UNIVERSIDADE ESTADUAL DE FEIRA DE SANTANA
Autorizada pelo Decreto Federal nº 77.496 de 27/04/76
Recredenciamento pelo Decreto nº17.228 de 25/11/2016
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
COORDENAÇÃO DE INICIAÇÃO CIENTÍFICA

XXIX SEMINÁRIO DE INICIAÇÃO CIENTÍFICA DA UEFS
SEMANA NACIONAL DE CIÊNCIA E TECNOLOGIA - 2025

Implementação de Modelos de Falhas em Ambiente Fogbed

Samara dos Santos Ferreira[1]; Antonio Augusto Teixeira Ribeiro Coutinho[2];
1.Bolsista PROBIC, Graduanda em Engenharia de Computação, Universidade Estadual de Feira de Santana,
e-mail: samaradosantosf@gmail.com
2. Orientador, Departamento de Tecnologia, Universidade Estadual de Feira de Santana,
e-mail: augusto@ecomp.uefs.br

PALAVRAS-CHAVE: Computação em Névoa; Emulação de Redes; Injeção de Falhas

INTRODUÇÃO

A computação em nuvem (*cloud computing*), apesar de sua ampla adoção para aplicações da Internet das Coisas (*Internet of Things*, IoT), apresenta desafios como alta latência e consumo excessivo de largura de banda (GAO et al., 2023). Como solução, o paradigma de computação em névoa (*fog computing*) estende a nuvem para mais perto dos dispositivos, processando dados localmente para reduzir o tempo de resposta e o tráfego da rede (IORGA et al., 2018).

A validação de sistemas em névoa é fundamental, porém o uso de plataformas de testes (*testbeds*) físicas é frequentemente inviável devido ao alto custo e complexidade. Ferramentas de emulação, como o Fogbed (COUTINHO et al., 2017), surgiram como alternativas para a prototipagem rápida. Contudo, uma lacuna importante nas ferramentas de emulação é a ausência de uma estrutura nativa para a injeção de falhas, uma capacidade crucial para testar a robustez de sistemas que operam em ambientes dinâmicos e sujeitos a falhas inesperadas (MADSEN et al., 2013).

Portanto, o objetivo deste trabalho é desenvolver uma API para a injeção programada de falhas, estendendo o emulador Fogbed. A proposta envolve a especificação e implementação de testes baseados em modelos de falhas que permitam a análise de sistemas de névoa com respeito à confiabilidade e à segurança.

METODOLOGIA

A criação da API de injeção de falhas foi executada em três fases principais e sequenciais, tendo como plataforma base o emulador Fogbed, que estende a biblioteca de emulação *Containernet* (PEUSTER et al., 2016) e Docker[1].

A primeira fase do projeto consistiu na pesquisa sobre modelos de falhas. Nela, foram estudados conceitualmente os tipos de falhas a serem implementadas no sistema. A abordagem partiu de dois tipos de falha básicos: a falha por parada (*Crash*), que simula o desligamento de um nó e a falha por desconexão (*Disconnect*), que interrompe sua comunicação com a rede. Em seguida, foi detalhado um modelo de disponibilidade,

[1] https://www.docker.com/

projetado para forçar um conjunto de nós a atingir um percentual de disponibilidade determinado através de um algoritmo baseado em intervalos de tempo (*time slots*).

A segunda fase foi a extensão da arquitetura do Fogbed. Esta etapa envolveu a modificação estrutural do emulador para integrar a nova funcionalidade e atualização do Fogbed para a nova versão do Ubuntu (24.04 LTS). As classes *Container* e *Instance* foram adaptadas para permitir a especificação dos testes de falhas e suas características através de parâmetros. Uma nova classe de controle, o *FailController*, foi criada para gerenciar a injeção das falhas durante os experimentos.

A terceira fase foi a implementação das falhas, em que os conceitos levantados na primeira etapa foram codificados. Foram desenvolvidas as classes *CrashFail* e *DisconnectFail*, além da *AvailabilityFail* para o teste de disponibilidade, permitindo a configuração de parâmetros como a taxa de disponibilidade e a duração de cada *slot*.

Por fim, para validar a API desenvolvida, foi conduzido um experimento usando a arquitetura SOFT-IoT (ANDRADE, 2020) em um ambiente emulado pelo Fogbed.

RESULTADOS E DISCUSSÃO

O cenário proposto para a realização dos experimentos consistiu em um *gateway* e três dispositivos IoT com configurações de disponibilidade distintas: 100%, 50% e 20%, como apresentado na Figura 1. Foram realizadas 30 execuções de 60 segundos cada para garantir a relevância estatística dos experimentos.

