UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

ALOISIO SOARES DE MELO FILHO

Proposta de um Injetor de Falhas Modular para Sistemas Computacionais

Recife

2025

ALOISIO SOARES DE MELO FILHO

Proposta de um Injetor de Falhas Modular para Sistemas Computacionais

TCC apresentado ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal de Pernambuco, Centro Acadêmico de Recife, como requisito para a obtenção do título de bacharel em Sistemas de Informação

Orientador: Prof. Dr. Jamilson Ramalho Dantas

Recife

2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

de Melo Filho, Aloisio Soares.
PROPOSTA DE UM INJETOR DE FALHAS MODULAR PARA SISTEMAS COMPUTACIONAIS / Aloisio Soares de Melo Filho. - Recife, 2025.
16 : il., tab.

Orientador(a): Jamilson Ramalho Dantas
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Sistemas de Informação - Bacharelado, 2025.
8.5.
Inclui referências.

1. Tolerância a falhas. 2. Computação na nuvem. 3. Injetores de falhas. I. Ramalho Dantas, Jamilson . (Orientação). II. Título.

000 CDD (22.ed.)

ALOISIO SOARES DE MELO FILHO

PROPOSTA DE UM INJETOR DE FALHAS MODULAR PARA SISTEMAS COMPUTACIONAIS

TCC apresentado ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal de Pernambuco, Centro Acadêmico de Recife, como requisito para a obtenção do título de bacharel em Sistemas de Informação

Aprovado em: 03/04/2025.

BANCA EXAMINADORA

_____

Profº. Dr. Jamilson Ramalho Dantas (Orientador)

Universidade Federal de Pernambuco

_____

Profº. Dr. Andson Marreiros Balieiro (Examinador Interno)

Universidade Federal de Pernambuco

PROPOSTA DE UM INJETOR DE FALHAS MODULAR PARA SISTEMAS COMPUTACIONAIS[1]

Proposal for a Modular Fault Injector for Computational Systems

Aloisio Soares de Melo Filho[2]
Orientação: Prof. Dr. Jamilson Ramalho Dantas[3]

RESUMO

Os testes de confiabilidade e disponibilidade de serviços em nuvem podem ser eficientemente realizados mediante a utilização de ferramentas de injeção de falhas, que permitem simular diversos cenários problemáticos sem comprometer os sistemas em produção. Uma abordagem particularmente eficaz consiste na implementação de um sistema dotado de interface gráfica, que simplifica significativamente o processo ao oferecer configuração intuitiva dos parâmetros e monitoramento em tempo real dos impactos causados pelas falhas simuladas. Através dessa solução, os usuários têm a capacidade de simular tanto falhas quanto procedimentos de recuperação em componentes de hardware e software, incluindo cenários complexos como quedas de nós específicos, redirecionamento dinâmico de tráfego e avaliação precisa dos mecanismos de tolerância a falhas implementados. A arquitetura proposta pelo estudo emprega um sistema web, com backend desenvolvido em Node.js para gestão e execução dos scripts de injeção de falhas, acoplado a um frontend em React.js que disponibiliza uma interface amigável com controle detalhado dos principais parâmetros de execução e visualização imediata dos resultados.

Palavras-chave: Tolerância a Falhas; Tolerância a falhas em computação em nuvem, computação de borda e sistemas embarcados; Injeção de falhas;

ABSTRACT

Reliability and availability testing for cloud services can be effectively conducted using fault injection tools, which enable simulation of various problematic scenarios without affecting production systems. A particularly efficient approach involves implementing a system with a graphical interface that significantly simplifies the process by offering intuitive parameter configuration and real-time monitoring of the impacts caused by simulated failures. This solution empowers users to simulate both hardware and software failure scenarios and recovery procedures, including complex situations such as specific node failures, dynamic traffic redirection, and precise evaluation of implemented fault tolerance mechanisms. The architecture proposed in the study employs a web-based system featuring a Node.js backend for managing and executing fault injection scripts, coupled with a React.js frontend that provides a user-friendly interface with detailed control over key execution parameters and immediate results visualization.

---

[1] Trabalho de Conclusão de Curso apresentado ao Curso de Sistemas de Informação da Universidade Federal de Pernambuco (UFPE), cuja banca de defesa foi composta pelos seguintes membros: Prof. Dr. Jamilson Ramalho Dantas; Prof. Dr. Andson Marreiros Balieiro, na seguinte data: 3 de abril de 2025.

[2] Graduando em Sistemas de Informação na UFPE.

[3] Professor do Centro de Informática da UFPE.

Keywords: Fault Tolerance;Fault Tolerance in Cloud Computing, Edge Computing, and Embedded Systems;Fault Injection;

1 INTRODUÇÃO

Os objetivos iniciais do uso de métodos de injeção de falhas são validar e avaliar sistemas. A validação envolve comparar o comportamento especificado com o sistema implementado, garantindo que ele funcione conforme o esperado na presença de falhas. Além disso, é necessário verificar se o sistema gerencia as falhas de forma eficaz. As ferramentas de injeção de falhas ajudam a aumentar a confiança na capacidade do sistema de fornecer o serviço planejado [3].

Existem várias ferramentas e técnicas consolidadas para a injeção de falhas em estudos. Feito as encontradas em [5], [9], [11]. Sendo que, elas exigem conhecimento técnico avançado para serem executadas. Além de, serem aplicados em contextos bem específicos tais como GPUs [9] ou Simulink [11].

