**UNIVERSIDADE FEDERAL DO CEARÁ**

**CAMPUS QUIXADÁ**

**CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO**

**PATRICK DA SILVA PEREIRA**

# DESENVOLVIMENTO DE UM *FRONT-END WEB* PARA UMA FERRAMENTA DE EXPERIMENTAÇÃO EM CENÁRIOS DE *MOBILE CLOUD COMPUTING*

**QUIXADÁ**

**2022**

PATRICK DA SILVA PEREIRA

# DESENVOLVIMENTO DE UM *FRONT-END WEB* PARA UMA FERRAMENTA DE EXPERIMENTAÇÃO EM CENÁRIOS DE *MOBILE CLOUD COMPUTING*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Paulo Antonio Leal Rego

QUIXADÁ

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

P494d Pereira, Patrick da Silva.
Desenvolvimento de um front-end web para uma ferramenta de experimentação em cenários de mobile cloud computing / Patrick da Silva Pereira. – 2022.
48 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2022.
Orientação: Prof. Dr. Paulo Antonio Leal Rego.

1. Aplicação Web. 2. Front-end e back-end. 3. Computação em nuvem móvel. 4. Testbed. I. Título.

CDD 005

PATRICK DA SILVA PEREIRA

DESENVOLVIMENTO DE UM *FRONT-END WEB* PARA UMA FERRAMENTA DE EXPERIMENTAÇÃO EM CENÁRIOS DE *MOBILE CLOUD COMPUTING*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação.

Aprovada em: __/__/____

BANCA EXAMINADORA

---

Prof. Dr. Paulo Antonio Leal Rego (Orientador)
Universidade Federal do Ceará (UFC)

---

Prof. Me. Francisco Anderson de Almada Gomes
Universidade Federal do Ceará (UFC)

---

Tec. Renan Alves Barbosa
Universidade Federal do Ceará (UFC)

À minha família, por sua capacidade de acreditar em mim e investir em mim. Mãe, seu cuidado e dedicação foi que deram, em alguns momentos, a esperança para seguir. Pai, sua história inspirou minha jornada, descanse em paz.

## AGRADECIMENTOS

Agradeço primeiramente à Deus, pelo dom da vida, a minha família, por me apoiar sempre na minha caminhada, aos amigos, pelo incentivo, e sempre me ajudar. Agradeço ao meu falecido pai, que sempre me apoiou nas minhas decisões e lutou sempre por mim e nunca me deixou faltar nada. Agradeço ao Dr. Paulo Antonio Leal Rego, pela paciência na conclusão deste trabalho e por sempre dar oportunidades e caminhos em que posso seguir. Por fim, agradeço minha esposa, por me apoiar nas horas de ansiedades e sempre está do meu lado.

“Alguns confiam em carros e outros em cavalos, mas nós confiamos no nome do Senhor, o nosso Deus.”

(BÍBLIA, Salmo, 20, 7)

## RESUMO

Os *smartphones* evoluíram bastante nos últimos tempos, mas apesar dessa evolução, ainda possuem restrições em relação à capacidade computacional, vida útil de bateria e conectividade com a rede. Com esta problemática, surgiu o paradigma *Mobile Cloud Computing*, que supre os recursos computacionais mais pesados na nuvem. Sua ideia principal é transferir uma tarefa do ambiente móvel para nuvem, fazer o processamento da tarefa e transmitir os resultados para o dispositivo móvel, assim, essa técnica é chamada *offloading* computacional. No decorrer do tempo, foram criados diversos *frameworks* para *offloading*, porém os cenários não condiziam com a realidade para fazer experimentos e avaliar soluções pesadas. *MCC Testbed* surgiu para suprir a necessidade de experimentos e execução de cenários personalizados, no entanto, era preciso de uma *interface* amigável para criação e gerenciamento de cenários de experimentação. Diante disso, o objetivo deste trabalho é criar uma aplicação *front-end web* para auxiliar desenvolvedores e pesquisadores na utilização do *MCC Testbed*. A aplicação foi desenvolvida utilizando *ReactJS* e tem funcionalidades como criar e executar cenários, acessar dispositivo via *Virtual Network Computing* (VNC), adicionar, listar e executar aplicações *Android* com formato *Android Application Pack* (APK) para testes em *offloading*, e consultar os *logs* executados nos testes na aplicação *Android*. Foram realizados experimentos em 3 casos de uso, no primeiro caso, criação de cenário, que calcula o tempo de criação e uso de *Central Process Unit* (CPU) e memória, segundo caso, execução de cenário, utiliza métricas de tempo de *upload*, tempo de *download* e tempo de execução no servidor, e o terceiro caso, uso de recursos com VNC habilitado, calcula o uso de recursos(CPU e memória) com VNC habitado e não habilitado. Os resultados dos experimentos demonstram ser possível utilizar a ferramenta substituindo a atual interface via *Command Line Interface* (CLI).



**Palavras-chave:** Aplicação Web. Frontend. MCC. Testbed. ReactJS.

## ABSTRACT

Smartphones have evolved a lot in recent times, but despite this evolution, they still have restrictions regarding computing power, battery life, and network connectivity. With this problem, the Mobile Cloud Computing paradigm emerged, which tries to supply the heaviest computing resources in the cloud. Its main idea is to transfer a task from the mobile environment to the cloud, process the task, and transmit the results to the mobile device, thus, this technique is called computational offloading. Over time, several offloading frameworks were created, but the scenarios did not match reality to experiment and evaluate heavy solutions. MCC Testbed emerged to meet the need for experiments and execution of custom scenarios, however, it needed a friendly interface for creating and managing experimentation scenarios. Therefore, the objective of this work is to create a front-end web application to help developers and researchers in the use of MCC Testbed. The application was developed using ReactJS and has features such as creating and running scenarios, accessing the device via Virtual Network Computing (VNC), adding, listing and running Android applications with Android Application Pack (APK) format for offloading tests, and consulting the executed logs. in tests in the Android application. Experiments were carried out in 3 use cases, in the first case, scenario creation, which calculates the creation time and use of Central Process Unit (CPU) and memory, in the second case, scenario execution, uses metrics of upload time, download time and run time on the server, and the third case, resource usage with VNC enabled, calculates the resource usage (CPU and memory) with VNC inhabited and not enabled. The results of the experiments demonstrate that it is possible to use the tool to replace the current interface via the Command Line Interface (CLI).



**Keywords:** Web Application. Frontend. MCC. Testbed. ReactJS.

## LISTA DE FIGURAS

## LISTA DE QUADROS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| ADB | *Android Debugging Bridge* |
| API | *Application Programming Interface* |
| APK | *Android Application Pack* |
| CLI | *Command Line Interface* |
| CPU | *Central Process Unit* |
| DOM | *Document Object Model* |
| GNS3 | *Graphical Network Simulator-3* |
| IP | *Internet Protocol* |
| JS | *JavaScript* |
| JSON | *JavaScript Object Notation* |
| MCC | *Mobile Cloud Computing* |
| REST | *Representational State Transfer* |
| UI | *User Interface* |
| VNC | *Virtual Network Computing* |
| WAN | *Wide Area Network* |

**SUMÁRIO**

# 1 INTRODUÇÃO

Apesar de os *smartphones* terem crescido no mundo, ele ainda possui restrições. De acordo com WU (2018) dispositivos móveis, como *tablets*, *smartphones* e *smartwatches*, têm recursos limitados em capacidade computacional, vida útil da bateria e conectividade de rede, o que os impede de executar aplicativos muito complexos. *Mobile Cloud Computing* (MCC) surge como um paradigma que supre os recursos computacionais dos dispositivos móveis executando os processamentos mais pesados na nuvem, deixando assim, a aplicação com uma menor tensão de processamento e consumo energético. De acordo com Raja *et al.* (2018), A MCC em sua forma mais simples se refere a uma infraestrutura onde tanto o armazenamento de dados quanto o processamento de dados acontecem fora do dispositivo móvel.

Em Biswas e Whaiduzzaman (2019), no MCC, a ideia principal é transferir uma tarefa do ambiente móvel para a nuvem, processar a tarefa e transmitir o resultado ao dispositivo móvel novamente. Essa técnica é chamada *offloading*, ela decide qual região do código deve ser executada durante o tempo de execução. Essa tecnica também pode ser fixa e definida a nível de programação. Ao aliviar as cargas, os *smartphones* são beneficiados como energia e tempo de execução.

Com o passar dos anos, vários *frameworks* de *offloading* foram propostos, porém, segundo Barbosa e Rego (2019), os cenários adotados não condizem com a realidade de um ambiente clássico de computação em nuvem, onde vários usuários interagem simultaneamente com um serviço e a demanda muda constantemente. Em (BARBOSA; REGO, 2019), foi criada uma ferramenta, *MCC Testbed*, para auxiliar os desenvolvedores e pesquisadores de *MCC* na criação de cenários personalizados e na execução de experimentos personalizados em um ambiente de teste virtual baseado em virtualização leve.

Na ferramenta *MCC Testbed*, notou-se a necessidade de uma representação visual no processo de criação e execução de cenários criados pela ferramenta, pois a experiência de usuário ainda é fornecida via CLI e *Application Programming Interface* (API). Nesse contexto, este trabalho propõe um modelo visual utilizando *frameworks* de *interface* de usuário para melhorar o processo de criação e execução de cenários personalizados em *offloading*, consumindo a API da ferramenta MCC *Testbed*, desenvolvida por Barbosa e Rego (2019).

## 1.1 Objetivo

Foi desenvolvido uma aplicação *web* com objetivo de auxiliar desenvolvedores e pesquisadores na criação e execução de cenários, acessar dispositivo via VNC remotamente, listar, enviar e executar uma aplicação *Android* com formato APK para testes em *offloading*, além de mostrar os *logs* de testes realizados na execução da aplicação *Android*.

A aplicação *web* foi implementada utilizando *ReactJS*, onde foi constituído para criação de interface, em que, essa aplicação representa visualmente a criação de servidores, *smartphones* e *switches*, segundo a ferramenta *MCC Testbed*. Foi consumido da *API MCC Testbed* todos os recursos necessários para o funcionamento da aplicação *front-end* deste trabalho, como executar e salvar cenário, dentre outros tipos de funcionalidades já mencionada anteriormente. O *ElectronJs* foi utilizado apenas para distribuição da ferramenta em diferentes sistemas operacionais.

O trabalho está organizado da seguinte maneira: a fundamentação teórica é apresentada no Capítulo 2. No Capítulo 3, são apresentados os trabalhos relacionados. O Capítulo 4, são apresentadas as metodologias e o Capítulo 5, explica como usar a ferramenta. No Capítulo 6, são discutidos os resultados da avaliação da ferramenta e por último a conclusão é conduzida no Capítulo 7.

# 2 FUNDAMENTAÇÃO TEÓRICA

Neste Capítulo são apresentadas as tecnologias que serão utilizadas e os conceitos em que será baseado o desenvolvimento deste trabalho.

## 2.1 *Mobile Cloud Computing*

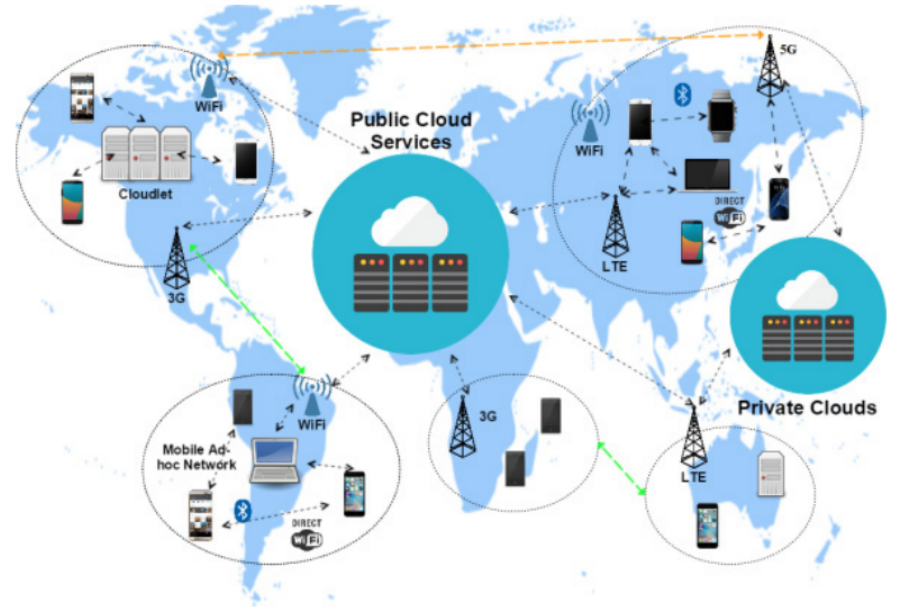
Conforme Vaidya *et al.* (2020), MCC é uma plataforma que combina os telefones móveis e os serviços de computação em nuvem. Como os *smathphone* têm muitas restrições, como poder de processamento, memória e duração da bateria, mas na computação móvel, a nuvem executa tarefas de alto desempenho e armazena abundantes dados. Usando essa técnica, os desenvolvedores conseguem desenvolver aplicativos projetados para *smartphones* sem se preocupar com as restrições de memória ou armazenamento, ou sistema operacional. Nessa arquitetura, as solicitações e informações do usuário são transmitidas aos processadores centrais por servidores móveis. Não há necessidade de instalar o aplicativo cliente no telefone do usuário neste caso.

