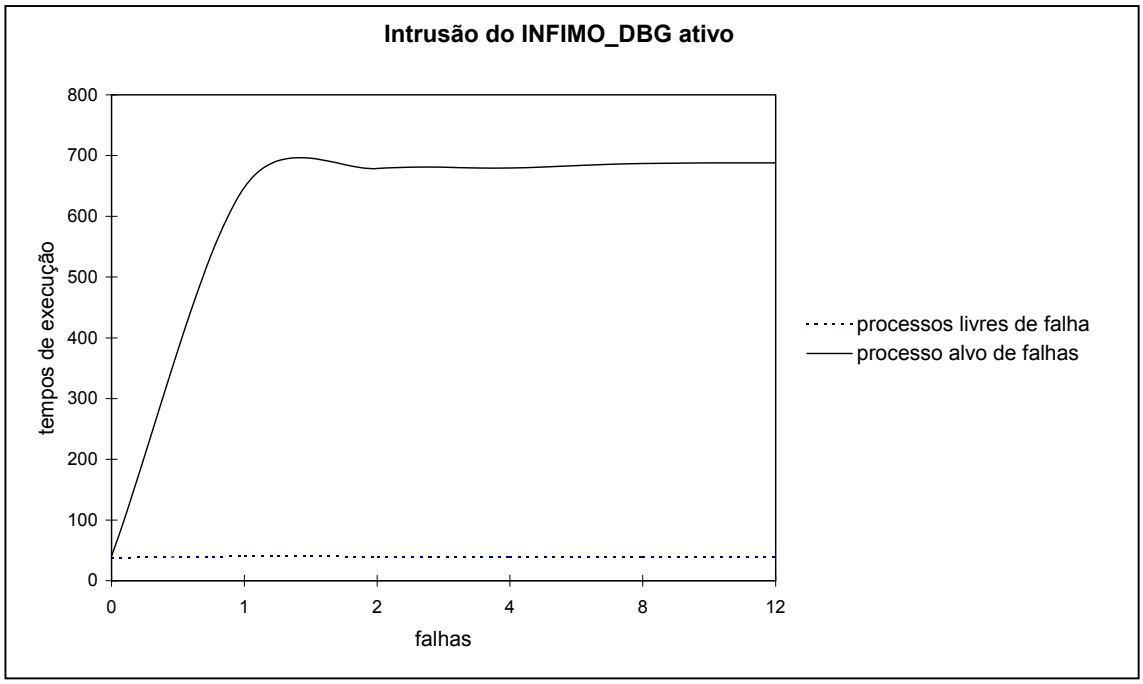

FIGURA 8.12 – Intrusão do INFIMO_DBG ativo

Atraso de recuperação da falha

Este experimento avalia o tempo gasto com o reenvio de um pacote perdido (devido a injeção da falha) e a recepção do ACK para este pacote. O tempo gasto considera ainda as operações de manipulação dos *timers*. Na Tabela 8.8 é exibido o atraso de recuperação da falha (em milisegundos). A Tabela mostra uma comparação entre os experimentos conduzidos, supondo-se a injeção de uma única falha por omissão em cada rodada de experimento.

De acordo com a Tabela 8.8, os valores da segunda coluna referem-se ao tempo em que o *timeout* expirou. Os valores de "Fim da execução", na terceira coluna, consideram o término da rodada, ou seja, após expirado o *timeout*, o processo alvo reenvia o pacote e aguarda pela confirmação de seu recebimento (*ACK*). Os valores da

última coluna da tabela, referenciados como atraso de recuperação da falha, exibem os tempos gastos com o reenvio e com a espera pelo *ACK*.

TABELA 8.8 – INFIMO_DBG: Atraso de recuperação da falha (em ms)

	Timeout expirou	Fim da execução	Atraso de recuperação
Valor Mínimo	619,460	636,339	10,081
Valor Máximo	645,866	656,428	30,117
Valor Médio	627,503	647,589	20,086
Desvio padrão	10,807	9,253	10,007

A tabela apresenta os valores mínimo, máximo, médio e desvio padrão para os tempos descritos anteriormente. O valor médio do atraso de recuperação da falha é de 20,086 milisegundos. Isto significa que em média, para a injeção de uma única falha, o tempo gasto com sua recuperação é de 20,086 milisegundos.

8.5.3 Intrusão Espacial

A intrusão espacial visa identificar se a inserção do código do injetor de falhas está comprometendo a semântica do protocolo alvo. Na ferramenta de injeção de falhas INFIMO_DBG não há necessidade de alteração das funções da biblioteca JRTPLIB, uma vez que os trechos de código que implementam as atividades de injeção de falhas não são incorporados ao protocolo alvo. A injeção de falhas é realizada através do *ptrace( )*.

Entretanto, como o protocolo alvo executa sob comando do *ptrace( )*, fez-se necessário uma modificação na estrutura de criação das *threads* de envio e resposta de pacotes. No INFIMO_TAP, para cada processo existe uma *thread* principal, uma *thread* de resposta e *n*-1 *threads* de envio, onde *n* é o número de processos participantes do protocolo. Para o INFIMO_DBG o código referente às *threads* de envio foi incorporado à *thread* principal. Com isso, o *ptrace( )* pode monitorar diretamente as chamadas de sistema que executam o envio de pacotes. Observou-se que o impacto desta alteração sobre os tempos de execução do protocolo alvo é imperceptível, ou talvez inexistente.

No INFIMO_DBG não há necessidade de execução de código extra no protocolo alvo, tampouco na biblioteca JRTPLIB, a fim de implementar as atividades de injeção de falhas. Entretanto, o processo alvo da injeção de falhas deve permitir ser monitorado pelo processo injetor de falhas, o que se consegue através da subordinação imposta na criação do processo alvo (processo filho) pelo processo injetor de falhas (processo pai).

9 Análise Comparativa

Este Capítulo tem como objetivo o estabelecimento de comparações. Sendo assim, o Capítulo encontra-se dividido em duas Seções. Na primeira Seção são comparadas as ferramentas de injeção de falhas do INFIMO entre si. Além da comparação em relação à cobertura de falhas e intrusão, descritas nos Capítulos anteriores, são analisadas as características de portabilidade e expansão para diferentes tipos de falhas.

Visando a avaliar as ferramentas de injeção de falhas do INFIMO com algumas das ferramentas de injeção de falhas propostas na literatura, na segunda Seção são analisadas características referentes ao mecanismo de injeção de falhas implementado, objetivo da injeção de falhas, modelo de falhas suportado e métricas de análise das ferramentas. As comparações foram feitas com base nas publicações das ferramentas de injeção de falhas, conforme resumo no Anexo.

9.1 Ferramentas de Injeção de Falhas do INFIMO

Esta Seção exibe comparações entre as duas ferramentas de injeção de falhas que, juntamente com o protocolo INFIMO_TAP, compõem o *toolkit* INFIMO. Embora as métricas cobertura de falhas e intrusão tenham sido individualmente exibidas nos experimentos descritos nos Capítulos 7 e 8, esta Seção apresenta um paralelo de tais métricas. Além disso, aspectos relacionados a portabilidade das ferramentas para validação de outros protocolos, bem como a portabilidade para execução em diferentes plataformas também são discutidos. A Seção encerra com a análise da viabilidade de expansão do suporte das ferramentas para diferentes tipos de falhas.

9.1.1 Cobertura de Falhas

Em tolerância a falhas, a medida de cobertura de falhas é de extrema importância, já que possibilita medir, de forma quantitativa, a eficiência do mecanismo de tolerância a falhas implementado no protocolo. As ferramentas de injeção de falhas do INFIMO preocupam-se em avaliar a cobertura de detecção de falhas relacionada ao mecanismo de detecção do *timeout*.

Neste trabalho a cobertura de falhas é analisada sob dois pontos de vista: cobertura x número de falhas e cobertura x número de processos. Experimentos que relacionam cobertura e número de falhas mantêm o mesmo número de processos no sistema e aumentam a quantidade de falhas injetadas. Similarmente, experimentos que avaliam a cobertura e número de processos aumentam o número de processos no sistema, mantendo o número de falhas constante.

Para experimentos que relacionam a cobertura e o número de falhas, as ferramentas de injeção de falhas INFIMO_LIB e INFIMO_DBG apresentam comportamento similar. Com até 15 falhas injetadas, tanto o injetor INFIMO_LIB como o INFIMO_DBG apresentam cobertura de 100%. Para o INFIMO_LIB (Seção 7.4.1.1),

ao serem injetadas 16 falhas a cobertura começa a apresentar redução. Para o INFIMO_DBG esta redução ocorre ao serem injetadas 17 falhas (Seção 8.5.1.1).

Nos experimentos com ambas as ferramentas de injeção de falhas, a diminuição na cobertura de falhas obtida pelo mecanismo de detecção do *timeout* do protocolo alvo se deve à complexidade na manipulação das falhas. Quanto mais falhas são injetadas, mais atividades o protocolo tem que executar. Para cada falha injetada em um pacote deve-se copiar o pacote, retirar seu *timer* da fila de *timers*, reenviar o pacote e adicionar o novo *timer* na fila de *timers*. Os injetores apresentam comportamento similar uma vez que a redução na cobertura se dá na injeção de 16 e 17 falhas para o INFIMO_LIB e o INFIMO_DBG, respectivamente.

Os experimentos que avaliam a cobertura em relação ao número de processos também apresentaram valores similares. Para ambas as ferramentas de injeção de falhas, o aumento na carga do sistema em termos de número de processos começa a interferir na cobertura a partir de execuções com 10 processos participando do protocolo. Isto pode ser visto nas Seções 7.4.1.2 e 8.5.1.2, que apresentam experimentos do INFIMO_LIB e do INFIMO_DBG, respectivamente.

Em ambas as ferramentas, o aumento do número de processos participantes do protocolo implica na manipulação de mais processos e, portanto, mais pacotes, no mesmo período de tempo. Quanto mais pacotes a serem manipulados, maior a complexidade no controle dos *timers* destes pacotes. Os experimentos mostram o limite no número de processos participantes do protocolo para a manutenção de cobertura máxima.

Portanto, a obtenção de altos percentuais de cobertura está relacionada à definição do *timeout*. Para protocolos cuja maior preocupação é a obtenção de alto percentual de cobertura, pode-se relaxar o valor do *timeout*. Entretanto, o relaxamento do valor do *timeout* pode prejudicar a característica tempo real do protocolo. Em contrapartida, protocolos que exigem maior rigor no tempo de detecção devem ser flexíveis quanto aos percentuais de cobertura obtidos ou usar de recursos redundantes de *hardware* para reduzir a probabilidade de falhas. A investigação de um protocolo ideal para aplicação tempo real foge ao escopo do trabalho, que concentra sua atenção em verificar se a injeção de falhas é viável para validação de tais protocolos.

9.1.2 Intrusão

A interferência dos injetores de falhas INFIMO_LIB e INFIMO_DBG sobre o protocolo alvo INFIMO_TAP é observada de forma temporal e espacial. A intrusão temporal corresponde ao aumento no tempo de execução do protocolo alvo devido às atividades de injeção de falhas. Intrusão espacial implica na modificação, por parte do injetor de falhas, do código do protocolo alvo.