Somado a isso, uma interface gráfica pode ser uma poderosa aliada na injeção de falhas para avaliar e testar qualquer tipo de sistema, oferecendo uma maneira visual, intuitiva e eficiente de planejar, executar e monitorar testes. Além de facilitar a configuração de cenários de falhas, dispensando a necessidade de programação, ela possibilita acompanhar em tempo real os impactos das falhas no sistema, como gráficos de desempenho, mapas de calor ou logs de eventos. Isso facilita a identificação de gargalos e pontos críticos na infraestrutura.

Por exemplo, através deste sistema, um usuário pode simular a queda de um nó específico em um cluster, monitorar o redirecionamento de tráfego em tempo real e identificar se os mecanismos de tolerância a falhas do sistema funcionaram corretamente, tudo de forma visual e interativa.

Este artigo apresenta o desenvolvimento de um injetor de falhas, estruturado como um sistema web modular. A solução proposta é composta por dois componentes principais: um backend, desenvolvido em *Node.js* [12], responsável por interagir com scripts simuladores de falhas, *hardware*, virtualização de servidores e infraestruturas de rede; e um frontend, escrito em *React.js* [13], que oferece uma interface gráfica para comunicação com o backend, permitindo a execução e o monitoramento das simulações de forma intuitiva e eficiente. Como estudo de caso, realizamos alguns testes iniciais em uma infraestrutura de computação na nuvem.

As principais contribuições incluem:

- A criação de uma ferramenta para simular falhas e reparo em infraestruturas computacionais de forma controlada, sem afetar os sistemas em produção;
- Interface gráfica amigável que torna o planejamento e monitoramento de falhas e reparo mais intuitivo, mesmo para usuários com pouco conhecimento técnico;
- O suporte à análise de desempenho e identificação de gargalos, facilitando melhorias nos mecanismos de tolerância a falhas e na infraestrutura;
- Uma abordagem modular que pode ser adaptada a diferentes cenários e requisitos de sistemas distribuídos;

O resto do trabalho está organizado como segue: No Capítulo 2 são apresentados os conceitos fundamentais para construção do ferramental. E também, no Capítulo 3 são discutidos alguns pontos em comum em trabalhos relacionados. Posteriormente, temos o capítulo 4 que expõe uma visão geral da ferramenta, algoritmos relacionados e como as partes integrantes conversam. Já no capítulo 5 são expostos detalhes mais técnicos da ferramenta. E por último, o capítulo 6 que conclui e entrega pontos possíveis de evolução para melhoria ferramental futura.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção descreve os conceitos fundamentais necessários para uma compreensão completa deste artigo.

2.1 Falhas de software ou hardware

Segundo [7], nem sempre um sistema desempenha a função para a qual foi concebido. Denominam-se fatores de confiabilidade as causas e consequências dos desvios em relação ao comportamento esperado do sistema. E um desses fatores é chamado falha, que consiste em um defeito físico, imperfeição ou anomalia que ocorre em componentes de hardware ou software [7]. Além disso, quando uma falha induz uma modificação errônea no estado de uma máquina, configura-se um erro [7].

Falhas de hardware de praticamente todos os tipos podem ser facilmente injetadas pelos dispositivos disponíveis para essa finalidade. Existem ferramentas especializadas de hardware capazes de inverter bits instantaneamente nos pinos de um chip, variar a alimentação de energia ou até mesmo bombardear o sistema ou chips com íons pesados - métodos considerados capazes de causar falhas semelhantes às falhas transitórias reais em hardware. Uma ferramenta de software cada vez mais popular é o injetor de falhas implementado por software, que altera bits nos registradores do processador ou na memória, produzindo assim os mesmos efeitos que falhas transitórias de hardware [7].

E também, temos as falhas de software em que segundo [7], "são sempre consequência de erros de projeto, seja na fase de especificação ou durante a codificação. Todo engenheiro de software sabe que um produto só pode ser considerado livre de bugs até que o próximo defeito seja encontrado. Muitas dessas falhas permanecem latentes no código e só se manifestam durante a operação, especialmente sob cargas de trabalho intensas ou incomuns, e em contextos temporais específicos.". E também, diz que "no processo de desenvolvimento de software, falhas podem ser introduzidas em todas as etapas: definição de requisitos, especificação de requisitos, projeto, implementação, testes e implantação."

Além disso, segundo [5] "as camadas de hardware e software interagem, permitindo que as falhas de hardware se propaguem para a camada de software, potencialmente causando falhas de software, como corrupção de dados, encerramento abrupto, falha de rede ou quebra de aplicativo. Assim, falhas de software podem resultar em falhas de software e hardware."

Portanto, a confiabilidade de um sistema está diretamente ligada à sua capacidade de lidar com falhas tanto de hardware quanto de software, considerando que ambas podem ocorrer de maneira isolada ou interdependente. As falhas de hardware, frequentemente induzidas por fatores externos ou limitações físicas, podem

desencadear erros no sistema que, por sua vez, impactam negativamente o comportamento da camada de software. Já as falhas de software, geralmente originadas de erros de projeto ou implementação, muitas vezes permanecem ocultas até que condições específicas as revelem durante a operação. Diante disso, compreender a origem, propagação e impacto das falhas em todas as camadas do sistema é essencial para o desenvolvimento de soluções mais robustas e confiáveis, capazes de mitigar riscos e assegurar o desempenho esperado mesmo em condições adversas.

2.2 Injetor de falhas

Ao se conduzir um experimento que tenha como objetivo a avaliação de dependabilidade, uma técnica importante a ser considerada, é a utilização de injeção de falha, pois falhas ocorrem de forma imprevisível dentro de um sistema, podendo levar muito tempo para a observação desse evento. A utilização de técnicas de injeção de falha auxilia no controle e monitoramento do experimento, observando o comportamento do sistema durante a ocorrência de eventos de falha.

