MininetFed 2.0: Uma Ferramenta Programável para Emulação e Análise de Aprendizado Federado em Redes

Daniel R. Trindade[1,2], Eduardo Zambon[1], Vinícius F. S. Mota[1], Ramon R. Fontes[3], Diego R. C. Dias[1], Giovanni Comarela[1], Rodolfo S. Villaça[1]

[1] Universidade Federal do Espírito Santo (Ufes)
{zambon, vinicius.mota, diego.dias, gc, rodolfo.villaca}@inf.ufes.br

[2]Instituto Federal do Espírito Santo (Ifes)
danielrt@ifes.edu.br

[3]Instituto Metrópole Digital (IMD) – Univ. Federal do Rio Grande do Norte (UFRN)
ramon.fontes@imd.ufrn.br

Abstract. *The massive generation of sensitive data by IoT devices requires privacy-focused approaches, such as Federated Learning (FL). However, evaluating these solutions is challenging due to computational and data heterogeneity, as well as inherent network limitations. This article presents MininetFed 2.0, an evolution of an FL emulation tool that introduces a programmable interface to simplify experiment creation. The tool enables modeling realistic environments by integrating FL, network emulation, and Docker containers, thereby facilitating the configuration of heterogeneous devices. The tool is validated through use cases focused on edge health monitoring (EHMS) and botnet detection in IoT devices (N-BaIoT).*

Resumo. *A geração massiva de dados sensíveis por dispositivos IoT exige abordagens focadas em privacidade, tais como Aprendizado Federado (FL). Contudo, avaliar essas soluções é desafiador devido à heterogeneidade computacional e de dados, além de limitações inerentes à rede. Este artigo apresenta o MininetFed 2.0, uma evolução de uma ferramenta de emulação de FL que introduz uma interface programável para simplificar a criação de experimentos. Ele permite a modelagem de ambientes realistas ao integrar FL, emulação de rede e contêineres Docker, viabilizando a configuração de dispositivos heterogêneos. A ferramenta é validada por meio de casos de uso voltados ao monitoramento de saúde na borda (EHMS) e detecção de botnets em dispositivos IoT (N-BaIoT).*

1. Introdução

O uso de dispositivos de Internet das Coisas (*Internet of Things* – IoT) tem crescido nos últimos anos, e por consequência, tais dispositivos estão gerando grandes volumes de dados que podem ser utilizados para o treinamento de modelos de aprendizado de máquina. Entretanto, em muitos cenários, esses dados são sensíveis e não podem ser compartilhados devido a restrições de privacidade, segurança ou regulamentação [Dritsas and Trigka 2025].

Como alternativa para reduzir o impacto desse problema, técnicas de aprendizado federado (*Federated Learning* – FL) podem ser aplicadas. No FL, o treinamento do

modelo é feito localmente nos dispositivos clientes e um servidor apenas é responsável por agregar esses modelos locais em um único modelo global [Yang et al. 2019]. Dessa forma, não há necessidade de compartilhamento dos dados em si, somente dos parâmetros dos modelos obtidos durante o treinamento. Entretanto, o FL apresenta diversos desafios, dentre os quais destacam-se a heterogeneidade dos dados distribuídos entre os clientes, as diferenças na capacidade computacional dos dispositivos e as limitações impostas pela rede de comunicação, tais como latência, largura de banda e perda de pacotes.

O MininetFed [Sarmento et al. 2024], apresentado no Salão de Ferramentas do SBRC 2024, é uma ferramenta para a proposta e estudo de algoritmos de FL integrado com o ambiente de emulação de rede Mininet [Mininet 2025]. A ferramenta permite emular dispositivos heterogêneos de rede por meio de contêineres Docker [Docker, Inc. 2024]. Os dispositivos e o agregador se comunicam de forma assíncrona, via protocolo MQTT (*Message Queuing Telemetry Transport*) [OASIS Standard 2014]. Entretanto, essa versão original da ferramenta, denominada a partir de agora como MininetFed 1.0, apresenta limitações práticas, tais como:

1. A configuração de experimentos exige a criação de arquivos de configuração complexos, adicionando uma etapa não usual para usuários do emulador Mininet;
2. O gerenciamento de imagens Docker deve ser feito manualmente pelo usuário;
3. Novos experimentos demandam uso excessivo de código "supérfluo" para configuração do ambiente experimental; e
4. A integração de extensões e novas funcionalidades exigem conhecimento detalhado da arquitetura interna do MininetFed.