Enquanto a intrusão temporal posibilita analisar a interferência nas limitações temporais do protocolo alvo, a intrusão espacial tem como objetivo avaliar o comportamento da semântica do protocolo alvo.

9.1.2.1 Intrusão Temporal

A análise da intrusão temporal dos injetores de falhas foi apresentada considerando ambiente local e distribuído e injetor ativo e inativo. Os itens a seguir comparam os experimentos realizados nas Seções 7.4.2 e 8.5.2. Diferentemente dessas Seções, a análise aqui apresentada somente considera o impacto sofrido pelo processo alvo da injeção de falhas.

a) intrusão do injetor inativo em ambiente local

Estes experimentos visam quantificar, através do tempo de execução, a intrusão da presença do injetor de falhas junto ao protocolo alvo.

Na ferramenta INFIMO_DBG (Seção 8.5.2.1), observa-se que o processo alvo apresenta aumento no tempo de execução quando comparado com valores de tempo apresentados pelo protocolo INFIMO_TAP. Na ferramenta INFIMO_LIB (Seção 7.4.2.1) o tempo de execução do processo alvo apresenta-se estável, quando comparado com o tempo de execução somente do protocolo. Neste contexto, pode-se afirmar que, em ambiente local, a presença do INFIMO_DBG, embora inativo, é mais intrusiva que a presença do INFIMO_LIB.

O INFIMO_LIB apresenta vantagem devido a forma como é implementado. O protocolo alvo e o injetor de falhas fazem parte de um único processo. No INFIMO_DBG, o protocolo e o injetor correspondem a dois processos concorrentes. Neste caso, devem ser considerados os tempos de chaveamento de contexto entre o processo injetor de falhas e o processo alvo. Estes tempos tornam a máquina mais lenta, interferindo, portanto, nos tempos de execução dos demais processos do protocolo.

b) intrusão do injetor inativo em ambiente distribuído

Estes experimentos analisam a intrusão do injetor de falhas inativo sobre o protocolo alvo em ambiente distribuído, estando o injetor inativo.

Para o INFIMO_LIB (Seção 7.4.2.2) os valores de tempo de execução somente do protocolo e do protocolo com injetor inativo mantém-se razoavelmente próximos. Com relação ao experimento anterior, que considera todos os procesos na mesma máquina, este experimento obtém vantagem do não compartilhamento do processador, apesar da disputa pelo acesso ao meio.

Para o INFIMO_DBG (Seção 8.5.2.2) os tempos de execução somente do protocolo e do protocolo com injetor inativo mantém-se bastante próximos. Com relação ao experimento local, este experimento também obtém vantagem do não compartilhamento do processador. Porém, no INFIMO_DBG, onde há chaveamento de contexto entre o processo injetor de falhas e o processo alvo, pode-se afirmar que o tempo exigido na disputa pelo acesso ao meio é mascarado pela disponibilidade dos processadores das máquinas onde se encontram os processos. Em ambas as ferramentas, o baixo número de colisões no acesso ao meio também beneficia o tempo de execução do protocolo.

c) intrusão da injeção de falhas em ambiente distribuído

Os experimentos descritos em 7.4.2.3 e 8.5.2.3 avaliam a intrusão da injeção de falhas em atividade, ou seja, exercendo atividades de injeção de falhas.

Para ambos os injetores, o comportamento tanto dos processos livres de falhas como do processo alvo da injeção de falhas apresenta-se uniforme. O processo alvo executa basicamente as seguintes operações:

- envio de pacotes;
- recebimento de ACK;
- manipulação de *timers* para controle do *timeout*;
- reenvio de pacotes;
- recebimento de ACK do reenvio.

A maioria das operações envolve escalonamento e/ou acesso ao meio. Em ambos os injetores, o processo alvo leva em média 680 milisegundos para efetuar todas as operações, considerando o valor *default* do *timeout*, que é de 500 milisegundos. Os experimentos de injeção de falhas consideram a introdução de 1 até 12 falhas. Em ambas as ferramentas de injeção de falhas a variação nos tempos de execução não é proporcional ao número de falhas injetadas.

d) atraso de recuperação da falha

O atraso de recuperação da falha consiste no tempo gasto com as operações de reenvio do pacote atingido pela falha injetada e espera pelo recebimento do respectivo ACK. Estas operações envolvem ainda a manipulação dos *timers* associados aos pacotes. Em ambos os injetores o atraso gerado é, em média, de 20 milisegundos. Este atraso considera a recuperação de uma única falha. Da mesma forma que no item anterior, a variação no atraso não é proporcional ao número de falhas injetadas.

9.1.2.2 Intrusão Espacial

A intrusão espacial está relacionada à intrusão temporal. Se o injetor de falhas apresenta código adicional a ser executado, estando este incorporado ou não ao protocolo alvo, este código deve ser executado, aumentando o tempo de execução. O aumento no tempo de execução corresponde ao aumento na intrusão temporal. Entretanto, a intrusão espacial não é proporcional à intrusão temporal, uma vez que o aumento de uma não implica no aumento da outra de forma linear.

No que se refere à intrusão espacial, o injetor de falhas INFIMO_LIB apesar de não modificar o protocolo alvo em si, apresenta modificações nas funções da biblioteca que implementa o protocolo (biblioteca JRTPLIB). Estas modificações consistem na alteração do código da função *SendPacket( )* e no acréscimo de duas novas funções: *ShouldFail( )* e *SetFaultScenary( )*. Como a biblioteca JRTPLIB situa-se no mesmo

nível do protocolo alvo, a intrusão nesta biblioteca implica em intrusão no próprio protocolo alvo.

Já na implementação do INFIMO_DBG não há necessidade de modificações nas funções da biblioteca JRTPLIB. As únicas modificações foram realizadas no protocolo INFIMO_TAP e consistem na estrutura de criação das *threads*. Ao invés da existência de uma *thread* principal e uma *thread* receptora que coordena a execução das *threads* de envio, o código da *thread* de envio foi incorporado à *thread* principal. Com isso, o *ptrace( )* atua sobre a *thread* de principal como se esta fosse um processo e não uma *thread*.

9.1.3 Portabilidade

A portabilidade das ferramentas de injeção de falhas que compõem o *toolkit* INFIMO é analisada sob dois pontos de vista: portabilidade para outros protocolos alvo e portabilidade para outras plataformas. A portabilidade para outros protocolos alvo se refere a possibilidade de utilização das ferramentas de injeção de falhas para validação de outros protocolos, além do INFIMO_TAP. Já a portabilidade para outras plataformas está relacionada a viabilidade de adaptação, com alterações mínimas, das ferramentas do INFIMO para outras plataformas, que não o Linux.

Sendo assim, em ambos os pontos de vista, a ferramenta INFIMO_DBG apresenta vantagens. Uma vez que não interfere no código do protocolo alvo, o injetor INFIMO_DBG pode ser utilizado para validar diversos protocolos, os quais, através do sistema operacional, possuam suporte do *ptrace( )*. Analogamente, O INFIMO_DBG pode ser adaptado para outras plataformas, desde que estas disponibilizem um mecanismo compatível com o *ptrace( )*.

Embora de forma mais restritiva, a ferramenta INFIMO_LIB também pode ser utilizada para validar outros protocolos. Entretanto, tais protocolos devem ter sido implementados sobre a biblioteca JRTPLIB, que por sua vez é implementada sobre o protocolo de transporte RTP. Tal restrição também se aplica à portabilidade para outras plataformas. A nova plataforma deve possuir suporte à biblioteca JRTPLIB.

9.1.4 Expansão para Diferentes Modelos de Falhas

Analisa-se a expansão do modelo de falhas sob quatro aspectos: tipo, classe, disparo e duração das falhas. Quanto ao tipo de falha, por definição, as ferramentas de injeção de falhas que compõem o *toolkit* INFIMO suportam falhas de comunicação. Analisando as possibilidades de expansão para outros tipos de falhas, pode-se afirmar que a ferramenta INFIMO_DBG apresenta maior flexibilidade. A utilização do *ptrace( )* permite o acesso e a manipulação de registradores, o que viabiliza a injeção de falhas de memória e processador.

A ferramenta INFIMO_LIB, implementada totalmente no nível da aplicação, não apresenta explicitamente a possibilidade de expansão do modelo de falhas. A implementação de falhas de memória e processador somente é possível se a biblioteca JRTPLIB viabilizar este tipo de acesso, o qual deve se dar através de chamadas de

sistema (*system calls*). Neste caso são necessárias diversas alterações nas funções da biblioteca, o que pode comprometer sua semântica. Além disso, a expansão para outros tipos de falhas conduz a um aumento na latência apresentada pelas falhas. O intervalo de tempo entre a injeção da falha e a posterior percepção da mesma pelos mecanismos de tolerância a falhas do protocolo alvo pode ser muito grande. Já com falhas de comunicação tal problema é amenizado, uma vez que a semântica das mesmas faz com que sejam percebidas pelo protocolo alvo com atraso menor.

No que se refere à classe de falhas, para o modelo de falhas de comunicação, as ferramentas do INFIMO visam a validação de falhas de *crash* e omissão, apresentando suporte a falhas de temporização.

Sob o ponto de vista de disparo das falhas, as ferramentas do INFIMO apresentam maior acurácia em injeção de falhas com disparo espacial. Para ambas as ferramentas, a injeção de falhas com disparo temporal não apresenta estabilidade no que se refere ao controle dos *timers*.

A expansão do modelo de falhas em relação à duração da falha esbarra com o objetivo das ferramentas: minimizar intrusão. Atualmente, as ferramentas implementam falhas transientes e intermitentes. A expansão para falhas permanentes foge ao escopo do INFIMO, uma vez que as mesmas são consideradas extremamente intrusivas.

9.2 INFIMO x Outras Ferramentas

Nesta Seção estão relacionadas as ferramentas de injeção de falhas que compõem o INFIMO com algumas das ferramentas de injeção de falhas descritas no Anexo. São elas: FIAT [SEG88], FERRARI [KAN95], FINE [KAO93], DEFINE [KAO94], DOCTOR [HAN95], FIRE [ROS98b], Xception [CAR98], RT-Xception [CUN2000], Orchestra [DAW96] e Fiesta [KRI98]. As comparações são feitas através de tabelas e baseiam-se nas publicações das ferramentas, conforme consta no Anexo. Neste sentido, analisa-se: a plataforma utilizada, os mecanismos de injeção de falhas implementados, o objetivo da validação, os modelos de falhas suportados e os resultados obtidos.

9.2.1 Mecanismos de Injeção de Falhas

A plataforma sobre a qual está implementada a ferramenta de injeção de falhas é ponto decisivo na forma de implementação da mesma. Dependendo do *hardware* e do sistema operacional disponíveis pode-se obter vantagens na implementação do mecanismo de injeção. A Tabela 9.1 compara as ferramentas de injeção de falhas com relação ao *hardware* e ao sistema operacional sob os quais estão construídas e os mecanismos de injeção de falhas que utilizam.

De acordo com a Tabela 9.1, as abordagens que usam injeção de falhas implementada em *software* cobrem uma vasta gama de sistemas alvo indo desde os sistemas de processamento de transações comerciais até os sistemas de tempo real.