Zhou e Buyya (2018) explana que MCC foi definido como um paradigma de computação restrito que considera as interações apenas entre dispositivos móveis e serviços de nuvem pública. Posteriormente, devido à instabilidade dos dispositivos móveis, bem como das redes sem fio, mais tipos de plataformas e recursos de computação foram introduzidos na MCC para obter serviços em nuvem móveis contínuos. Com isso, foi apresentado visão sobre uma heterogenia computação em nuvem, como mostrado na Figura 1. Com base nas definições anteriores e novos modelos, conforme a Figura 1, definiu a MCC da seguinte forma:

> A *Mobile Cloud Computing* é um paradigma de computação que permite que dispositivos móveis com recursos restritos utilizem recursos de computação heterogêneos (por exemplo, nuvens públicas, nuvens privadas e *MANETs*) em vários tipos de redes sem fio (por exemplo, rede celular, *WiFi*, *Bluetooth* e *Femtocell*) para fornecer usuários de dispositivos móveis com um serviço móvel contínuo, sob demanda e escalonável que oferece uma rica experiência de usuário. (ZHOU; BUYYA, 2018)

Dentro deste contexto, o Akherfi *et al.* (2016) afirma que o MCC atraiu a atenção dos empresários como uma solução comercial benéfica e útil que minimiza os custos de desenvolvimento e execução de aplicativos móveis, permitindo que os usuários móveis adquiram a tecnologia mais recente convenientemente sob demanda.

Figura 1 – Visão geral do ambiente de serviço de nuvem móvel.




Fonte: Zhou e Buyya (2018)

### *2.1.1 Offloading*

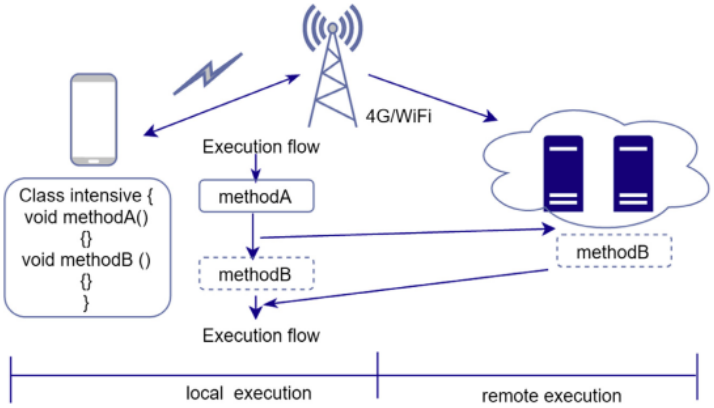
*Offloading* é uma técnica baseada em processamento remoto e migração de dados em nuvem computacional. Esta técnica é usada para tentar diminuir a tensão de consumo energético e recurso computacional que o *smartphone* produz quando aplicações de pequeno a grande porte estão executando. Deixando-as essas aplicações mais leves para executa-las no *smartphone*, devido ao compartilhamento de recursos com a nuvem.

De acordo com Akherfi *et al.* (2016), a Figura 2 ilustra o ambiente que suporta o *offloading*. Nesta visão geral, o dispositivo móvel decide fazer o *offloading* do método B para um servidor em nuvem ou uma máquina poderosa. A nuvem aqui fornece os recursos de computação virtual para executar os componentes de *offloading*. A máquina poderosa pode ser um servidor ou *cluster* em um centro de computação, uma grade de computação ou um servidor virtual na nuvem.

Segundo Biswas e Whaiduzzaman (2019), a tomada de decisão na execução de *offloading* é uma tarefa complexa. A decisão de *offloading* pode ser afetada pela rede, dispositivo móvel, modelo de aplicativo, tráfego de nuvem e usuário. Em um ambiente diferente, a decisão de *offloading* pode ser alterada. Nosso objetivo é reduzir a utilização de energia e o tempo de execução. As entidades que podem afetar a tomada de decisão de *offloading* são fornecidas abaixo:

- **Rede**: um dispositivo móvel pode usar uma rede diferente (por exemplo, 3G, 4G, LTE

Figura 2 – Uma visão geral do processo de *offloading*.




Fonte: Akherfi *et al.* (2016)

e Wi-Fi) em momentos diferentes devido à sua mobilidade. As diferentes redes usam diferentes tecnologias de transmissão, com diferentes velocidades de conexão e requisitos de energia. Essa heterogeneidade da rede afeta o esquema de *offloading*. Por exemplo, os dispositivos móveis preferem se conectar ao Wi-Fi disponível, apesar da rede móvel (3G / HSDPA, 4G). Na conexão Wi-Fi, os *smartphones* exigem menos consumo de bateria.

- **Dispositivo móvel**: diferentes dispositivos móveis têm diferentes configurações e poder de processamento. Dia a dia, os *smartphones* e *tablets* estão se tornando mais poderosos. Um dos mais recentes *smartphones Xiaomi* 12 Pro tem um processador *Octa-core* de 2,2 *Ghz*, 8 GB de *RAM*, suporte para armazenamento de dados de 256 GB e bateria de 4600mAh. Portanto, os *smartphones* dominantes podem exigir menos descarga.
- **Modelo de aplicativo**: O *design* e os objetivos do aplicativo definem seu modelo. O modelo também pode diferir como segurança, dependência de nuvem, disponibilidade de rede e particionamento. Portanto, a decisão de *offloading* de computação depende da natureza do aplicativo. Suponha que seja um aplicativo de pesquisa por voz que pesquisa o item desejado a partir da entrada de voz do usuário através de seu *smartphone*. Para reconhecimento de padrões de voz, milhões de dados de treinamento são necessários, os quais não podem ser armazenados localmente. Portanto, o modelo de aplicativo afeta a decisão de *offloading* para seus vários fins.
- **Tráfego de nuvem**: a nuvem deve fornecer serviço contínuo para um número ilimitado de solicitações. No entanto, as solicitações enviadas de muitos dispositivos móveis em simultâneo podem criar uma longa fila. Uma nova solicitação terá que esperar um determinado período antes que as outras solicitações sejam processadas. Portanto, a decisão de *offloading* é afetada pelo tráfego que atinge a nuvem. Além disso, a nuvem,

seus recursos e qualidade de serviço são cruciais para *offloading* para a tomada de decisões.

- **Usuário**: Um usuário pode não desejar usar o serviço em nuvem para privacidade de dados, custo de comunicação, custo de serviço em nuvem, etc., portanto, ele pode habilitar e desabilitar o serviço em nuvem. Além disso, o objetivo de um usuário, como economia de energia, serviço de qualidade e eficácia do aplicativo, pode afetar a decisão de *offloading*.

### *2.1.2 Cloudlet*

De acordo com Akherfi *et al.* (2016), o *offloading* para a nuvem nem sempre é uma solução, devido às altas latências da *Wide Area Network* (WAN), principalmente para aplicativos com restrições de tempo real. Portanto, a nuvem deve ser movida para mais perto do usuário móvel na forma de *cloudlets*. Um *cloudlet* é um ambiente de hospedagem para tarefas *offloading* implantadas em recursos remotos, tão diferentes quanto servidores individuais ou sistemas paralelos. *Cloudlets* são máquinas virtuais (VM) baseadas na escalabilidade, mobilidade e elasticidade do suporte. Eles estão localizados próximo de um salto para dispositivos móveis.

### *2.1.3 MCC Testbed*

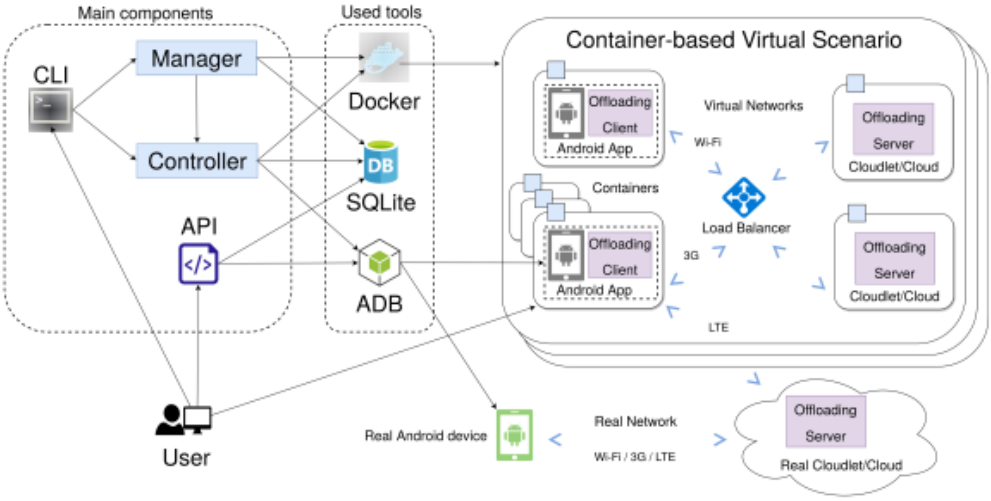
Segundo Barbosa e Rego (2019), o *MCC Testbed* permite que os usuários criem cenários de rede complexos e realistas para experiências com MCC. O *testbed* foi desenvolvido usando a linguagem de programação *Python* com três componentes principais: *CLI*, Gerente e Controlador. O gerente é responsável por criar e destruir cenários (um grupo de contêineres), enquanto o Controlador é responsável pela listagem e nome processo de validação, controle de execução de cenário e conexão para os contêineres.

Para criar tais cenários, a ferramenta utiliza softwares publicamente disponíveis, como *Android Debugging Bridge* (ADB), responsável pela comunicação com dispositivos *Android*; *Docker*, uma tecnologia de software que realiza virtualização ao nível de sistema operacional, podendo criar e gerenciar *containers*, e *SQLite*, usado para armazenar todas as informações sobre contêineres e cenários. Os desenvolvedores/pesquisadores podem criar dispositivos Android baseados em Docker, servidores remotos (*cloudlets* ou instâncias de nuvem pública) e balanceadores de carga *Nginx*, que podem ser acessados via CLI e VNC. Além disso, uma API também está disponível para permitir que desenvolvedores /pesquisadores executem experimentos de forma programática.

A Figura 3 mostra como os principais componentes da ferramenta interagem. O

usuário usa uma CLI interativa para se comunicar com o gerenciador e o controlador. O gerenciador se conecta ao banco de dados *SQLite*, controlador e usa o *Docker*. O controlador se conecta ao banco de dados *SQLite*, usa *Docker* e a ferramenta ADB. Por fim, o usuário usa a API para criar *scripts Python* para automatizar a execução de métodos dos aplicativos instalados nos contêineres *Android*. A ferramenta está sendo reestrutura para um novo padrão e este trabalho vai se basear na nova versão da ferramenta para produzir a *interface* de usuário.

Figura 3 – Visão geral dos principais componentes da ferramenta *MCC Testbed*




Fonte: Barbosa e Rego (2019)

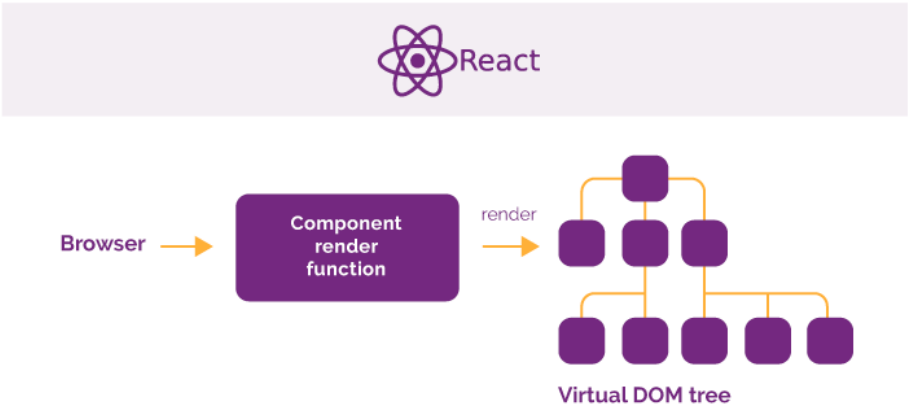
## 2.2 *ReactJS*

Em conformidade com Sousa e Gonçalves (2020), o *React* (também denominado *React.js* ou *ReactJS*) é uma biblioteca *JavaScript* de código aberto com enfoque na criação de *interfaces* de utilizadores (*frontend*) em páginas *webs*. É mantido pelo *Facebook*, *Instagram*, outras empresas, bem como uma comunidade de desenvolvedores individuais. Segundo Thakkar (2020), os componentes são a espinha dorsal do *React*. Eles são como funções, que recebem *props* (entrada), elementos de *User Interface* (UI) de saída e podem ser reutilizados como e quando necessário em outros arquivos. Mesmo que sejam como funções, você não precisa invocá-los. Eles podem ser usados como elementos HTML.