Este trabalho apresenta o **MininetFed 2.0**, uma evolução do MininetFed 1.0, que preserva os objetivos principais da proposta original, ao mesmo tempo em que introduz uma interface totalmente programável, permitindo que o usuário crie novos experimentos de FL de forma rápida e com pouca codificação. Dentre as principais contribuições deste trabalho, destacam-se:

- Refatoração do *framework* MininetFed, resultando em uma nova versão, o MininetFed 2.0, mantendo uma estrutura mais limpa e organizada;
- Introdução de uma interface programável familiar à comunidade de redes para definição de experimentos federados, diminuindo a curva de aprendizado;
- Integração transparente entre FL, emulação de rede e execução distribuída em contêineres *docker* de modo mais integrado ao experimento;
- Criação de experimentos e testes de novas técnicas de FL com menor esforço de configuração e codificação; e
- Possibilidade de simular diferentes distribuições de dados entre os clientes federados, se aproximando de situações encontradas em ambientes reais.

O restante do texto está organizado da seguinte forma: a Seção 2 apresenta os trabalhos relacionados. A Seção 3 apresenta a arquitetura do MininetFed 2.0, bem como o funcionamento do fluxo de FL implementado. A seguir, a aplicação da ferramenta é ilustrada com dois casos de uso na Seção 4. A Seção 5 apresenta as conclusões e trabalhos futuros. Os artefatos necessários para reprodução dos experimentos estão indicados no Apêndice A. Por fim, o Apêndice B detalha a implementação de um dos casos de uso.

2. Trabalhos Relacionados

Existem variadas ferramentas para a realização de experimentos em aprendizado federado. Dentre elas, o Flower [Tan et al. 2020] se destaca por oferecer uma API simples e extensível, facilitando a prototipagem e a implementação de diferentes estratégias federadas, mas sem possibilitar a emulação da rede de comunicação entre clientes e servidor. Por sua vez, o TensorFlow Federated [Bonawitz et al. 2019] apresenta uma forte base matemática e conceitual, mas sua utilização é mais complexa e sua integração com outros *frameworks* de aprendizado de máquina é difícil. O FedML [He et al. 2020] disponibiliza uma grande quantidade de técnicas de FL já implementadas, mas seu uso também é mais complexo e oferece menor flexibilidade para customizações. Por fim, o OpenFL [Reinhold et al. 2021] possui mecanismos de segurança já incorporados, porém com pouca flexibilidade, sendo menos adequado para experimentação.

Apesar das suas contribuições individuais, percebe-se que nenhuma dessas ferramentas provê ambientes que permitam simular ou mesmo emular cenários mais próximos da realidade, nos quais estejam presentes simultaneamente restrições de rede e heterogeneidade computacional e de dados entre os dispositivos.

3. MininetFed 2.0

A versão original do MininetFed, neste artigo denominada MininetFed 1.0, é uma ferramenta desenvolvida para viabilizar a emulação de ambientes de FL por meio de dispositivos de borda heterogêneos e configuráveis. No entanto, como já mecionado, esta versão apresenta algumas limitações técnicas, que foram sanadas no MininetFed 2.0. A Figura 1 apresenta uma visão geral da arquitetura e do funcionamento do MininetFed 2.0. A ferramenta atua como um orquestrador de experimentos de FL, fornecendo um ambiente controlado para configuração, execução e gerenciamento de experimentos distribuídos.

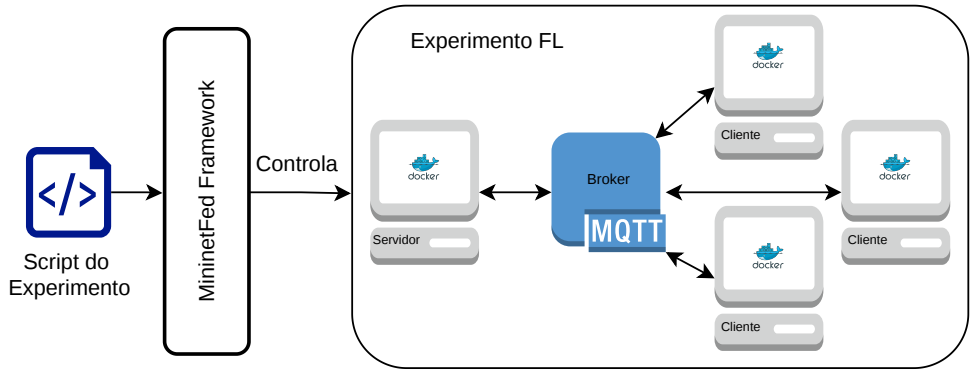


Figura 1. Visão geral do MininetFed2.0

O sistema disponibiliza uma interface de programação baseada no modelo Mininet [Mininet 2025]. Por meio do *script* de experimento, o usuário especifica a topologia do ambiente federado, incluindo a quantidade de clientes, o servidor federado e o *broker* responsável pela comunicação entre os nós. Também são definidos nesse *script* os parâmetros do treinamento federado, tais como o número de rodadas de treinamento, a estratégia de agregação de modelos e as políticas de seleção de clientes.

