# Injeção Distribuída de Falhas de Comunicação com Suporte à Controle e Coordenação de Experimentos

**Gustavo M. Oliveira, Sérgio L. Cechin, Taisy S. Weber**

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

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

***Abstract.*** *Fault injection intents to insert artificial faults in systems under test to assess their behavior in abnormal situations. However, few efforts have been focused to support distributed systems. Thus, the difficulties in reproducing common fault scenarios in these systems, such as network partitioning, remain open. Therefore, this paper presents mechanisms for more expressive descriptions of distributed fault scenarios with a high abstraction level. Complementing these mechanisms, it is presented an overall coordinator of experiments to keep a centralized control of the targets, and to coordinate testing of critical points in the system.*

***Resumo.*** *Injeção de falhas consiste na inserção artificial de falhas em sistemas sob teste para avaliar seu comportamento diante de situações anormais. Contudo, poucos esforços voltam-se para sistemas distribuídos. Desta forma, a reprodução de cenários de falhas comuns em tais sistemas, como particionamentos de rede, permanece em aberto. Logo, este trabalho apresenta mecanismos para descrições mais expressivas de cenários de falhas distribuídas em diversos pontos da rede com um alto nível de abstração. Unido a este é apresentado um coordenador global de experimentos para um controle centralizado das estações-alvo, bem como para realização de testes coordenados em pontos críticos do sistema.*

## 1. Introdução

O crescimento da adoção de sistemas computacionais para a execução de atividades banais no dia-a-dia, que vão desde simples aplicativos de *e-commerce* até complexos mecanismos de controle de tráfego aéreo, traz consigo um aumento da dependência sobre estas aplicações e, por consequência, na demanda por sistemas com características de dependabilidade. Desta forma, a inclusão de técnicas de tolerância a falhas com o objetivo de elevar o grau de dependabilidade destes sistemas já é uma prática comumente adotada, uma vez que quando não detectadas, falhas podem ter consequências catastróficas durante a vida útil de um sistema. Quando conectados por meio de um canal de comunicação, o caráter distribuído de um sistema acarreta em cuidados acerca da dependabilidade, como colapsos de nodo, atrasos na entrega de pacotes, particionamento de rede, entre outros fatores inexistentes em sistemas auto-contidos. Assim, a complexidade do sistema é elevada de tal maneira que a implementação de procedimentos de tolerância a falhas seja imprescindível, e não apenas mais um atributo associado ao sistema [Dawson et al. 1997, Buchacker and Sieh 2001, Galla et al. 2004, Drebes 2005].

De acordo com a literatura, comparada às demais abordagens propostas, como modelos analíticos e formais, a avaliação de dependabilidade de sistemas através da injeção de falhas também é apresentada como uma solução factível e eficiente, pois não depende da ocorrência natural das falhas e permite alta controlabilidade sobre o ambiente de testes [Clark and Pradhan 1995, Martins et al. 2002, Looker et al. 2005]. A injeção de falhas pode ser feita tanto por *hardware* quanto por *software*. Entretanto, a primeira delas foge ao escopo deste trabalho. Apesar de possuírem limitações de acesso e possibilidades de provocar interferências no desempenho do sistema, injetores de falhas por *software* não necessitam de periféricos com custos elevados e ainda permitem a aplicação da técnica em serviços específicos com maior facilidade [Hsueh et al. 1997].

Grande parte das ferramentas de injeção de falhas não é acessível à comunidade ou, se disponibilizadas, apresentam documentação pobre e, muitas vezes, são pouco funcionais em razão de terem sido implementadas para avaliação de ambientes específicos.

Injetores de falhas de comunicação tornaram-se ferramentas muito valiosas no âmbito de avaliação e análise de dependabilidade de sistemas. Contudo, injetores de falhas de comunicação não suportam, obrigatoriamente, a injeção distribuída de falhas, referentes à realização de experimentos de forma coordenada e cooperativa para execução de uma determinada tarefa em ambientes distribuídos.

A carência de suporte a sistemas distribuídos, por parte dos injetores de falhas encontrados na literatura [Dawson and Jahanian 1995, Stott et al. 2000, Drebes 2005], dá foco para o estudo de um tipo especial de falha de comunicação. Trata-se do particionamento de rede, uma fragmentação da rede em sub-redes desconectadas que impede a coordenação de atividades entre estações presentes em partições distintas da rede [Birman 1997, Veríssimo and Rodrigues 2001].

A figura 1 exemplifica o problema descrito acima. Neste ambiente, três processos iniciam com o mesmo estado **S0**. Um eventual particionamento de rede mantém as estações **X** e **Y** conectadas, enquanto **Z** encontra-se isolada em outra partição. Em seguida, a mensagem **m1** é processada por **X** e **Y** que passam para o estado **S1**. Na outra partição, a mensagem **m2** é processada pela estação **Z**, alterando seu estado atual para **S2**. Quando o particionamento de rede é tratado, **S1** e **S2** são diferentes acarretando em uma divergência entre estados. Logo, são necessárias estratégias para tratar conflitos gerados pelo particionamento que, por sua vez, precisam ser provocados para que estas estratégias possam ser avaliadas.