Técnicas de injeção de falhas permitem a observação do comportamento do sistema através de experimentos controlados, nos quais a presença de falhas é induzida explicitamente nos componentes do sistema. Existem dois tipos de implementação para injetores de falhas: injetores de falha em nível de hardware, e injetores de falha em nível de software [7].

Injeção de falhas baseada em software é uma abordagem flexível, pois pode ser direcionada a aplicativos de software, sistemas operacionais e hardware. No entanto, essa abordagem é limitada até onde os comandos de software podem alcançar. Essa técnica permite a injeção de falhas que podem ocorrer em dois momentos diferentes: (a) tempo de compilação, o aplicativo é modificado antes de ser carregado e executado, erros são injetados no código-fonte do aplicativo de destino simulando efeitos de falhas; (b) tempo de execução, requer um software específico para disparar as falhas. Um dos mecanismos comumente usados para disparar eventos de falha é conhecido como time-out. Este método usa um temporizador (hardware ou software) para controlar a injeção da falha. Quando o tempo predeterminado termina, a falha é injetada [4].

2.3 Websockets

A área de conectividade do HTML5 (HyperText Markup Language) inclui tecnologias como WebSocket, Server-Sent Events e Cross-Document Messaging. Essas APIs (Application Programming Interface) foram incluídas na especificação do HTML5 para ajudar a simplificar algumas áreas onde as limitações dos navegadores impediam os desenvolvedores de aplicativos web de criar o comportamento avançado que desejavam ou onde o desenvolvimento web estava se tornando excessivamente complexo. Um exemplo de simplificação em HTML5 é uma mensagem entre documentos [2].

Com o WebSocket, sua requisição HTTP se torna uma única solicitação para abrir uma conexão WebSocket (seja WebSocket padrão ou WebSocket sobre TLS - Transport Layer Security, anteriormente conhecido como SSL - Secure Sockets Layer) e reutiliza a mesma conexão tanto do cliente para o servidor quanto do servidor para o cliente. O WebSocket reduz a latência porque, uma vez estabelecida

a conexão WebSocket, o servidor pode enviar mensagens à medida que elas se tornam disponíveis. O servidor não precisa aguardar uma solicitação do cliente. Da mesma forma, o cliente pode enviar mensagens para o servidor a qualquer momento [2].

WebSockets permitem uma troca de mensagens bidirecional e em tempo real entre o frontend e o backend. Isso é útil em cenários onde o injetor de falhas pode simular interrupções ou alterações rápidas que precisam ser refletidas instantaneamente no frontend para testes ou monitoramento.

2.4 Diagrama de sequência

A UML (Unified Modeling Language) é uma linguagem visual utilizada para modelar softwares baseados no paradigma de orientação a objeto. Podemos classificar a referida como uma modelagem de propósito geral com a habilidade de ser agregada a todos os domínios da aplicação. Esta linguagem tornou-se padrão, visto que é adotada internacionalmente pela indústria de engenharia de software. A necessidade de modelar um software,surge em decorrência da documentação extremamente detalhada que o sistema precisa ao ser concebido, dessa forma a modelagem é uma forma eficiente de documentá-lo [1].

Dentre os diagramas UML existentes,o diagrama de sequência é comportamental e preocupa-se com a ordem temporal em que as mensagens são trocadas entre os objetos envolvidos em um determinado processo. Em geral baseia-se em um caso de uso definido pelo diagrama de mesmo nome e apoia-se no diagrama de classes em um determinado processo [1].

3 TRABALHOS RELACIONADOS

Este capítulo apresenta alguns dos trabalhos encontrados na literatura que envolvem ferramentas de injeção de falhas em diversos contextos. Em um dos estudos [5], propôs uma avaliação e validação de modelagem hierárquica por meio de cadeias de Markov de tempo contínuo (CTMC) e diagramas de blocos de confiabilidade (RBD) para analisar a disponibilidade de um sistema MEC (Multi-edge computing) baseado em uma rede definida por software (SDN). Em que usa uma ferramenta de injeção de falhas baseada em software para transmitir falhas e reparos no banco de testes para verificar a disponibilidade de um sistema, garantindo que ele pudesse lidar com falhas não fatais.

No contexto de soluções de recuperação de desastres, [6] utilizou a injeção de falhas para verificar os benefícios e desvantagens de tal solução durante as fases de design e implantação inicial. Medidas de confiabilidade obtidas no banco de testes podem ser usadas para ajustar componentes específicos da solução, para verificar modelos analíticos e de simulação, bem como para fornecer bases para a definição de acordos de nível de serviço com os clientes.

Noutro contexto, [4] apresenta uma ferramenta de injeção de falhas, denominada EucaBomber, para estudos de confiabilidade e disponibilidade na plataforma de computação em nuvem Eucalyptus. O EucaBomber permite definir a distribuição de probabilidade associada ao tempo entre eventos gerados. A eficiência do EucaBomber é verificada por meio de cenários de teste onde falhas e reparos são injetados em uma nuvem privada Eucalyptus. Os resultados experimentais são

cruzados com resultados estimados a partir de um Diagrama de Blocos de Confiabilidade, usando os mesmos parâmetros de entrada do teste experimental. Os cenários de teste também ilustram como a ferramenta pode auxiliar administradores e planejadores de sistemas em nuvem a avaliar as políticas de disponibilidade e manutenção do sistema.