As ferramentas de injeção de falhas do INFIMO apresentam como vantagem a portabilidade no que se refere à plataforma de implementação. Por estar implementado

sobre máquinas PC e utilizar ambiente Linux, as ferramentas INFIMO_LIB e INFIMO_DBG apresentam-se tão portável quanto o Linux. Em relação ao mecanismo de injeção de falhas implementado, as ferramentas de injeção de falhas do INFIMO derivam suas abordagens de algumas características das ferramentas enfocadas na Tabela 9.1, como FERRARI, RT-Xception, FIESTA e Orchestra.

A observação da Tabela 9.1 permite afirmar que a maioria das ferramentas de injeção de falhas utiliza a manipulação de código e/ou a corrupção de valor como mecanismo de injeção de falhas. Fazem parte deste conjunto as ferramentas FIAT, FINE, DEFINE, DOCTOR, FIRE e Orchestra. Já as ferramentas FERRARI, Xception, RT-Xception e FIESTA fazem uso de recursos de depuração para a injeção das falhas.

TABELA 9.1 – Mecanismos de injeção de falhas

Ferramenta	***Hardware***	**Arquitetura/ SO**	**Mecanismos de IF**
INFIMO_LIB INFIMO_DBG	PC	Linux	Alteração de bibliotecas da aplicação ou desvio de chamadas de sistema através do *ptrace( )*
FIAT	IBM PCRT	-	Inserção de código na aplicação e no SO
FERRARI	*Sun Sparc*	*SunOS* 4.1.1	Alteração dinâmica de código e dados através de interrupções de *software*
FINE	*Sun Sparc*	*SunOS* 4.1.2	Modificação de rotinas de manipulação de interrupções. Falhas são injetadas pela alteração de código assembler
DEFINE	*Sun Sparc*	*SunOS*	Modificação de imagens executáveis dos programas; manipulação de interrupções de *clock* modificado
DOCTOR	Barramento VME	HARTOS	Execução concorrente de processos, alteração de controle de fluxo e substituição de código
FIRE	Multiplata-forma	Sistemas orientados a objeto	Corrupção de valores dos atributos e/ou argumentos dos métodos reflexivos.
Xception	PowerPC 601	PARIX	Programação direta do *hardware* depurador no processador alvo. Usa *breakpoints* e *timers* para disparar a injeção de falhas
RT-Xception	PC Pentium	SMX v. 3.2	Diretamente programando o *hardware* depurador no processador alvo, utilizando rotinas especializadas
Orchestra	*Sun Sparc*	*Real-Time Mach*	Camada de injeção de falhas entre duas camadas de uma pilha de protocolos para filtrar e manipular mensagens
FIESTA	*Sun Sparc*	VxWorks	Injeção de falhas via depurador tempo real

9.2.2 Objetivos da Validação

A validação por injeção de falhas por *software* (SWIFI) pode estar vinculada a inúmeros objetivos, dependendo do alvo da validação. A Tabela 9.2 exibe os principais objetivos, os quais referem-se à validação, de cada uma das ferramentas de injeção de falhas.

Os objetivos da validação por injeção de falhas estão fortemente relacionados às características de dependabilidade oferecidas pelo protocolo alvo. Adicionalmente, algumas ferramentas preocupam-se com limitações temporais. Conforme mostra a Tabela 9.2, as ferramentas de injeção de falhas do INFIMO preocupam-se em validar o mecanismo de tolerância a falhas oferecido pelo pelo protocolo alvo. Portanto, as ferramentas visam a validação do mecanismo de detecção de *timeout* do protocolo alvo, o qual possui restrições temporais.

As ferramentas INFIMO_LIB e INFIMO_DBG apresentam similaridade com grande parte das ferramentas no que se refere aos objetivos da validação. Isto porque, exceto as ferramentas FINE e DEFINE, as quais preocupam-se com a propagação de falhas no *kernel* do Unix, todas as ferramentas visam testar mecanismos de detecção de erros do protocolo alvo.

TABELA 9.2 – Objetivos dos experimentos de SWIFI

Ferramenta	**Objetivos**
INFIMO_LIB INFIMO_DBG	Validação do mecanismo de detecção do *timeout* do protocolo INFIMO_TAP
FIAT	Teste de mecanismos de detecção e recuperação de erros e avaliação quantitativa da dependabilidade do sistema sob teste
FERRARI	Teste de mecanismos de detecção e recuperação de erros
FINE	Estuda a propagação de falhas no *kernel* do Unix
DEFINE	Estuda a propagação de falhas no *kernel* do Unix e apresenta capacidade distribuída
DOCTOR	Teste dos mecanismos de dependabilidade distribuídos tempo real
FIRE	Validação de aplicações orientadas a objeto usando reflexão computacional
Xception	Ambiente de injeção de falhas por *software* e monitoração sem preocupação com limitações temporais
RT-Xception	Ambiente de injeção de falhas por *software* e monitoração com interferência mínima na aplicação alvo
Orchestra	Teste do comportamento tolerante a falhas e da temporização de aplicações tempo real distribuídas
FIESTA	Teste do mecanismo de detecção do *timeout* no caso de falhas de *crash*

9.2.3 Modelos de Falhas

A Tabela 9.3 compara os diferentes modelos de falha utilizados em abordagens de injeção de falhas implementadas por *software*. Os modelos de falhas da maioria dos sistemas tolerantes a falhas cobrem somente falhas físicas. Falhas humanas tais como falhas de projeto, falhas de implementação e falhas na interação entre o operador e o sistema de computação não são consideradas. São comparados o tipo, a localização, a duração e o disparo das falhas em ferramentas de injeção de falhas por *software*.

Diversas são as maneiras de se definir um modelo de falhas. A Tabela 9.3 classifica os modelos de falhas das ferramentas de injeção de acordo com a metodologia do INFIMO, apresentada na Seção 5.4 Neste sentido, observa-se o tipo de falha a ser validado pelo injetor, sua duração e disparo. Com relação às classes de falhas, algumas ferramentas preocupam-se somente com falhas de *crash*, enquanto outras adicionalmente apresentam suporte para validação de falhas de temporização. Devido à especificidade desta classificação, a Tabela 9.3, restringe-se à comparação dos tipos de falhas validados.

TABELA 9.3 – Modelos de falhas

Ferramenta	**Tipo/Localização**	**Duração**	**Disparo**
INFIMO_LIB INFIMO_DBG	Comunicação (processos)	Transiente e intermitente	Espacial
FIAT	Memória, comunicação e registradores da CPU	Intermitente e permanente	Espacial
FERRARI	Memória, processador e barramentos	Transiente e permanente	Espacial e temporal
FINE	Memória, processador e barramentos	Transiente e permanente	Espacial e temporal
DEFINE	Memória, processador e barramentos	Transiente e permanente	Espacial e temporal
DOCTOR	Memória, processador e comunicação	Transiente, intermitente e permanente	Espacial e temporal
FIRE	Corrupção de valores de atributos	Transiente, intermitente e permanente	Espacial
Xception	Memória, processador e barramentos	Transiente e permanente	Espacial e temporal
RT-Xception	Memória, processador e barramentos	Transiente e permanente	Espacial e temporal
Orchestra	Comunicação (processos ou links)	Transiente, intermitente e permanente	Espacial e temporal
Fiesta	Memória, processador e barramentos	Transiente	Espacial e temporal

9.2.4 Resultados

A Tabela 9.4 apresenta a forma pela qual as ferramentas de injeção de falhas exibem seus resultados experimentais. São comparadas as aplicações utilizadas na validação por injeção de falhas, bem como as métricas analisadas nos experimentos.

Conforme exibe a Tabela 9.4, a maioria das ferramentas de injeção de falhas visa analisar a latência e a cobertura de detecção de erros obtida nos experimentos. A definição das métricas de análise das ferramentas está fortemente relacionada aos objetivos dos experimentos das mesmas (exibidos na Tabela 9.2).

Ferramentas cujas aplicações analisam a cobertura de detecção falha/erro são FIAT, FERRARI, DOCTOR, Xception, Orchestra e Fiesta. Dentre estas ferramentas, FIAT, FERRARI e DOCTOR analisam ainda a latência de detecção das falhas/erros. RT-Xception é a ferramenta que dá mais enfoque à intrusão temporal imposta pelas atividades de injeção de falhas, embora Fiesta e DOCTOR também tenham esta preocupação.

TABELA 9.4 – Resultados obtidos

Ferramenta	**Aplicação**	**Métricas de Análise**
INFIMO_LIB INFIMO_DBG	Protocolo de troca de pacote (INFIMO_TAP)	Cobertura de detecção e intrusão (com base no tempo de execução)
FIAT	Carga de trabalho do *checkpoint*, multiplicação de matrizes	Cobertura de detecção, latência, efeitos dos tipos de falhas
FERRARI	Multiplicação de matrizes, algortimos de sorting	Cobertura de detecção, latência
FINE	*Kernel* do Unix	Propagação de falhas
DEFINE	*Kernel* do Unix	Propagação de falhas em ambiente distribuído
DOCTOR	Utilização de primitivas s*end/receive*	Cobertura, latência e MTTF
FIRE	Aplicação distribuída orientada a objetos	Capacidade para apontar falhas na aplicação alvo
Xception	Cálculo π, multiplicação de matrizes, resolução de equação de Laplace	Cobertura de detecção de erro, cobertura *fail silence*
RT-Xception	Execução de ciclos: INPUT, COMPUTE e OUTPUT; algoritmos de controle de temperatura	Intrusão baseada na variação dos tempos de execução
Orchestra	Protocolos de comunicação	Cobertura
FIESTA	Aplicações ADA distribuídas embutidas	Cobertura de detecção de erro e rotinas de recuperação de erro

9.3 Comparativo

O INFIMO *toolkit* foi desenvolvido para a realização de experimentos que analisem a intrusão de injetores de falhas na validação da dependabilidade de um protocolo com característica tempo real.

O Capítulo apresentou comparações entre as próprias ferramentas de injeção de falhas do INFIMO e entre estas e algumas ferramentas propostas na literatura (conforme descrito no Anexo).

As ferramentas de injeção de falhas do INFIMO foram desenvolvidas em paralelo com o protocolo alvo da validação (INFIMO_TAP). Portanto, há disponibilidade do código do protocolo.

Ambas as ferramentas de injeção de falhas mantêm a integridade do sistema operacional que as suporta. Com relação à integridade do protocolo alvo, a ferramenta INFIMO_LIB realiza modificações na biblioteca que o implementa. O INFIMO_DBG mantém íntegros tanto o protocolo alvo, como a biblioteca JRTPLIB.

A portabilidade das ferramentas de injeção de falhas para outras plataformas está vinculada à portabilidade do Linux. A portabilidade para validação de diferentes protocolos alvo para o INFIMO_DBG está garantida desde que o protocolo execute em ambiente compatível com o Unix e que, portanto, ofereça função de depuração. Para o INFIMO_LIB é necessário que o protocolo seja suportado pela biblioteca JRTPLIB.

A intrusão temporal do INFIMO_LIB e do INFIMO_DBG, ambos em atividade, apresenta-se de forma similar. A detecção da primeira falha implica impacto temporal proporcional ao *timeout* do protocolo alvo. Entretanto, falhas sucessivas não ocasionam aumento linear do tempo de execução, o qual é utilizado para medir a intrusão temporal.