Além da descrição e injeção de falhas de particionamento de rede, o presente trabalho é motivado pelas seguintes observações:

1. a necessidade de coordenação entre estações participantes do experimento seguindo alguns estados, a fim de garantir que rotinas específicas, definidas na carga de falhas, sejam executadas corretamente;

2. a necessidade de gatilhos bem definidos (temporais, por estado parcial do sistema, ou ainda por número de pacotes) para marcar o início de atividades de injeção de falhas, bem como alterar o comportamento do sistema em tempo de execução;

3. a necessidade de coletar dados acerca do comportamento do sistema diante de

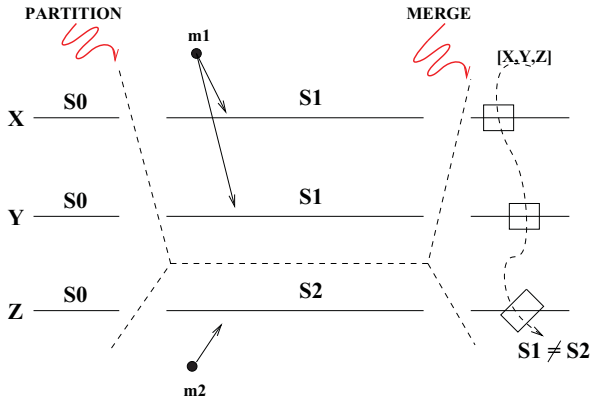


**Figure 1. Particionamento de rede**

falhas, para que seja possível identificar fragilidades no sistema.

Com base em questionamentos, necessidades e inadequações das ferramentas atuais levantados para a avaliação de dependabilidade de sistemas distribuídos, este trabalho tem por objetivo apresentar uma proposta para execução de testes voltados para avaliação de dependabilidade em sistemas distribuídos, considerando o (i) desenvolvimento de uma representação para descrição de cenários distribuídos de falhas em um alto nível de abstração; o (ii) desenvolvimento de um injetor distribuído de falhas, focando falhas de comunicação com ênfase em particionamento de rede; e, por fim, o (iii) desenvolvimento de um coordenador global de experimentos para sincronização, monitoramento e coleta de dados acerca do ambiente de testes.

O restante deste trabalho organiza-se como segue. Na seção a seguir é realizada uma abordagem acerca de injetores de falhas de comunicação encontrados na literatura, apontando suas características principais e limitações que inviabilizam seu reaproveitamento neste trabalho. A seção 3, por sua vez, apresenta o processo de execução de testes de particionamento de rede quando utilizados injetores presentes na literatura. Já a seção 4 introduz uma nova linguagem para possibilitar a especificação dos cenários-alvo deste trabalho. Na seção 5 é detalhada uma arquitetura para desenvolvimento de um novo injetor de falhas de comunicação. Por fim, a seção 6 finda este trabalho apresentando as considerações finais.

## 2. Trabalhos Relacionados

Com o passar dos anos, diversas ferramentas auxiliares para testes de protocolos de rede foram desenvolvidas com o objetivo de avaliar não apenas sua implementação, como também seu desempenho. Dentre as ferramentas mais populares destacam-se o (i) *DBS* (*Distributed Benchmark System*) [Murayama and Yamaguchi 1997], que permite múltiplos envios de dados de forma coordenada para posterior avaliação do comportamento do protocolo *TCP* (*Transmission Control Protocol*); o (ii) *Dummynet* [Rizzo 1997], uma ferramenta com alto grau de flexibilidade para gerenciamento de banda a partir da simulação de condições desejáveis no tráfego de rede, como atrasos, filas, descartes de pacotes, entre outros; por fim, uma ferramenta de propósito geral, o (iii) *NIST Net* [Carson and Santay 2003], que permite a emulação de características críticas presentes

em redes de larga escala, como congestionamentos e descarte de pacotes, para fins de avaliação do desempenho dinâmico em redes *IP* (*Internet Protocol*).

Apesar da alta disseminação destas ferramentas de teste, as necessidades de avaliação dos requisitos mínimos para alcançar dependabilidade não são satisfeitos. Mesmo com o suporte à reprodução de comportamentos característicos de ambientes de comunicação, como o descarte e atraso de mensagens, ferramentas de teste não suportam a construção de um cenário de falhas elaborado para a realização de testes bem definidos de acordo com o gerente de testes, de maneira que funções específicas do alvo sob teste possam ser executadas corretamente. Para tanto, diferentes injetores de falhas de comunicação vêm sendo propostos pela comunidade científica a fim de garantir a realização de testes, seguindo especificações definidas pelo gerente de testes, seja para avaliação de protocolos de comunicação, seja para avaliação de aplicações distribuídas.

ORCHESTRA [Dawson et al. 1996] consiste em avaliar e validar características temporais e de dependabilidade de protocolos distribuídos. Sua arquitetura baseada em camadas, onde nenhuma distinção é feita entre protocolos de nível de aplicação, comunicação, ou físico, integra-se a uma linguagem de alto nível para descrição de cenários de falhas. Entretanto, apesar de possuir um poderoso sistema para descrição de cenários de falhas de comunicação através de *Tcl scripts*, ORCHESTRA não suporta distribuição de falhas para emulação de particionamentos de rede. Além disso, não permite injeção de falhas coordenadas, isto é, a capacidade de injetar falhas em um nodo de acordo com o comportamento de um segundo nodo participante do experimento.

