Pythia: Uma Plataforma de Emulação de Mobilidade em Névoas Computacionais

Vinícius Alcântara[1], **Pedro Cruz**[1], **Nadjib Achir**[2], **Aline C. Viana**[2], **Luís Henrique M. K. Costa**[1]

[1]GTA/COPPE/Poli, Universidade Federal do Rio de Janeiro, Brasil
[2]INRIA Saclay - Palaiseau, França

{alcantara,cruz,luish}@gta.ufrj.br
{aline.viana, nadjib.achir}@inria.fr

Resumo. *A utilização de aplicações de processamento intensivo em em dispositivos móveis (User Equipment – UEs) carece do auxílio de recursos computacionais externos. Entretanto, aplicações sensíveis à latência exigem recursos próximos aos UEs. No padrão Multi-Access Edge Computing, recursos são espalhados geograficamente na rede das operadoras de telefonia móvel. Contudo, as decisões de alocação de recursos tornam-se complexas com a mobilidade dos UEs, podendo afastá-los dos recursos que os servem. Este trabalho apresenta a plataforma Pythia, uma ferramenta de emulação da mobilidade dos UEs do ponto de vista das aplicações. Seu principal objetivo é suportar o desenvolvimento de estratégias de alocação e de aplicações cientes da mobilidade.*

Abstract. *The use of processing-intensive applications in mobile devices (User Equipment – UE) requires assistance from external computational resources. Nevertheless, latency-sensitive applications require resources close to the UEs. In the Multi-Access Edge Computing standard, resources are spread geographically across the mobile operator's network. However, resource allocation decisions become complex with UE mobility, which might move them away from resources they are using. This work presents the platform Pythia, a tool capable of emulating the UE mobility from an application point of view. Its main objective is supporting the development of resource allocation strategies and mobility-aware applications.*

1. Introdução

O atual cenário de desenvolvimento de aplicações para dispositivos móveis celulares requer recursos para além da capacidade local dos aparelhos. Assim, para aplicações mais complexas, os equipamentos de usuário (UEs) frequentemente devem acessar recursos externos através da Internet em um procedimento denominado *task offloading*. Quando esses recursos estão concentrados em servidores distantes do usuário, são classificados como recursos em nuvem. Contudo, a distância entre os UEs e os servidores pode resultar em uma maior latência, prejudicando a Qualidade de Experiência (*Quality of Experience* - QoE) percebida pelos usuários. A disponibilização de recursos na borda da rede, conhecida como computação na névoa (do inglês *fog computing*), garante proximidade geográfica e topológica entre os UEs e os recursos computacionais. Assim, diminui-se a latência na comunicação [Silva et al. 2020].

O padrão *Multi-Access Edge Computing* (MEC), estabelecido pelo *European Telecommunications Standards Institute* (ETSI), descreve uma arquitetura de computação em névoa na qual os recursos são distribuídos pela rede e disponibilizados pelas operadoras de redes móveis (*Mobile Network Operators* - MNO). Pelo padrão MEC, para que um UE acesse recursos computacionais, uma aplicação executada pelo UE (UEApp) realiza uma solicitação ao sistema MEC. O sistema MEC então instancia uma aplicação MEC (MECApp) em um servidor MEC (MEC Host). A aplicação MECApp, por sua vez, é capaz de atender às requisições da aplicação UE, executando tarefas e aliviando a carga do UE. A escolha do servidor MEC é realizada pelo sistema MEC e é baseada em uma estratégia de alocação de recursos do sistema. Entretanto, a otimização de uma estratégia de alocação de recursos torna-se mais complexa com a presença de mobilidade dos UEs.

A mobilidade de usuários introduz em uma topologia dinâmica na qual o sistema MEC deve ser capaz de detectar padrões de deslocamento e decidir a estratégia que garanta a maior QoE ao usuário [Cruz et al. 2022]. Para o desenvolvimento de estratégias focadas na QoE, é necessário haver formas de validar o desempenho da estratégia nesse aspecto. Este trabalho apresenta a Pythia, uma plataforma de emulação de redes com um serviço MEC na presença de mobilidade. O objetivo principal da plataforma é emular a mobilidade dos UEs com vistas a investigar estratégias de alocação de recursos. Na Pythia, UEApps podem se comunicar com MECApps de modo que a latência da comunicação segue um traço previamente configurado. Assim, é possível reproduzir condições de rede similares para experimentos com diferentes estratégias de alocação.