A intrusão espacial das ferramentas de injeção de falhas do INFIMO sobre o protocolo alvo está relacionada à manutenção da integridade do alvo. Conforme mencionado anteriormente, o INFIMO_LIB modifica as funções da biblioteca JRTPLIB. O INFIMO_DBG realiza modificações na estrutura de *threads* do protocolo alvo. Entretanto, a idéia de ambos é manter a semântica do protocolo.

Quando comparadas com outras ferramentas de injeção de falhas, as ferramentas INFIMO_LIB e INFIMO_DBG apresentam similaridades. Isto porque as mesmas foram projetadas e implementadas com base no estudo de diferentes ferramentas. Quanto ao mecanismo de injeção de falhas, as ferramentas de injeção de falhas do INFIMO derivam das ferramentas FERRARI [KAN95], RT-Xception [CUN2000], DOCTOR [HAN95], Fiesta [KRI98] e Orchestra [DAW96]. As ferramentas FERRARI e Fiesta utilizam depurador; as ferramentas RT-Xception, Fiesta e DOCTOR preocupam-se com tempo real e a ferramenta Orchestra enfoca falhas de comunicação.

O objetivo da maioria das ferramentas visa validação de mecanismos de tolerância a falhas. As ferramentas de injeção de falhas do INFIMO visam a validar o mecanismo de tolerância a falhas do protocolo alvo, ou seja, o mecanismo de detecção do *timeout* oferecido pelo INFIMO_TAP.

O modelo de falhas suportado pelas ferramentas INFIMO_LIB e INFIMO_DBG, conforme mencionado anteriormente, consiste em falhas de comunicação, tal como a ferramenta Orchestra.

Dentre as métricas de análise enfocadas pelas ferramentas, a intrusão temporal corresponde à principal preocupação das ferramentas de injeção de falhas do INFIMO. Ferramentas como Fiesta, DOCTOR e RT-Xception também possuem esta preocupação, embora o enfoque do INFIMO_LIB e do INFIMO_DBG apresente-se mais próximo do enfoque da ferramenta RT-Xception.

10 Conclusões

A injeção de falhas por *software* consiste na criação de código que simule a ocorrência das falhas necessárias para a validação do sistema sob teste. Com injeção de falhas é possível medir a eficiência dos mecanismos de detecção, correção e recuperação de erros no sistema. Vantagens da injeção de falhas por *sofware* estão associadas ao baixo custo, baixa complexidade de desenvolvimento, portabilidade e fácil expansibilidade para novos tipos de falhas. Além disso, não apresenta problemas com interferências externas e riscos de danificar o sistema sob teste.

Pode-se citar como desvantagem o fato de o injetor ser um código a ser executado. Este código deve estar em algum lugar e eventualmente ser executado, de forma que o sistema se comportará de forma ligeiramente diferente de quando o injetor não estiver sendo utilizado. Algumas chamadas de sistema podem demorar mais para retornar, o sistema operacional pode demorar mais tempo para escalonar os processos, menos memória livre estará disponível.

Esta tese de doutorado apresenta um *toolkit* para experimentos de injeção de falhas em protocolos com característica tempo real. O *toolkit* apresentado, denominado INFIMO, explora uma combinação de abordagens de implementação visando reduzir a intrusão provocada por um módulo extra no sistema: o módulo injetor de falhas. O trabalho utiliza recursos tanto da aplicação como do sistema operacional para implementação da injeção de falhas, preocupando-se em otimizar o tempo de execução exigido pelo injetor de falhas. O INFIMO visa analisar a intrusão imposta pelas ferramentas de injeção de falhas na aplicação alvo.

O *toolkit* é composto por um protocolo de comunicação (INFIMO_TAP) e duas ferramentas de injeção de falhas (INFIMO_LIB e INFIMO_DBG). O INFIMO_TAP implementa o mecanismo de tolerância a falhas a ser validado pelas ferramentas de injeção de falhas. Este mecanismo corresponde à detecção do *timeout* associado à troca de pacotes. Conforme já explicado no decorrer do trabalho, o objetivo da implementação de duas ferramentas de injeção de falhas é possibilitar a comparação, sob os pontos de vista de cobertura de falhas e intrusão, das duas abordagens de implementação. As abordagens prevêem a utilização de recursos da própria aplicação a ser validada e recursos do sistema operacional.

O diferencial das ferramentas de injeção de falhas do INFIMO em relação às ferramentas estudadas (ver Anexo) está na preocupação com restrições temporais e no escopo de falhas a ser validado, o qual consiste em falhas de comunicação. As principais contribuições deste trabalho são:

a) INFIMO *toolkit*

O *toolkit* do INFIMO permite aos projetistas realizarem experimentos de injeção de falhas com o intuito de refinar os mecanismos de detecção de falhas do protocolo alvo da validação. As características de projeto do INFIMO incluem suporte para validação de protocolos com característica tempo real, portabilidade e preocupação com intrusão temporal e espacial.

A maioria dos injetores de falhas propostos na literatura preocupa-se com falhas de CPU, memória e barramento. O INFIMO volta sua atenção para falhas de comunicação, as quais apresentam latência pequena, já que o intervalo de tempo entre a injeção da falha e a sua percepção pelo mecanismo de tolerância a falhas do protocolo é reduzido.

b) estabelecimento de métricas de avaliação de injetores de falhas

O INFIMO estabelece dois tipos de comparação para suas ferramentas. O primeiro tipo compara as ferramentas de injeção de falhas do INFIMO entre si. A comparação das ferramentas INFIMO_LIB e INFIMO_DBG foi realizada de forma experimental e analisa:

- a cobertura de falhas em relação ao aumento no número de falhas injetadas;
- a cobertura de falhas em relação ao aumento no número de processos participantes do protocolo;
- a intrusão temporal da presença do injetor de falhas inativo;
- a intrusão temporal das atividades de injeção de falhas;
- a intrusão espacial do injetor de falhas;
- a portabilidade do injetor de falhas para outros protocolos alvo;
- a portabilidade do injetor de falhas para outras plataformas;
- a expansão do injetor de falhas para suporte a diferentes modelo de falhas.

O segundo tipo de comparação visa contrastar as ferramentas de injeção de falhas do INFIMO com ferramentas de injeção de falhas disponíveis. Esta comparação tem como base as publicações das ferramentas. Neste sentido, avalia-se:

- os mecanismos de injeção de falhas implementados pelas ferramentas;
- os objetivos da validação por injeção de falhas;
- os modelos de falhas os quais as ferramentas se propõem validar;
- os resultados obtidos com os experimentos de injeção de falhas.

c) análise da intrusão dos injetores de falhas

O principal objetivo do INFIMO é avaliar a intrusão imposta por ferramentas de injeção de falhas em um protocolo com característica tempo real. Esta avaliação é realizada com base nos valores de tempo de execução apresentados pelo protocolo alvo. Primeiramente analisa-se o tempo de execução do protocolo alvo em si (INFIMO_TAP). A seguir, analisam-se os tempos de execução do protocolo juntamente com cada um dos injetores de falhas, estando os mesmos inativos e ativos. Experimentos com injetor inativo permitem analisar a intrusão do injetor de falhas. Já experimentos com injetor ativo visam analisar a intrusão das atividades de injeção de falhas.

Para efeitos de comparação são consideradas execuções das ferramentas sob dois enfoques: em ambiente local e em ambiente distribuído. Execuções locais simulam

situações onde diversos processos de um ambiente distribuído encontram-se na mesma máquina e devem, portanto compartilhar os mesmos recursos.

d) suporte para protocolos com característica tempo real

Mínima intrusão é o principal objetivo de injetores de falhas para validação de protocolos com característica tempo real. Com isso, o projeto de ferramentas de injeção de falhas para ambiente tempo real deve, da mesma forma, ter esta preocupação. Esta tese descreve abordagens que efetivamente exploram características da aplicação e do sistema operacional para tentar compensar a intrusão dos métodos de injeção de falhas.

Embora trate de um protocolo com limitações temporais pouco rígidas, pode-se analisar a viabilidade de utilização das ferramentas de injeção de falhas do INFIMO para validação de protocolos com restrições temporais críticas. Deve fazer parte da análise um exame detalhado dos atrasos impostos pelas ferramentas, assim como uma reavaliação da especificação do *timeout* associado ao recebimento de pacotes.

Dentre as ferramentas de injeção de falhas investigadas, as quais encontram-se resumidas no Anexo, as que se preocupam com características tempo real são RT-Xception, DOCTOR e FIESTA. O RT-Xception [CUN2000] conta com um *kernel* tempo real (SMX v.3.2.) e utiliza-se das características de desempenho e depuração do mesmo para ativar a injeção de falhas. Entretanto, o escopo de falhas enfocado pelo RT-Xception concentra-se principalmente em falhas de memória e processador.

A ferramenta DOCTOR [HAN95] preocupa-se com características tempo real, contudo somente funções essenciais são realizadas no mesmo processador do protocolo alvo e são usados métodos de injeção de falhas relativamente simples.

A ferramenta FIESTA [KRI98], implementada sobre o sistema operacional tempo real comercial VxWorks 5.3, visa a avaliação da dependabilidade de sistemas tempo real embutidos. Para tanto, FIESTA conta com o depurador tempo real do sistema embutido, utilizando-se da garantia de correção temporal presente no sistema.

e) expansão para diferentes modelos de falhas

As ferramentas de injeção de falhas do INFIMO possibilitam, com algumas alterações, a definição e implementação de outros modelos de falhas, tais como falhas de memória, CPU e barramento. Esta afirmação aplica-se principalmente à ferramenta INFIMO_DBG, já que conforme mostra a ferramenta FERRARI [KAN95], o *ptrace( )* apresenta facilidade no acesso e manipulação de tais componentes.

Para a ferramenta INFIMO_LIB é preciso avaliar as possibilidades de acesso a estes componentes por meio da biblioteca JRTPLIB. Por ser uma biblioteca implementada no nível da aplicação há necessidade de viabilizar acesso aos componentes de mais baixo nível, o que envolve ainda a flexibilidade do sistema operacional utilizado como suporte.

Outra possibilidade de expansão envolve a inclusão da duração permanente de falha. Este último aspecto não foi enfatizado neste trabalho por caracteristicamente introduzir carga no sistema afetando, com isso, o comportamento temporal do protocolo alvo.

f) portabilidade

A característica de portabilidade das ferramentas de injeção de falhas do INFIMO pode ser enfocada sob dois aspectos: portabilidade para outros protocolos alvo e portabilidade para outras plataformas. As ferramentas de injeção de falhas do INFIMO mostram-se portáveis para validação de outros protocolos, desde que implementados sobre a biblioteca JRTPLIB.

O INFIMO *toolkit* pode ser portado para outras plataformas compatíveis com Linux. Entretanto, a complexidade que envolve esta portabilidade deve ser criteriosamente investigada.

g) demonstração da viabilidade do INFIMO *Toolkit*

Para demonstrar a viabilidade das abordagens propostas pelo INFIMO, um protótipo do INFIMO *toolkit* foi construído. Com base na implementação do protótipo, foram conduzidos diversos experimentos. Tais experimentos possibilitaram a verificação da viabilidade (ou da não viabilidade) da utilização da abordagem sob diversos cenários. A denominação protótipo para o INFIMO refere-se à idéia de continuidade do trabalho, uma vez que diversas otimizações e trabalhos relacionados estão sendo sugeridos. O grupo de tolerância a falhas da UFRGS tem mostrado grande interesse em injeção de falhas. Espera-se que este trabalho venha incentivar ainda mais este interesse.