Em outra ferramenta, NFTAPE [Stott et al. 2000], é proposto um *framework* voltado para avaliação de dependabilidade de sistemas distribuídos com o emprego de uma arquitetura modular com suporte a múltiplos modelos de falhas, múltiplos mecanismos de disparo de falhas, múltiplos alvos e métodos versáteis de coleta e reporte de erros. Todavia, sua arquitetura baseada em componentes torna sua utilização inviável para avaliação de aplicações distribuídas através do suprimento de mensagens passantes na rede. Nestes, a integração de mecanismos de disparo e a injeção de falhas é desejado, uma vez que falhas tendem a ser injetadas sobre pacotes de acordo com seu conteúdo. Em outras palavras, tanto o mecanismo de disparo, quanto o módulo responsável pela injeção necessitam acesso ao mesmo fluxo de comunicação.

VIRTUALWIRE [De et al. 2003], por sua vez, é uma ferramenta de injeção de falhas e análise de sistemas capaz de emular qualquer conjunto de falhas de comunicação a partir de uma abstração para uma rede virtual em cima da rede física usada em um ambiente experimental. Sua estrutura suporta a especificação de tipos de falhas assim como o seu respectivo evento de disparo através de *scripts* independentes. Já sua natureza distribuída é efetivada através da implementação de um protocolo de controle que gerencia o injetor presente em múltiplos nodos, caracterizando sua alta escalabilidade. Por outro lado, assim como ORCHESTRA, VIRTUALWIRE não suporta a descrição de particionamento de rede, alvo do presente trabalho.

LOKI [Chandra et al. 2004] é um injetor de falhas para aplicações distribuídas que assume a ocorrência de defeitos em sistemas distribuídos e sua possível dependência de um estado global do mesmo para o disparo de falhas. Seu processo de execução inclui uma máquina de estados responsável pela manutenção da visão parcial do estado

global do sistema, pela injeção de falhas de acordo com estados específicos, e pela coleta de informações acerca de troca de estados. Para construção do ambiente de falhas, LOKI adota uma linguagem de alto nível para manutenção do estado local de estações em teste. Entretanto, LOKI requer instrumentação de código da aplicação-alvo. Além disso, a descrição de cenários é baseada somente em estados globais do sistema, dificultando a construção de cenários de particionamentos de rede que independam do estado global do sistema.

Também voltado para avaliação de dependabilidade de aplicações distribuídas, FAIL-FCI [Hoarau and Tixeuil 2005] propõe uma linguagem abstrata de alto nível para construção de cenários de falhas, (*FAIL - FAult Injection Language*), integrada a uma plataforma para injeção distribuída de falhas (*FCI - FAIL Cluster Implementation*). Seu modelo de injeção de falhas adota uma abordagem baseada em *software* de depuração, de maneira que a aplicação sob testes pode ser interrompida ao executar funções específicas e retomada posteriormente de acordo com a descrição do cenário de falhas construído. Não obstante, sua arquitetura distribuída e escalável dá-se a partir da comunicação explícita entre agentes presentes em cada estação, possibilitando a reprodução de comportamentos específicos da aplicação. Entretanto, os mecanismos adotados por esta ferramenta apresentam-se inviáveis para avaliação de aplicações distribuídas em cenários de particionamentos de rede, uma vez que o suprimento de mensagens do fluxo de comunicação é obstruído pelo *software* de depuração.

Desenvolvido pelo grupo de tolerância a falhas da UFRGS, FIONA (*Fault Injector Over Network Applications*) [Jacques-Silva et al. 2004] é um injetor de falhas para validação experimental de mecanismos tolerantes a falhas existentes em aplicações distribuídas implementados em *Java*, com base no protocolo *UDP*. Sua extensão [Jacques-Silva et al. 2006] propõe uma arquitetura distribuída, que permite uma configuração centralizada de múltiplos cenários de falhas, bem como suporte para uma gama maior de modelos de falhas associados a sistemas distribuídos, incluindo falhas de particionamento de rede.

Assim como FIONA, FIERCE (*Fault Injection Environment for Remote Communication Evaluation*) [GERCHMAN and WEBER 2006] e FIRMI (*Fault Injector for RMI*) [Vacaro and Weber 2006] também voltam-se para validação experimental de aplicações *Java*, diferenciando-se apenas pelo pelo protocolo de comunicação adotado para troca de mensagens (*UDP*, *TCP* e *RMI - Remote Method Invocation*, respectivamente). FIERCE e FIRMI seguem o mesmo padrão de FIONA. Entretanto, FIERCE tem seu modelo de falhas baseado na utilização de códigos de erro retornados pela biblioteca de comunicação do sistema operacional e o conhecimento prévio dos estados que os módulos do protocolo vigente podem assumir, conforme proposto por Neves [Neves and Fuchs 1997].

Por fim, FIRMAMENT [Drebes 2005] é um injetor destinado à validação experimental de técnicas de tolerância a falhas de protocolos de comunicação e sistemas distribuídos. Sua descrição de cenários de falhas dá-se por meio de uma linguagem de baixo nível que gera uma especificação em *bytecode* como saída e é interpretado pela ferramenta através de sua máquina virtual, proporcionando maior poder para construção de cenários. Seu modelo de injeção de falhas explora a arquitetura da interface de programação *Netfilter* para acessar o fluxo de execução dos protocolos baseados em *IPv4* e *IPv6*. Portanto,

FIRMAMENT pode ser carregado em qualquer dispositivo executando versões recentes desse sistema sem a necessidade de recompilação do núcleo. Entretanto, não apresenta uma arquitetura distribuída, nem uma linguagem para descrição de cenários de falhas com suporte para reprodução de falhas de particionamento de rede.