Conforme a comitiva do StackoverFlow (2021), no ano de 2021, o *ReactJS* ultrapassou *JQuery* como *framework web* mais utilizado e sendo mais utilizado atualmente. Segundo Rawat e Mahajan (2020), o React tem um estilo amplo entre as comunidades de desenvolvedores, devido à sua simplicidade e ao processo de desenvolvimento simples, porém eficaz.

Uns dos motivos de utilizar o *React* neste trabalho são processo de desenvolvimento simples e seu desempenho, pois trabalha com a virtual *Document Object Model* (DOM). Segundo Thakkar (2020), como o *HTML* é gerado usando JavaScript, o React não teria acesso ao DOM antes de ser gerado; portanto, ele mantém uma representação virtual das visualizações e, em seguida, a usa para comparar as alterações na UI. Ou seja, a biblioteca trabalha com uma cópia virtual do DOM da página *web*, copiando-o para a memória do computador, manipulando os elementos em tela com um ótimo desempenho. O fluxo de funcionamento do *ReactJs* é mostrado na Figura 4.

Figura 4 – Fluxo de funcionamento do *ReactJS*




Fonte: RubyGarage (2020)

De acordo com RubyGarage (2020), o ecossistema do *ReactJS* é composto por:

- A própria biblioteca *React* mais *React-DOM* para manipulação de *DOM*.
- *React-router* para implementação de rotas.
- *JSX*, uma linguagem de marcação especial que mistura *HTML* com *JavaScript*.
- *React Create App*, uma *interface* de linha de comando que permite configurar rapidamente um projeto *React*.
- Bibliotecas *Axios* e *Redux*, bibliotecas baseadas em *JS* que permitem organizar a comunicação com o *backend*.
- *React Developer Tools* para navegadores *Chrome* e *Firefox*.
- *React Native* para desenvolvimento de aplicações móveis nativas para *iOS* e *Android*.

### *2.2.1 JSX*

Segundo Thakkar (2020), a criação de elementos *HTML* usando a abordagem *JavaScript* reduz significativamente a legibilidade do código, especialmente quando você está criando

uma grande composição. *JSX* é uma sintaxe como HTML em JavaScript que nos permite criar elementos HTML sem invocar o método *React.createElement()*. No entanto, em segundo plano, o que acontece é que os elementos *JSX* são compilados automaticamente para as chamadas do método *createElement()*. O principal benefício de usar *JSX* é que ele aumenta significativamente a legibilidade do código e permite que os desenvolvedores escrevam código estruturado. Para renderizar os elementos *JSX*, usamos o mesmo método *render()* fornecido pelo *ReactDOM*.

### *2.2.2 Hooks*

Um recurso do *React* muito utilizado são os *Hooks*. *Hooks* foram adicionados no *React* na versão 16.8. Eles permitem que você use o *state* e outros recursos do *React* sem escrever uma classe. *Hooks* resolvem uma variedade de problemas aparentemente separados em React que encontramos ao longo de cinco anos escrevendo e mantendo milhares de componentes. Esteja você aprendendo *React*, usando diariamente, ou até mesmo se prefere outra biblioteca com um modelo de componente parecido, você reconhecerá alguns destes problemas (FACEBOOK, 2019).

# 3 TRABALHOS RELACIONADOS

Neste Capítulo são apresentados os trabalhos relacionados, conceituando seus principais pontos e como estão relacionados a este trabalho.

## 3.1 *MiniNAM: A Network Animator for Visualizing Real-Time Packet Flows in Mininet*

O trabalho (KHALID *et al.*, 2017) apresenta um utilitário chamado MiniNAN, que fornece animação em tempo real de redes criadas pelo emulador *Mininet*. *Mininet*, de acordo com Khalid *et al.* (2017), é um dos emuladores de rede mais conhecidos na pesquisa e na academia. O utilitário *MiniNAN* propõe uma solução de observar e monitorar visualmente os pacotes a fluir pela topologia de rede criada, apesar de o *Mininet* conseguir emular redes tradicionais e definidas por software, ele ainda não conseguia fazer isto. O utilitário inclui todos os componentes necessários para iniciar, visualizar e modificar fluxos de rede *Mininet* em tempo real. O *MiniNAM* foi desenvolvido usando *Tkinter* e *Mininets Python Api* e fornece uma interface gráfica de usuário que permite a modificação dinâmica de preferências e filtros de pacotes: usuário pode visualizar fluxos seletivos com opções para colorir pacotes de código com base no tipo de pacote e/ou nó de origem.

As diferenças entre o trabalho proposto e o trabalho de (KHALID *et al.*, 2017), são que o primeiro é um *front-end* voltado para criar cenários de rede complexos e realistas para experiências com *MCC* e segundo é um *front-end* voltado criar topologias de rede, observando e monitorando visualmente pacotes a fluir pela topologia de rede criada.

## 3.2 *Cytoscape StringApp: Network Analysis and Visualization of Proteomics Data*

O trabalho de (DONCHEVA *et al.*, 2018) apresenta um aplicativo, *stringApp*. Esse aplicativo é uma integração entre *STRING*(é o banco de dados, que fornece redes de proteínas para mais de 2000 organismos, incluindo interações físicas de dados experimentais e associações funcionais de caminhos selecionados, fazendo mineração automática de texto e métodos de previsão.) e o *Cytoscape*(É um software que trabalha com grandes redes e oferece maior flexibilidade como análise de rede, importação, e visualização de dados adicionais). Com isto, o aplicativo consegue facilitar a importação de redes *STRING* para o *Cytoscape*, mantendo a aparência e muitos recursos de *STRING* e integrando dados de bancos de dados associados. Nesse trabalho, apresentam recursos do *stringApp* e mostram como eles podem ser usados para realizar

análises de rede complexas e tarefas de visualização em um conjunto de dados proteômicos típico, tudo por meio da interface de usuário do *Cytoscape*.

Segundo Doncheva *et al.* (2018), *stringApp* foi projetado para servir como uma ponte entre dois recursos bem conhecidos e amplamente utilizados, o banco de dados *STRING* para redes de associação de proteína — proteína com controle de qualidade e a plataforma de *software Cytoscape* para integração, análise e visualização de dados de rede. Assim, o objetivo principal do *stringApp* é recuperar dados de rede de *STRING*, importá-los para o *Cytoscape* e manter a aparência e maior parte de funcionalidade de banco de dados *STRING*, enquanto ao mesmo permite que os usuários analisem a rede com o conjunto completo que o *Cytoscape* apresenta e integra-o com os seus próprios dados.

O *stringApp* é implementado em *Java* utilizando o *Cytoscape* 3.6 *App API*. O aplicativo tem duas funções principais: servir como uma ponte entre o *Cytoscape* e as *APIs* de serviço da *web* de *STRING* e os bancos de dados relacionados, e fornecer visualizações semelhantes às do servidor da *web STRING*, bem como recursos adicionais como o painel lateral e as visualizações de enriquecimento. Essas duas funções trabalham juntas para trazer grande parte da riqueza do site *STRING* para o *Cytoscape*, que permite então que a rede e todos os dados associados sejam analisados com o *Cytoscape* e suas centenas de outros aplicativos. Por exemplo, o aplicativo *clusterMaker2* pode ser muito útil para agrupar. redes *STRING*, conforme mostrado no caso de uso abaixo.

A funcionalidade de ponte do *stringApp* usa várias *APIs* de serviço da *web RESTful* para consultar os bancos de dados e recuperar redes. No caso de consultas de proteínas e proteínas/compostos, o aplicativo primeiro resolve os termos de consulta inseridos para os identificadores internos do banco de dados usando o *STRING* e *STITCH API* padrão. Para consultas de doenças, ele entra em contato com o *API* do banco de dados *DISEASES* duas vezes, primeiro para resolver o nome da doença inserido para um identificador de doença e, segundo, para recuperar a lista de proteínas associadas à doença.

A principal diferença entre o trabalho (DONCHEVA *et al.*, 2018) e o este trabalho é, *stringApp* é *front-end* voltado para análise de rede complexa de proteínas enquanto este trabalho é um *front-end* voltado para criação de cenários de redes complexos e realistas para experiências com MCC.

## 3.3 *Visualization of Traffic Flows in a Simulated Network Environment to investigate abnormal Network Behavior in complex Network Infrastructures*

(CATAL *et al.*, 2019) apresenta um trabalho com base na necessidade de uma investigação realista de novas tecnologias de rede (por exemplo, no ambiente *IPv6*) na fase de planejamento de rede para novas infraestruturas e novos desenvolvimentos de redes existentes, este artigo identifica a falta de possibilidades de visualização adequadas para processos de tráfego como um grande obstáculo para teste e análise de *fault/error/failure/alarm* durante a fase de simulação do projeto e planejamento da rede. Assim, uma abordagem de solução é apresentada, como ferramentas de simulação de rede realista como *Graphical Network Simulator-3* (GNS3), que emulam roteadores reais, podem ser estendidas pela visualização de pacotes. A abordagem é baseada em vários componentes distribuídos que permitem o uso potencial da tecnologia desenvolvida em diferentes ferramentas de simulação.

O foco dos autores Catal *et al.* (2019) está, portanto, em uma função importante que se destina a facilitar a análise da fase de desenvolvimento e avaliação da adoção de novas tecnologias de rede. Além do software de simulação de rede disponível gratuitamente, adequado para vários fins educacionais, existem alguns desenvolvimentos no mercado que atendem aos requisitos das considerações acima, e também podem cobrir necessidades de natureza industrial.

Na área de *back-end*, há uma subdivisão em três subcomponentes. A unidade funcional primária é descrita pelo serviço *POST*, que pré-processa os dados que chegam do *Frontend* e persiste os registros, ou seja, transfere os dados para o *Data Store*. O armazenamento de dados é visto conscientemente como um componente separado. Para aumentar o desempenho, um banco de dados não relacional*(NoSQL)* é usado aqui, o desempenho do qual pode ser otimizado por *clustering* de banco de dados, se necessário. O serviço *GET* consegue recuperar os dados armazenados do *data store*, bem como receber dados diretamente do serviço *POST* e processá-los posteriormente sem armazenamento intermediário. O processo real de processamento de pacotes de rede ocorre no serviço *GET*. O serviço de *POST* e *GET* implementando usando *Java* servlet e *Data Store* consiste no banco de dados *NoSQL Cassandra*, que pode ser acessado com a ajuda de bibliotecas *Java*.

O *front-end* contém o componente *Tool Adapter*, que executa a função de ler os pacotes de rede dentro do subcomponente *Data Adapter* (onde os dados podem ser pré-processados) e, finalmente, transmite os dados processados para o ambiente de simulação e o *Visualization Adapter* para visualização. Um subcomponente adicional de comunicação de *back-end* assume

a função de interface de comunicação intermediária, que encaminha os dados importados para o *back-end* por um protocolo estabelecido (ou seja, *JSON*) para processamento posterior e também recebe os dados processados de volta para o *front-end*. O *VA* foi implementado usando GNS3 para simulação de ambiente e o ambiente pode ser manipulado usando a linguagem de programação *python*.

As diferenças entre o trabalho proposto e o trabalho de (KHALID *et al.*, 2017), são que o primeiro é um *front-end* voltado para criar cenários de rede complexos e realistas para experiências com MCC e segundo é *front-end* voltado para facilitar a análise da fase de desenvolvimento e avaliação da adoção de novas tecnologias de rede.

## 3.4 Comparativo entre os trabalhos relacionados e o proposto.

O Quadro 1 apresenta um comparativo entre os trabalhos com base em quatro critérios: utiliza biblioteca para representação de rede, cria um *back-end* para *front-end* se comunicar, utiliza emuladores *Android* para criar os cenários de testes e se exibe uma presentação visual no lado cliente.

Quadro 1 – Comparação entre os trabalhos relacionados e o trabalho proposto

| | (KHALID *et al.*, 2017) | (DONCHEVA *et al.*, 2018) | (CATAL *et al.*, 2019) | Trabalho Proposto |
|---|---|---|---|---|
| Utiliza biblioteca para representação de rede? | Sim | Sim | Sim | Sim |
| Cria um *back-end* para *front-end* se comunicar? | Não | Não | Sim | Não |
| Utiliza emuladores *Android* para criar os cenários de testes? | Não | Não | Não | Sim |
| Exibe uma representação visual no lado cliente? | Não | Não | Não | Sim |

Fonte: Elaborado pelo autor.

# 4 METODOLOGIA

Este capítulo apresenta os procedimentos metodológicos utilizados para criar a ferramenta, responsável por criar e executar testes personalizados em cenários de MCC. É importante ressaltar que deve-se entender bem o funcionamento das tecnologias utilizadas neste trabalho, para poder obter um bom software. A Figura 5 mostra os passos para construção deste trabalho.