A emulação de redes é de interesse tanto para o desenvolvimento científico quanto para o desenvolvimento de aplicações comerciais. O simulador NS-3[1] possui ênfase em simulação, mas entrega algumas funcionalidades de emulação. O trabalho de Imputato *et al.* utiliza a biblioteca netmap[2] para emular condições de rede sem realizar chamadas ao núcleo do sistema operacional, melhorando o desempenho da emulação do ns-3 [Imputato et al. 2017]. Porém, o ns-3 ainda possui a desvantagem de simular componentes que por vezes são desnecessários. Outro trabalho relevante é o Kollaps, que emula redes dinâmicas e aumenta seu desempenho focando na emulação ponto-a-ponto, ignorando estados internos da rede e de seus elementos [Gouveia et al. 2020]. O Kollaps utiliza contêineres para isolar aplicações e utiliza a biblioteca NetEm [Hemminger et al. 2005] para alterar as políticas de tratamento de pacotes entre as aplicações, emulando as condições de rede entre elas. O Kollaps tem foco em topologias estáticas, dando suporte limitado a topologias dinâmicas. A Pythia também utiliza contêineres em conjunto com o NetEm, mas foca em topologias dinâmicas, com um tempo de atualização curto entre eventos. Também é utilizada a mesma filosofia de ignorar o estado interno da rede para dar ainda maior dinamicidade à rede, podendo emular as condições de rede experimentadas por aplicações executadas por UEs em movimento. Ao contrário do ns-3 e do Kollaps, a Pythia suporta o padrão MEC e sua API AppList [ETSI 2020].

O restante deste texto está organizado da seguinte maneira: a Seção 2 introduz as funcionalidades da Pythia, enquanto a Seção 3 apresenta sua arquitetura. A Seção 4 realiza uma análise de desempenho da ferramenta. A Seção 5 descreve a demonstração

[1] `https://www.nsnam.org/`

[2] `http://info.iet.unipi.it/~luigi/netmap/`

da Pythia. A Seção 6 conclui o trabalho e apresenta os trabalhos futuros.

2. Funcionalidades da Pythia

A plataforma Pythia executa MECApps e UEApps, e emula o comportamento da rede de acesso MEC, alterando a característica das conexões em função da movimentação dos UEs. O objetivo principal da Pythia é oferecer uma maneira de estudar as estratégias de alocação de recursos de uma infraestrutura MEC, levando em conta as aplicações e a topologia dinâmica da rede. Adicionalmente, também é possível avaliar a QoE de aplicações sob o efeito da topologia dinâmica. A emulação da Pythia permite que as mesmas condições de rede sejam reproduzidas para diferentes estratégias de alocação e diferentes aplicações. Assim, é garantida a reprodutibilidade de experimentos, possibilitando a comparação entre diferentes estratégias em diferentes cenários.

Para utilização da Pythia, é necessário o download no github[3]. O arquivo `README.md` disponível no diretório principal possui instruções de instalação. Inicialmente, é necessária a instalação do Docker segundo as instruções da equipe de desenvolvimento do Docker[4]. O segundo passo é a construção (*build*) das imagens para a execução do emulador. A Pythia é desenvolvida para o sistema operacional Debian GNU/Linux *trixie*[5]. O código possui um ambiente virtual que instala as bibliotecas Python necessárias. O vídeo com o passo-a-passo para instalação e utilização está disponível publicamente[6].

O usuário da Pythia deve fornecer um arquivo com os parâmetros do experimento, chamado *scenario.xml*. O experimento deve descrever os UEApps, os UEs, os MEC Hosts e os MECApps do cenário. Cada UEApp deve estar relacionado ao UE que o executa, assim como cada MECApp deve estar relacionado ao MEC Host que o executa. A Seção 3 detalha a representação dessas entidades no funcionamento interno da Pythia. O código disponível possui alguns cenários de exemplo para guiar usuários.

Parâmetro de entrada	**Significado**
ue	Um UE do cenário. Pode executar várias UEApps simultaneamente.
ue_app	Uma UEApp. Fornecida pelo usuário na forma de imagem Docker.
mec_host	Um MECHost do sistema emulado. Pode executar várias MECApps simultaneamente.
mec_app	Uma MECApp. Fornecida pelo usuário como uma imagem Docker.
link	Representa as condições de rede entre um UE e um MECHost. Permite a manipulação de latência, de taxa de download e de taxa de upload.