[11] apresenta um kit de ferramentas de código aberto para injeção automatizada de falhas e geração de mutantes em modelos Simulink. No qual permite a injeção de falhas em partes específicas, suportando tipos comuns de falhas e operadores de mutação cujos parâmetros podem ser personalizados para controlar o tempo de atuação e persistência da falha. Sinalizadores adicionais permitem que o usuário ative os blocos de falhas individuais durante o teste para observar seus efeitos na confiabilidade geral do sistema. Além de possuir um estudo de caso do domínio da aviônica.

E também nessa linha, [9] apresenta a ferramenta NVBitFI para injeção de falhas em programas de GPU. Essa ferramenta executa a instrumentação de código dinamicamente e seletivamente para instrumentar o conjunto mínimo de kernels dinâmicos de destino. E como não requer acesso ao código-fonte fornece melhorias em desempenho e usabilidade.

Um outro artigo, de [10] , apresenta o GOOFI (Generic Object-Oriented Fault Injection), uma nova ferramenta de injeção de falhas com alta adaptabilidade para diferentes sistemas-alvo e técnicas de injeção. Desenvolvido em Java com banco de dados SQL compatível, o GOOFI oferece portabilidade entre plataformas. Além de possuir uma interface gráfica amigável para o usuário. A versão atual suporta duas técnicas principais: Injeção de Falhas Implementada por Software (pré-execução) e Injeção de Falhas Implementada por Cadeia de Varredura (Scan-Chain).

Os estudos anteriores propõem ferramentas de injeção de falhas utilizadas em contextos diversos de atuação. Porém, elas exigem um conhecimento técnico avançado para serem utilizadas, mesmo possuindo uma interface gráfica amigável.

4 FERRAMENTAL PROPOSTO

4.1 Visão Geral

A utilização de técnicas de injeção de falha auxiliam no controle e monitoramento do experimento, observando o comportamento do sistema durante a ocorrência de eventos de falha [8]. E ainda, além dessas técnicas exigirem um conhecimento técnico avançado, por parte do usuário, não costumam oferecer interfaces amigáveis. Este estudo propõe um sistema que utiliza essa técnica, oferecendo uma interface intuitiva para usuários não técnicos. Por ser uma ferramenta baseada em navegador web, ela é compatível com leitores de tela, garantindo acessibilidade para pessoas com deficiência visual. A arquitetura do sistema é dividida em dois módulos funcionais devidamente nomeados: frontend (interface do usuário) e backend (lógica de processamento).

O frontend desempenha um papel fundamental, servindo como interface com o usuário e inicia o fluxo de injeção de falhas. Apresenta uma formulário, uma área que exibe mensagens oriundas do backend e uma área com validações dos campos.

E ainda, se o usuário preencher o formulário e clicar no botão intitulado "inject" aciona um função de evento que verifica se todos os campos necessários no formulário foram preenchidos corretamente. Caso alguma validação falhe, mensagens de erro são exibidas para o usuário. Posteriormente, após a validação bem-sucedida, uma conexão WebSocket é aberta para se comunicar com o backend, enviando as configurações necessárias para execução do injetor de falhas. Por fim, o frontend exibe em tempo real as mensagens enviados pelo backend,como status das operações, progresso da execução, resultados, garantindo que o usuário acompanhe todas as etapas do processo.

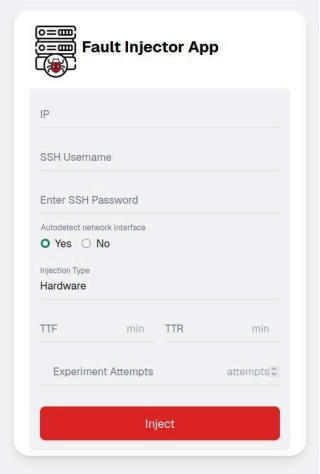
Já o backend recebe as solicitações do frontend, processa as instruções e executa as ações relacionadas à injeção de falhas. Primeiramente, assim que os parâmetros de configuração chegam via WebSocket, o backend valida os dados e inicia os preparativos para a execução. Além disso, caso o usuário opte por selecionar no formulário do frontend a funcionalidade de auto detecção de interfaces, o backend executa uma função de detecção da interface de rede no sistema alvo, que estabelece uma conexão SSH com o servidor-alvo e executa comandos para identificar os IDs das interfaces de hardware disponíveis. É necessário a identificação dessa interface de rede, pois são necessárias para execução dos algoritmos de injeção de falha e reparo.

Além disso, o backend também é responsável pelo agendamento de falhas e reparos, definindo e programando a execução dos comandos necessários com base nos parâmetros recebidos. Em seguida, para garantir o acompanhamento do estado do servidor, uma função que monitora periodicamente se o sistema alvo está ativo ou inativo. Por fim, para manter o usuário sempre informado, mensagens de status, logs e resultados das operações são continuamente enviados ao frontend via WebSocket, garantindo informações atualizadas em tempo real.

4.2 Diagrama de sequência

O fluxo de ações descritas pelo diagrama de sequência são iniciadas na interação do usuário com o formulário do frontend ilustrado na Figura 1.

Figura 1 – Formulário do frontend



Fonte: Elaborado pelo próprio autor (2025).

