PUCRS

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
PROGRAMA DE PÓS-GRADUAÇÃO
CIÊNCIA DA COMPUTAÇÃO

BRUNO SCHMIDT MARQUES

IMPLEMENTAÇÃO E AVALIAÇÃO DE MODELOS DE MOBILIDADE DE DISPOSITIVOS NO SIMULADOR MYIFOGSIM

Porto Alegre
2019

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

IMPLEMENTAÇÃO E AVALIAÇÃO DE MODELOS DE MOBILIDADE DE DISPOSITIVOS NO SIMULADOR MYIFOGSIM

BRUNO SCHMIDT MARQUES

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Tiago Ferreto

Porto Alegre
2019

Ficha Catalográfica

M357i Marques, Bruno Schmidt

Implementação e avaliação de modelos de mobilidade de dispositivos no simulador MyiFogSim / Bruno Schmidt Marques . – 2019.

68 p.

Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Tiago Coelho Ferreto.

1. Fog Computing. 2. Simulador. 3. Redes. 4. MyiFogSim. 5. Edge Computing. I. Ferreto, Tiago Coelho. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS com os dados fornecidos pelo(a) autor(a).

Bibliotecária responsável: Salete Maria Sartori CRB-10/1363

Bruno Schmidt Marques

Implementação e avaliação de modelos de mobilidade de dispositivos no simulador MyiFogSim

Tese/Dissertação apresentada como requisito parcial para obtenção do grau de Doutor/Mestre em Ciência da Computação do Programa de Pós-Graduação em Ciencia da Computação, Escola Politécnica da Pontifícia Universidade Católica do Rio Grande do Sul.

Aprovado em 28 de março de 2019.

BANCA EXAMINADORA:

Prof. Dr. Jorge Luis Victória Barbosa (PIPCA/UNISINOS)

Prof. Dr. Fabiano Passuelo Hessel (PPGCC/PUCRS)

Prof. Dr. Tiago Coelho Ferreto (PPGCC/PUCRS - Orientador)

IMPLEMENTAÇÃO E AVALIAÇÃO DE MODELOS DE MOBILIDADE DE DISPOSITIVOS NO SIMULADOR MYIFOGSIM

RESUMO

A Computação em Névoa (*Fog Computing*) propõe uma arquitetura capaz de flexibilizar e otimizar aplicações distribuídas, hospedando as mesmas mais próximas de seus consumidores. Entretanto, a dificuldade de acesso a ambientes reais de *Fog Computing* faz com que pesquisadores utilizem simuladores para desenvolver novas contribuições nessa área. Uma das principais deficiências encontradas nos simuladores de *Fog Computing* atuais é a simplificação do modelo de mobilidade dos dispositivos. Este trabalho apresenta a implementação de modelos de mobilidade para dispositivos móveis em ambientes de *Fog Computing* por meio da extensão do simulador *MyiFogSim*. Além disso, é analisado o impacto que diferentes modelos de mobilidade têm sobre a escalabilidade e latência das aplicações, e também sobre o escalonador de máquinas virtuais do *MyiFogSim*.

Palavras-Chave: *Fog Computing*, modelos de mobilidade, simulação, MyiFogSim.

IMPLEMENTATION AND EVALUATION OF DEVICE MOBILITY MODELS ON THE MYIFOGSIM SIMULATOR

ABSTRACT

Fog Computing proposes an architecture capable of flexibilizing and optimizing distributed applications, hosting them closer to its consumers. However, the difficulty of accessing real *Fog Computing* environments causes researchers to use simulators to develop new contributions in this area. One of the main shortcomings found in the current 'Fog Computing' simulators is the simplification of the device mobility model. This work presents the implementation of mobility models for mobile devices in *Fog Computing* environments through the extension of the *MyiFogSim* simulator. In addition, the impact of different mobility models on the scalability and latency of applications, as well as the *MyiFogSim* virtual machine scheduler, is analyzed.

Keywords: Fog Computing, mobility models, simulation, MyiFogSim.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

1. INTRODUÇÃO

A Internet das Coisas (*Internet of Things - IoT*) consiste num paradigma computacional que promete conectar bilhões de novos dispositivos à Internet nos próximos anos. Estes dispositivos têm capacidade de processamento e armazenamento limitadas devido ao seu tamanho físico e por questões de portabilidade. Assim, muitos deles utilizam serviços e aplicações de Computação em Nuvem para superar essa limitação, expandindo suas capacidades (AaJD+16, BMZA12, AIM10).