Tabela 1. Parâmetros de entrada da Pythia.

Além de descrever os dispositivos e aplicações relativos ao cenário, o experimento também deve descrever as conexões entre os UEs e os MEC Hosts. É possível definir os parâmetros das conexões de maneira temporal. Por exemplo, pode-se definir que a latência entre um determinado UE e um determinado MEC Host é de 500 ms a partir de 1 s de emulação, mas que ela se torna 200 ms a partir de 20 s de emulação. Essa dinâmica permite emular o efeito do movimento de UEs com relação à rede.

[3] `https://github.com/GTA-UFRJ/pythia`
[4] `https://docs.docker.com/desktop/`
[5] `https://www.debian.org/releases/testing/index.html`
[6] `https://www.youtube.com/watch?v=kEY2-_nyUuM`

3. Arquitetura da Plataforma

A ferramenta modela quatro componentes principais da arquitetura MEC: UEApp, Virtual UE (vUE), Virtual MEC Host (vMECHost) e MECApp, todos implementados como contêineres Docker [Merkel 2014]. A comunicação entre os componentes se dá através de três redes: Rede UE, Rede interna e Rede MEC. As três redes são subredes IP implementadas com *bridges* no Docker, cada uma com uma faixa de ips definida no arquivo de configuração. A Figura 1 ilustra os componentes da plataforma Pythia e as redes que os conectam.

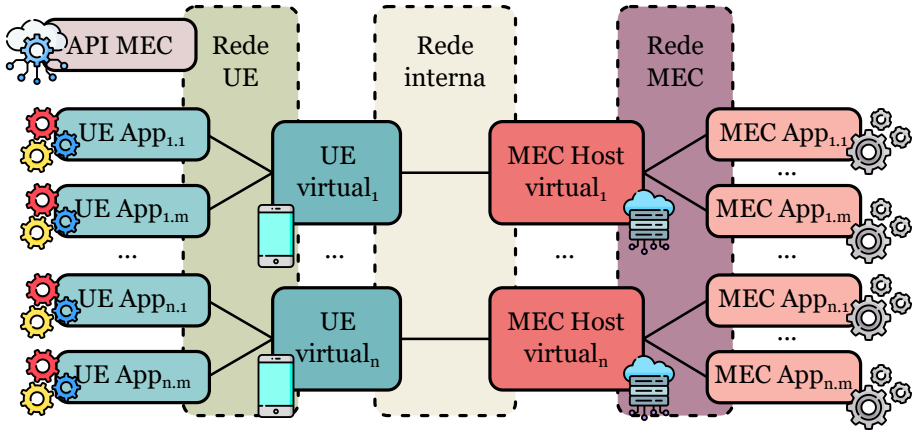


Figura 1. Os componentes da plataforma Pythia.

Os componentes da arquitetura desempenham as seguintes funções:

- **UEApp**: é uma aplicação desenvolvida pelo usuário da Pythia, a ser executada por um ou mais UEs. Junto com a MECApp, a UEApp é o alvo da emulação e, por isso, uma das entradas da Pythia. A UEApp deve ser capaz de realizar *offloading* de tarefas para uma MECApp. Para execução no ambiente da Pythia, deve estar na forma de uma imagem de contêiner Docker.
- **MECApp**: é uma aplicação desenvolvida pelo usuário da Pythia, a ser executada por MEC hosts. No caso geral, ela é projetada para atender aos pedidos de *offloading* de tarefas por UEApps. Junto com a UEApp, a MECApp é o alvo da emulação. Para execução no ambiente da Pythia, deve estar na forma de uma imagem de contêiner Docker. Assim como o caso das UEApps, pode haver a execução simultânea de várias MECApps bem como a execução de várias instâncias de uma mesma MECApp.
- **vUE**: o vUE representa um UE do cenário que se deseja emular (por exemplo, um smartphone), podendo haver um ou vários no mesmo cenário de emulação. Internamente à Pythia, o vUE é um contêiner Docker que é parte integrante da plataforma. O como um UE pode executar mais de uma aplicação simultaneamente, o vUE pode estar vinculado a uma ou mais aplicações.
- **vMECHost**: o vMECHost representa um MEC host do cenário que se deseja emular. Um MEC Host é um servidor de recursos computacionais, podendo haver vários no mesmo cenário de emulação. Assim como o vUE, o vMECHost é um contêiner Docker cuja imagem é fornecida junto com o resto do código da Pythia.