O MininetFed 2.0 também permite configurar aspectos relacionados à simulação de rede, tais como latência, largura de banda e perda de pacotes, utilizando os mecanismos oferecidos pelo Mininet. Paralelamente, características dos dispositivos podem

ser simuladas por meio de contêineres Docker, permitindo controlar recursos como CPU, memória e limitações de hardware. Cada nó participante do treinamento federado é executado em um contêiner Docker independente. O gerenciamento dessas imagens, incluindo a sua criação, configuração e atualização, é realizado automaticamente pelo *framework*, de forma transparente ao usuário. Essa abordagem facilita a replicação de experimentos e possibilita a simulação de ambientes heterogêneos típicos de cenários de FL.

As subseções a seguir discutem com maiores detalhes a arquitetura do MininetFed 2.0, as suas funcionalidades e como se dá o fluxo de treinamento federado.

3.1. Arquitetura do MininetFed 2.0

A Figura 2 apresenta os principais componentes da arquitetura do MininetFed 2.0. O sistema é organizado em dois módulos principais: **MininetFed.sim** e **MininetFed.core**.

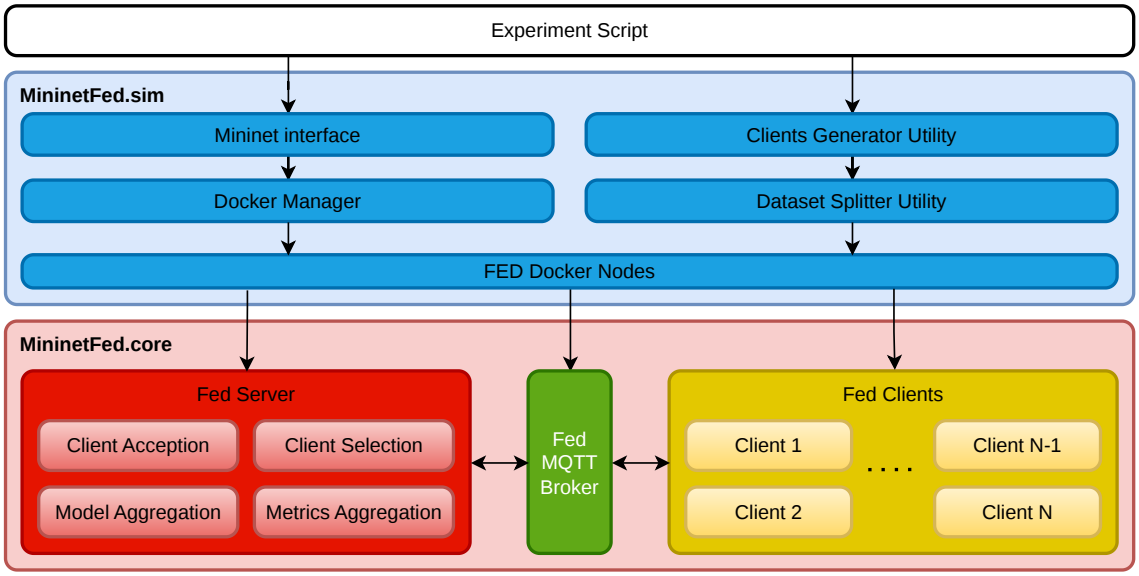


Figura 2. Arquitetura detalhada do MininetFed2.0

O módulo **MininetFed.sim** é responsável pelo gerenciamento da simulação do ambiente de FL. O usuário interage exclusivamente com esse módulo por meio de um *script*, no qual são definidas as características gerais do experimento, incluindo parâmetros do treinamento federado, topologia da rede e características dos dispositivos simulados. Internamente, o módulo **MininetFed.sim** é composto pelos seguintes componentes:

- **Mininet Interface**: fornece uma interface de programação no estilo do Mininet para definição da topologia e configuração do experimento. No MininetFed 1.0, tais configurações eram realizadas por meio de arquivos YAML, o que tornava o processo menos flexível. A nova interface, totalmente programável em Python como no Mininet, permite uma definição mais natural do ambiente experimental, aproveitando o modelo de programação amplamente utilizado do Mininet.
- **Docker Manager**: responsável pelo gerenciamento das imagens Docker utilizadas pelos nós do sistema. Cada nó do MininetFed 2.0 é executado em um contêiner independente. O usuário precisa apenas especificar as dependências de software necessárias, e o gerenciador automatiza a criação, atualização e configuração das

imagens. Estas são criadas apenas uma vez e atualizadas automaticamente sempre que alterações nas dependências são detectadas. No MininetFed 1.0, esse gerenciamento precisava ser realizado manualmente, o que demandava maior tempo e esforço. O *framework* também disponibiliza imagens padrão para os nós do MininetFed 2.0, embora o usuário também possa utilizar suas próprias imagens.
- **Fed Docker Nodes**: representam os nós do sistema federado executados em contêineres Docker. Esses nós encapsulam as implementações dos componentes de FL definidos no módulo **MininetFed.core**. Atualmente, são suportados três tipos de nós: servidor federado (**Fed Server**), cliente federado (**Fed Client**) e *broker* de comunicação (**Fed MQTT Broker**).