## 3. Injeção de Particionamento

Para elaboração desta proposta, pensou-se, primordialmente, na possibilidade de extensão de ferramentas já existentes. Entretanto, concluiu-se ser praticamente impossível o simples reaproveitamento destes trabalhos, seja pela ausência de suporte distribuído em sua arquitetura, seja pela baixa, ou até inexistente, disponibilidade destas ferramentas para a comunidade, ou ainda pelo baixo poder de expressividade suportado pelos seus respectivos mecanismos de descrição de falhas.

A figura 2 representa um sistema distribuído em quatro instantes de tempo diferentes, cada um deles representando as conexões estáveis e aquelas em falha, caracterizando o particionamento da rede original. Após uma quebra de *link* no instante de 60 segundos a rede sofre uma fragmentação, de modo que ficam agrupadas e isoladas de um lado as estações **1, 2** e **3**, e em outra sub-rede as estações **4, 5** e **6**. Passados mais 60 segundos, no instante de 2 minutos, o sistema sofre seu segundo particionamento. Desta vez, a rede fragmenta-se em três sub-redes distintas, mantendo conectadas as estações **1** e **3** na primeira partição, as estações **2, 4** e **6** na segunda e, por fim, a terceira partição isolando a estação **5**. Já no instante de 3 minutos, a rede sofre seu último particionamento, onde a estação **6** perde contato com as demais estações de sua partição e passa a comunicar-se somente com a estação **5**.

### 3.1. Emulação de Particionamentos

Supondo que se deseja testar uma aplicação distribuída no cenário de falhas mostrado na figura 2, é requisitada uma prévia descrição para posterior emulação destes particionamentos. Durante um experimento, para possibilitar a reprodução de situações anômalas em sistemas distribuídos, é necessária a integração de um coordenador de experimentos, para efetuar o preparo do ambiente de testes, com um módulo de injeção de falhas e uma nova linguagem para descrição de cenários de falhas em um alto nível de abstração para obter maior expressividade. Este trabalho apresenta uma nova linguagem de descrição de falhas para o suporte a este tipo de especificação, detalhado a seguir.

### 3.2. Descrição de Particionamentos

Para possibilitar a realização de testes com falhas de particionamento de rede a linguagem proposta deve apresentar alguns requisitos peculiares a sistemas distribuídos. É necessário, em tempo de pré-execução, a declaração das estações para efetivar o mapeamento destas que farão parte do ambiente experimental. Esta abordagem estática impede a inserção ou exclusão de novas estações do experimento em tempo de execução, mas ambientes dinâmicos fogem ao escopo deste trabalho e a solução para esse problema será abordada em trabalho futuro.

A especificação de eventos deve propiciar um modelo simples e prático, de modo que permita ao gerente de testes a visualização do ambiente construído a partir de sua descrição. Para satisfazer esse requisito foram definidas primitivas temporais e baseadas

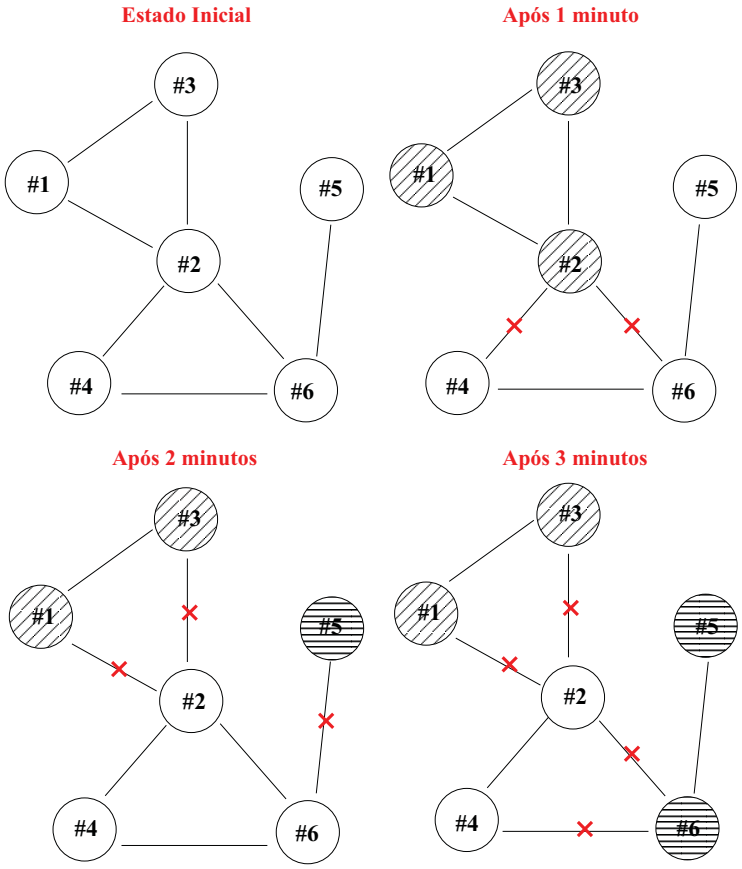


**Figure 2. Exemplo de particionamento de rede**

em Teoria de Conjuntos, para demarcar o instante de ativação da falha e a localização das mesmas, respectivamente, visando modelar os particionamentos desejados.