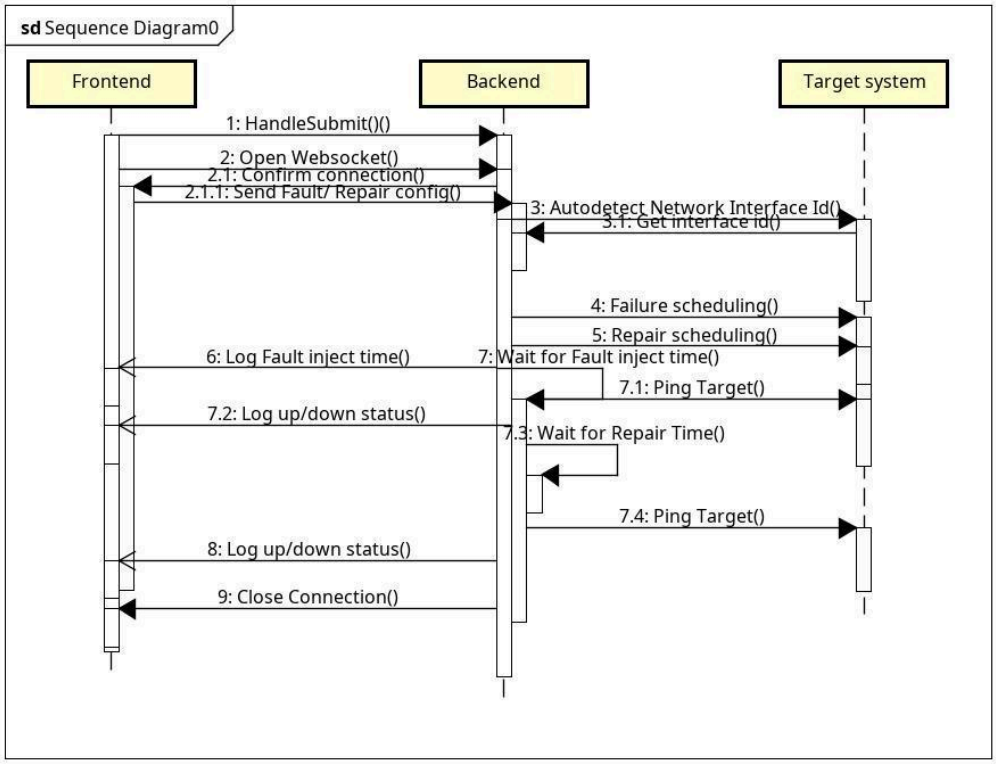
No diagrama de sequência, ilustrado na Figura 2, é possível observar um fluxo completo de ações envolvendo a interação entre o frontend, backend e servidor alvo.

O processo se inicia no primeiro retângulo, representando o frontend, onde o usuário interage com um formulário. Este formulário permite a inserção de informações essenciais para configurar o injetor de falhas. Após preencher os campos requeridos, o usuário clica no botão rotulado com *Inject*.

O clique no botão aciona a função *handleSubmit()*, responsável por validar os dados inseridos no formulário. Nessa etapa, são verificadas condições como o preenchimento de campos obrigatórios e a consistência dos valores fornecidos. Caso todas as validações sejam aprovadas, uma conexão WebSocket é estabelecida com o backend — representado pelo segundo retângulo do diagrama. Durante todo o processo, notificações são exibidas em tempo real na interface do frontend, oferecendo retorno contínuo ao usuário.

Se o usuário optar por ativar a funcionalidade de detecção automática de identificação, o backend executa a função *detectHardwareInterfaceId()*, encarregada de identificar interfaces de hardware no servidor-alvo. Essa função utiliza parâmetros previamente definidos e estabelece uma conexão via SSH para acessar o servidor. Após a conexão, comandos específicos são executados com o objetivo de coletar os IDs das interfaces de rede disponíveis.

Figura 2 – Diagrama de sequência



Fonte: Elaborado pelo próprio autor (2025).

Na sequência, comandos são enviados ao servidor-alvo para agendar falhas e reparos. Essas ações são cuidadosamente coordenadas e, ao serem concluídas, o backend envia ao frontend uma mensagem com o status da operação, utilizando a mesma conexão WebSocket, garantindo visibilidade ao usuário.

Após o agendamento, o backend entra em modo de monitoramento, respeitando o intervalo de tempo definido. Durante este período, é executada uma função de ping

para verificar a disponibilidade do servidor-alvo. O resultado deste monitoramento também é enviado ao frontend, como parte do feedback ao usuário. Finalizadas todas as etapas, o backend encerra a conexão e conclui sua interação com o sistema.

4.3 Pseudocódigos

A Figura 3, contém o pseudocódigo que descreve o fluxo principal do sistema de injeção de falhas, na parte do backend, destacando as etapas de validação, agendamento de falhas e recuperações, e comunicação com o cliente via WebSocket. Ele serve como uma visão geral da lógica do código, facilitando a compreensão e a documentação do sistema.

No princípio, o servidor WebSocket é iniciado e aguarda conexões de clientes. Quando um cliente se conecta, o servidor loga a conexão. O servidor então recebe uma mensagem do cliente, que contém os detalhes da requisição. Se solicitado, o servidor tenta detectar a interface de rede da máquina alvo; se não conseguir, enviar retorna uma mensagem de erro e fecha a conexão. Em seguida, o servidor entra em um loop para realizar múltiplas tentativas de injeção de falhas. Durante este processo, o servidor gera tempos para falha e recuperação, agenda a falha via SSH e envia mensagens ao cliente. Após a falha, o servidor verifica o status do servidor alvo com ping e envia os resultados ao cliente. Na sequência, o servidor agenda a recuperação via SSH, verifica novamente o status do servidor alvo e envia os resultados ao cliente. Finalmente, após todas as tentativas, o servidor fecha a conexão WebSocket.