Entretanto, a distância entre os recursos de nuvem e os dispositivos de IoT implica em uma alta latência de comunicação, o que inviabiliza a implementação de certas classes de aplicações neste tipo de arquitetura, tais como aplicações de tempo-real (AaJD+16). A Computação em Névoa (*Fog Computing*) é um paradigma computacional que surge como uma abordagem para resolver este e outros problemas, por meio da oferta de recursos computacionais de forma descentralizada, permitindo a disponibilização de aplicações e serviços próximos aos seus usuários (BMZA12).

A disponibilidade de ambientes reais para testes é pequena. Tal restrição limita não somente as pesquisas sobre esse novo paradigma, mas também o desenvolvimento de aplicações e serviços mais eficientes para esse tipo de ambiente (MGG+17). Sendo assim, diversos pesquisadores utilizam simuladores para desenvolvimento e avaliação de soluções para ambientes de *Fog Computing* (AGC17). Estes simuladores apresentam limitações, especialmente no que tange a mobilidade de dispositivos. Alguns trabalhos utilizam modelos de mobilidade limitada para avaliação de proposta de escalonadores de máquinas virtuais, o que pode impactar no desempenho das aplicações (BDB+17).

Neste contexto, este trabalho utiliza como base o simulador *MyiFogSim* (LHCB17) para implementação do seguinte conjunto de modelos de mobilidade de dispositivos móveis: *Random Waypoint*, *Reference Point Group*, *Nomadic Community*, *Pursue* e *Obstacles*. Além disso, a escalabilidade e latência de uma aplicação foi analisada com os modelos implementados e os resultados comparados com os do modelo de direção e velocidade fixas, apresentados em (BH04).

As contribuições deste trabalho são:

1. Implementação e validação de uma nova arquitetura interna no simulador *MyiFogSim* através da implementação dos modelos de mobilidade *Random Waypoint*, *Reference Point Group*, *Nomadic Community*, *Pursue* e *Obstacles*;

2. Avaliação dos modelos de mobilidade utilizando algoritmos para escalonamento de serviços em ambientes de *Fog Computing*;

3. Ferramenta para visualização do padrão de movimentação dos dispositivos no ambiente de *Fog Computing*;

4. Análise do impacto de modelos de mobilidade no escalonador do *MyiFogSim*.

Este trabalho está organizado da seguinte maneira. O Capítulo 2 apresenta a fundamentação teórica, contextualizando *Fog Computing*, simuladores computacionais e modelos de mobilidade. O Capítulo 3 elenca os trabalhos relacionados ao tema, isto é, simuladores e emuladores aplicados ao contexto de *Fog Computing*. O Capítulo 4 detalha a implementação da modelagem e protótipo. Por fim, o Capítulo 5 especifica a metodologia utilizada, apresenta e análise os resultados obtidos. As conclusões e trabalhos futuros são apresentadas no Capítulo 6.

2. FUNDAMENTAÇÃO TEÓRICA

Este Capítulo descreve brevemente a fundamentação teórica das seguintes áreas: (i) Internet das Coisas (IoT), (ii) Computação em nuvem, (iii) Computação em névoa, (iv) Simuladores e emuladores e (v) Modelos de mobilidade.

2.1 Internet das Coisas (IoT - *Internet of Things*)

A Internet das Coisas é uma área que envolve engenharia e computação para que objetos, referenciados como coisas, sejam conectados o tempo todo e estejam sempre acessíveis em qualquer lugar (AIM10). Isto define uma ideia de presença constante dos objetos, tais como telefones celulares, marcadores (*tags*) e leitores, numa rede virtual, onde trocam informações entre si a fim de atingir um objetivo comum. É esperado que bilhões de novos dispositivos conectem-se à Internet das Coisas nos próximos anos. Alguns pesquisadores apontam que o número deve dobrar a cada ano durante a próxima década (Con17b, GRdC14, Con17a).

O surgimento de tecnologias de comunicação sem fio permitiram que ciência e indústria desenvolvessem sensores sem fio de tamanho reduzido. Esse avanço foi essencial para o surgimento de diversas aplicações da Internet das Coisas, especialmente nas áreas de transporte e logística, saúde e ambientes inteligentes (SHFW10). Alguns destes dispositivos já conectam-se diretamente a Internet, o que pode modificar o enfoque de conexão e reduzir uma série de *middlewares* utilizados (WV16).