Figura 5 – Ciclo de desenvolvimento


Fonte: Elaborado pelo autor

## 4.1 Análise da Problemática

Neste passo, buscou-se estudar os fundamentos de MCC e *offloading*, assim como o funcionamento do *MCC Testbed* em sua nova estrutura para entender quais melhorias e funcionalidades podem ser implementadas para tornar a interação do usuário com o sistema fácil, permitindo criar cenários e fazer testes de forma simplificada.

## 4.2 Análise de requisitos

Neste passo, levantamos os requisitos funcionais e não funcionais da ferramenta junto aos pesquisadores de *offloading* que desenvolveram o *MCC Testbed*.

Os Requisitos funcionais levantados são os seguintes:

1. O usuário deve poder criar cenário;
2. O usuário deve poder pesquisar cenários;

3. O usuário deve poder fazer *upload* e *download* de cenários;
4. O usuário deve poder configurar o *Internet Protocol* (IP) do servidor para se comunicar com a API *MCC Testbed*;
5. O usuário deve poder adicionar nó ao cenário do tipo cliente, servidor, *switch* ou *load-balance*;
6. O usuário deve poder configurar nó no cenário;
7. O usuário deve poder deletar nó do cenário;
8. O usuário deve poder conectar nó a outro nó;
9. O usuário deve poder configurar a conexão entre dois nós;
10. O usuário deve poder deletar a conexão de nó entre nó;
11. O usuário deve poder executar um cenário;
12. O usuário deve poder salvar cenário;
13. O usuário deve poder parar execução de cenário;
14. O usuário deve poder acessar área de gerenciamento de aplicativos;
15. O usuário deve poder enviar um aplicativo *Android* APK para os clientes;
16. O usuário deve poder executar um aplicativo *Android*;
17. O usuário deve poder baixar *logs* das aplicações;
18. O usuário deve poder visualizar a tela de um dispositivo cliente;

Requisitos não-funcionais:

1. O sistema *front-end* pode ser executado em qualquer sistema operacional;

## 4.3 Definição de arquitetura

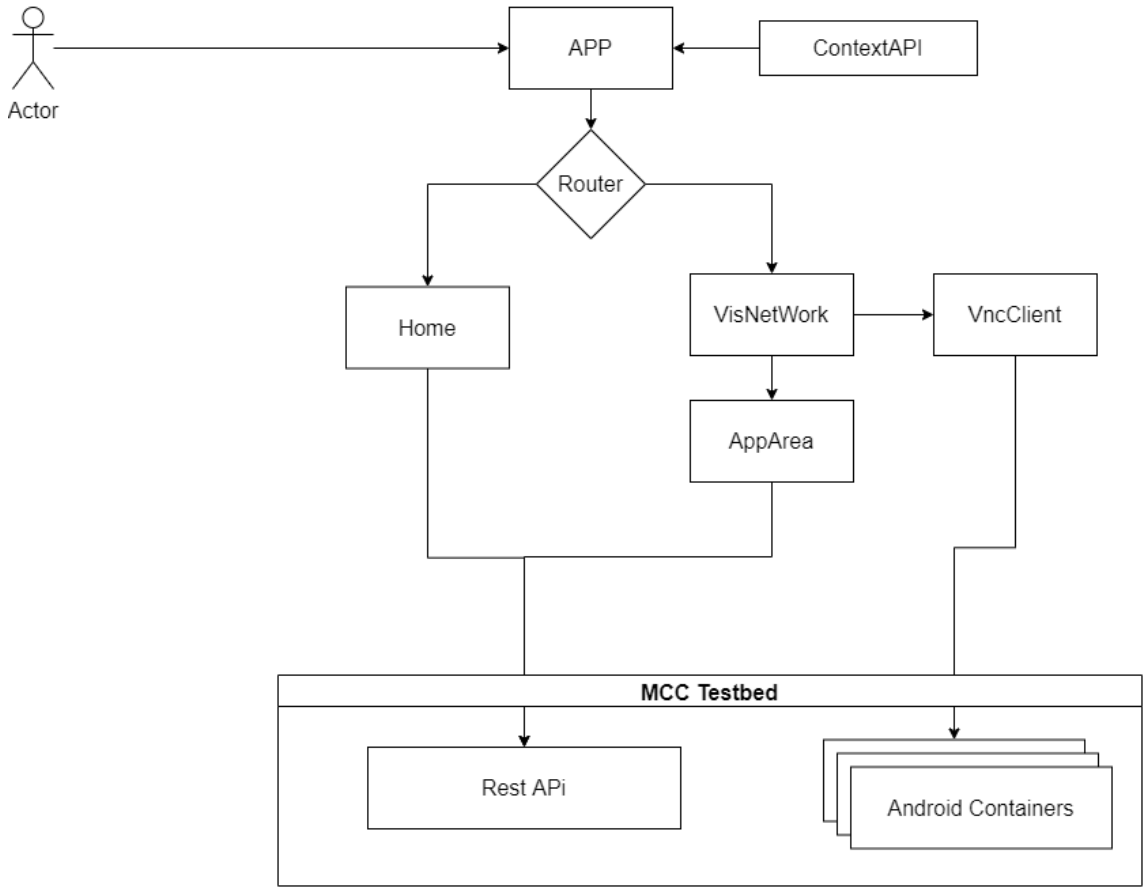
A arquitetura da ferramenta proposta foi desenvolvida utilizando a linguagem *Javascript*, foi utilizado a biblioteca *ReactJS* para construções de interfaces e utilizamos o recurso *ContextAPI* para o controle de estados da aplicação. A arquitetura é apresentada na Figura 6, ela demonstra como está organizado a estrutura de componentes do projeto. Nele, o usuário utiliza *APP* para acessar a interface da ferramenta, *APP* utiliza o *ContextAPI*, e *ContextAPI* é dividida em 3 categorias de contextos: *ApiContext*, utilizada para guardar informações do *IP* do servidor da *MCC Testbed*, *ScenarioContext*, utilizado para manipulação dos cenários da ferramenta e por fim, *GraphContext*, utilizado para manipular dados da criação de topologia de rede, com a biblioteca *vis-network-react*. O *Router* representa as rotas do projeto, na quais são *Home* e *VisNetWork*.

A *Home* representa a página inicial do projeto, ela apresenta componentes de criação de cenários, pesquisar por cenário salvo, *upload* de cenário, e configuração do servidor para adicionar o IP onde está localizada a API. A *VisNetwork*, representa a página de criação e execução de cenário, responsável por criar variados categorias de topologias de rede e executar os cenários criados.

O *VncClient* é componente do *VisNetwork* e utilizado pelo usuário quando o cenário está em execução e seleciona o nó. Para ser possível ter acesso ao dispositivo, *VisNetwork* faz uma requisição ao servidor para a porta do dispositivo, e por fim, *VncClient* via *iframe*, acessa o VNC do lado cliente.

O *AppArea* é o componente responsável por lidar pelo gerenciamento de testes automatizados com *offloading*, podendo salvar um aplicativo *Android* com formato APK, executar aplicativo salvo adicionando as suas configurações e observar os *logs* executados nos dispositivos.

Figura 6 – Arquitetura utilizada no Projeto




Fonte: Elaborado pelo autor

## 4.4 Implementação

Neste passo, desenvolvemos a ferramenta conforme os requisitos funcionais e não funcionais, onde utilizamos as bibliotecas listadas abaixo:

1. *ReactJs* para criação de interface de usuário;
2. *react-router-dom* para criação de rotas dinâmicas;
3. *vis-network-react* para criar de redes dinâmicas organizadas automaticamente e personalizáveis;
4. *material-ui* é uma biblioteca de componentes com designs prontos sendo utilizada no projeto para agilizar o tempo de desenvolvimento;
5. *styled-components* para criação de componentes, de forma que consegue criar estilos *CSS* com *JavaScript* (JS) e modificar o *CSS* de componentes prontos como do *material-ui*;
6. *axios* é uma biblioteca cliente *HTTP* e utilizada para fazer a comunicação com o servidor *MCC Testbed* API;
7. *electron*, utilizado para criar aplicações *desktop*;
8. *electron-builder* utilizado empacotar e construir um aplicativo *electron* pronto para distribuição em sistemas operacionais como *MAC OS*, *Linux*, e *Windows*.;

Todas as bibliotecas utilizadas neste trabalho são de código aberto.

### *4.4.1 Testes e Deploy*

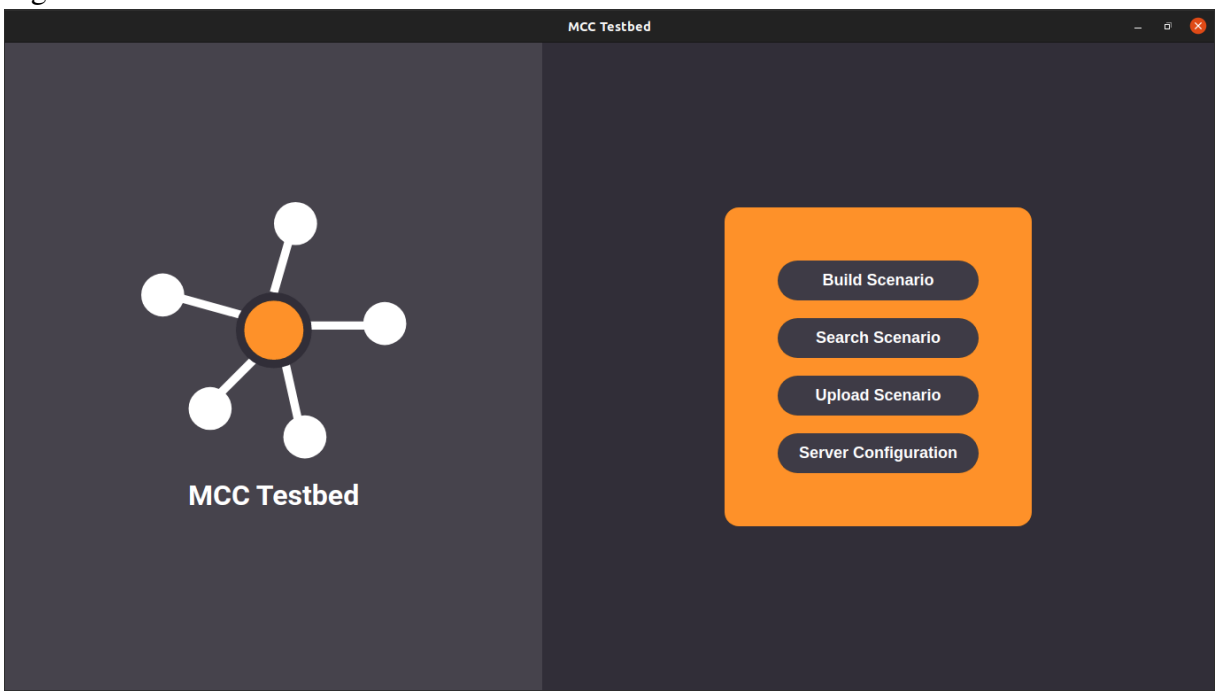
Neste passo, foram realizados testes funcionais com os desenvolvedores do *MCC Testbed*, para eles poderem testar, foram criados instaladores *.deb* para *Linux*, *.exe* para *Windows* e *.pkg* para *MAC OS*, disponibilizados no repositório da ferramenta. Após aprovada nos testes, foi disponibilizada na loja do *snap*, utilizada em boa parte das distribuições *Linux*, onde também pode ser encontrada a ferramenta *MCC Testbed*. Um dos motivos para ser publicada no *snap*, implica em ser uma loja gratuita para publicação.

# 5 A FERRAMENTA

A ferramenta desenvolvida neste trabalho visa facilitar o uso do *MCC Testbed* e, para isso, cria interfaces visuais para as funcionalidades presente na API *Representational State Transfer* (REST) da ferramenta. Assim, os usuários podem criar e executar cenários, ver estado do dispositivo, listar, enviar e executar aplicações para API, além de observar os *logs* de testes. Neste Capítulo é apresentado todo o funcionamento da ferramenta desenvolvida neste trabalho.

## 5.1 Tela inicial

Figura 7 – Tela inicial




Fonte: Elaborada pelo autor

Na tela inicial, ilustrada na Figura 7, são definidos 4 componentes no menu, *Build scenario*, *Search scenario*, *Upload scenario* e *Server Configuration*. A definição de cada componente está definido na lista abaixo:

- **Build scenario**: é um componente de criação de cenários, onde quando clicado, aparecerá um *modal*, onde é solicitado que o usuário insira o nome do cenário e, assim, seguir para a página de criação de cenário.
- **Search scenario**: é um componente de buscar cenários salvos no servidor, aparecerá um *modal*, onde quando selecionado um cenário salvo, vai para página de criação de cenário.
- **Upload scenario**: é um componente que busca cenários salvos no computador do usuário,

que quando selecionado um cenário, vai para página de criação de cenário.