Figura 3 – Pseudocódigo

Algorithm 1 Injetor de falhas e reparos de hardware

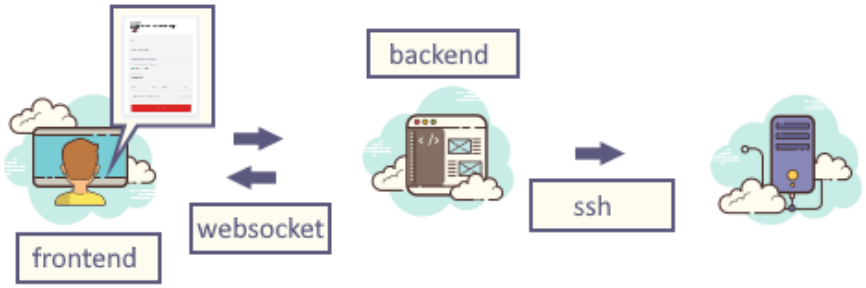
```
1:  Início
2:  Aguardar mensagem WebSocket
3:  if autoDetecção de Interface de Rede ativada then
4:      Executar detectHardwareInterfaceId()
5:      if nenhuma interface detectada then
6:          Enviar status de erro e fechar conexão
7:          Encerrar
8:      end if
9:  end if
10: Definir experimentCount como 0
11: while experimentCount ¡ experimentAttempts do
12:     Incrementar experimentCount
13:     Gerar tempos para falha (timeToFail) e reparo (timeToRepair)
14:     if falha na geração dos tempos then
15:         Enviar status de erro e fechar conexão
16:         Encerrar
17:     end if
18:     Enviar mensagem de início da falha
19:     Executar comando SSH para injetar falha na interface de rede
20:     Aguardar até timeToFail
21:     for i = 0 to 9 do
22:         Verificar status do servidor via ping
23:         Enviar status atualizado ao frontend
24:         Aguardar 5 segundos
25:     end for
26:     Enviar mensagem de início do reparo
27:     Executar comando SSH para restaurar a interface de rede
28:     Aguardar até timeToRepair
29:     for j = 0 to 9 do
30:         Verificar status do servidor via ping
31:         Enviar status atualizado ao frontend
32:         Aguardar 5 segundos
33:     end for
34:     Exibir mensagem de conclusão da iteração
35: end while
36: Fechar conexão WebSocket
37: Fim =0
```

Fonte: Elaborado pelo próprio autor (2025).

5 TESTES INICIAIS

A proposta deste estudo é oferecer uma interface amigável para um injetor de falhas. Essa interface, chamada de frontend, foi construída usando Javascript e a biblioteca React com o framework Nextjs. E a parte do backend, foi usado o Nodejs. A conexão do frontend com o backend é feita através de websockets, pois permitem que o servidor e o cliente (frontend) troquem dados de forma bidirecional sem a necessidade de fazer múltiplas requisições HTTP. Isso significa que o servidor pode enviar dados para o cliente a qualquer momento, sem que o cliente precise solicitar informações continuamente (diferente do modelo tradicional de "request-response" com HTTP). Uma vantagem de ter essa separação entre as aplicações é para permitir que fiquem independentes, para que num caso futuro este backend possa ser acionado via aplicação de celular. A visão geral da ferramenta pode ser observada na Figura 4.

Figura 4 – Visão geral da aplicação



Fonte: Elaborado pelo próprio autor (2025).

5.1 Pré-requisitos e instalação

A ferramenta é composta por duas partes separadas doravante chamadas de frontend e backend que precisam ser executadas concomitantemente. Primeiramente, para que a ferramenta funcione corretamente é necessário estar num sistema operacional Linux e mais especificamente no Ubuntu 22.04 ou Ubuntu 24.04, onde foi testado. Além disso, precisa estar com os pacotes instalados: *nodejs, npm, sshpass e o littler*. Que podem ser instalados através dos comandos (se executados no Ubuntu):

sudo apt-get install nodejs npm sshpass litter

Os servidores virtualizados testados foram o Ubuntu Server 24.04 e o Fedora 41 Server. Ambos com pacotes openssh instalados e configurados. A ferramenta precisa de usuário e senha em nível de super usuário para que possam ser executados os comandos. Além de possuir o pacote instalado chamado de *at*. Nas distribuições Linux supramencionadas ele já vem instalado. Ele é necessário para agendar o comando de falha e reparo da ferramenta. Um outro ponto é que a data e hora do servidor alvo precisa ser e ter o mesmo fuso horário e horário da máquina que executa a ferramenta para que os temporizadores sejam executados no tempo correto.

Para que o frontend funcione corretamente é necessário instalar também o pnpm, disponível em https://pnpm.io/installation. A instalação pode ser feita da seguinte forma: primeiramente executar o comando, em um terminal, *pnpm install* e aguarde finalizar a instalação dos pacotes. Após a instalação dos pré-requisitos citados acima, executar, num terminal, o comando *npm run dev* para executar a ferramenta. O código fonte da ferramenta pode ser baixado em https://github.com/aloisiosmelo/injetor-de-falhas.

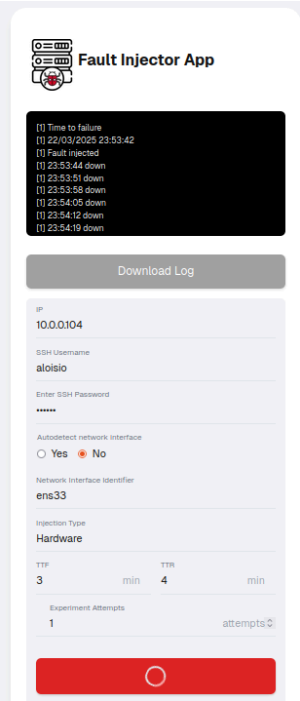
Para que o backend instale e execute basta executar o comando *npm install* e *node main*, respectivamente.