Assim como a Internet, dispositivos de IoT móveis como *smartphones* e *tablets* têm um papel importante e fazem parte do nosso cotidiano. Entretanto, esses dispositivos possuem capacidade limitada para atingir o grau de mobilidade desejado pelos seus usuários, então eles utilizam recursos de Computação em Nuvem) para expandir suas capacidades (AaJD+16, MKB18, OSMF17). Assim, surgiu *Mobile Cloud Computing* (MCC), que estuda como permitir que os aparentes recursos infinitos da computação em nuvem estejam disponíveis aos dispositivos móveis de maneira elástica. Na prática, entretanto, o MCC depende de cenários raros onde a conexão de rede é rápida e estável (OBL15).

2.2 Computação em Nuvem

A Computação em Nuvem é um modelo de arquitetura que permite acesso sob demanda a diversos recursos de computação, tais como redes, servidores, espaço de armazenamento, aplicações e serviços, por meio da Internet (MG11). Os *datacenters* de *Cloud*

Computing proveem recursos virtualmente ilimitados utilizando o modelo de cobrança "*pay-as-you-go*". Segundo o NIST(*National Institute of Standards and Technology*) o modelo de computação em nuvem é composto por cinco características principais, três modelos de serviço e quatro modelos de implantação (*deployment*) (MG11). As cinco principais características do modelo de Computação em Nuvem são:

- **Acesso sob demanda:** o consumidor pode, de maneira unilateral, provisionar sistemas computacionais como servidores de aplicação ou de armazenamento de dados conforme a sua necessidade sem interagir com outro humano.
- **Amplo acesso à Internet:** os sistemas computacionais provisionados estão disponíveis através de mecanismos comuns e padronizados de acesso a Internet, promovendo assim um acesso heterogêneo, que vai desde dispositivos simples(*smartphones*, *tablets*) a sistemas computacionais complexos (notebooks, *workstations*).
- **Agrupamento de recursos:** os recursos computacionais físicos do provedor são disponibilizados aos clientes em um modelo multi-inquilino, isto é, múltiplos clientes compartilham o mesmo *hardware* físicos. A alocação do *hardware* designado a cada cliente é feita de maneira dinâmica e pode ser refeita conforme a demanda. Não existe senso de independência de localização, já que o cliente não possui conhecimento preciso quanto a localização que a sua infraestrutura está sendo processada. Algumas nuvens podem disponibilizar uma estrutura de alto nível de abstração, como o país ou o *datacenter* que os serviços vão utilizar.
- **Elasticidade:** a capacidade dos recursos alocados pode aumentar ou diminuir rapidamente, e em alguns casos essa progressão pode ser automática. Do ponto de vista do cliente os recursos são ilimitados e podem ser alocados a qualquer momento.
- **Medição dos serviços:** serviços de computação em nuvem são medidos de diferentes maneiras, dependendo do tipo de serviço que está sendo oferecido, como processamento, armazenamento, largura de banda, ou atividade por uso de contas. Os recursos podem ser monitorados, medidos e podem gerar relatórios, criando transparência para ambos cliente e provedor de serviço.

Uma característica comum em infraestrutura de nuvem tradicional é que a mesma seja centralizada, havendo um número reduzido de *datacenters*, mas cada um com muita capacidade computacional e de armazenamento. Essa abordagem centralizada implica em uma alta latência de comunicação entre os recursos da nuvem e os dispositivos dos usuários consumidores.

A característica de centralização de infraestrutura inviabiliza algumas aplicações de se beneficiarem da computação em nuvem. Algumas das características que estas aplicações necessitam são: (*i*) baixa latência, (*ii*) grande capacidade computacional próxima

aos dispositivos que a necessitam, e (*iii*) questões relacionadas a segurança das informações trafegadas na rede (CZ16).

2.3 Computação em Névoa (*Fog Computing*)

A *Fog Computing* tenta disponibilizar os recursos solicitados na extremidade lógica (borda) da rede. Esta abordagem apresenta como grande desafio o desenvolvimento de aplicações em um cenário cuja topologia de rede não é confiável e muda constantemente (OBL15). O ambiente de *Fog Computing* se caracteriza por baixa latência, proximidade dos servidores aos clientes finais, largura de banda abundante para transferência de dados e consciência em tempo real de informações da rede e localidade (HPS+15).