A implementação do INFIMO apresenta limitações, as quais incluem a irrelevância do destino dos pacotes na comunicação. Para o INFIMO não importa para onde os pacotes são direcionados. Essa abstração torna-se possível já que a falha de comunicação é injetada no envio do pacote e o controle do *timeout*, relacionado ao recebimento do pacote, é realizado na origem do mesmo.

Além disso, o INFIMO restringe sua implementação para falhas de *crash* e omissão, apresentando suporte para falhas de temporização. Assim como grande parte das ferramentas de injeção de falhas, as ferramentas do INFIMO também não suportam a injeção de falhas simultâneas.

Com relação ao processos injetor de falhas e ao processo alvo existe obrigatoriedade de que ambos estejam executando na mesma máquina. Adicionalmente, para a ferramenta INFIMO_LIB deve-se possuir acesso à biblioteca JRTPLIB e ao protocolo RTP; e para a ferramenta INFIMO_DBG deve-se contar com a disponibilidade da função de monitoração *ptrace( )*, padrão Unix.

10.1 Questões Relacionadas

Ao longo da implementação do INFIMO, algumas questões foram formuladas. Dentre elas, pode-se citar as seguintes:

1) As falhas não intencionais, incluindo falhas de temporização, podem ser introduzidas no sistema pelos mecanismos de injeção de falhas ?

Diversos pesquisadores da área de tolerância a falhas afirmam que nenhum programa é 100% correto e seguro. Com base nesta afirmação, pode-se concluir que um programa criado para injetar falhas em outro programa (programa protocolo alvo) está sujeito à inserção de falhas não previstas em sua especificação. Isto porque, de acordo com a própria afirmação, tanto o programa protocolo alvo como o programa injetor de falhas não estão livres de falhas. Este fato pode dificultar a análise de cobertura das falhas obtidas com os experimentos pois, além das falhas injetadas intencionalmente, deve-se prever a possibilidade de manipulação de falhas inerentes aos programas envolvidos.

2) O protocolo alvo estará menos previsível em função da injeção de falhas ?

   A tarefa de medir o quanto as atividades de injeção de falhas interferem no protocolo alvo é bastante complexa. O que as ferramentas fazem, em geral, é avaliar a forma de implementação das atividades de injeção de falhas. Sob o ponto de vista espacial, quanto menos intrusivas as implementações, menor o risco de modificar o contexto e, portanto o comportamento do protocolo alvo. Entretanto, em injeção de falhas por *software*, a intrusão temporal está quase sempre presente, em maior ou menor grau. Dependendo da intrusão, o protocolo pode ter seu comportamento mais ou menos modificado.

3) A intrusão temporal dos experimentos de injeção de falhas pode ser precisamente quantificada ?

   Por mais exaustivos que sejam os experimentos não se pode afirmar que os mesmos podem ser precisamente quantificados, principalmente em sistemas de computação convencional, caracteristicamente pouco acurados, considerando o problema de sincronismo. Além disso, a intrusão temporal é medida através do tempo de execução do protocolo, o que pode tornar os valores ainda menos precisos.

10.2 Trabalhos Futuros

Dentre os possíveis trabalhos provenientes do INFIMO, destacam-se:

- execução das ferramentas de injeção de falhas do INFIMO com outros protocolos de comunicação;
- exibição das experimentações com o auxílio do *sniffer*;
- implementação de uma interface gráfica para o INFIMO;
- geração automática de relatórios com base nos arquivos de *log*;
- análise automatizada das métricas de comparação das ferramentas de injeção de falhas, utilizando técnicas de inteligência artificial;
- expansão do modelo de falhas para suporte a falhas múltiplas;
- expansão da implementação para permitir injeção distribuída de falhas.

Anexo Trabalhos Relacionados

Esta Seção apresenta resumos das principais características de algumas das ferramentas propostas na literatura, através das quais obteve-se base para a definição e implementação do INFIMO. Os resumos apresentados cobrem ferramentas de injeção de falhas representativas de várias técnicas, algumas das quais foram utilizadas na implementação do INFIMO. A Seção descreve ainda o estado da arte da pesquisa em injeção de falhas no Grupo de Tolerância a Falhas da UFRGS.

FIAT

O injetor FIAT (*Fault-Injection-Based Automated Testing*) combina a flexibilidade de controle de *software* com a emulação de *hardware* para avaliar a dependabilidade de sistemas distribuídos tolerantes a falhas. FIAT usa injeção de falhas implementada por *software* para setar e limpar *bytes* nas imagens dos programas, ou seja, FIAT injeta erros diretamente na memória. Os programas executam em uma rede de máquinas configuradas para modelar uma determinada arquitetura de sistema [SEG88].

Este injetor foi desenvolvido na Universidade de Carnegie Mellon, nos Estados Unidos, utilizando-se quatro IBM RT PC's conectados via *token ring*. FIAT tem sido usado para medir cobertura e latência, classificar falhas e investigar os efeitos dos tipos de falhas e carga nestas medidas.

O objetivo do injetor é a validação de sistemas distribuídos tempo real. Neste contexto, FIAT visa a descoberta de deficiências nos mecanismos de detecção e recuperação de erros do sistema e a avaliação quantitativa da dependabilidade do sistema sob teste.

O injetor FIAT é composto por dois componentes ligados por uma rede de comunicação: FIM (*Fault Injection Management*) e FIRE (*Fault Injector Receptors*). O FIM é um programa de controle global, responsável por todas as fases do experimento. Além de suportar o desenvolvimento do experimento, o FIM realiza a coleta e a análise de dados e controla o experimento em tempo de execução. O FIRE roda sob controle do FIM e oferece a plataforma de execução para o sistema distribuído tempo real sob teste. Algumas das funções do FIRE são a coleta de resultados experimentais e o envio de informações para o FIM.

FIAT tem sido utilizado para estudar o impacto das falhas no nível de aplicação. A seleção da localização da falha é feita pelo usuário no nível de aplicação, enquanto a localização física dentro da memória é obtida através de informações do compilador ou carregador. O injetor FIAT não é capaz de injetar falhas transientes.

FERRARI

O FERRARI (*Fault and ERRor Automatic Real-time Injector*) é um injetor de falhas por *software* desenvolvido na Universidade do Texas [KAN95]. FERRARI visa

avaliar sistemas complexos através da emulação de falhas de *hardware* por *software*. Para tanto, possui habilidade para injetar erros transientes e falhas permanentes.

O injetor FERRARI emula um grande número de falhas de *hardware*, assim como erros de fluxo, permite controle no tempo, localização, tipo e duração da falha ou erro. É capaz de medir cobertura e latência, controlando um grande número de rodadas de experimento.

FERRARI utiliza o *ptrace( )* do Unix para corromper a imagem de memória do processo, em tempo de execução, e inserir instruções de interrupção de *software* nos endereços de instrução específicos onde as falhas devem ser ativadas.

No FERRARI dois processos executam concorrentemente: o processo injetor e o processo que implementa o programa alvo. O mecanismo de injeção é baseado na comunicação entre estes dois processos, permitindo que o processo injetor altere o estado de execução do programa alvo.

O injetor é composto por quatro módulos: inicialização e ativação, informação do usuário, injeção de erro/falha e coleta e análise de dados. Estes quatro módulos são controlados por um módulo gerente, o qual coordena a operação e a interação dos mesmos.

O módulo de inicialização e ativação prepara o programa alvo para a injeção de erros/falhas. O módulo de informação do usuário obtém os parâmetros do experimento. Alguns destes parâmetros são: propriedades de dependabilidade a serem medidas (cobertura e latência), duração da falha/erro, modo do experimento (localização e tempos para a injeção ou especificação randômica), modelos de erro/falha, dentre outros.

Cabe ao módulo de injeção de erro/falha a introdução de diferentes tipos de erros transientes e falhas permanentes. Os mecanismos de injeção são idênticos, entretanto, há diferença na duração do erro/falha. Erros transientes possuem a duração de um ciclo de instrução, enquanto falhas permanentes possuem a duração de vários ciclos de instrução, podendo se propagar por toda a execução da aplicação.

A injeção de falhas realizada pelo FERRARI pode se dar de três maneiras: corrupção de memória, corrupção espacial e corrupção temporal. Na corrupção de memória, falhas são injetadas antes do início da execução do programa e podem permanecer durante toda sua execução. Na corrupção espacial a injeção da falha ocorre em um endereço específico. Já na corrupção temporal, erros são injetados no intervalo de diversas instruções.

As atividades de injeção realizadas pelo FERRARI obedecem o seguinte:

- aplicação alvo é executada sem que seja efetuada a injeção de erros e sua saída é armazenada em um arquivo para futura comparação;
- erro/falha desejado é introduzido no sistema enquanto a aplicação executa;
- erros detectados pelos mecanismos de tolerância a falhas do sistema são armazenados;
- se nenhum mecanismo de detecção é disparado, a saída da atual rodada é comparada com a saída da aplicação;
- a diferença entre as duas saídas indica um erro que não foi detectado por nenhum mecanismo de tolerância a falhas.

O módulo de coleta e análise de dados mede e registra a resposta do sistema para cada erro/falha injetado. Com os resultados coletados são realizados cálculos estatísticos com relação as medidas de cobertura, latência e tipo de mecanismo de detecção de erros utilizado para cada experimento.

Resultados de experimentos conduzidos em *Sun Sparc Workstations* rodando *SunOS* 4.1 mostram que a cobertura obtida é altamente dependente dos modelos de falha e erro escolhidos e dos mecanismos de detecção de erros disponíveis.

FINE e DEFINE

O FINE (*Fault Injection moNitoring Environment*) é uma ferramenta de injeção de falhas que foi desenvolvida na Universidade de Illinois [KAO93]. O FINE estuda a propagação de falhas no *kernel* do Unix. Segundo Kao, a maioria dos estudos de injeção de falhas concentra-se no impacto final das falhas no sistema com ênfase nas medidas de cobertura e latência da falha. O que realmente acontece após uma falha ser injetada e como uma falha se propaga em um sistema são questões consideradas irrelevantes pelos projetistas de injetores de falhas por *software*.

Além de estudar a propagação das falhas, o FINE pode ser utilizado para avaliar a dependabilidade do sistema, assim como a eficiência dos mecanismos de detecção e recuperação de erros. O injetor FINE necessita do código fonte da aplicação alvo e, segundo Cunha [CUN2000], causa muita sobrecarga (*overhead*).

O FINE justifica a escolha do sistema operacional como alvo de estudo devido a todas as aplicações do usuário necessitarem do suporte do sistema operacional. Neste sentido, falhas no sistema operacional ocasionam, consequentemente, falhas nas aplicações.

Embora o objeto de estudo seja o sistema operacional, projetistas do FINE afirmam que a metodologia para injeção também pode ser utilizada em aplicações do usuário. O FINE introduz falhas *online* no *kernel* do Unix e acompanha o fluxo de execução do sistema. Para identificar a propagação das falhas, é realizada uma comparação entre os dados faltosos e dos dados livres de falha, considerando a mesma carga de trabalho.