Para o funcionamento do frontend, é necessário instalar o gerenciador de pacotes pnpm (disponível em https://pnpm.io/installation). A instalação das dependências é realizada através do comando *pnpm install* em um terminal, aguardando a conclusão do processo. Em seguida, a ferramenta pode ser iniciada com o comando *npm run dev*. O código-fonte está disponível no repositório https://github.com/aloisiosmelo/injetor-de-falhas. Quanto ao backend, sua configuração requer a execução dos comandos *npm install* para instalação das dependências e *node main* para inicialização do serviço. Código-fonte está disponível no repositório https://github.com/aloisiosmelo/backend-injetor-de-falhas.

5.2 Frontend

O frontend é composto por um formulário com os campos da máquina virtual alvo. São eles usuário do ssh, senha do ssh, IP, tipo de injeção (no caso para este estudo será apenas hardware), tempo de falha em minutos, tempo de reparo em minutos, tentativas do experimento. E também, um campo em que o usuário pode marcar por detecção automática do identificador da interface de rede ou digitar. Isso pode ser visto na Figura 5.

Figura 5 – Formulário do injetor recebendo mensagens do backend.

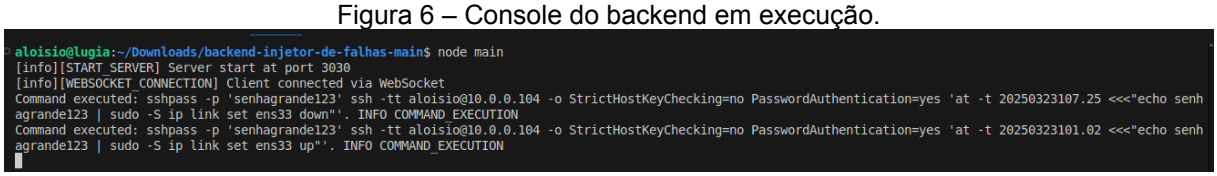


Fonte: Elaborado pelo próprio autor (2025).

5.3 Backend

Toda lógica de processamento da ferramenta está no backend. Como pode ser observado na Figura 6, abaixo. Ele mostra algumas informações sobre a execução de comandos e erros se houver. Ele usa os dados oriundos do formulário do frontend e se conecta à máquina virtual alvo para executar as falhas e reparos. Isso acontece via ssh. Essa conexão com a máquina alvo acontece em três momentos. Inicialmente, se o usuário optar por auto detecção da interface de rede, ela se conecta via ssh e executar o comando *sshpass -p " + sshPassword + " ssh " + sshUsername + "@" + ip + " ifconfig -a"*. Sendo que, este comando vem acrescido de parâmetros e posterior tratamento de string com o intuito de pegar apenas o necessário.

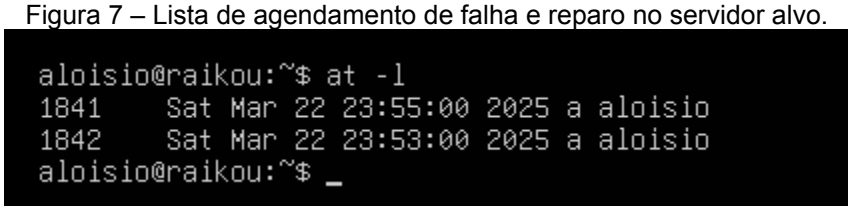
Figura 6 – Console do backend em execução.



Fonte: Elaborado pelo próprio autor (2025).

Posteriormente, executa os comandos de agendamento através do programa “at” que roda o comando no tempo agendado. Este agendamento na máquina é necessário para que o reparo ou falha possa ser executado em tempos independentes. Na Figura 7, é possível ver o retorno de agendamento diretamente no servidor alvo da ingestão de falha e reparo.

Figura 7 – Lista de agendamento de falha e reparo no servidor alvo.



Fonte: Elaborado pelo próprio autor (2025).

A injeção de falha e reparo acontece simulando um desligamento e religamento da interface rede. E que pode ser monitorada através de ping. Este ping é feito através do módulo de Nodejs com o mesmo nome. E essa injeção de falhas é feita utiliza o comando:

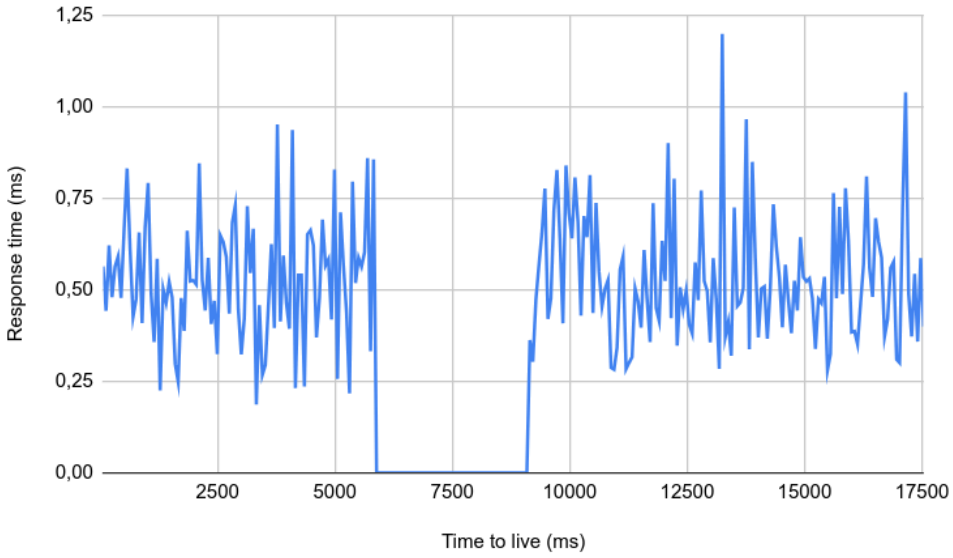
sshpass -p senhaSSHComRoot ssh -tt usuarioSSHComRoot@ip -o StrictHostKeyChecking=no PasswordAuthentication=yes "at -t tempoUsandoUmMetodoEspecificoDeConversaoDeTempoParaFerramenta \(<<<\) "echo senhaSSHComRoot | sudo -S ip link set identificadorDaInterfaceDeRede down"