- **Server Configuration**: é um componente de configuração de servidor, que quando clicado, aparecerá um *modal*, onde solicita que o usuário insira o IP de onde o servidor esteja em execução para utilizar a ferramenta. O modal desse componente sempre aparecerá quando ainda não tem o servidor salvo, porque a ferramenta precisa do servidor *testbed* em execução para prosseguir com seu funcionamento.

## 5.2 Tela *Network*

Na tela de *network*, são compostos por dois estados de tela, um em estado de criação, demonstrado na Figura 8, onde é permitido a criação do cenário com uma topologia válida e outro em estado de execução, demonstrado na Figura 9, onde é permitido visualizar o VNC dos dispositivos, adicionar APK que contenham configuração para executar *offloading*, executar os testes com APK, e mostrar os *logs* de cada dispositivos. As telas são detalhadas nas seções seguintes.

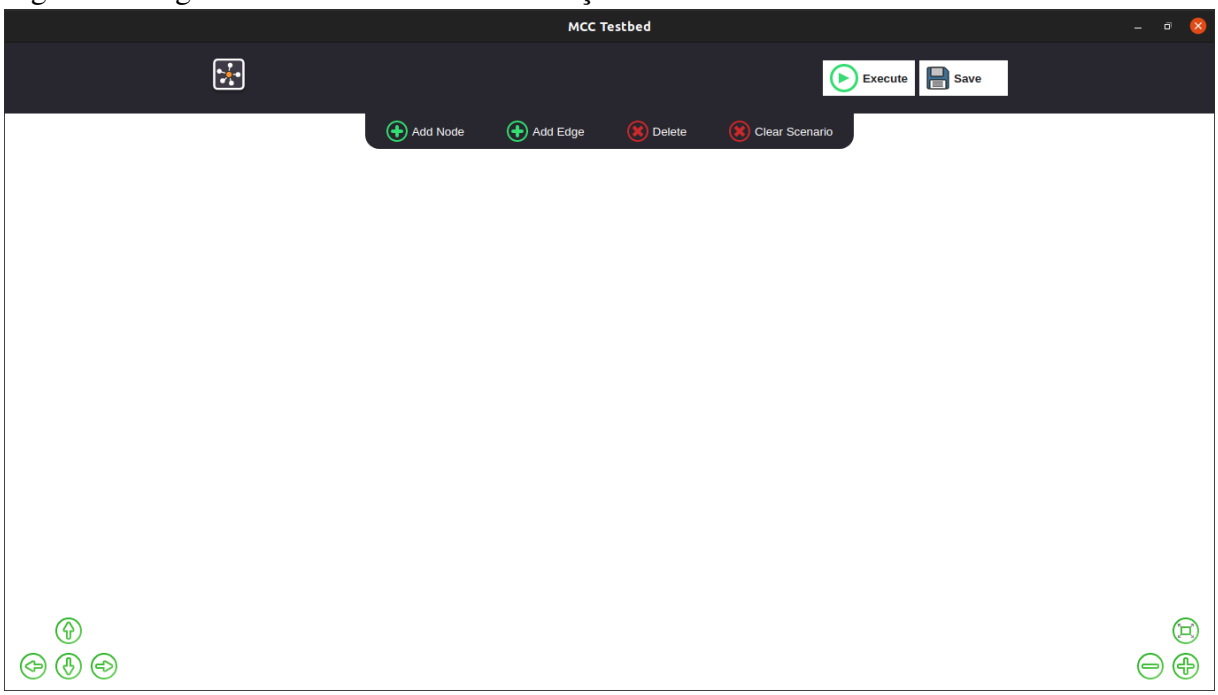
Os componentes gerais da tela *Network* são demonstrados na lista abaixo:

- **Logo**: componente utilizado para voltar para tela inicial;
- ***Execute***: botão utilizado para executar um cenário na *api testbed*;
- ***Stop***: botão utilizado para parar um cenário na *api testbed*;
- ***Save***: botão utilizado para salvar um cenário na *api testbed*;
- ***App Area***: botão utilizado para configurações de testes automatizados, disponível apenas em modo de execução de cenário;

### *5.2.1 Tela Network com estado de criação*

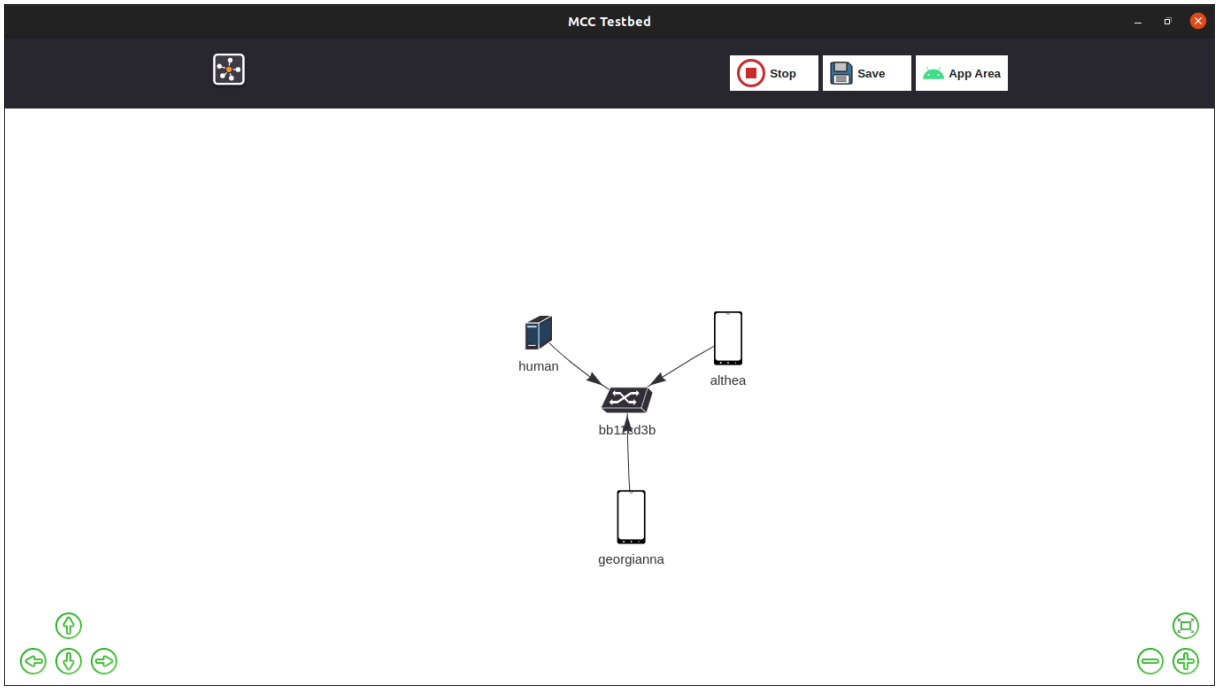
Esta tela, representada na Figura 8, neste estado permite manipular a criação de topologias de redes válidas, onde podemos criar nodes de tipos *smartphone*, *server*, *load balance* e *switch*, estes são os dispositivos disponíveis para criação de um nó. Para criação de um nó, é necessário clicar primeiramente em *Add Node*, em seguida, clicar na superfície em branca da tela, onde se encontra o componente de manipulação de *network*, aparecendo um modal de configurações do nó, como mostrar a Figura 10. A configuração é composta por quatro possibilidades, *smartphone*, *server*, *load balance* e *switch*, onde para cada escolha, é uma configuração diferente. Essa configuração é mostrada na lista abaixo.

Figura 8 – Página Network em estado de criação




Fonte: Elaborada pelo autor

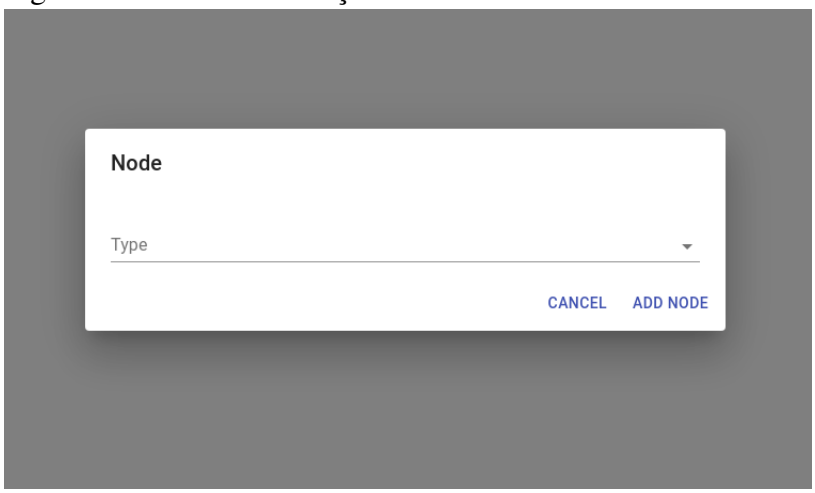
Figura 9 – Página Network em estado de execução




Fonte: Elaborada pelo autor

- **Smartphone**: este nó é composto por um tipo *Smartphone*, um nome(único) e um IP(único, podendo ser de 10.0.0.1 a 10.0.0.254);
- **Server**: este nó é composto pelo tipo *Server*, um nome(único) e um IP(único, podendo ser de 10.0.0.1 a 10.0.0.254);
- **Load Balance**: este nó é composto pelo tipo *Load-Balance*, um nome(único) e um

Figura 10 – *Modal* de criação de nó



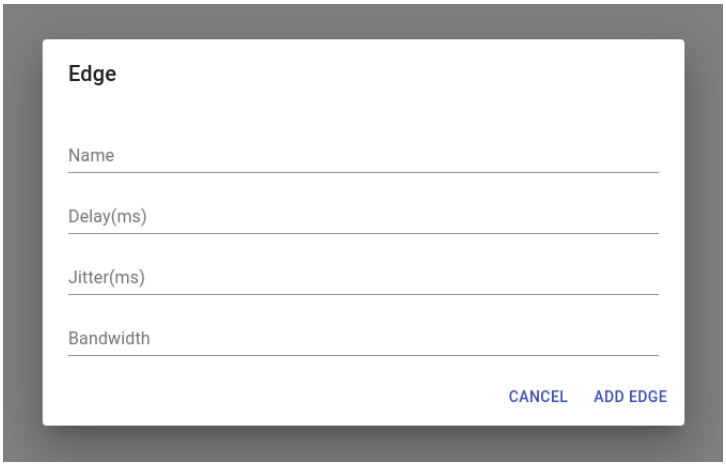

Fonte: Elaborada pelo autor

IP(único, podendo ser de 10.0.0.1 a 10.0.0.254);

- **Switch**: este nó é composto pelo tipo *Switch* e um nome(único e criado automaticamente);

Para fazer uma conexão de um nó a outro nó, é necessário clicar em *Add Edge*, em seguida, clicar e segurar o *cursor* até o nó desejado, com esse procedimento, mostra um modal de configurações da conexão, mostrado na Figura 11. Este modal mostra os parâmetros de nome, *delay*(é o atraso para cada link), *jitter*(é o atraso no recebimento entre os pacotes) e *bandwidth*(largura de banda).

Figura 11 – *Modal* de criação de conexão

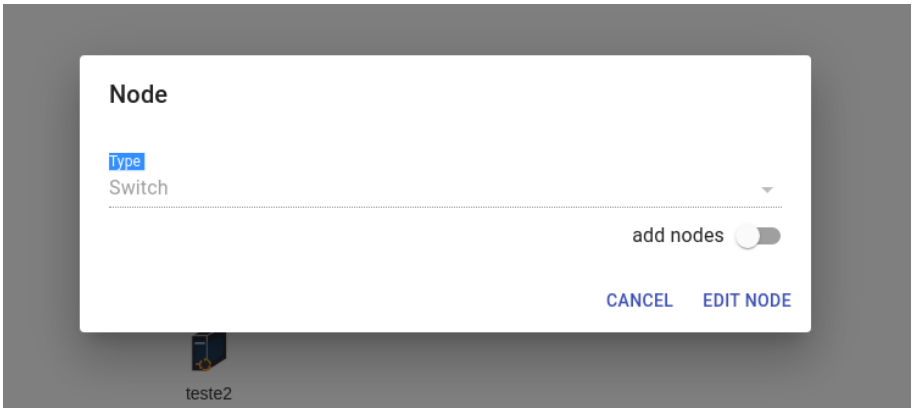



Fonte: Elaborada pelo autor

Para edição do nó, basta dar um clique duplo no nó desejado, em seguida, é mostrado um modal para alterar as configurações do nó, menos o tipo já escolhido no momento de criação do nó. A Figura 12 ilustra como é feita a alteração. Nesse exemple da Figura 12, mostra como é feita a alteração do *Switch*, como ele não tem alteração de nome e IP, ele apresenta