O FINE injeta tanto falhas de *hardware*, as quais consequentemente induzem a erros de *software*, como falhas de *software*.

FINE é composto por quatro módulos, os quais correspondem a quatro processos na aplicação alvo. São eles: injetor de falhas, monitor de *software*, gerador de carga de trabalho e controlador. O injetor de falhas, como o próprio nome sugere é responsável pela introdução de falhas de *hardware* e *software*. O monitor de *software* controla o fluxo de execução e as variáveis chave do *kernel*. Cabe ao gerador de carga de trabalho a ativação das falhas introduzidas e a geração da carga de trabalho, a qual consiste em chamadas de sistema para o *kernel*. O módulo controlador realiza a tarefa de gerenciar o experimento, especificando o tipo de falha para o injetor de falhas, as variáveis/dados chave para o monitor de *software* e a especificação da carga de trabalho para o gerador de carga de trabalho.

Por executarem na camada de aplicação do sistema, todos os módulos podem ser afetados após as falhas serem injetadas no *kernel*.

Com base nos modelos, é realizada uma análise através de cadeias de Markov. O objetivo desta análise é avaliar a perda de desempenho devido a uma falha injetada.

Resultados mostram que a injeção de falhas de memória e falhas de *software* geralmente apresentam latência longa, enquanto a injeção de falhas de barramento e CPU são percebidas quase que imediatamente pelo sistema.

DEFINE [KAO94] é uma evolução do FINE, a qual inclui capacidades distribuídas. O injetor DEFINE modifica as imagens executáveis dos programas para emular falhas de memória, como por exemplo pela inserção de interrupção de *software* em localizações específicas de memória no segmento de texto.

O DEFINE também deriva do FINE na introdução de um manipulador de interrupção de *clock* modificado por *hardware* para injetar falhas de CPU e barramento com disparos temporais. DEFINE também é capaz de injetar alguns tipos de falhas de *software*, ou seja, falhas de implementação/projeto de *software*.

DOCTOR

O injetor de falhas DOCTOR (*integrateD software fault injeCTiOn enviRonment*) [HAN95] foi implementado no sistema distribuído tempo real HARTOS. DOCTOR avalia dependabilidade através de medidas como cobertura, latência e MTTF.

DOCTOR pode ser aplicado em sistemas tempo real. Portanto, em tais sistemas, onde o tempo é o principal recurso, as atividades de injeção de falhas e coleta de dados relevantes devem ser realizadas com mínima sobrecarga na aplicação alvo. Para minimizar este problema, somente funções essenciais são realizadas no mesmo processador sob teste e são usadas técnicas de injeção de falhas relativamente simples, o que garante a portabilidade da ferramenta. Para aumentar a acurácia e minimizar a sobrecarga da coleta de dados, foi projetado um *hardware* dedicado para este fim.

O injetor consiste em três módulos principais: EGM, ECM e FIA. Além destes, existem ainda os módulos DCM, DAM, HMON e SWG. As descrições de tais módulos encontram-se abaixo:

- EGM (*Experiment Generation Module*) – realiza a geração de carga de trabalho;
- ECM (*Experiment Control Module*) – executa o controle externo;
- FIA (*Fault-Injection Agent*) – efetua a injeção de falhas propriamente dita;
- DCM (*Data Collection Module*) – realiza a coleta de dados;
- DAM (*Data Analysis Module*) – é responsável pela análise de dados;
- HMON (*Hardware Monitor*) – obtém uma temporização de dados mais acurada e com menor sobrecarga. Utilizado em substituição ao DCM;
- SWG (*Synthetic Workload Generator*) – gera cargas sintéticas, também chamadas “cargas artificiais”.

O injetor DOCTOR suporta três tipos de falhas: falhas de memória, falhas de CPU e falhas de comunicação. Além do modelo básico de falhas, DOCTOR também oferece uma conveniente interface para usuário que permite a especificação temporal da injeção de falhas, habilitando assim a construção de cenários de falhas mais complexos. O comportamento temporal de uma falha pode ser especificado como transiente, intermitente ou permanente.

Falhas de memória são emuladas via alteração parcial/total do conteúdo. A localização da falha pode ser especificada pelo usuário ou escolhida randomicamente. Falhas de CPU podem ocorrer em registradores de dados ou de endereços, na unidade de busca de dados, nos registradores de controle, na unidade de decodificação de *opcode*, ULA, etc.

Falhas de comunicação podem causar a perda, a alteração, a duplicação ou o atraso de mensagens. A perda de mensagens pode ser efetuada intermitentemente, através de uma distribuição de probabilidade especificada pelo usuário, ou cada mensagem pode ser perdida durante um determinado período. A alteração de mensagens ocorre de forma similar à atividade de injeção de falhas de memória, ou seja, por corrupção de *bits*. O usuário pode especificar se o erro deve ser injetado no corpo ou no cabeçalho de uma mensagem. Já o atraso de mensagens deve obedecer a um tempo determinístico ou probabilístico. O usuário pode ainda definir falhas de comunicação adicionais, as quais correspondem a combinações dos tipos pré definidos.

O controle temporal da falha pode ser implementado da seguinte forma: uma falha transiente deve ser injetada apenas uma vez, já uma falha intermitente deve ser injetada repetidamente, podendo o usuário especificar a distribuição de probabilidade da mesma, que pode ser, por exemplo, uniforme, exponencial ou normal.

FIRE

O injetor de falhas FIRE (*Fault Injection using a Reflexive Architecture*) [ROS98b] propõe o uso de reflexão computacional para minimizar a perturbação na aplicação alvo no teste de aplicações orientadas a objeto. FIRE utiliza injeção de falhas por *software* ativada em tempo de execução (*runtime*).

A ferramenta FIRE foi implementada usando a linguagem C++ e o pré-processador Open C++1.2 e visa validar aplicações desenvolvidas em C++ ou Open C++ 1.2.

A reflexão computacional foi utilizada para implementar uma biblioteca de injeção e monitoração, a qual é linkada à aplicação alvo. A ferramenta usa as facilidades oferecidas pelo protocolo de meta-objetos do Open C++ 1.2 para injetar falhas e observar seus efeitos.

FIRE é constituído por três componentes principais:

- *application*: é uma aplicação tolerante a falhas orientada a objetos, que deve ser submetida a uma fase de preparo antes da validação.
- *meta-level library*: contém meta-objetos denominados *schedulers*, cada um composto por um injetor e um sensor. O injetor é responsável pela

introdução de falhas pré-especificadas no objeto injetável, enquanto o sensor realiza a coleta de dados de saída.

- *controller*: executa o controle do experimento. Inicia a aplicação, controla injetores e sensores e armazena resultados. É composto por três objetos: *injection manager*, *monitor* e *user interface*.

O injetor FIRE apresenta um novo mecanismo de ativação, denominado interceptação de mensagens, oferecido pelo Open C++ 1.2. O método de injeção de falhas do FIRE visa a corrupção de valores dos atributos e/ou argumentos dos métodos reflexivos.

FIRE considera falhas de *software* internas e externas, as quais podem se apresentar de modo transiente, intermitente ou permanente. O alvo dos experimentos é uma aplicação distribuída orientada a objetos que implementa uma pilha tolerante a falhas. Esta aplicação utiliza dois mecanismos para tolerar falhas de *software*: blocos de recuperação e programação n-versões.

Xception e RT-Xception

A ferramenta Xception [CAR98] pode injetar falhas com interferência mínima na aplicação alvo. Xception utiliza recursos de um depurador de *hardware* e um monitor de *software*. Com isso, o objetivo do Xception é registrar informação detalhada sobre o comportamento do processador alvo após a injeção da falha. Alguns exemplos destas informações são o número de leituras e escritas na memória, o número de instruções executadas, etc. Além disso, Xception pode monitorar outros aspectos, tais como a possibilidade de detectar se alguma área de memória foi acessada após a falha ou se alguma função do programa foi executada.

Outro aspecto importante é que, devido ao Xception operar muito próximo do *hardware*, as falhas injetadas podem afetar qualquer processo rodando na aplicação alvo, incluindo o *kernel*. É também possível injetar falhas em aplicações nas quais o código fonte não está disponível.

A aplicação alvo considerada pelo Xception é formada pelo processador, memória e barramentos de dados e endereço. Falhas injetadas podem emular diretamente falhas físicas afetando as seguintes unidades: barramento de dados, barramento de endereço, unidade de ponto flutuante, unidade de inteiro, unidade de gerenciamento de memória, registradores de propósito geral, unidade de processamento de desvio e memória principal [CAR95].

O impacto na carga da aplicação alvo é muito pequeno tanto no tempo como no espaço. Com relação ao tempo, a exceção que dispara a injeção da falha é programada no processador do depurador antes do início da aplicação alvo (*off-line*). Portanto, o processador é executado em alta velocidade até ser interrompido pela exceção disparada para realizar a injeção.

Com relação ao espaço, com Xception a aplicação não precisa ser alterada. Além disso, o código do injetor ocupa pouco espaço de memória.

Na implementação atual, o Xception foi integrado ao *kernel* PARIX, um sistema operacional semelhante ao Unix para máquinas paralelas. O Xception consiste de três módulos:

- *Kernel* – módulo *linkado* estaticamente ao *kernel* do sistema. Consiste de manipuladores de exceção e código para realização da injeção de falhas;
- *Fault setup* – biblioteca de funções cuja tarefa é receber os parâmetros de falha do *host* (via troca de mensagens) e passá-los para o *kernel*;
- EMM (*Experiment Manager Module*) – este módulo não executa na aplicação alvo. Oferece uma interface para o usuário para definição da falha, controle automático do experimento de injeção de falhas e coleta de resultados.

O Xception considera falhas de memória e CPU, podendo a duração da falha ser transiente ou permanente, embora este último tipo de falha apresente um grau de complexidade de implementação bastante elevado.

O injetor RT-Xception [CUN2000] é uma versão do Xception especialmente projetado para sistemas tempo real. RT-Xception roda em SMX v.3.2 (*Simple Multitasking Executive*), um *kernel* tempo real da *Micro Digital Inc, USA*. A plataforma alvo é um PC Pentium.

A exemplo do Xception, a arquitetura do RT-Xception baseia-se no modelo cliente-servidor. Ela compreende um núcleo de injeção, que roda na aplicação alvo e é responsável pela inserção das falhas, e uma aplicação GUI, responsável pela gerência do experimento de injeção de falhas.

O modelo de falhas suportado pelo RT-Xception também segue o modelo do Xception, ou seja, visa a introdução de falhas transientes nas unidades funcionais do processador alvo. Estas unidades incluem unidade de inteiro, unidade de ponto flutuante, unidade de gerenciamento de memória, barramento de dados, barramento de endereço, registradores de propósito geral e memória.

O RT-Xception usa as características de desempenho e depuração existentes nos processadores modernos para disparar a injeção de falhas e emular os efeitos de uma falha. É um injetor ativado em tempo de execução(*runtime*), porém com mínima intrusão no que se refere ao disparo da falha.