Na figura 3 é apresentada a descrição, usando a linguagem proposta, das falhas de particionamento que foram exemplificadas na figura 2. A necessidade do desenvolvimento desta linguagem, deu-se pelas dificuldades em encontrar injetores de falhas de comunicação na literatura com suporte à descrições de falhas de particionamento. Embora em estágio inicial de desenvolvimento, verificou-se que, em todos os casos onde foi aplicada, ofereceu uma expressividade adequada, além da simplicidade e praticidade citadas anteriormente.

Nesta nova linguagem, conforme aparece na figura 3, a descrição é iniciada com a especificação das estações que participarão do experimento (linha 1). Sucessivo a este, na linha 2, é demarcado o início do experimento. Após um intervalo temporal de 60 segundos decorridos desde o início do experimento, especificado pela primitiva *after* (linha 3), inicia-se a especificação do comportamento do sistema conforme descrito pelo gerente de testes. A linha 4, por sua vez, é responsável pela reprodução do primeiro particionamento sofrido pelo sistema, fragmentando o mesmo em duas sub-redes distintas e isoladas. No instante de dois minutos após o início do teste (linha 6), o sistema volta a ser fragmentado, aumentando de 2 para 3 o número de particionamentos (linha 7). Passados 60 segundos, já no instante de 3 minutos após o início do experimento (linha 9), o sistema sofre sua

última fragmentação, conforme apresentado na linha 10. Já a linha 13 demarca o instante em que o sistema sofre o *merge*, onde a rede não apresenta mais particionamentos e retorna ao seu estado inicial. Por fim, as linhas 15 e 16 indicam o tempo total de duração do experimento que, após 300 segundos, será encerrado automaticamente.

### 3.3. Arquitetura

Definida a linguagem para especificação de cenários de falhas de particionamento de rede, faz-se necessária sua integração a um injetor de falhas para reprodução dos comportamentos especificados na carga de falhas. Além disso, o fato de operar sobre sistemas distribuídos exige a presença de um módulo para efetuar a coordenação de experimentos, uma vez que a não sincronização de eventos entre estações pode comprometer a integridade de resultados. Não obstante, a presença de um controlador de experimentos é importante para manter um domínio centralizado dos testes, uma vez que sua ausência sujeita o gerente de testes a (i) percorrer cada estação para carregar a ferramenta, bem como sua respectiva carga de falhas e, posteriormente, com o sistema pronto para execução, (ii) percorrer novamente pelas estações do experimento para iniciar a execução dos testes.

```
1:  @declare { HOST_1, HOST_2, HOST_3, HOST_4, HOST_5, HOST_6 }

2:  START:
3:      after (60s) do
4:          partition {HOST_1, HOST_2, HOST_3} {HOST_4, HOST_5, HOST_6};
5:      end
6:      after (120s) do
7:          partition {HOST_1, HOST_3} {HOST_2, HOST_4, HOST_6} {HOST_5};
8:      end
9:      after (180s) do
10:         partition {HOST_1} {HOST_2, HOST_3, HOST_4} {HOST_5, HOST_6};
11:     end
12:     after (240s) do
13:         partition {HOST_1  HOST_2, HOST_3, HOST_4, HOST_5, HOST_6};
14:     end
15: STOP:
16:     after (300s);
```

**Figure 3. Exemplo de descrição de um cenário de particionamento de rede.**

Além disso, após a inicialização do experimento, o gerente de testes também encontra dificuldades para o início do experimento de acordo com o cenário desejado em virtude das limitações de configuração de cenário do respectivo injetor de falhas. Assim que liberada a execução de ações previstas na carga de falhas do injetor, o ambiente permanecerá no mesmo estado permanentemente, visto que a ausência de um controlador para delegação de funções ou coleta de dados do ambiente durante a execução de testes dificulta o comportamento dinâmico do injetor de falhas. Por fim, o gerente de testes teria

de percorrer novamente por cada estação para efetuar o encerramento dos testes. Para tanto, criou-se uma arquitetura voltada para solução destes problemas e é apresentada a seguir.

Conforme apresenta a figura 4, após a criação do *script* de carga de falhas por parte do gerente de testes, este é submetido para uma unidade (***parser***) responsável pela validação deste *script* que, posteriormente, é submetido para outra unidade (***fault-load generator***) encarregada do preparo da carga de falhas específica para cada *host* do experimento. Essa carga determina o momento preciso de inicialização do injetor, a duração temporal da execução de ações do injetor de falhas, como também os canais de comunicação que o módulo de injeção de falhas (***F.I.M*** - *Fault Injection Module*) deve atuar, suprimindo mensagens para emulação de um *crash* de *link*.

Em conjunto com um módulo de coleta de dados, (***data collector***), o ***faultload generator*** responsabiliza-se também pela obtenção de informações acerca do ambiente, como o número de ciclos de relógio de cada *host*. Desta forma é possível realizar a compensação entre os diferentes ciclos de relógio de cada estação. Ou seja, cada *script* tem o conteúdo de seus gatilhos para o disparo de falhas alterado para fins de coordenação de atividades entre estações. Por fim, estas informações são repassadas para uma unidade central de controle e coordenação do experimento, o ***controller***.