O componente **Mininet Interface** utiliza o Mininet para fornecer a emulação da rede e integra-se ao **Docker Manager** para realizar a simulação de dispositivos e o gerenciamento dos contêineres. Além desses componentes principais, o módulo **MininetFed.sim** inclui dois utilitários que auxiliam na preparação dos experimentos:

- **Dataset Splitter Utility**: permite dividir um conjunto de dados em múltiplos subconjuntos destinados aos diferentes clientes federados. O usuário pode definir o tipo de distribuição dos dados entre os clientes, incluindo cenários onde os subconjuntos seguem a mesma distribuição do conjunto de dados original (*iid*) ou onde os subconjuntos tem distribuição desbalanceada (*non-iid*).
- **Clients Generator Utility**: utiliza os subconjuntos gerados pelo **Dataset Splitter Utility** e o código de cliente definido pelo usuário para criar automaticamente as estruturas de diretórios dos clientes federados. Cada cliente gerado contém uma cópia do código de treinamento local e o seu respectivo subconjunto de dados.

Estes novos utilitários não estavam disponíveis no MininetFed 1.0, são opcionais e visam simplificar a configuração dos experimentos, sem impedir que o usuário execute esses procedimentos externamente ao *framework*.

O módulo **MininetFed.core**, por sua vez, contém toda a lógica do FL. Nele estão implementados os principais componentes do sistema: servidor federado, clientes federados e *broker* de comunicação:

- **Fed Server**: é o servidor responsável por controlar o fluxo do treinamento federado, incluindo o registro e aceitação de clientes, seleção de clientes de cada rodada, agregação dos modelos locais e de métricas. O *framework* disponibiliza uma implementação padrão do servidor que usa o algoritmo de agregação de pesos FedAvg [McMahan et al. 2017] e que aceita e seleciona todos os clientes (mais detalhes na Seção 3.2). O usuário pode, entretanto, implementar suas próprias políticas de treinamento e agregação estendendo a classe base do servidor.
- **Fed Client**: é o cliente que executa o treinamento local do modelo. Além do *script* de treinamento, somente o código do cliente federado precisa ser fornecido. Esse código consiste em implementar um conjunto mínimo de métodos responsáveis pelo treinamento e avaliação local.
- **Fed MQTT Broker**: é o nó *broker* responsável pela comunicação entre o servidor e os clientes. Isso é feito por meio do protocolo MQTT, sendo disponibilizada uma implementação padrão baseada no *broker* Mosquitto [Eclipse Foundation 2024]. Para garantir a interoperabilidade, o usuário pode trocar a implementação padrão por qualquer alternativa baseada no protocolo MQTT.

3.2. Fluxo de Aprendizado Federado

A Figura 3 apresenta o diagrama de fluxo do processo de treinamento federado no MininetFed 2.0. No lado esquerdo da figura é mostrada a linha do tempo do servidor federado, enquanto no lado direito é apresentada a linha do tempo dos clientes federados. Entre eles são exibidas as mensagens trocadas por meio do protocolo MQTT e as estruturas de dados utilizadas na comunicação.