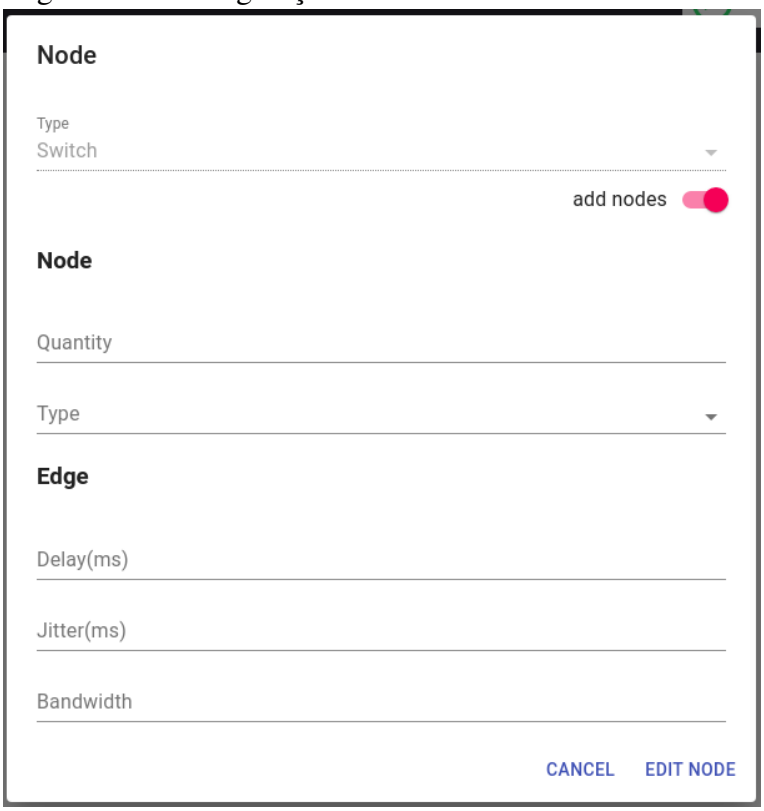
apenas a configuração de *add nodes*. Essa funcionalidade, serve para adicionar vários nós de um determinado tipo, que pode ser *Smartphone*, *Switch* ou *Server*, com configurações de nomes e IPs aleatoriamente sem repetições dos mesmo, adicionando também a configuração de conexão entre os nós adicionados e o nó de edição. Essa configuração de *add nodes* está disponível apenas para *Switch*, *Server* e *Load Balance*. A Figura 13 demonstra a configuração de *add nodes*.

Figura 12 – Modal de alteração de nó.




Fonte: Elaborada pelo autor

Figura 13 – Configuração *add nodes*.




Fonte: Elaborada pelo autor

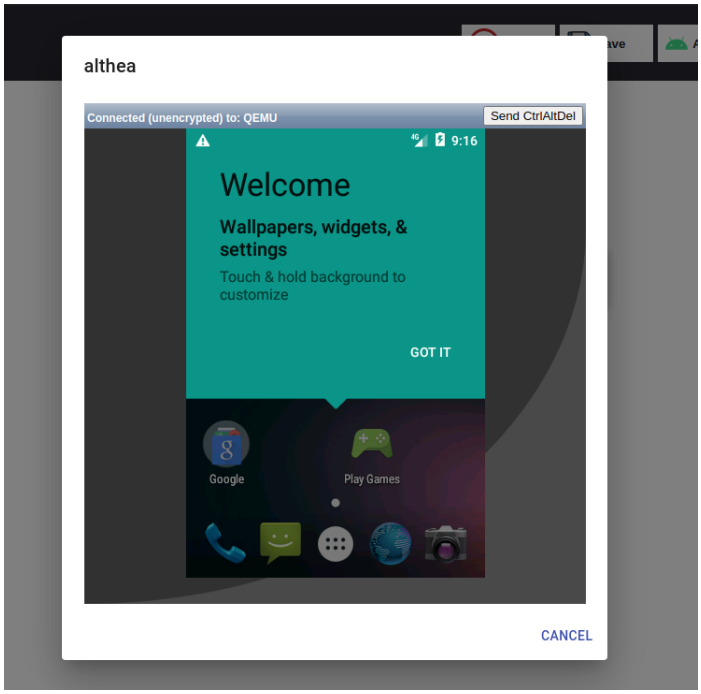
A tela *network* no modo de criação, também permitir excluir nós e conexões. Para

excluir, basta clicar no nó(*smartphone*, *server* ou *switch*) ou na conexão, e em seguida no botão de *Delete*. Caso necessite limpar todo o cenário, basta clicar em *Clear Scenario*.

### *5.2.2 Tela Network com estado de execução*

Esta tela, representada na Figura 9, neste estado de execução, permite criar testes automatizados em *offloading* utilizando a *api testbed* e visualizar o lado cliente(*smartphone*) via *VNC*. Para visualizar o lado cliente, basta dar um clique duplo no *smartphone*, ele abre um *modal*, onde contém o lado cliente via *VNC*, assim, como mostrada na Figura 14.

Figura 14 – *Modal* de conexão VNC

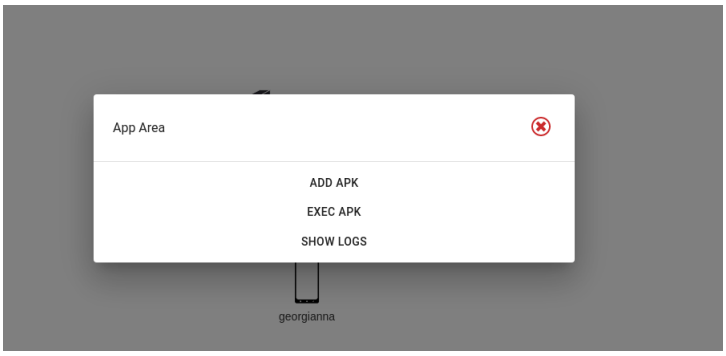



Fonte: Elaborada pelo autor

O componente *App Area*, representado na Figura 15, é responsável pelo gerenciamento de execuções de testes automatizados com *offloading*, este componente permite, adicionar uma *apk* para teste em *offloading*, executar uma *apk* como teste e mostrar os *logs* de execução dos testes automatizados. O Detalhamento dos componentes é mostrado na lista abaixo.

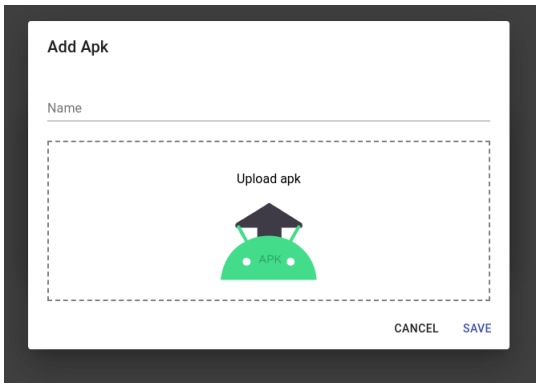
- ***ADD APK***: este componente quando clicado, abre um *modal*, a partir deste *modal*, clicando na área onde está pontilhada, permite fazer o *upload* da aplicação *Android* no formato APK a partir do computador onde a ferramenta está sendo executada, assim, como é mostrado na Figura 16;
- ***EXEC APP***: este componente quando clicado, abre um *modal*, onde mostra vários pa-

Figura 15 – *Modal* de gerenciamento de execuções de testes




Fonte: Elaborada pelo autor

Figura 16 – *Modal* de adicão de aplicativo *Android* no formato APK

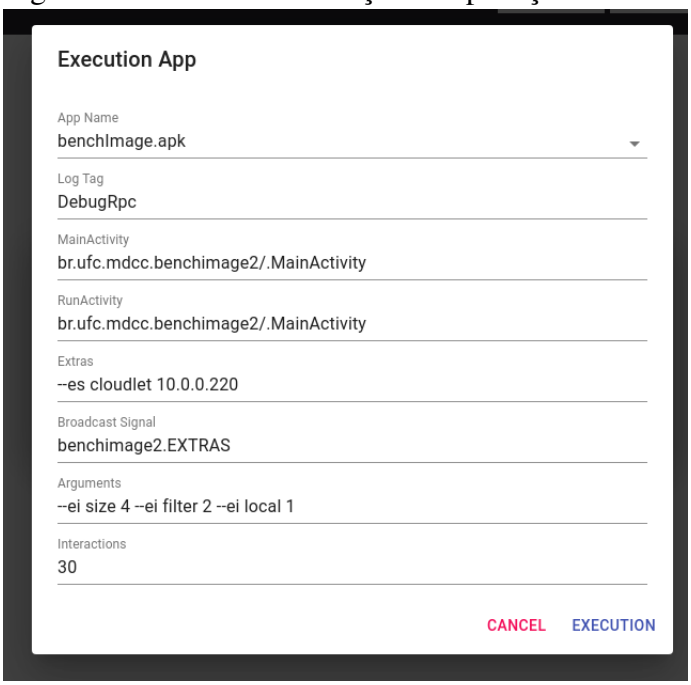



Fonte: Elaborada pelo autor

râmetros de configuração, esses parâmetros são necessários para execução da aplicação *Android*, essa configuração é mostrada na Figura 17. Os parâmetros é explicado na lista abaixo:

  - ***App Name***: Nome do aplicativo *Android* adicionado no componente *ADD APK*;
  - ***Log Tag***: informações de onde é utilizada os *logs* para o ciclo de execução dos testes;
  - ***MainActivity***: Onde está localizado o *MainActivity* no *apk*;
  - ***RunActivity***: Onde está localizado o executor da *Activity*;
  - ***Extras***: Parâmetros extras enviado para execução do teste;
  - ***Broadcast Signal***: indica onde está localizado o *Broadcast* para enviar informações do sistema operacional do *smartphone*;
  - ***Arguments***: Argumentos utilizados para indicar configurações de execuções no *APK*;
  - ***Interactions***: São o número de interações que será realizado por cliente, que no caso, *smartphone*;
- ***SHOW LOGS***: este componente quando clicado, abre um *modal*, onde mostra uma tabela de resultados dos testes, com uma lista de *device*, onde cada item, tem uma lista de *logs*,

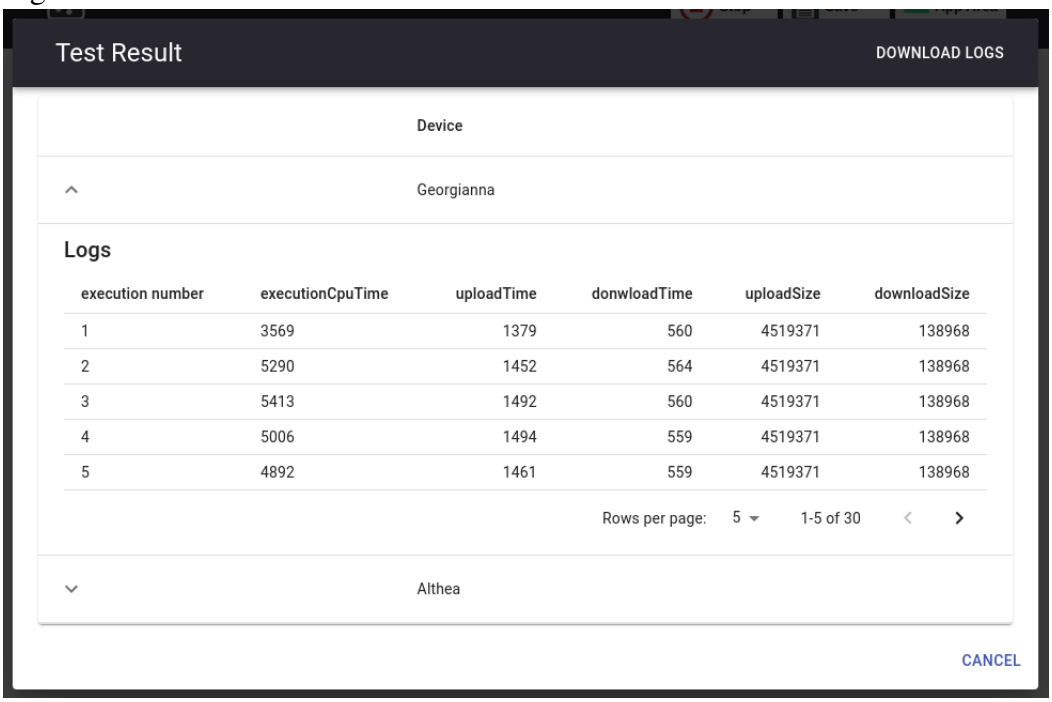
Figura 17 – *Modal* de execução de aplicação *Android*




Fonte: Elaborada pelo autor

quando é clicado na seta para baixo, é mostrada a lista de *logs*, quando clicado na seta para cima, é ocultada a lista de *logs*. Por padrão, a lista de logs é ocultada. E também, podemos fazer o *download* dos *logs* em formato *JavaScript Object Notation* (JSON). A demonstração desse componente é mostrada na Figura 18;

Figura 18 – *Modal* de resultados de testes




Fonte: Elaborada pelo autor

# 6 EXPERIMENTAL

Neste capítulo, são apresentados 3 estudos de caso para avaliação da ferramenta e resultados obtidos pelo mesmo. Para avaliação da ferramenta, é utilizado um computador com as características listadas no Quadro 2:

Quadro 2 – Característica do equipamento utilizado

| Característica | |
|---|---|
| Modelo do *hardware* | *Lenovo ideapad* 320-15IKB |
| Memória | 20GiB |
| Processador | Intel® Core™ i5-7200U CPU @ 2.50GHz |
| Gráficos | Mesa Intel® HD Graphics 620 (KBL GT2) |
| Capacidade de Disco | 720,2 GB |
| Nome do SO | Ubuntu 21.10 |
| Tipo do SO | 64 bits |
| Versão do GNOME | 40.4.0 |
| Sistemas de Janelas | X11 |

Fonte: Elaborado pelo autor

## 6.1 Criação de Cenário

Neste primeiro estudo de caso, é elaborado um experimento, utilizando a topologia rede *WIFI*. O primeiro passo deste experimento, é calculado o tempo de criação do cenário antes de qualquer execução e por fim, calcular o uso de CPU e memória em relação à ferramenta proposta neste trabalho. Os detalhes do experimento são mostrados no Quadro 3.