Obtidas as informações necessárias do gerente de testes, o sistema passa a operar automaticamente desde a preparação do ambiente, até o encerramento dos experimentos. Antes de iniciar as atividades, para cumprimento dos requisitos mínimos de execução de testes coordenados em sistemas distribuídos, a unidade controladora responsabiliza-se pela distribuição das informações fornecidas pelo gerente de testes às estações participantes do experimento e aguarda até que todas as estações repassem seus estados para a unidade de controle.

Com a preparação do cenário completa, a unidade controladora dá início ao experimento. Uma vez em execução, cada estação (delimitada por linhas pontilhadas) irá conter um módulo responsável pela injeção de falhas (***F.I.M.***) a fim de reproduzir comportamentos no ***target*** definidos na carga de falhas (*script*). O ***target***, por sua vez, é a aplicação distribuída sob teste, ou seja, a aplicação alvo que será avaliada de acordo com seu comportamento na presença de falhas de particionamento, emuladas pela interceptação de mensagens pelo módulo injetor de falhas. Para representar adequadamente o cenário descrito na carga de falhas, a unidade de controle passa a operar em conjunto com o módulo de monitoramento, a fim de verificar e avaliar se o estado atual do sistema encontra-se conforme especificado pelo gerente de testes na carga de falhas do injetor.

Como pode ser visto, há a necessidade de um protocolo de controle experimental para possibilitar a troca de mensagens em busca de um ambiente coordenado. Entretanto, este modelo deve ser estruturado com cautela, uma vez que tais mensagens de controle após o início de um experimento estão suscetíveis às ações do injetor, como o descarte de mensagens, comprometendo a corretude do experimento em relação a sua especificação.

Quando encerrado o experimento, todas as operações realizadas por cada estação são coletadas pelo módulo responsável e enviadas para a unidade de controle. Assim como no ponto de partida, informações também são coletadas ao fim do experimento e submetidas a unidade controladora que, por sua vez, agrupará estas informações e criará

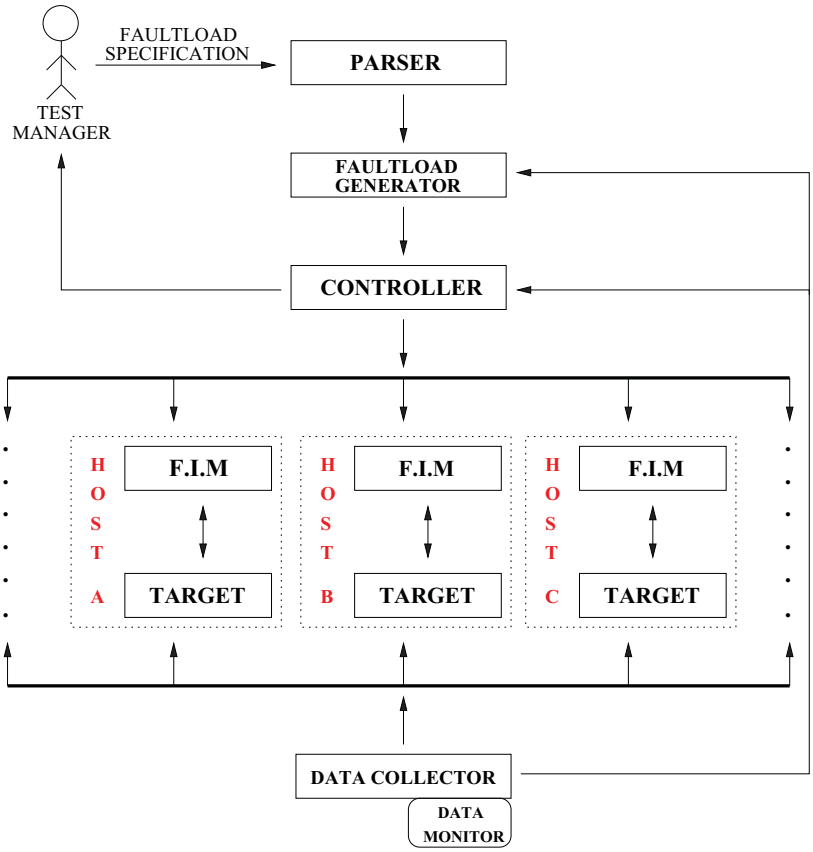


**Figure 4. Arquitetura proposta**

um relatório para visualização de procedimentos realizados em tempo de execução e resultados por parte do gerente de testes.

Para possibilitar a coordenação destes procedimentos, optou-se pela utilização de técnicas de compensação de relógios em virtude do curto tempo de duração do experimento, ou seja, são feitas alterações nas especificações presentes na carga de falhas de acordo com os relógios de cada estação. Desta maneira, a oscilação dos relógios das estações do início até o fim do experimento é relativamente aceitável, não comprometendo a realização de eventos coordenados, quando colocados em uma linha do tempo global. Apesar disso, ao final de cada experimento, os resultados apresentados pelo módulo de coleta de dados são analisados pelo agente de monitoramento. Desta maneira é possível identificar as variações de relógio das máquinas sob teste e permitir a avaliação da integridade do experimento.