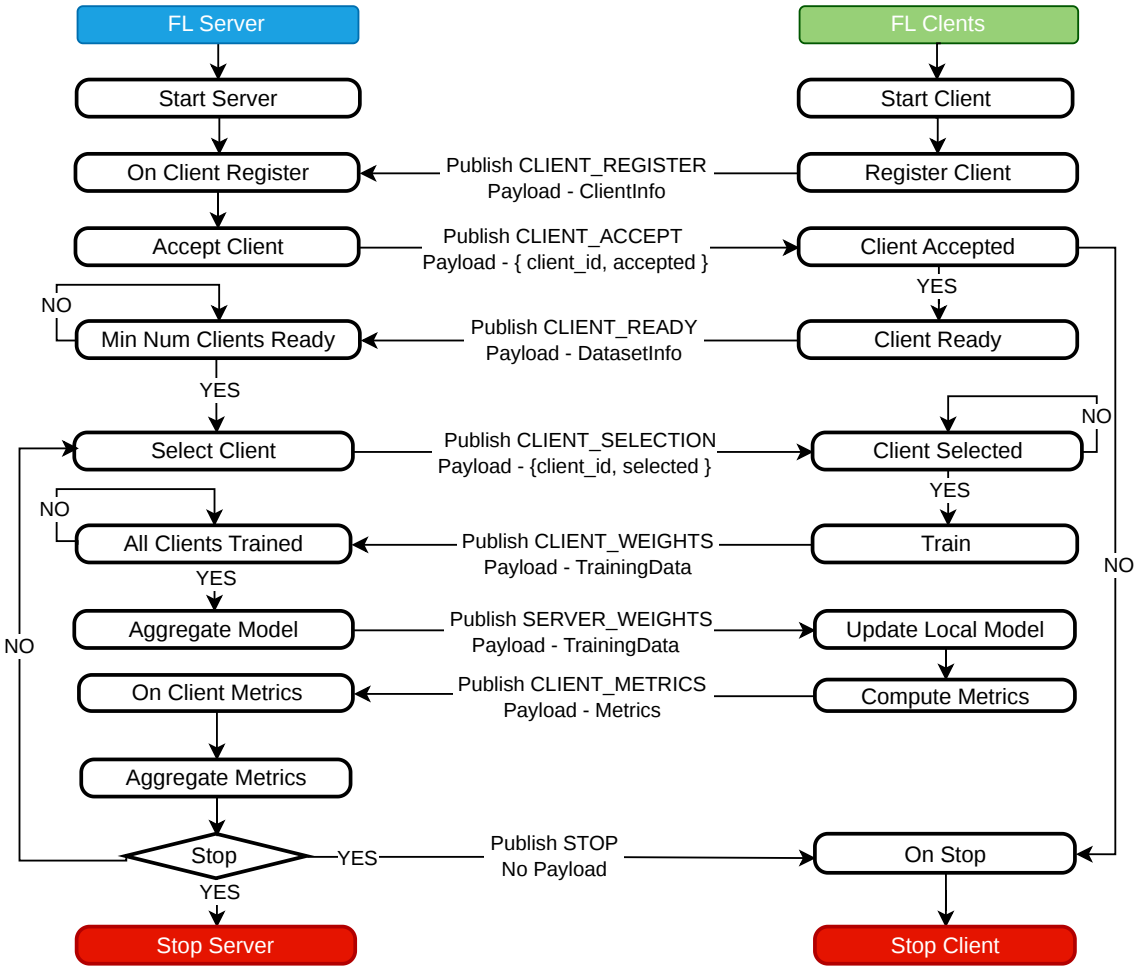


Figura 3. Fluxo do treinamento federado no MininetFed 2.0

O treinamento inicia-se com a instanciação do servidor e dos clientes federados. Os clientes enviam ao servidor uma mensagem de registro (**CLIENT_REGISTER**), acompanhada de informações encapsuladas na estrutura *ClientInfo*. Ao receber as mensagens de registro, o servidor aplica uma política de aceitação de clientes baseada nas informações que estes enviaram, determinando quais clientes poderão participar do treinamento. Essa etapa é nova e não existe no MininetFed 1.0. Por padrão, todos os clientes são aceitos, mas o usuário pode definir suas próprias regras. Por exemplo, o cliente pode enviar informações como capacidade de CPU ou memória, permitindo que o servidor decida sobre a participação de um dado cliente. Após essa etapa, o servidor publica a mensagem **CLIENT_ACCEPT**, indicando os clientes aceitos.

Quando aceito, o cliente carrega seus dados locais e sinaliza que está pronto para

o treinamento por meio da mensagem **CLIENT_READY**. Nesse momento, ele também pode enviar informações sobre o *dataset* usando a estrutura *DatasetInfo*. Como exemplo, o número de amostras disponíveis no *dataset*, utilizado por algoritmos de agregação como o FedAvg, pode ser enviado ao servidor.

O treinamento inicia-se quando um número mínimo de clientes sinalizam estarem prontos, conforme definido no experimento. No início de cada rodada, o servidor aplica uma política de seleção de clientes e comunica o resultado por meio da mensagem **CLIENT_SELECTION**. A política padrão no MininetFed 2.0 seleciona todos os clientes disponíveis, mas o usuário pode definir outras estratégias.

Os clientes selecionados realizam o treinamento local do modelo e enviam seus pesos ao servidor por meio da mensagem **CLIENT_WEIGHTS**, utilizando a estrutura *TrainingData*. O servidor aguarda os modelos locais e aplica um algoritmo de agregação para gerar o modelo global. No MininetFed 2.0 o método padrão é o FedAvg, embora o usuário possa implementar outros algoritmos.

Após a agregação, o modelo global é enviado aos clientes por meio da mensagem **SERVER_WEIGHTS**. Cada cliente avalia o modelo localmente e envia suas métricas ao servidor por meio da mensagem **CLIENT_METRICS**, utilizando a estrutura *Metrics*. O servidor agrega as métricas locais para obter métricas globais e, com base em um critério de parada definido pelo usuário, decide se o treinamento deve continuar ou ser finalizado. O usuário pode definir qual métrica será utilizada na avaliação e configurar mecanismos como *early stopping*, que interrompem o treinamento caso não haja melhoria após um número determinado de rodadas.