- **Rede UE**: Uma rede Docker que conecta as UEApps aos vUEs. Cada UEApp e cada vUE possui uma interface virtual de rede conectada a essa rede. Os UEApps têm acesso às outras entidades somente pela Rede UE.
- **Rede MEC**: Uma rede Docker análoga à Rede UE. Ela conecta os vMECHosts aos MECApps. Cada MECApp e cada vMECHost possui uma interface virtual de rede conectada a essa rede. Os MECApps têm acesso às outras entidades somente pela Rede MEC.
- **Rede interna**: Uma rede Docker que conecta os vUEs aos vMECHosts. Além das respectivas conexões à Rede UE e à Rede MEC, os vUEs e vMECHosts possuem uma interface virtual conectada à rede MEC.
- **API MEC**: Esse componente é um serviço capaz de emular a API de um sistema MEC, respondendo às requisições de UEApps de acordo com o padrão ETSI. Atualmente, a ferramenta possui implementada a API AppList [ETSI 2020], que permite a uma UEApp descobrir as aplicações disponíveis no sistema MEC.

Como ilustrado na Figura 1, a conexão entre uma UEApp e sua respectiva MECApp é feita através de um vUE e um vMECHost. Em termos de implementação, a UEApp possui uma interface virtual, que é conectada à Rede UE. A tabela de roteamento do contêiner da UEApp é configurada para que seu *gateway* seja o vUE correspondente. De maneira análoga, as MECApps possuem apenas uma interface, conectada à Rede MEC. Para cada MECApp, seu vMECHost correspondente é configurado como seu *gateway*.

O vUE e o vMECHost conectam-se por meio da Rede interna. Cada vUE, ao receber um pacote por sua interface na Rede UE, encaminha o pacote ao vMECHost adequado através da Rede Interna. O caminho inverso também é configurado nos vMECHost's. A Rede Interna é responsável por toda a manipulação do tráfego de rede. Os parâmetros de latência, capacidade e percentual de perda são adicionados na transmissão dos pacotes pela interface conectada à Rede Interna, tanto do vUE quanto do vMECHost.

A manipulação do tráfego é realizada utilizando-se da ferramenta NetEm (Network Emulator) [Hemminger et al. 2005]. O NetEm é parte da ferramenta tc[7] e separa o tráfego de saída de cada interface em diferentes filas. Os pacotes presentes em cada fila sofrem, então, políticas de modificação de tráfego relativos a cada fila, com parâmetros definidos pelo usuário (latência, perda, duplicação de pacotes, etc.). Na Pythia, essas filas estão presentes na interface da rede interna do vUE e do vMECHost. Entretanto, cada vUE pode apresentar conexões com múltiplos vMECHosts, e o mesmo vale para o vMECHost. Assim, a fim de evitar que parâmetros de uma conexão diferente sejam aplicados, a Pythia implementa uma fila para cada enlace virtual existente. O filtro das filas é baseado no protocolo IP de modo que ao analisar o IP de destino de cada pacote é possível atribuí-lo a uma fila específica.

A Pythia possui duas fases de operação: inicialização (*bootstrap*) e emulação. A fase de inicialização instancia todos os contêineres e inicializa a execução dos contêineres relativos aos vUEs e vMECHosts. Durante a inicialização, as conexões do cenário são registradas e as filas do NetEm são criadas. As filas, no entanto, só recebem os parâmetros de alterações no tráfego de rede uma vez que a emulação começa.

A segunda fase, de emulação, inicializa os contêineres relativos às UEApps e ME-

[7] `https://man7.org/linux/man-pages/man8/tc.8.html`

Cenário	Nº de UEs	Nº de MEC Hosts	N° de Eventos
1	1	1	6
2	2	2	24
3	4	4	96
4	8	8	384
5	16	16	1536

Tabela 2. Resumo dos cenários de avaliação da ferramenta.

CApps, executando-os enquanto percorre o laço de emulação. O laço de emulação altera os parâmetros de conexão nos instantes adequados. As mudanças de conexão são agrupadas em uma fila de eventos e, no momento em que o tempo de emulação se aproxima de um dos tempos de mudança de links do arquivo de cenário, ocorre uma chamada de função para a mudança dos parâmetros das conexões. Após a alteração, o evento é retirado da fila. A divisão entre as fases de inicialização e emulação garante que, durante a emulação, apenas as tarefas pertinentes sejam executadas. Assim, são evitadas sobrecargas que poderiam reduzir a fidelidade da emulação.