### 3.4. Injetor de Falhas

O novo injetor proposto neste trabalho está sendo implementado como um módulo de núcleo do sistema operacional Linux, o que oferece acesso aos fluxos de entrada e saída de pacotes, como também a possibilidade de se testar sistemas baseados nos protocolos *IPv4* e *IPv6*.

Para reprodução de comportamentos descritos na carga de falhas, o módulo injetor de falhas atua na interceptação de mensagens. Esta, por sua vez, é realizada com o auxílio

da arquitetura de interface de programação *Netfilter* [Russel and Welte 2002], conforme proposto em [Drebes 2005]. Com isso, há ganhos de portabilidade desta ferramenta, uma vez que o módulo responsável pela injeção de falhas pode ser carregado em quaisquer dispositivos executando versões recentes desse sistema, sem a necessidade de recompilação do núcleo.

Atualmente, o desenvolvimento desta solução encontra-se em estado avançado. O gerente de testes já é capaz de descrever seu cenário de falhas e submeter para o ***parser***. Com a validação dos dados submetidos pelo usuário, o ***faultload generator*** já efetua o filtro das informações e as deixa prontas para o ***controller*** que distribuirá as cargas de falhas específicas para suas respectivas estações presentes no experimento.

Com a preparação do sistema completa, cada módulo de injeção de falhas presente em cada estação interpreta sua carga de falhas separadamente. No momento, o módulo de coleta de dados, bem como a unidade de controle do experimento estão sendo definidos e implementados. Cuidados especiais devem ser tomados em relação à comunicação entre esses módulos, visto que a troca de mensagens entre eles poderia sofrer descarte de mensagens pela atuação do injetor de falhas, que opera sobre todas as mensagens que circulam pela pilha de protocolos do kernel. Algumas soluções possíveis estão sendo atualmente exploradas. Uma delas é a possibilidade de identificação unívoca das mensagens do protocolo de controle e monitoramento, de modo que estas mensagens sejam devidamente reconhecidas pelo injetor de falhas. Outra possibilidade seria o uso de dois canais de comunicação independentes no ambiente de testes experimentais, um para o fluxo normal de atividades da aplicação alvo sob teste, e outro para a troca de mensagens de controle e monitoramento. Essa segunda possibilidade facilitaria a atuação dos filtros de seleção do injetor, mas implicaria na necessidade de instalar um ambiente de testes diferente do ambiente natural de operação da aplicação alvo.

## 4. Considerações Finais

Sistemas computacionais com elevados requisitos de dependabilidade devem ser submetidos a procedimentos de testes bem elaborados para avaliar o seu comportamento sob falhas, de modo que estes forneçam um serviço consistente e operante mesmo em situações anômalas.

O uso de injetores de falhas de comunicação para avaliação de aplicações baseadas em troca de mensagens não infere no descarte de outras técnicas já existentes de teste de aplicações, uma vez que injetores de falhas de comunicação não atuam sobre todas as características que compõe um dado sistema.

Este trabalho apresentou uma proposta para possibilitar a descrição e emulação de cenários de falhas bastante pertinentes em sistemas distribuídos em um alto nível de abstração. Além disso, foi proposta uma arquitetura para construção de um novo injetor de falhas de comunicação distribuído com suporte a coordenação de experimentos, que permite a realização de testes coordenados em pontos críticos do sistema. Ao passo que a implementação desta solução se aproxima do fim, testes serão realizados sobre sistemas de sincronização de réplicas de dados como forma de validação experimental desta proposta.

Uma próxima fase de desenvolvimento deste trabalho poderia incluir a extensão da linguagem de descrição de falhas de particionamento para suportar mais descrições de

falhas de comunicação. A inclusão de falhas como o atraso, duplicação e corrompimento de mensagens, bem como a possibilidade de disparo de falhas de acordo com com o fluxo de mensagens na rede, ou ainda através do estado específico de uma determinada estação, contribuiria para o desenvolvimento de um injetor de falhas mais abrangente e de maior aplicabilidade. Logo, apenas um injetor de falhas poderia ser empregado para avaliação de protocolos e aplicações distribuídas de uma só vez, eximindo o gerente de testes do uso de um injetor de falhas de comunicação e outro injetor de falhas unicamente para emulação de falhas de particionamento.

Além disso, outro trabalho interessante seria estudar a possibilidade de incorporação de mecanismos de injeção de mensagens para geração de fluxo, desobrigando o gerente de testes da utilização de *softwares* auxiliares para realização desta tarefa. Injeção de mensagens é particularmente útil para a emulação de ataques de segurança. A necessidade dessa nova funcionalidade em injetores de falhas de comunicação foi apontada em trabalhos que utilizaram *FIRMAMENT* para testes de vulnerabilidades de segurança [Martins et al. 2008].

Por fim, outra extensão interessante do trabalho é estudar como suportar diferentes plataformas de operação para possibilitar a avaliação de sistemas que operam sobre ambientes móveis baseados em *IP*.

## Referências

Birman, K. P. (1997). *Building secure and reliable network applications*. Manning Publications Co., Greenwich, CT, USA.

Buchacker, K. and Sieh, V. (2001). Framework for testing the fault-tolerance of systems including os and network aspects. *High-Assurance Systems Engineering, IEEE International Symposium*, page 95.