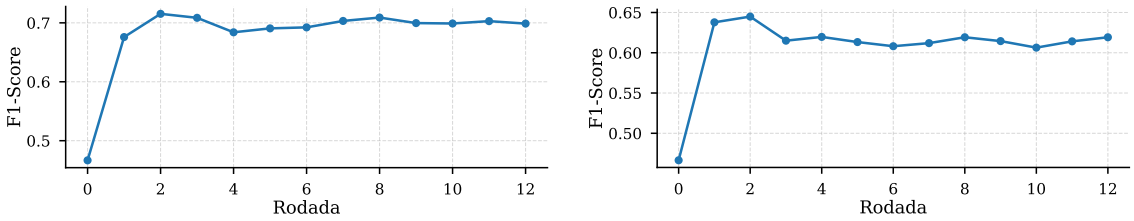
As estruturas *ClientInfo*, *DatasetInfo*, *TrainingData* e *Metrics* padronizam a comunicação entre os componentes do sistema, mantendo flexibilidade para que o usuário inclua diferentes tipos de informação. Isso facilita a experimentação com novos métodos de seleção de clientes, agregação de modelos ou coleta de métricas no MininetFed 2.0.

4. Casos de Uso

Como exemplos de aplicação, foram realizados experimentos de FL utilizando dois conjuntos de dados: **EHMS** (*Edge Health Monitoring System*) [Washington University in St. Louis 2020] e **N-BaIoT** (*Detection of IoT Botnet Attacks*) [Meidan et al. 2018].

O conjunto de dados **EHMS** é voltado ao monitoramento de saúde em ambientes IoT, contendo medições biomédicas e métricas de rede para tarefas de detecção de anomalias e classificação de eventos. O experimento foi configurado com um servidor federado e quatro clientes, considerando dois cenários de distribuição de dados: *iid*, no qual os dados seguem a distribuição original, e *non-iid*, gerado via distribuição de Dirichlet com $\alpha = 0.8$. Como mostrado na Figura 4, ambos os cenários convergem em 12 rodadas; contudo, o cenário *non-iid* apresenta maior oscilação e desempenho inferior (F1 ≈ 0.65) em comparação ao cenário *iid* (F1 ≈ 0.7). O Apêndice B apresenta parte do código para este caso de uso.

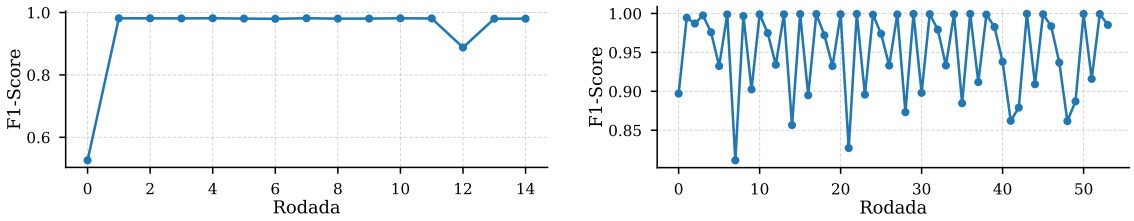
O conjunto de dados **N-BaIoT** contém dados de tráfego real, coletados de 9 dispositivos IoT comerciais infectados por botnets. Os dados já se encontram naturalmente particionados por dispositivo, não sendo necessária a divisão artificial dos dados. Devido



(a) iid **(b) non-iid**

Figura 4. Curvas de aprendizado para o dataset EHMS

a restrições computacionais, foram selecionados 7 dos 9 dispositivos, resultando assim em 7 clientes. Dois modos de treinamento foram avaliados: (i) política de seleção padrão de clientes (Figura 5a), na qual todos os clientes participam a cada rodada, e (ii) um servidor personalizado com seleção em fila circular, onde apenas 3 clientes treinam por rodada de forma intercalada (Figura 5b). Essa nova política de seleção foi criada para demonstrar a capacidade do MininetFed 2.0 na experimentação de novas técnicas de FL. No modo de seleção padrão, a convergência ocorre em 14 rodadas; já na seleção em fila, o treinamento demanda 53 rodadas e apresenta maior instabilidade.



(a) seleção padrão **(b) seleção em fila**

Figura 5. Curvas de aprendizado para o dataset N-BaloT

5. Conclusão

Este trabalho apresentou o MininetFed 2.0, uma evolução do *framework* MininetFed 1.0. A nova versão preserva os objetivos centrais da anterior, ao mesmo tempo em que introduz uma interface programável mais simples e flexível, reduzindo significativamente o esforço necessário para configuração e execução de experimentos.

A integração entre FL, emulação de rede por meio do Mininet e execução distribuída com contêineres Docker permite a criação de cenários experimentais que consideram simultaneamente restrições de comunicação e heterogeneidade computacional dos dispositivos. A ferramenta permite também emular clientes com diferentes distribuições de dados. Essas características tornam o MininetFed 2.0 uma ferramenta adequada para investigação de problemas práticos do FL, que não são plenamente contemplados por *frameworks* tradicionais.

Agradecimentos

Este trabalho possui financiamento parcial das seguintes agências: Fapes (2026-B74MN, 2023-RWXSZ, 2025-1H3FP); MCTI/Fapesp/CGI.br (Porvir-5G); CNPq; e Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Referências

Bonawitz, K. et al. (2019). Towards federated learning at scale: System design. *Proceedings of Machine Learning and Systems (MLSys)*.