Orchestra

O injetor Orchestra [DAW96][DAW98] permite a manipulação das mensagens trocadas entre os processos participantes do protocolo. Orchestra foi inicialmente desenvolvido sob o sistema operacional Mach e, mais tarde, portado para outras plataformas, as quais incluem *Solaris* e *SunOS*.

O injetor Orchestra atua no nível de mensagens, o que permite que o projetista de sistemas injete falhas nos protocolos distribuídos através da manipulação de mensagens. Orchestra permite as seguintes operações sobre mensagens:

- filtragem de mensagens: permite interceptar e analisar mensagens;

- manipulação de mensagens: possibilita a perda, o atraso, a reordenação, a duplicação ou a modificação de uma mensagem;
- introdução de mensagens: permite introduzir mensagens no sistema.

O objetivo da injeção de falhas no nível de mensagens inclui teste em sistemas distribuídos. Neste caso, deseja-se forçar o sistema a determinados estados para garantir que caminhos de execução específicos serão tomados. Além disso, quando se testa as capacidades de tolerância a falhas de um sistema distribuído, um comportamento específico pode ser exigido de um protocolo, o que é impossível alcançar sob condições normais. Adicionalmente, projetistas, em geral, necessitam de uma metodologia que não instrumente o código fonte sob teste, condição ideal para quando não se dispõe do código fonte da aplicação alvo.

O injetor de falhas Orchestra permite a especificação do experimento através de *scripts*, que consistem de instruções executadas pelas camadas adjacentes ao protocolo sob teste, visando conduzir a computação do sistema a um determinado estado e injetar diversos tipos de falhas.

Orchestra é implementado com uma camada extra, denominada PFI (*Protocol Fault Injection*), localizada abaixo do protocolo sob teste. A camada PFI é capaz de filtrar e manipular as mensagens trocadas entre os participantes no protocolo sob teste [DAW94].

Orchestra usa *scripts* para determinar as ações do injetor enquanto mensagens passam através da camada PFI. Estes *scripts*, implementados em *runtime*, podem ser determinísticos ou probabilísticos. Interceptando mensagens entre duas camadas numa pilha de protocolos, a camada de injeção de falhas pode atrasar, abandonar, reordenar, duplicar e modificar mensagens. Além disso, novas mensagens podem ser introduzidas no sistema, com o objetivo de testar a resposta da aplicação alvo, conduzindo a execução do sistema para determinados caminhos. A vantagem da utilização de *scripts* interpretados é que estes permitem alterações de teste sem recompilar a aplicação alvo ou o *software* de injeção de falhas.

FIESTA

O injetor de falhas FIESTA (*Fault Injection for Embedded System Target Application*) [KRI98] foi implementado sobre o sistema operacional tempo real comercial VxWorks 5.3. FIESTA aborda o problema de projetar injetores de falhas por *software* para avaliação da dependabilidade de sistemas embutidos tempo real. O FIESTA apresenta uma metodologia de projeto genérica, a qual possibilita uma rápida prototipação de injetores de falhas implementados por *software*.

De acordo com Krishnamurthy, a maioria das abordagens propostas para injetores de falhas não são apropriadas para plataformas tempo real embutidas porque são fortemente relacionadas com o ambiente de execução, exigem *hardware* adicional e apresentam funções bastante específicas. FIESTA se esforça para seguir o padrão POSIX, garantindo a portabilidade da aplicação e procura alterar o mínimo possível o comportamento da aplicação alvo, mantendo uma posição não intrusiva em relação ao sistema embutido [KRI98].

FIESTA foi implementado com base nos recursos do depurador tempo real do sistema embutido, utilizando-se da garantia de correção temporal presente no sistema. O injetor FIESTA dividiu a funcionalidade da atividade de injeção de falhas em dois domínios separados. O depurador tempo real manipula as tarefas críticas distribuídas e de tempo real separadamente do gerente de injeção de falhas. A interface entre a ferramenta de injeção de falhas e a aplicação alvo é realizada por meio do depurador.

O FIESTA é composto pelos seguintes módulos: o depurador tempo real, o injetor propriamente dito e a aplicação alvo da injeção de falhas.

O injetor de falhas FIESTA apresenta uma interface GUI, desenvolvida em Tcl/Tk, a qual assegura eficiência e portabilidade. FIESTA é capaz de injetar falhas de memória e CPU, ambas de forma transiente, e foi projetado para validar o mecanismo de detecção do *timeout* em caso de falha de *crash*.

Experimentos realizados com o injetor FIESTA mostraram que, em média, há 64% de probabilidade que uma aplicação sobreviva a uma falha transiente. Embora tal valor não seja aceitável para a maioria das aplicações tempo real, segundo Krishnamurthy, este valor pode ser utilizado pelo projetista para otimizar o mecanismo de tolerância a falhas utilizado.

Trabalhos Locais Relacionados

Esta Seção descreve brevemente as ferramentas desenvolvidas e em desenvolvimento pelo Grupo de Tolerância a Falhas da UFRGS. Injeção de falhas tem sido utilizada na implementação e validação de técnicas de tolerância a falhas tanto em sistemas distribuídos convencionais, como em sistemas tempo real.

A ferramenta ADC [BAR95] [BAR96] consiste em um ambiente para avaliação de protocolos de difusão confiável. Esta ferramenta foi desenvolvida em C (padrão Unix, ambiente *SunOS*), com suporte do *HetNOS* [BAC94], um sistema operacional heterogêneo. O ADC roda em um ambiente simulado sob controle de um módulo central. A injeção de falhas é implementada através desse módulo, de acordo com uma probabilidade definida por um modelo de entrada. Este modelo especifica o cenário de falhas a ser validado, ou seja, o cenário define as classes de falhas as quais o protocolo está apto a suportar. A técnica de injeção de falhas utilizada pelo ADC é classificada como injeção de falhas por simulação e corresponde a uma solução específica para a implementação em questão.

A técnica de recuperação de processos, implementada pela ferramenta FIX [TEI95] prevê um módulo de injeção de falhas acionado diretamente a partir de chamadas de sistema do Linux. Esse módulo fornece primitivas que podem ser usadas para a construção de ferramentas de injeção de falhas em protocolos de recuperação de processos.

A ferramenta AFIDS [SOT97a] [SOT97b] possibilita a construção de ferramentas de injeção de falhas implementadas por *software* para sistemas distribuídos através do fornecimento de uma biblioteca de classes básica e genérica em C++. Esta biblioteca disponibiliza uma estrutura que suporta a geração de parâmetros de falhas, a injeção efetiva da falha e seu controle e, finalmente, a coleta de dados dos experimentos

e sua análise. Visa também permitir a validação de protocolos tolerantes a falhas para sistemas distribuídos, seja através de eliminação ou prevenção de falhas. A aplicação de AFIDS é ilustrada por uma ferramenta de injeção de falhas que utiliza um objeto injetor em cada um dos processos que compõem o protocolo sob teste. AFIDS e esta ferramenta são implementadas em C++, usando *sockets* no sistema operacional Linux.

A ferramenta de injeção de falhas ComFIRM [BAR99a] [LEI2000] foi construída através da alteração de rotinas do *kernel* do sistema operacional Linux. Tais alterações possibilitam a inserção do injetor de falhas de comunicação. Desta forma, o injetor de falhas pode interceptar todos os pacotes de rede transmitidos e recebidos pela máquina. Com isso, pode-se duplicar, suprimir ou atrasar pacotes no nível do sistema operacional.

A ferramenta FIDe [GON2000] implementa injeção de falhas pela utilização de recursos presentes na maioria dos sistemas Unix, ou seja, técnicas de depuração. O injetor FIDe controla a execução de uma aplicação e direciona a introdução de falhas nesta aplicação de acordo com regras pré estabelecidas. FIDe utiliza a chamada de sistema *ptrace( )* do Unix para controlar a execução de uma aplicação alvo e injetar falhas em registradores através de técnicas de depuração. A injeção de falhas consiste na alteração dos parâmetros de resposta fornecidos por uma chamada de sistema da aplicação alvo.

Bibliografia

[ARL90] ARLAT, J. et al. Fault-injection for dependability validation: a methodology and some applications. **IEEE Transactions on Software Engineering**, New York, v.SE-16, n. 2, p. 166-181, Feb. 1990.

[BAC94] BARCELLOS, A M. P. et al. Um ambiente para programação de aplicações distribuídas em redes de workstations. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, SBRC, 12., 1994, Curitiba, PR. **Anais...** Curitiba: SBC, 1994. 598p.

[BAR95] BARCELOS, P. P. A.; WEBER, T. S. Experimentação e Avaliação de Protocolos de Difusão Confiável. In: SIMPÓSIO DE COMPUTADORES TOLERANTES A FALHAS, SCTF, 6., 1995, Canela, RS. **Anais...** Porto Alegre: SBC, 1995. 277p.

[BAR96] BARCELOS, P. P. A. **ADC – Ambiente para Experimentação e Avaliação de Protocolos de Difusão Confiável**. 1996. 113p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[BAR99a] BARCELOS, P. P. A; LEITE, F. O.; WEBER, T. S. Building a Fault Injector to Validate Fault Tolerant Communication Protocols. In: PARALLEL COMPUTING SYMPOSIUM, PCS, 1999, Ensenada, Baja California, México. **Proceedings...** [S.l.:s.n.], 1999.

[BAR99b] BARCELOS, P. P. A. **Análise da Viabilidade da Implementação de um Injetor de Falhas de Comunicação Não Intrusivo**. 1998. 68p. Proposta de Tese (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[BEC98] BECK, M. et al. **Linux Kernel Internals**. Wokingham: Addison Wesley, 1998.

[CAR95] CARREIRA, J.; MADEIRA, H.; SILVA, J. G. Assessing the Effects of Communication Faults on Parallel Applications. In: INTERNATIONAL COMPUTER PERFORMANCE AND DEPENDABILITY SYMPOSIUM, 1995, Erlangen, Germany. **Proceedings...** [S. l.]: IEEE, 1995. p. 214-223.

[CAR98] CARREIRA, J., MADEIRA, H., SILVA, J. G. Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers. **IEEE Transactions on Software Engineering**, New York, v.SE-24, n. 2, Feb. 1998.

[CHI89] CHILLAREGE, R.; BOWEN, N. S. Understanding Large-System Failures: A Fault-Injection Experiment. In: INTERNATIONAL

SYMPOSIUM ON FAULT-TOLERANT COMPUTING SYSTEMS, FTCS, 19., 1989, Los Alamitos, California. **Proceedings...** [S. l.]: IEEE CS Press, 1989. p.356-363.

[CHO92] CHOI, G. S.; IYER, R. K. FOCUS: An Experimental Environment for Fault Sensitivity Analysis. **IEEE Transactions on Computers**, New York, v. 41, n. 12, p. 1515-1526, Dec. 1992.

[CLA93] CLARK, J. A.; PRADHAN, D. K. React: A Synthesis and Evaluation Tool for Fault-Tolerant Multiprocesor Architectures. ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM, 1993, Piscataway, N. J. **Proceedings...** [S. l.]: IEEE Press, 1993. p. 428-435.

[CLA95] CLARK, J. A.; PRADHAN, D. K. Fault Injection – A Method for Validating Computer-System Dependability. **Computer**, New York, v. 28, n. 6, p. 47-56, June 1995.