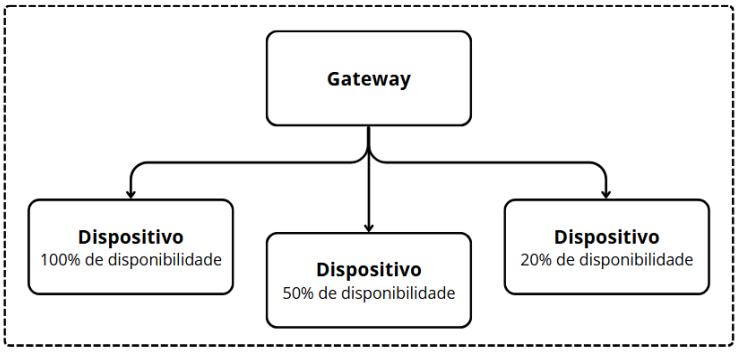

Figura 1: Estrutura do experimento no cenário emulado.

A análise dos resultados confirmou a precisão da ferramenta. O dispositivo configurado para 50% de disponibilidade atingiu uma média real de 50,50%, enquanto o de 20% alcançou 19,70%. A análise de métricas como Tempo Médio Entre Falhas (*Mean Time Between Failures*, MTBF), que indica o tempo em que o sistema opera corretamente (Equação 1), e Tempo Médio de Reparo (*Mean Time to Repair*, MTTR), que representa o tempo para corrigir uma falha (Equação 2), apresentaram valores medidos próximos dos tempos esperados para os ciclos de conexão e desconexão, conforme detalhado na Tabela 1.

$$MTBF = \frac{\text{somatorio de tempo em operacao}}{\text{numero de paradas}} \quad (1)$$

$$MTTR = \frac{\text{somatorio de tempo de reparo}}{\text{numero de paradas}} \quad (2)$$

Tabela 1. Análise das métricas de confiabilidade MTBF e MTTR, para os testes dos dispositivos IoT com 50% e 20% de disponibilidade, respectivamente.

Dispositivo	**Ciclo Esperado**	**Tempo de *Slot***	**MTBF**	**MTTR**
50% disponibilidade	1 *slot* ON e 1 *slot* OFF	2	2,01	1,97
20% disponibilidade	1 *slot* ON e 4 *slots* OFF	3	2,95	3,00

Na Figura 2 é possível visualizar os dispositivos 2 e 3 falhando ao decorrer do tempo. O dispositivo 1 não é exibido, pois operou sem falhas durante o experimento. Para o dispositivo 2, nota-se intervalos de tempo em bom funcionamento semelhantes com os intervalos em que esteve parado, o que resulta em MTBF e MTTR médios próximos ao tempo de *slot* de 2 segundos, conforme a Tabela 1.