Docker, Inc. (2024). Docker documentation: Docker container. `https://docs.docker.com`.

Dritsas, E. and Trigka, M. (2025). Federated learning for IoT: A survey of techniques, challenges, and applications. *Journal of Sensor and Actuator Networks*, 14(1):9.

Eclipse Foundation (2024). Eclipse Mosquitto: An open source MQTT broker. `https://mosquitto.org/`.

He, C., Annavaram, M., and Avestimehr, S. (2020). FedML: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*.

McMahan, B. et al. (2017). Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Meidan, Y. et al. (2018). detection_of_IoT_botnet_attacks_N_BaIoT. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5RC8J.

Mininet (2025). Mininet: An instant virtual network on your laptop.

OASIS Standard (2014). MQTT version 3.1.1. `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html`.

Reinhold, J., Reinsch, R., Li, Y., et al. (2021). OpenFL: The open federated learning library. *arXiv preprint arXiv:2105.06413*.

Sarmento, E. M. et al. (2024). MininetFed: A tool for assessing client selection, aggregation, and security in federated learning. In *2024 IEEE 10th World Forum on Internet of Things (WF-IoT)*, pages 1–6. IEEE.

Tan, D. J., Qi, Y., et al. (2020). Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*.

Washington University in St. Louis (2020). Edge health monitoring system (EHMS) dataset. `https://www.cse.wustl.edu/~jain/ehms/index.html`.

Yang, Q. et al. (2019). Federated Machine Learning: Concept and Applications.

A. Artefatos para Reprodução

Os artefatos para reprodução (códigos fonte do Mininet 2.0 e dos casos de uso da Seção 4) se encontram em
`https://github.com/lprm-ufes/MininetFed-2.0-SBRC-2026`.
No repositório há instruções detalhadas de instalação do MininetFed 2.0 e execução dos casos de uso.

B. Projeto MininetFed 2.0 exemplo para o dataset EHMS

A Listagem 1 mostra um exemplo básico de projeto usando o MininetFed 2.0 para FL usando o *dataset* EHMS. O código completo se encontra no *link* do GitHub indicado no

Apêndice A, juntamente com instruções de como rodar o experimento. Neste exemplo, o servidor usado é o padrão já fornecido pelo MininetFed 2.0. Portanto, somente é necessário definir o código do cliente federado e o *script* do experimento. O projeto é formado por três arquivos:

- **ehms_fed.py**: contém o código do *script* de experimento, com as configurações do FL e definição da topologia de rede.
- **ehms_trainer.py**: contém o código que roda o treinamento localmente em cada cliente.
- **client_requirements.txt**: contém a lista de pacotes Python dos quais o código cliente depende, e que devem ser instalados no container do cliente.

A Listagem 2 mostra o código do *script* de experimento (**ehms_fed.py**). Esse código fornece uma visão geral das etapas necessárias para execução do treinamento federado usando o MininetFed 2.0. No corpo do código há comentários explicando cada uma das etapas. Não serão mostrados aqui os códigos referentes ao cliente (**ehms_trainer.py** e **client_requirements.txt**), mas uma versão devidamente comentada pode ser acessada através do repositório do projeto.

Listagem 1. Estrutura do projeto para o experimento EHMS federado

```
MininetFed-EHMS-Example/
|-- ehms_fed.py
|-- dataset/
|   |-- wustl-ehms-2020_with_attacks_categories.csv
`-- client_code/
    |-- ehms_trainer.py
    `-- client_requirements.txt
```