Quadro 3 – Detalhes do experimento de criação de cenário

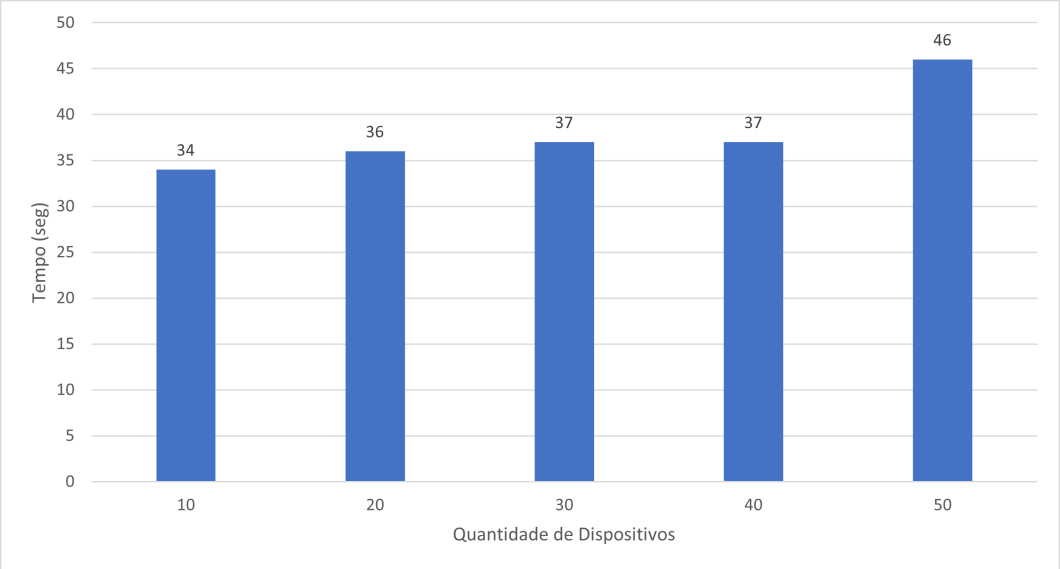
| Fator/Parâmetro | Nível |
|---|---|
| Topologia de Rede | *WIFI* |
| Quantidade de dispositivos móveis | 10, 20, 30, 40 e 50 |
| Métricas | CPU e memória |

Fonte: Elaborado pelo autor

Na Figura 19, são mostrados os resultados de tempo de criação do cenário antes da execução do mesmo. Podemos notar uma pequena variação entre a criação do cenário com menos e mais dispositivos. Essa pequena variação de tempo se deu por conta da funcionalidade

de *add nodes* que ele pode criar vários nós filhos de um mesmo tipo para um nó pai.

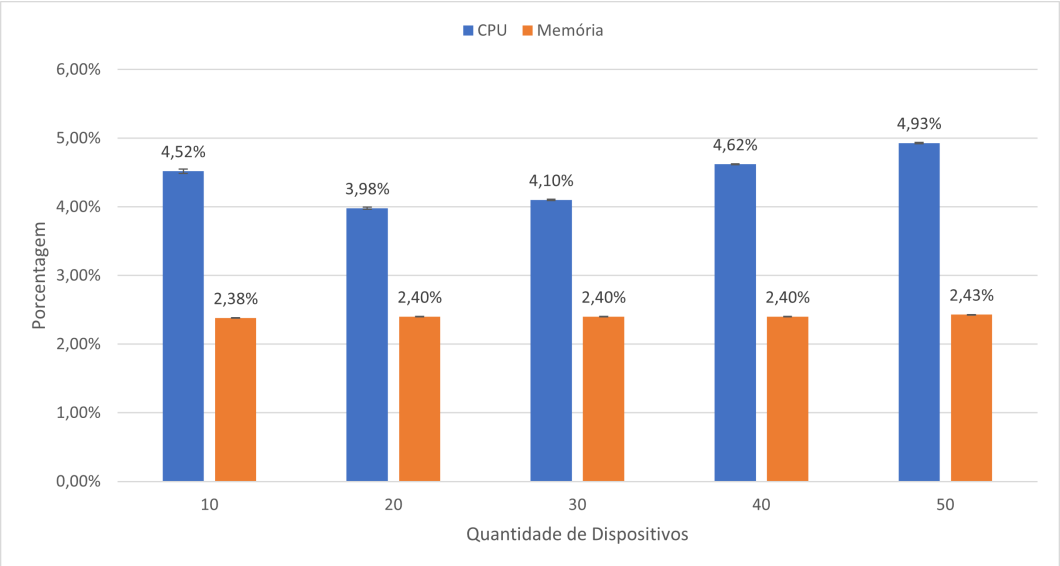
Figura 19 – Tempo de criação de cenário antes da execução com topologia *WIFI*




Fonte: Elaborada pelo autor

Em relação a este experimento, durante a criação do cenário, obtemos resultados de CPU e memória, estes dados são mostrados na Figura 20. Foi utilizado um *script* em *Shell* para capturar o uso de CPU e memória. O cenário com 10 dispositivos, teve 2558 capturas, com 20 dispositivos, 2921 capturas, com 30 dispositivos, 2813 capturas, com 40 dispositivos, 2738 capturas e por fim, com 50 dispositivos, 2433 capturas. A figura apresenta a média e um intervalo de confiança de 95%. Podemos notar que houveram pequenas variações em relação ao uso de CPU e memória durante a criação do cenário pela ferramenta.

Figura 20 – Uso de *CPU* e memória antes da execução com topologia *WIFI*

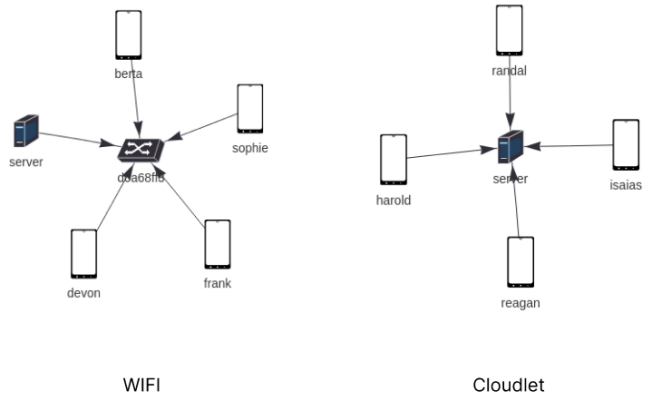



Fonte: Elaborada pelo autor

## 6.2 Execução de Cenário

No segundo estudo de caso, é elaborado um experimento, utilizando as topologias de rede *WIFI* e *Cloudlet*. Um exemplo de topologia rede *WIFI* e *Cloudlet* é ilustrado na Figura 21, na parte de rede *WIFI*, ela apresenta as conexões dos dispositivos e servidor com o *Switch*, onde os dispositivos fazem *offloading* para um *Cloudlet*, no caso, nó chamado *server*. Para topologia de rede *Cloudlet* a conexão entre o servidor é feita diretameta, ou seja, sem passar por algum *Switch*.

Figura 21 – Exemplo com topologias de rede *WIFI* e *Cloudlet*




Fonte: Elaborada pelo autor

O objetivo deste experimento, é comparar os cenários de testes de *offloading* automatizados utilizando a aplicação *BenchImage* (aplicativo de processamento de imagens que processa variadas resoluções de imagem aplicando variados tipos filtros), com métricas de tempo de *upload*, tempo de *download* e tempo de execução no servidor, onde os testes variam a quantidade de dispositivos e tamanho da imagem. No Quadro 4, mostra os fatores e níveis utilizados para avaliação desta ferramenta:

Na Figura 22, é mostrado os resultados de tempo de execução com topologia *WIFI* e na Figura 23, com topologia *Cloudlet*. Foram testados variações de 1, 2, 4 e 6 dispositivos, variando entre resolução de imagem de 2MP, 4MP e 8MP. Ao todo, nos dois cenários, *WIFI* e *Cloudlet*, foram executados 2340 vezes a aplicação. Cada dispositivo, com variação de resolução de imagem, foram executadas 30 vezes, depois calculou-se a média e intervalo de confiança de 95% do tempo de execução e transferência(*upload* e *download*).

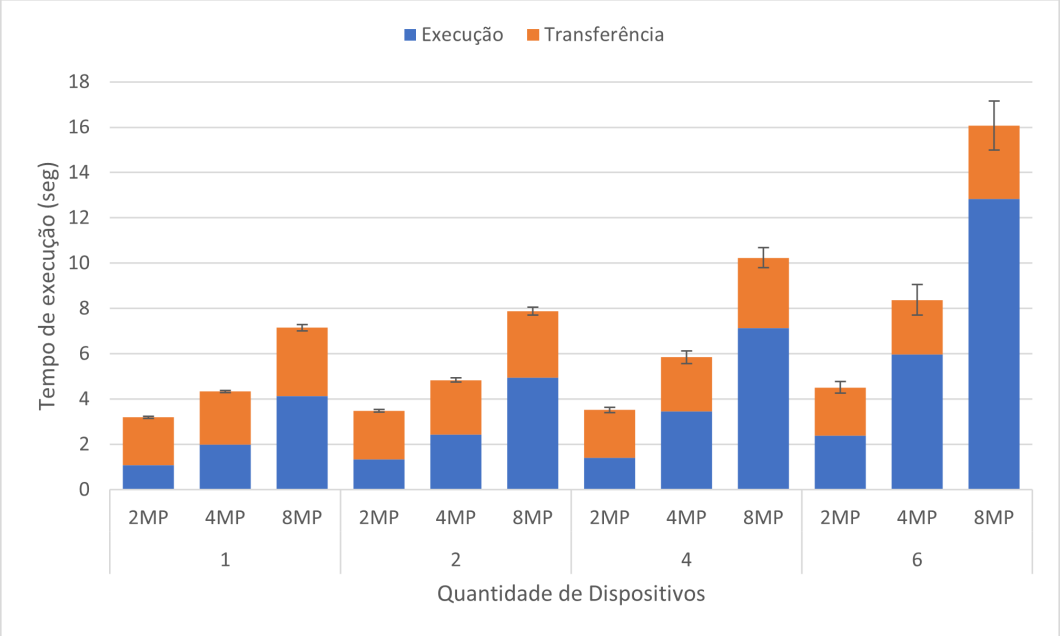
Os resultados apresentados nos gráficos *WIFI* mostrado na Figura 22 e *Cloudlet*, mostrado na Figura 23, podemos observar, conforme a quantidade de dispositivos aumenta,

Quadro 4 – Fatores/Parâmetros e níveis do experimento

| Fator/Parâmetro | Nível |
|---|---|
| Topologia de Rede | *WIFI* e *Cloudlet* |
| Quantidade de dispositivos móveis | 1, 2, 4 e 6 |
| Resolução de imagem | 2MP, 4MP e 8MP |
| *Delay* | 50ms |
| *Jitter* | 5ms |
| *Bandwidth* | 500Mbits |
| Aplicação utilizada | *BenchImage* |