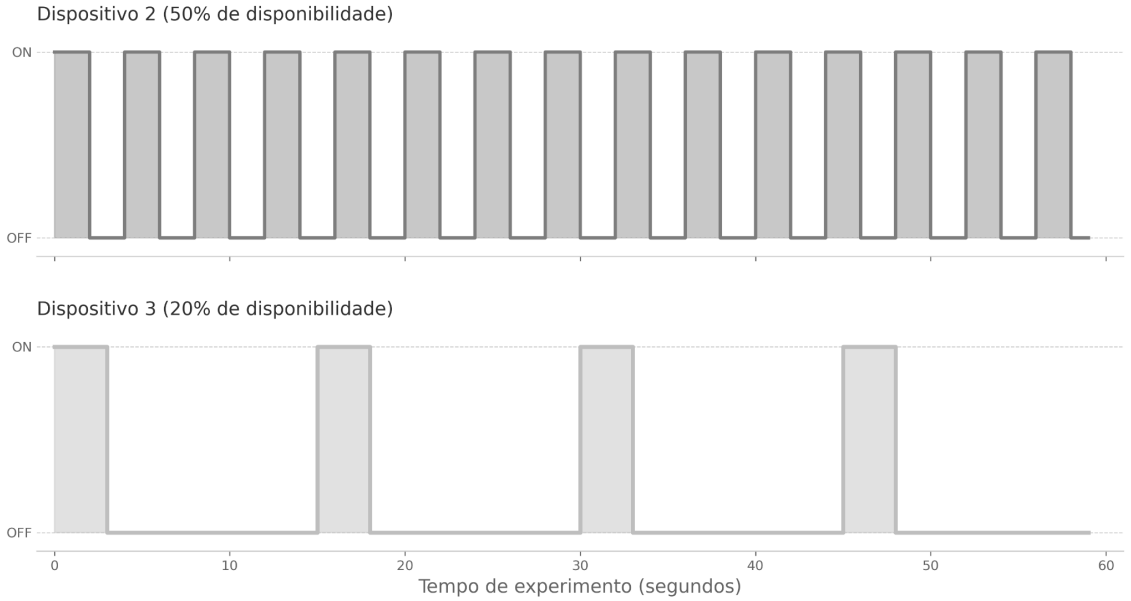

Figura 2: Validação experimental do modelo de disponibilidade, exibindo o status da conexão (ON/OFF) para os dispositivos ao longo de 60 segundos de experimento.

No dispositivo 3, o tempo de *slot* foi definido para 3 segundos, valor que o MTBF médio chegou próximo, assim como o MTTR dividido pelos 4 *slots* que o nó esteve desconectado, como pode ser visto na Tabela 1.

Nos experimentos, o impacto da injeção de falhas nos dispositivos SOFT-IoT também foi avaliado. Observou-se um atraso perceptível para que um dispositivo retomasse o envio de dados após a reconexão, indicando dificuldades de recuperação. A degradação do desempenho foi verificada comparando com os valores ideais de dados enviados para cada dispositivo, conforme a Tabela 2.

Assim, quanto menor a disponibilidade do dispositivo, menor é o percentual alcançado do número médio de dados enviados em relação ao ideal imaginado por ele.

Tabela 2. Percentual do número médio de dados enviados alcançado pelos dispositivos em relação ao valor ideal de dados enviados.

Dispositivo	Número Médio de Dados Enviados	Valor Ideal de Dados Enviados	Percentual Alcançado
100% disponibilidade	52,73	59	89,37%
50% disponibilidade	23,23	29	80,10%
20% disponibilidade	5,90	11	53,64%

CONCLUSÃO

Este trabalho expandiu com sucesso o sistema de emulação Fogbed ao integrar uma API de código aberto para a injeção programada de falhas, implementando e validando um modelo focado em disponibilidade. Os experimentos revelaram que a arquitetura SOFT-IoT apresenta dificuldades na recuperação automática e perda de eficiência quando exposta a falhas. Tais resultados ressaltam a importância da ferramenta desenvolvida, que se mostrou eficaz para identificar e quantificar deficiências de tolerância a falhas em sistemas de computação em névoa, contribuindo assim para o desenvolvimento de soluções mais robustas.

REFERÊNCIAS

GAO, Y. et al. Coverage Guided Fault Injection for Cloud Systems. In: IEEE/ACM INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE), 45., 2023, Melbourne. Proceedings. Melbourne: IEEE/ACM, 2023. p. 2211-2223. DOI: 10.1109/ICSE48619.2023.00186.

IORGA, M. et al. Fog Computing Conceptual Model. Gaithersburg, MD: National Institute of Standards and Technology, 2018. (NIST Special Publication, 500-325). Disponível em: https://doi.org/10.6028/NIST.SP.500-325. Acesso em: 17 jun. 2025.

COUTINHO, et al. An architecture for fog computing emulation. In: WORKSHOP EM CLOUDS E APLICAÇÕES (WCGA). SBC, 2017.

MADSEN, H. et al. Reliability in the utility computing era: towards reliable Fog computing. In: INTERNATIONAL CONFERENCE ON SYSTEMS, SIGNALS AND IMAGE PROCESSING (IWSSIP), 20., 2013, Bucharest. Anais [...]. Bucharest: IEEE, 2013. p. 43-46. DOI: 10.1109/IWSSIP.2013.6623445.

PEUSTER, M., KARL, H., AND ROSSEM, S. V. MeDICINE: Rapid Prototyping of Production-Ready Network Services in Multi-PoP Environments. In: IEEE CONFERENCE ON NETWORK FUNCTION VIRTUALIZATION AND SOFTWARE DEFINED NETWORKS (NFV-SDN), Palo Alto, CA, USA, pp. 148-153. doi: 10.1109/NFV-SDN.2016.7919490. (2016).

ANDRADE, Leandro et al. Fog of things: Fog computing in internet of things environments. In: SPECIAL TOPICS IN MULTIMEDIA, IOT AND WEB TECHNOLOGIES. Cham: Springer International Publishing, 2020. p. 23-50.