Listagem 2. Arquivo ehms_fed.py

```
import ...

n_clients = 4
client_code_path = "client_code/"

# Configuracoes do treinamento federado
server_args = {
    ServerOptions.MIN_CLIENTS: n_clients,
    ServerOptions.NUM_ROUNDS: 100,
    ServerOptions.STOP_VALUE: 0.99,
    ServerOptions.PATIENT: 10,
    ServerOptions.TARGET_METRIC: MetricType.F1,
    ServerOptions.CLIENT_ACCEPTOR: ClientAcceptorType.ALL_CLIENTS,
    ServerOptions.CLIENT_SELECTOR: ClientSelectorType.ALL_CLIENTS,
    ServerOptions.MODEL_AGGREGATOR: AggregatorType.FED_AVG
}

def run_experiment():
    # Carrega o dataset, o divide entre 4 clientes
    # seguindo a mesma distribuicao do original
    client_paths = create_federated_client_datasets(
        dataset_source="dataset/wustl-ehms-2020_with_attacks_categories
            .csv",
        target_col="Label",
```

```
        n_clients=n_clients,
        split_mode="iid",
        code_src_dir=client_code_path,
    )

    # Carrega o dataset, o divide entre 4 clientes seguindo uma
    # distribuicao desbalanceada em relacao a original
    #client_paths = create_federated_client_datasets(
    #    dataset_source="dataset/wustl-ehms-2020_with_attacks_categories.csv",
    #    target_col="Label",
    #    n_clients=n_clients,
    #    split_mode="non_iid",
    #    alpha=0.8,
    #    code_src_dir=client_code_path,
    #)

    # Cria uma imagem com o framework MininetFed 2.0 ja instalado,
    # alem dos pacotes dos quais o codigo cliente e dependente
    client_dimage = build_fed_node_docker_image("torch_client", client_code_path + "client_requirements.txt")["tag"]

    # Cria a simulacao usando o modelo baseado no Mininet
    net = MininetFed()
    try:
        # Adiciona um switch
        s1 = net.addSwitch(name="s1", failMode='standalone')

        # Adiciona o broker de comunicacao entre os clientes
        broker = net.addHost(name="broker", cls=FedBrokerNode)
        net.addLink(s1, broker)

        # Adiciona o servidor federado
        server = net.addHost(
            name="server",
            cls=FedServerNode,
            server_args=server_args
        )
        net.addLink(s1, server)

        # Adiciona os clientes federados. Cada cliente e mapeado para
        # uma das pastas retornada por create_federated_client_datasets
        # e usa a imagem docker criada por build_fed_node_docker_image
        for i in range(n_clients):
            net.addHost(
                name=f"client{i}",
                cls=FedClientNode,
                script="ehms_trainer.py",
                dimage=client_dimage,
                client_folder=client_paths[i]
            )
            net.addLink(s1, net.get(f"client{i}"))

        # dispara a simulacao da rede Mininet
        print(f'*** Starting network...\n')
        net.build()
```

```
        net.addNAT(name='nat0', linkTo='s1', ip='192.168.210.254').
            configDefault()
        s1.start([])

        # roda o experimento de FL
        net.runFed()
    finally:
        # termina o experimento quando ocorre o criterio
        net.stop()

if __name__ == "__main__":
    run_experiment()
```

Para a execução do experimento, o usuário deve executar o seguinte comando:

```
$ sudo python ehms_fed.py
```

O MininetFed 2.0 deve ser executado com privilégios de administrador do sistema (*sudo*). Isso é uma exigência do Mininet, para que seja possível emular os apectos relacionados à rede na simulação.

A Figura 6 mostra os artefatos produzidos pelo comando anterior. Na execução do *script* de treinamento, o *dataset* é dividido entre os clientes seguindo a distribuição definida pelo usuário e são criadas as pastas referentes a cada cliente (pastas *clients/client<i>*). São criadas também pastas para o nós *broker* (pasta *broker_output/*) e servidor (pasta *server_output/*). Em seguida, com o início do treinamento federado em si, são abertas janelas de terminais onde é possível acompanhar a execução. Cada janela aberta representa um dos nós da federação (1 terminal para o servidor, 1 para o broker e 4 para os clientes).

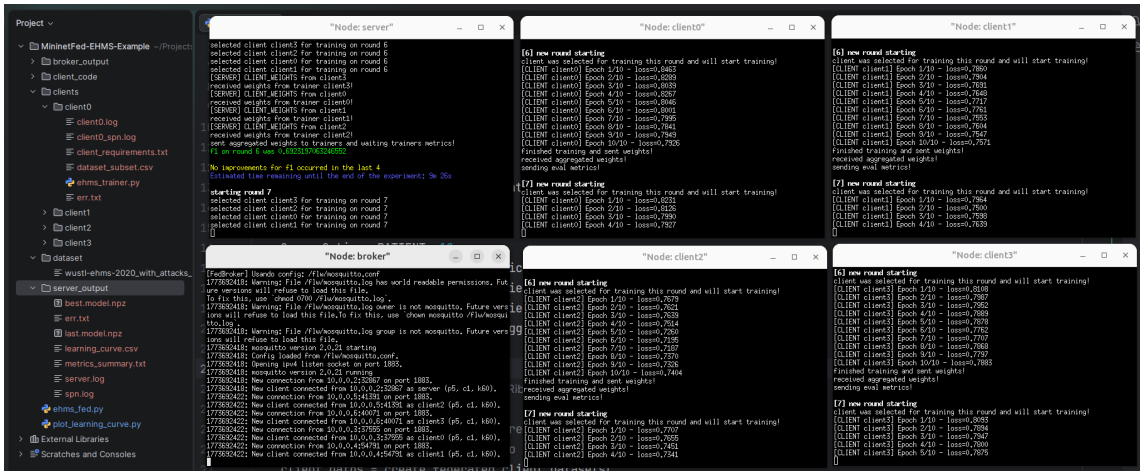


Figura 6. Execução do treinamento federado para o dataset EHMS

Ao final do treinamento, são salvos dentro do pasta de saída do servidor os arquivos contendo os resultados do experimento. Dentre eles, destacam-se:

- **metrics_summary.txt**: contém as métricas de desempenho do treinamento.
- **learning_curve.csv**: contém as métricas de desempenho do treinamento para cada rodada.
- **best.model.npz**: pesos do melhor modelo obtido durante o treinamento.