Carson, M. and Santay, D. (2003). Nist net: a linux-based network emulation tool. *SIGCOMM Comput. Commun. Rev.*, 33(3):111–126.

Chandra, R., Lefever, R., Joshi, K., Cukier, M., and Sanders, W. (2004). A global-state-triggered fault injector for distributed system evaluation. *IEEE Trans. Parallel Distrib. Syst.*, 15(7):593–605.

Clark, J. A. and Pradhan, D. K. (1995). Fault injection: A method for validating computer-system dependability. *IEEE Computer*, 28(6):47–56.

Dawson, S. and Jahanian, F. (1995). Probing and fault injection of protocol implementations. In *ICDCS '95: Proceedings of the 15th International Conference on Distributed Computing Systems*, page 351, Washington, DC, USA. IEEE Computer Society.

Dawson, S., Jahanian, F., and Mitton, T. (1996). Orchestra: A fault injection environment for distributed systems. Technical report, In 26th International Symposium on Fault-Tolerant Computing (FTCS).

Dawson, S., Jahanian, F., and Mitton, T. (1997). Experiments on six commercial tcp implementations using a software fault injection tool. *Softw. Pract. Exper.*, 27(12):1385–1410.

De, P., Neogi, A., and Chiueh, T.-c. (2003). Virtualwire: A fault injection and analysis tool for network protocols. In *ICDCS '03: Proceedings of the 23rd International Con-*

*ference on Distributed Computing Systems*, page 214, Washington, DC, USA. IEEE Computer Society.

Drebes, R. J. (2005). Firmament: Um módulo de injeção de falhas de comunicação para linux. Master's thesis, Instituto de Informática, UFRGS, Porto Alegre.

Galla, T. M., Hummel, K. A., and Pallierer, R. (2004). Software implemented fault injection for safety-critical distributed systems by means of mobile agents. In *HICSS '04: Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, page 90302.1, Washington, DC, USA. IEEE Computer Society.

GERCHMAN, J. and WEBER, T. S. (2006). Emulando o comportamento de tcp/ip em um ambiente com falhas para teste de aplicações de rede. In *Anais do VII Workshop de Testes e Tolerância a Falhas*, Curitiba, BR.

Hoarau, W. and Tixeuil, S. (2005). A language-driven tool for fault injection in distributed systems. In *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 194–201, Washington, DC, USA. IEEE Computer Society.

Hsueh, M., Tsai, T. K., and Iyer, R. K. (1997). *Fault Injection Techniques and Tools*. *Computer*, 30(4):75–82.

Jacques-Silva, G., Drebes, R. J., Gerchman, J., Trindade, J. M. F., Weber, T. S., and Jansch-Porto, I. (2006). A network-level distributed fault injector for experimental validation of dependable distributed systems. In *COMPSAC '06: Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, pages 421–428, Washington, DC, USA. IEEE Computer Society.

Jacques-Silva, G., Drebes, R. J., Gerchman, J., and Weber, T. S. (2004). Fiona: A fault injector for dependability evaluation of java-based network applications. In *NCA '04: Proceedings of the Network Computing and Applications, Third IEEE International Symposium*, pages 303–308, Washington, DC, USA. IEEE Computer Society.

Looker, N., Munro, M., and Xu, J. (2005). A comparison of network level fault injection with code insertion. In *COMPSAC '05: Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05) Volume 1*, pages 479–484, Washington, DC, USA. IEEE Computer Society.

Martins, E., Morais, A., and Cavalli, A. (2008). Generating attack scenarios for the validation of security protocol implementations. In *2nd. Brazilian Workshop on Systematic and Automated Software Testing (SAST)*, pages 21–33, Campinas, Brazil. SBC.

Martins, E., Rubira, C. M. F., and Leme, N. G. M. (2002). Jaca: A reflective fault injection tool based on patterns. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 483–482, Washington, DC, USA. IEEE Computer Society.

Murayama, Y. and Yamaguchi, S. (1997). Dbs: a powerful tool for tcp performance evaluations. In *In SPIE Proceedings of Performance and Control of Network Systems*.

Neves, N. and Fuchs, W. K. (1997). Fault detection using hints from the socket layer. In *SRDS '97: Proceedings of the 16th Symposium on Reliable Distributed Systems (SRDS '97)*, page 64, Washington, DC, USA. IEEE Computer Society.

Rizzo, L. (1997). Dummynet: a simple approach to the evaluation of network protocols. *SIGCOMM Comput. Commun. Rev.*, 27(1):31–41.

Russel, R. and Welte, H. (2002). Linux netfilter hacking howto.

Stott, D. T., Floering, B., Kalbarczyk, Z., and Iyer, R. K. (2000). Nftape: A framework for assessing dependability in distributed systems with lightweight fault injectors. In *IPDS '00: Proceedings of the 4th International Computer Performance and Dependability Symposium*, page 91, Washington, DC, USA. IEEE Computer Society.

Vacaro, J. C. and Weber, T. S. (2006). Injeção de falhas na fase de teste de aplicações distribuídas. In *XX Simpósio Brasileiro de Engenharia de Software*, volume 1, pages 161–176, Florianópolis, BR. SBC.

Veríssimo, P. and Rodrigues, L. (2001). *Distributed Systems for System Architects*. Springer, Boston, USA, 1 edition.