A virtualização dos componentes ocorre com a plataforma Docker [Merkel 2014]. Contudo, a fim de possibilitar uma possível migração para outra plataforma, como Kubernetes[8], existe uma camada de abstração, o arquivo `docker_utils.py`, que previne a interação direta entre a emulação e as funções do Docker.

A API MEC da Pythia é inicializada durante a fase de inicialização e executada ao longo da fase de emulação. Ela representa, do ponto de vista das UEApps, o Sistema MEC. A API possui uma interface na rede UE e é responsável por responder às requisições do padrão ETSI. Atualmente, a API é capaz de listar os MECApps disponíveis para os UEs, seguindo a API ETSI *ApplicationList* [ETSI 2020].

4. Análise de Desempenho

A plataforma é avaliada com relação ao seu uso de memória e de CPU (processador). Para avaliar essas características, são desenvolvidos 5 cenários experimentais. Os cenários estão resumidos na Tabela 2. Cada linha da tabela representa um cenário, enquanto as colunas representam o número de dispositivos, que podem ser UEs ou MEC Hosts. Cada UE executa uma UEApp e cada MEC Host executa uma MECApp. A UEApp e a MECApp utilizadas são, respectivamente, o cliente e o servidor da aplicação docker contida no diretório `detect_latency`. Os códigos utilizados para os experimentos estão disponíveis no repositório do código, em uma *branch* denominada `tests`.

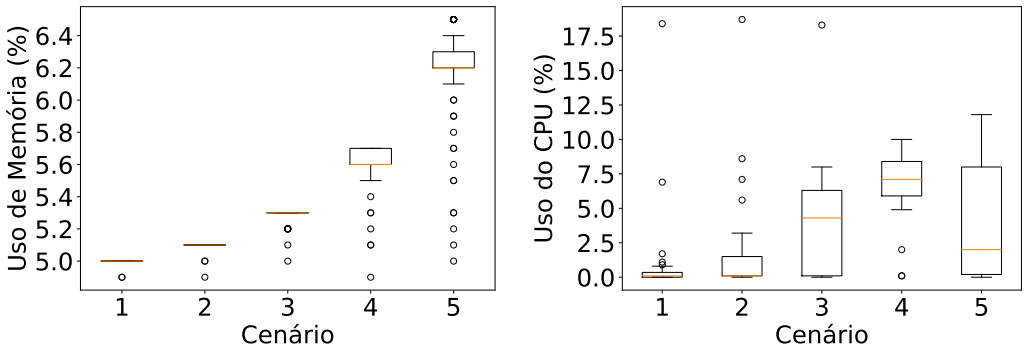
Os cenários são executados em uma máquina com processador Intel i7-12700 de 12 núcleos, frequência de 4,9 GHz. A máquina possui memória RAM DDR3 de 64 GB e armazenamento por SSD de 1 TB. A Figura 2 apresenta os resultados da utilização de CPU e de memória para cada um dos cenários. Uma vez que o processador possui 12 núcleos, a utilização da CPU pode chegar a 1.200%. As medidas são obtidas através da biblioteca Psutil[9], utilizando um processo separado para realizar medidas a cada segundo.

A Figura 2(a) ilustra o uso de memória em cada cenário, no formato de diagrama

[8] `https://kubernetes.io/pt-br`

[9] `https://psutil.readthedocs.io/en/latest/`

de caixas. É perceptível o aumento de utilização de memória com o aumento da quantidade de UEs e MECHosts emulados. A Figura 2(b) ilustra o uso de CPU em cada cenário. Pode-se observar a crescente utilização de CPU por cada cenário, exceto no Cenário 5. Uma hipótese para explicar tal fenômeno é que, a fase de *bootstrap* do Cenário 5 ocupa uma proporção mais significativa do experimento, reduzindo a média de uso da CPU. Assim, são medidos os tempos de *bootstrap* de cada cenário, cujos resultados estão na Figura 3, confirmando a hipótese.



(a) Utilização de memória em cada cenário. (b) Utilização de CPU em cada cenário.

Figura 2. Uso de recursos nos experimentos.