[CRI91] CRISTIAN, F. Understanding Fault-Tolerant Distributed Systems. **Communications of the ACM**, New York, v. 34, n. 2, p. 57-78, Feb. 1991.

[CUN2000] CUNHA, J. C.; RELA, M. Z.; SILVA, J. G. **RT-Xception:** Fault-Injection for Real-Time. Coimbra: Centro de Informatica e de Sistemas da Universidade de Coimbra, 2000. (TR-2000/002).

[DAW94] DAWSON, S.; JAHANIAN, F. **Probing and Fault Injection of Protocol Implementations**. Michigan: University of Michigan, 1994. (CSE-TR-217-94).

[DAW96] DAWSON, S.; JAHANIAN, F.; MITTON, T. **ORCHESTRA**: A Fault Injection Environment for Distributed Systems. Ann Arbor, Michigan: University of Michigan, 1996. (CSE-TR-318-96).

[DAW98] DAWSON, S. **Message Level Fault Injection in Distributed Systems**. 1998. 141 p. Tese (Doutorado em Ciência da Computação) – Science Computer and Engineering Department, University of Michigan, Michigan.

[FON97] FONSECA, K. V. O. **Uma Metodologia de Configuração do Suporte de Comunicação de Sistemas Tempo Real Críticos**. 1997. 115 p. Tese (Doutorado em Engenharia Elétrica) – Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis.

[FUC96] FUCHS, E. **Software Implemented Fault Injection**. 1996. 147 p. Tese (Doutorado em Engenharia Elétrica) – Institut für Technische Informatik, Technische Universität Wien, Viena.

[GON2000] GONÇALVES, L. C. R.; WEBER, T. S. Injeção de Falhas Via Depuradores. In: WORKSHOP DE SOFTWARE LIVRE, WSL, 1., 2000, Porto Alegre, RS. **Anais...** Porto Alegre: SBC, 2000.

[HAN95] HAN, S; SHIN, K. G.; ROSENBERG, H. A. DOCTOR: An integrateD sOftware fault injeCTion enviRonment for Distributed real-time systems. In: INTERNATIONAL COMPUTER PERFORMANCE AND DEPENDABILITY SYMPOSIUM, 1995, Erlangen, Germany. **Proceedings...** [S. l.]: IEEE, 1995. p. 204-213.

[HSU97] HSUEH, M.; TSAI, T. IYER, R. K. Fault Injection Techniques and Tools. **Computer**, New York, v. 30, n. 4, p. 75-82, Apr. 1997.

[JEN94] JENN, E. et al. Fault Injection into VHDL Models: The MEFISTO Tool. In:. INTERNATIONAL SYMPOSIUM ON FAULT TOLERANT AND COMPUTING SYSTEMS, FTCS, 24., 1994, Austin, Texas. **Proceedings...** Washington, D.C.: IEEE, 1994. p.336-344.

[JRT99] LIESENBORGS, J. **Manual da JRTLIB**. Disponível em: <http://lumumba.luc.ac.be/jori/jrtplib/jrtplib.html>. Acesso em 24 abr. 2000.

[KAN95] KANAWATTI, G. A. et al. FERRARI: A flexible software-based fault and error injection system. **IEEE Transactions on Computers**, New York, v. 44, n. 2, p. 248-260, Feb. 1995.

[KAO93] KAO, W. et al. FINE: A Fault Injection and Monitoring Environment for Tracing the Unix System Behaviour under Faults. **IEEE Transactions on Software Engeneering**, New York, v.SE-19, n. 11, Nov. 1993.

[KAO94] KAO, W.; IYER, R. K. DEFINE: a distributed fault injection and monitoring environment. In: WORKSHOP ON FAULT-TOLERANT PARALLEL AND DISTRIBUTED SYSTEMS, 1994. **Proceedings...** [S.l.]: IEEE, 1994.

[KAR94] KARLSSON, J. et al. Using Heavy-Ion Radiation to Validate Fault-Handling Mechanisms. **IEEE Micro**, New York, v. 14, n. 1, p. 8-23, Feb. 1994.

[KER90] KERNIGHAN, B. W. **C, a linguagem de programação**: padrão ANSI. Rio de Janeiro: Campus, 1990.

[KOP94] KOPETZ, H. TTP – A Protocol for Real Time Systems. **Computer**, New York, p. 14-23, Jan. 1994.

[KRI98] KRISHNAMURTHY, N. et al. A design methodology for software fault injection in embedded systems. In: INTERNATIONAL WORKSHOP ON DEPENDABLE COMPUTING AND APPLICATIONS, IFIP, 1998,

Johannesburg, South Africa. **Proceedings...** Johannesburg:[s.n.], 1998. p. 237-248.

[LAP85] LAPRIE, J. C. Dependable Computing and Fault Tolerance: Concepts and Terminology. In:. INTERNATIONAL SYMPOSIUM ON FAULT TOLERANT AND COMPUTING SYSTEMS, FTCS, 15., 1985, Ann Arbor, USA. **Proceedings...** Washington, D.C.: IEEE, 1985. p. 2-11.

[LEI2000] LEITE, F.; WEBER, T. S. Injeção de Falhas de Comunicação em Linux. In: WORKSHOP DE SOFTWARE LIVRE, WSL, 1., 2000, Porto Alegre, RS. **Anais...** Porto Alegre: SBC, 2000.

[MAL94] MALCOLM, N.; ZHAO, W. The Timed-Token Protocol for Real-Time Communications. **Computer**, New York, v. 27, n. 1, p. 35-41, Jan. 94.

[MAN2001] MANFREDINI, R. A. **Condução de Experimentos de Injeção de Falhas em Bancos de Dados Replicados**. 2001. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[MAR93] MARTINS, E. Validação Experimental da Tolerância a Falhas: a Técnica de Injeção de Falhas. In: SIMPÓSIO DE COMPUTADORES TOLERANTES A FALHAS, SCTF, 5., 1993, São José dos Campos, SP. **Anais...** São José dos Campos: INPE, 1993. v.2, p. 56-70.

[OLI97] OLIVEIRA, R. S. **Escalonamento de Tarefas Imprecisas em Ambiente Distribuído**. 1997. 137 p. Tese (Doutorado em Engenharia Elétrica) – Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis.

[PRA96] PRADHAN, D. **Fault-Tolerant Computer System Design**. Englewood Cliffs, New Jersey: Prentice-Hall, 1996.

[RAM94] RAMAMRITHAM, K.; STANKOVIC, J. A. Scheduling Algorithms and Operating Systems Support for Real-Time Systems. **Proceedings of the IEEE**, New York, v. 82, n. 1., p. 55-67.

[ROS96] ROSENBERG, J. B. **How Debuggers Work:** Algorithms, Data Structures, and Architecture. New York: John Wiley & Sons, 1996.

[ROS98a] ROSA, A. A.; MARTINS, E. Using reflexive programming to inject faults into object-oriented systems. In: INTERNATIONAL WORKSHOP ON DEPENDABLE COMPUTING AND ITS APPLICATIONS, IFIP, 1998, Johannesburg, South Africa. **Proceedings...** Johannesburg: [s.n.],1998. p. 227-236.

[ROS98b] ROSA, A. A.; MARTINS, E. Using a Reflexive Architecture to Validate

Object-oriented Applications by Fault Injection. In: WORKSHOP ON REFLECTIVE PROGRAMMING IN C++ AND JAVA, 1998. Vancouver, Canada, **Proceedings...** [S.l.:s.n.], 1998.

[SCH96] SCHULZRINNE, H. G. et al. **RTP:** A Transport Protocol for Real-Time Applications. RFC 1889. 1996. Disponível em: <http://www.faqs.org/rfcs/rfc1889.html>. Acesso em: 02 maio 2000.

[SCH97] SCHULZRINNE, H. G. **RTP – Real-Time Transport Protocol**. 1997. Disponível em: <http://www.cs.columbia.edu/~hgs/RTP>. Acesso em: 02 maio 2000.

[SEG88] SEGALL, Z. et al. FIAT – Fault injection based automated testing environment. In: INTERNATIONAL SYMPOSIUM ON FAULT TOLERANT COMPUTING SYSTEMS, FTCS, 18., 1998, Los Alamitos, California. **Proceedings...** New York: IEEE, 1988. p. 102-107.

[SHI94] SHIN, K. G.; RAMANATHAN, P. Real-Time Computing: A New Discipline of Computer Science and Engineering. **Proceedings of the IEEE**, New York, v. 82, n. 1, p. 6-24, 1994.

[SIE98] SIEWIOREK, D.; SWARZ, R. S. **Reliable Computer Systems – Design and Evaluation.** 3rd ed. [S.l.]: Editorial, Sales, and Customer Service Office, 1998.

[SIL94] SILBERSCHATZ, A.; GALVIN, P. B. **Operating Systems Concepts**. [S.l.]: Addison Wesley, 1994.

[SOT97a] SOTOMA, I.; WEBER, T. S. Desenvolvimento de uma Ferramenta para a Injeção de Falhas em Sistemas Distribuídos baseada em AFIDS. In: SIMPÓSIO DE COMPUTADORES TOLERANTES A FALHAS, SCTF, 7., 1997, Campina Grande, PB. **Anais...** Porto Alegre: II da UFRGS, 1997. p. 337-352.

[SOT97b] SOTOMA, I.; WEBER, T. S. AFIDS: Arquitetura para Injeção de Falhas em Sistemas Distribuídos. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, SBRC, 15., 1997, São Carlos, SP. **Anais...** São Carlos: UFSCar, SBC, 1997. p. 294-309.

[STA88] STANKOVIC, J. A; RAMAMRITHAM, K. **Hard Real Time Systems**. New York: IEEE, 1988.

[STE97] STEFANI, M. R.; MARTINS, E. Análise de Traço e Geração de Diagnósticos para Testes Baseados em Injeção de Falhas por Software. In: SIMPÓSIO DE COMPUTADORES TOLERANTES A FALHAS, SCTF, 7., 1997, Campina Grande, PB. **Anais...** Porto Alegre: II da UFRGS, 1997. p. 321-336.

[TAN97] TANENBAUM, A. S. **Computer Networks**. Englewood Cliffs, New Jersey: Prentice-Hall, 1997.

[TEI95] TEIXEIRA JÚNIOR, C. A.; WEBER, T. S. FIX – uma ferramenta para recuperação de aplicações distribuídas no sistema operacional Unix. In: SIMPÓSIO DE COMPUTADORES TOLERANTES A FALHAS, SCTF, 6., 1995, Canela, RS. **Anais...** Porto Alegre: II da UFRGS, 1995.

[VAR95] VARDANEGA, T. et al. On the Development of Fault-Tolerant On-Board Control Software and its Evalutation by Fault Injection. In: INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, FTCS, 25., 1995. **Proceedings...** [S.l.: s.n.], 1995. p. 510- 515.

[VER93] VERÍSSIMO, P. J. E. Real Time Communication. In: MULLENDER S. J.; VERÍSSIMO, P. J. E. **Distributed Systems**. 2nd ed. [S.l.]: Addison-Wesley, 1993. p. 447-490.

[VER2001] VERÍSSIMO, P. J. E.; RODRIGUES, L. **Distributed Systems for Systems Architects**. [S.l.]: Kluwer Academic Publishers, 2001.