Fonte: Elaborado pelo autor

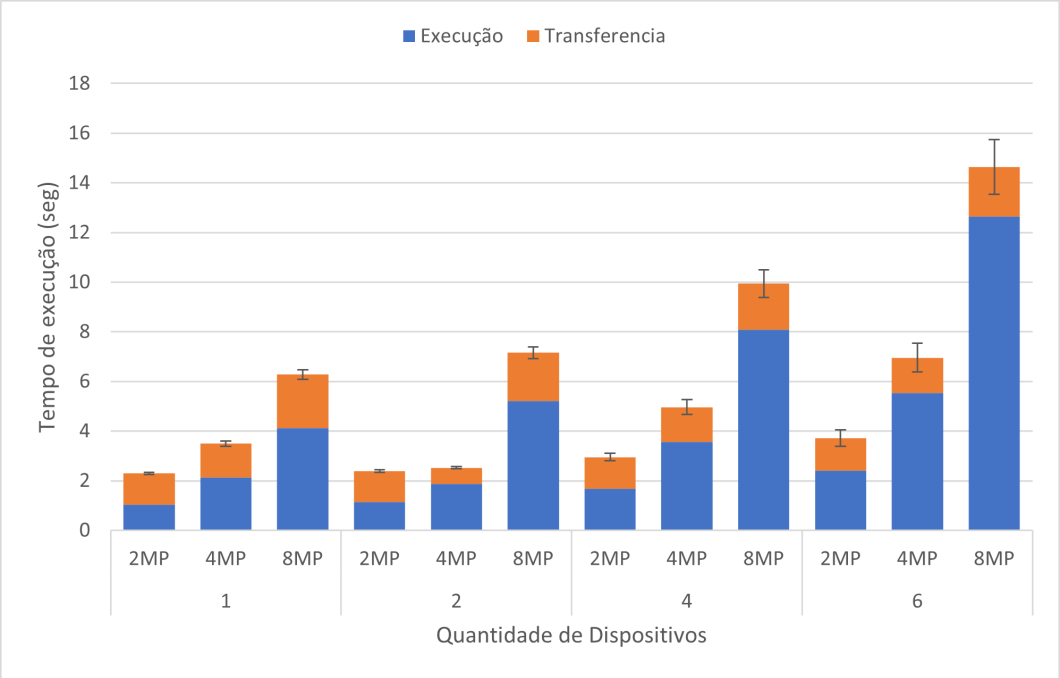
Figura 22 – Tempo de execução com topologia *WIFI*




Fonte: Elaborada pelo autor

também aumenta o tempo de execução e transferência(*upload* e *download*). Em comparação entre o tempo de execução na topologia de rede *WIFI* e *Cloudlet*, notamos que o tempo de execução são semelhantes, mas o tempo de transferência, maior na rede *WIFI*, sendo em alguns casos, até o dobro do tempo. Devemos considerar, na topologia *WIFI*, os dados saem do dispositivo, passando pelo *Switch* e chegando ao servidor, servidor calcula processamento da imagem, e depois envia de volta para o dispositivo passando pelo *Switch*, e durante esse procedimento, toda vez que ele passa por uma conexão, ele sofre *delay*, assim, aumentando o tempo de transferência diferenciando da rede *Cloudlet*, que os dispositivos estão diretamente conectados ao servidor.

Figura 23 – Tempo de execução com topologia *Cloudlet*




Fonte: Elaborada pelo autor

## 6.3 Uso de recursos com VNC habilitado.

No terceiro estudo de caso, é elaborado um estudo sobre uso de recursos com VNC habilitado, calculando métricas em relação ao uso de *CPU* e memória da ferramenta ao utilizar VNC e sem utilizar o mesmo, com o aplicativo de matrizes (aplicativo que calcula dimensões de matrizes). No Quadro 5, mostra os fatores e níveis utilizados para avaliação desta ferramenta.

Quadro 5 – Fatores/Parâmetros e níveis do experimento

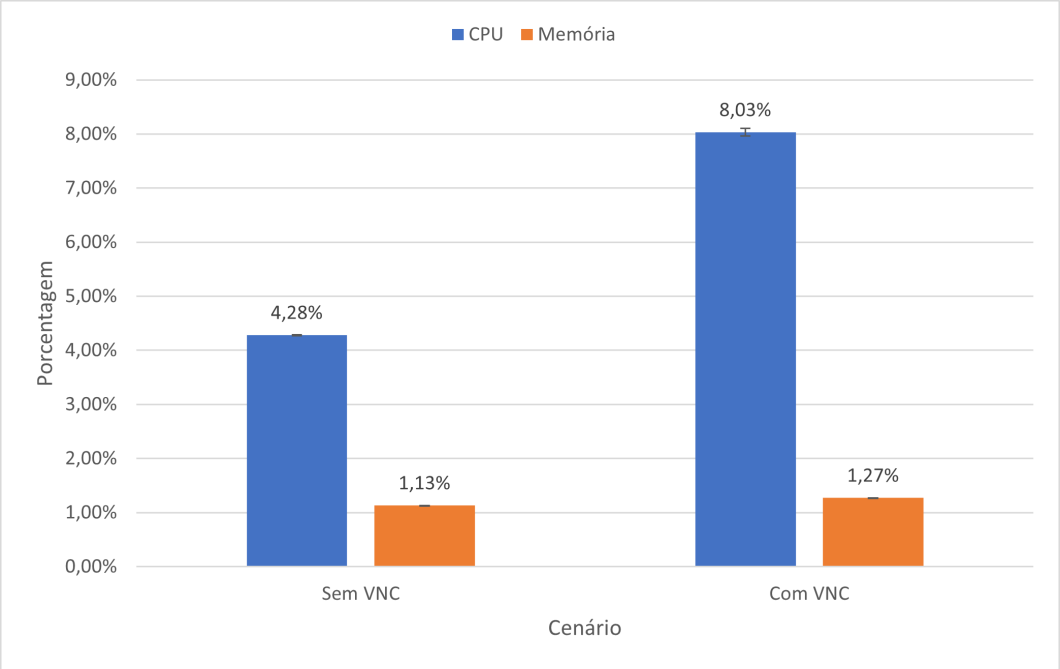
| Fator/Parâmetro | Nível |
|---|---|
| Topologia de Rede | *Cloudlet* |
| Quantidade de dispositivos móveis | 10 |
| Dimensão das matrizes utilizadas | 400x400 |
| *Delay* | 50ms |
| *Jitter* | 5ms |
| *Bandwidth* | 500Mbits |
| Aplicação utilizada | *MatrixOperation* |

Fonte: Elaborado pelo autor

Na Figura 24, é mostrado os resultados do uso de *VNC* habilitado e não habilitado com topologia de rede *Cloudlet*. Foi utilizado um *script* em *shell* para capturar o uso de CPU e memória. Foram 7408 capturas de uso de *CPU* e memória durante um intervalo de 12 minutos,

no período de execução da aplicação matrizes com dimensão de matrizes 400x400 sem o uso de VNC. Com o uso de VNC, foram 6395 capturas em um intervalo de 12 minutos e 17 segundos. Diante dessas capturas, foram realizados os cálculos de médias e um intervalo de 95%.

Figura 24 – Uso de *CPU* e memória em relação ao *VNC*




Fonte: Elaborada pelo autor

Podemos notar uma grande diferença em relação ao uso do *VNC*, o gráfico demonstra que o uso do *VNC* habilitado é quase o dobro de *CPU* de quando não está habilitado. Nos resultados de memória, houve pequenas variações entre os dois cenários, pouco significativas.

## 6.4 Limitações dos Experimentos.

Neste trabalho, foram feitos apenas três estudos de caso, e com algumas diferenças de configurações, isso se deu, por conta do limite de recursos presente na máquina utilizada para realizar os experimentos. Além dos experimentos realizados, ainda podem ser feito diversos experimentos com a ferramenta desenvolvida, como executar uma aplicação *Android* com mais dispositivos, alternar configurações de rede para diferentes topologias, etc.

# 7 CONCLUSÕES E TRABALHOS FUTUROS

A ferramenta *MCC Testbed* foi criada com a necessidade de auxiliar desenvolvedores e pesquisadores de MCC a criar cenários e executar experimentos personalizados de *offloading*, no entanto, apesar de ter suprido a necessidade, a criação e execuções cenários ainda é fornecida via CLI. No contexto onde *MCC Testbed* se encontra, este trabalho teve como objetivo desenvolver uma aplicação *web* utilizando *ReactJS* para melhorar e simplificar o processo de criação e execução de cenários personalizados em *offloading* utilizando a API da ferramenta *MCC Testbed*.

A aplicação *web* criada ao longo deste trabalho atende todos os requisitos funcionais e não funcionais abordados na seção 4.2, no Capítulo 4, visando o atendimento para comunidade, os objetivos propostos. A solução é disponibilizada no repositório *tricksil/mcc-testbed-ui*[1] e API da ferramenta *MCC Testbed* é disponibilizada no repositório *alvesRenan/containernet-androidTestbed*[2].

Após implementada a ferramenta, foram realizados experimentos com 3 casos de usos, no primeiro caso, criação de cenário, obteve resultados de tempo de criação de cenário, que houve uma pequena variação na criação dos cenários e o uso de CPU e memória, de forma, que também houve pequenas variações no uso de recursos da máquina. No segundo caso, execução de cenário, obtemos os resultados de tempo de execução e tempo de transferência, foi notado, de acordo que a quantidade de dispositivos aumenta, também aumenta o tempo de execução e tempo de transferência. Por último, o terceiro caso, uso de recursos com VNC habilitado, os resultados obtidos demonstram que o uso *CPU* e memória com VNC e quase o dobro quando não utiliza VNC. Isso demonstra ser possível utilizar a ferramenta substituindo a atual interface via CLI.

Como trabalhos futuros, seria interessante avaliar experiência de usuário ao utilizar a ferramenta desenvolvida, atualmente, apenas foi avaliado o desempenho. Adicionar uma paginação para visualizar mais dispositivos via VNC, atualmente só é possível ver apenas uma conexão com o cliente. Salvar dados do formulário de configuração de execução de aplicação *Android*, pois atualmente, ao executar uma aplicação, ele apaga os dados do formulário. Adicionar gráficos com intervalo de confiança sobre os resultados dos *logs*. Criar funcionalidades de cálculo de uso de *CPU* e memória na aplicação e realizar contagem de tempo na criação e execução de cenário adicionando como métricas na ferramenta e demonstrando esses dados em gráficos.

[1] https://github.com/tricksil/mcc-testbed-ui
[2] https://github.com/alvesRenan/containernet-androidTestbed

# REFERÊNCIAS

AKHERFI, K.; GERNDT, M.; HARROUD, H. Mobile cloud computing for computation offloading: Issues and challenges. **Applied Computing and Informatics**, p. 1–16, 2016.

BARBOSA, R. A.; REGO, P. A. L. A mobile cloud computing testbed based on lightweight virtualization. **IEEE**, Fortaleza, Ceará, Brazil, p. 1–6, 2019.

BISWAS, M.; WHAIDUZZAMAN, M. Efficient mobile cloud computing through computation offloading. **International Journal of Advancements in Technology**, p. 1–7, 2019.

CATAL, F.; TCHOLTCHEV, N.; HöFIG, E.; HOFFMANN, A. Visualization of traffic flows in a simulated network environment to investigate abnormal network behavior in complex network infrastructures. **The 10th International Conference on Ambient Systems, Networks and Technologies (ANT 2019) / The 2nd International Conference on Emerging Data and Industry 4.0 (EDI40 2019) / Affiliated Workshops**, Berlin, Germany, p. 279–287, 2019.

DONCHEVA, N. T.; MORRIS, J. H.; GORODKIN, J.; JENSEN, L. J. Cytoscape stringapp: Network analysis and visualization of proteomics data. **American Chemical Society Publications**, Copenhagen, Dinamarca, p. 623–632, 2018.

FACEBOOK. **Introdução aos Hooks**. 2019. Portal Corporativo. Disponível em: https://pt-br.reactjs.org/docs/hooks-intro.html . Acesso em: 13 abril, 2021.

KHALID, A.; QUINLAN, J.; SREENAN, C. Mininam: A network animator for visualizing real-time packet flows in mininet. **Conference on Innovations in Clouds, Internet and Networks (ICIN)**, Cork, Ireland, p. 229–231, 2017.

RAJA, C. V.; CHITRA, K.; JONAFARK, M. A survey on mobile cloud computing. **National Science Foundation (NSF)**, p. 2096–2100, 2018.

RAWAT, P.; MAHAJAN, A. N. Reactjs: A modern web development framework. **International Journal of Innovative Science and Research Technology**, p. 698–702, 2020.

RUBYGARAGE. **The Best JS Frameworks for Front End**. 2020. Portal Corporativo. Disponível em: https://rubygarage.org/blog/best-javascript-frameworks-for-front-end . Acesso em: 14 abril. 2021.

SOUSA, M.; GONçALVES, A. humanportal – um caso de estudo usando react.js. **Iberian Conference on Information Systems and Technologies (CISTI)**, Tucson Arizona, United States. 2012, p. 1–6, 2020.

STACKOVERFLOW. **2021 Develop Survey**. 2021. Portal Corporativo. Disponível em: https://insights.stackoverflow.com/survey/2021#overview . Acesso em: 05 maio, 2022.

THAKKAR, M. **Building React Apps with Server-Side Rendering**. Vadodara, Gujarat, India: Apress, 2020.

VAIDYA, S.; SHAH, N.; VIRANI, K.; DEVADKAR, P. K. A survey: Mobile cloud computing in education. **Proceedings of the Fifth International Conference on Communication and Electronics Systems (ICCES 2020)**, Mumbai, India. 2020, p. 655–659, 2020.

WU, H. Multi-objective decision-making for mobile cloud offloading: A survey. **IEEE Access**, p. 3962–3976, 2018.

ZHOU, B.; BUYYA, R. Augmentation techniques for mobile cloud computing: A taxonomy, survey, and future directions. **ACM Computing Surveys**, Melbourne, Australia. 2018, p. 1–38, 2018.