Para reparo, usa-se o mesmo comando, trocando *down* por *up* (com contornos de restrição). O backend executa o comando, envia mensagens ao frontend via WebSocket, gera logs e retorna: falhas (se ocorrerem), data/hora do agendamento (ou falha), status do servidor (via ping, a cada 5s, 10 vezes) e o número da tentativa atual.

Temos na Figura 8, um gráfico que demonstra a disponibilidade, com dados obtidos do programa ping executado via terminal tendo como parâmetro o ip do servidor alvo

da injeção de falha e reparo. No eixo y temos o tempo de resposta do servidor alvo e no eixo x temos o tempo de vida ou disponibilidade, ambos em milisegundos (ms). Inicialmente, acontece uma disponibilidade até meados de 5000ms (eixo x), demonstrando que o sistema está disponível antes do agendamento da falha. Posteriormente, indisponibilidade até o tempo próximo de 10000 ms (eixo x), denotando que o agendamento da falha foi efetuado com sucesso. E mais adiante, o sistema torna-se disponível novamente constatando que a injeção de reparo foi executada com sucesso.

Figura 8 – Gráfico de disponibilidade.



Fonte: Elaborado pelo próprio autor (2025).

6 CONCLUSÕES E TRABALHOS FUTUROS

Esta ferramenta foi desenvolvida para proporcionar uma interface intuitiva destinada à injeção de falhas e execução de reparos, com potencial para diversas melhorias futuras. Entre as possíveis evoluções, destacam-se: a implementação de novos tipos de injetores (como funcionalidades para desabilitar ou habilitar serviços), o desenvolvimento de um aplicativo móvel para acesso direto ao backend e execução remota dos injetores, a incorporação de métodos avançados de benchmarking (incluindo testes de estresse e carga), além de otimizações de código para ampliar a compatibilidade tanto da ferramenta quanto dos sistemas alvo.

REFERÊNCIAS

[1] GILLEANES T A GUEDES. UML : uma abordagem prática. São Paulo: Novatec, 2008.

[2] WANG, V.; SALIM, F.; MOSKOVITS, P. The Definitive Guide to HTML5 WebSocket. [s.l.] Apress, 2013.

[3] PAULO. Performance, Reliability, and Availability Evaluation of Computational Systems, Volume 2. [s.l: s.n.].

[4] SOUZA, D. et al. A Tool for Automatic Dependability Test in Eucalyptus Cloud Computing Infrastructures. Computer and information science, v. 6, n. 3, 21 maio 2013.

[5] NASCIMENTO, E. et al. Modeling Availability in Softwarized MEC: Integrating a Fault Injection Tool for Effective Validation. Proceedings of the 13th Latin-American Symposium on Dependable and Secure Computing, p. 38–48, 26 nov. 2024.

[6] MATOS, R.; ANDRADE, E. C.; MACIEL, P. Evaluation of a disaster recovery solution through fault injection experiments. 2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC), p. 2675–2680, 1 out. 2014.

[7] HAISSAM ZIADE; AYOUBI, R.; VELAZCO, R. A survey on fault injection techniques. 1 jan. 2004.

[8] VIDAL-GOMEL, C.; ROGALSKI, J. La conceptualisation et la place des concepts pragmatiques dans l'activité professionnelle et le développement des compétences. Activites, v. 04, n. 1, 15 abr. 2007.

[9] TSAI, T. et al. NVBitFI: Dynamic Fault Injection for GPUs. p. 284–291, 1 jun. 2021.

[10] NASCIMENTO, E. et al. Modeling Availability in Softwarized MEC: Integrating a Fault Injection Tool for Effective Validation. Proceedings of the 13th Latin-American Symposium on Dependable and Secure Computing, p. 38–48, 26 nov. 2024.

[11] BARTOCCI, E. et al. FIM: fault injection and mutation for Simulink. Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 7 nov. 2022.

[12] NODE.JS. Node.js. Disponível em: https://nodejs.org/en. Acesso em: 16 abr. 2025.

[13] META OPEN SOURCE. React. Disponível em: https://react.dev/. Acesso em: 16 abr. 2025.

AGRADECIMENTOS

Dedico este trabalho, com todo o amor e gratidão, à minha mãe — minha base, meu exemplo, minha maior inspiração. Foi ela quem me criou praticamente sozinha, com coragem, sacrifício e uma força que jamais poderei retribuir à altura. Cada conquista minha carrega a marca do esforço dela.

Agradeço profundamente à minha esposa, que foi abrigo nos momentos difíceis e parceira incansável em cada passo desta caminhada. Seu apoio, paciência e incentivo foram fundamentais para que eu chegasse até aqui.

À minha sogra, meu sincero agradecimento por ter me dado forças e, com tanto carinho, cuidado da minha filha durante a elaboração deste trabalho. Sua ajuda foi essencial para que eu pudesse me dedicar com mais tranquilidade.

Estendo minha gratidão à minha família, por estar sempre presente, torcendo e apoiando nos bastidores, e aos amigos e professores que a vida universitária me deu — laços construídos ao longo do curso que levarei para sempre comigo.