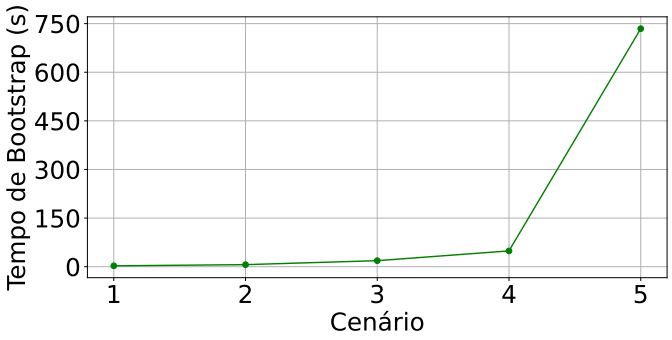


Figura 3. Evolução do tempo de boostrap.

O estudo do tempo de mudança das conexões é realizado registrando a diferença entre o tempo presente no arquivo `scenario.xml` e o tempo real da emulação, além do tempo entre os eventos configurados para o mesmo instante. O tempo médio de mudança de conexões é de 162 ± 40 ms, para todos os 2.046 eventos de todos os cenários. Esse tempo é fortemente influenciado pela função do Docker para executar os comandos de manipulação no tráfego de rede nos contêineres.

5. Demonstração

A demonstração da ferramenta envolve a execução da versão aberta do jogo "Quake III Arena"[10] como UEApp e MECApp. Os visitantes serão convidados(as) a jogar contra um bot (jogador controlado por computador) sob diferentes perfis de latência, simulando diferentes cenários. Assim, a experiência de jogo poderá ser sentida diretamente.

[10] `https://github.com/id-Software/Quake-III-Arena`

Para executar essa demonstração, será necessário uma máquina, um monitor, um teclado e um mouse. Os requisitos mínimos da máquina são: processador Intel i5 ou similar, memória de 8 GB e sistema operacional Debian *trixie*. Também é necessário acesso à Internet para instalação de bibliotecas e outras dependências de software.

6. Conclusão

A análise do comportamento de dispositivos móveis e suas aplicações em um contexto de mobilidade é crucial para a formulação de técnicas de otimização da alocação de recursos no paradigma MEC. Entretanto, os emuladores tradicionais se tornam alternativas pouco eficientes por não possuírem foco na mobilidade de usuários. Este trabalho propõe a Pythia, uma ferramenta leve computacionalmente e ágil. A Pythia emula as condições de rede do ponto de vista de uma aplicação UE, sendo capaz de emular a mobilidade do UE. Ela pode ser utilizada por operadoras para desenvolver novas estratégias de alocação de recursos, assim como por desenvolvedores de aplicações para estimar o efeito da mobilidade na QoE. O trabalho apresenta uma análise do desempenho da Pythia e propõe uma demonstração.

Como trabalhos futuros, planeja-se expandir a plataforma para integrar o uso de múltiplas máquinas na emulação e possibilitar o uso de aplicações externas não geradas pela plataforma. Também pretende-se elaborar uma estratégia de alocação de recursos propícia ao ecossistema MEC e suas limitações motivadas pelo uso de dispositivos móveis, a partir dos dados da plataforma.

7. Agradecimentos

Este trabalho foi parcialmente financiado pela Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro - FAPERJ e utiliza ícones por Freepik, de `flaticon.com`.

Referências

Cruz, P., Achir, N., and Viana, A. C. (2022). On the edge of the deployment: A survey on multi-access edge computing. *ACM Computing Surveys*, 55(5):1–34.

ETSI (2020). Multi-access edge computing (mec); device application interface. Technical Report ETSI GS MEC 016 V2.2.1, European Telecommunications Standards Institute, Sophia Antipolis, France.

Gouveia, P., Neves, J., Segarra, C., Liechti, L., Issa, S., Schiavoni, V., and Matos, M. (2020). Kollaps: decentralized and dynamic topology emulation. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–16.

Hemminger, S. et al. (2005). Network emulation with netem. In *Linux conf au*, volume 5.

Imputato, P., Avallone, S., and Pecorella, T. (2017). Network emulation support in ns-3 through kernel bypass techniques. In *Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools*, pages 259–260.

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239).

Silva, T. P., Rocha, A., Batista, T. V., da Silva Lopes, F. A., Delicato, F. C., and Pires, P. F. (2020). Plataformas de fog computing: da teoria à prática. *Minicursos do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*.