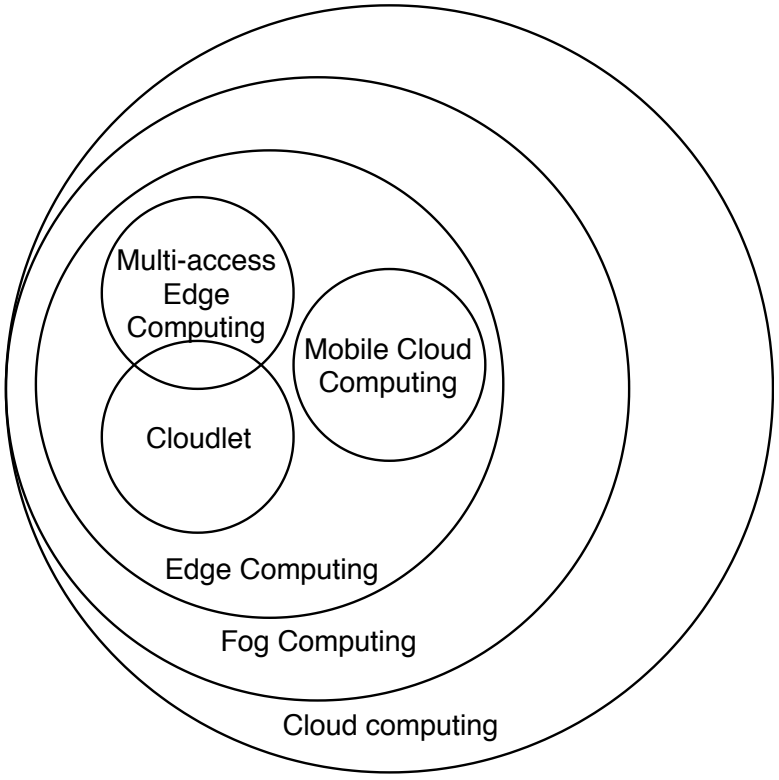


Figura 2.1: Classificação do escopo de *Fog Computing* e de paradigmas computacionais relacionados (elaborado pelo autor).

Existem diversos paradigmas computacionais relacionados a *Fog Computing*, tais como *Edge Computing*, *Mobile Cloud Computing* (MCC) e *Multi-access Edge Computing* (MEC). *Edge Computing* é um conjunto de tecnologias que permitem que parte da computação seja realizada na borda da rede, isto é, próximo as fontes produtoras e consumidoras de dados (usuários), mas não está associada a computação móvel. Já a MCC está relacionada a computação móvel, mas não faz uso das tecnologias de *Edge Computing*. *Cloudlets* são réplicas de *datacenters* de *Cloud Computing* em tamanho reduzido que são hospedados na borda da rede, sendo uma das tecnologias utilizadas pela *Edge Computing*, mas que

não possui relação com a computação móvel. O escopo da *Fog Computing* abrange tanto as tecnologias de disponibilização de recursos da *Edge Computing* próximos aos usuários quanto a computação destes dados nos dispositivos finais, tal como a MCC (YFN+18). A *Fog Computing* se caracteriza, portanto, em uma estrutura hierárquica e dinâmica, que utiliza tecnologias de *Edge Computing* para disponibilizar os recursos próximos aos usuários e realizar parte da computação nos dispositivos móveis de maneira semelhante a MCC. A Figura 2.1 representa os escopos de cada paradigma computacional e as suas respectivas relações.

A *Fog Computing* pode lidar com tarefas de requisições de rede, processamento computacional, armazenamento, cache e balanceamento de carga, trazendo benefícios como a diminuição no tempo de resposta e redução de consumo de energia de 30 a 40% (LGL+15).

2.4 Simulação e emulação

A simulação é uma técnica utilizada para análise de desempenho de sistemas computacionais. Se o sistema a ser analisado não está disponível, o que ocorre frequentemente em estágios iniciais do desenvolvimento do mesmo, um modelo de simulação disponibiliza um método fácil para estimar ou comparar diversas alternativas. Em alguns casos, mesmo que o sistema esteja disponível, a simulação é capaz de medir uma gama mais ampla de ambientes e cargas de trabalho (Jai91).

A simulação de um *hardware* ou *firmware* é chamada de emulação (Jai91). São exemplos de emuladores: os emuladores de terminal, que simulam um tipo de terminal em outro terminal, e os emuladores de processadores, que traduzem as instruções de um tipo de processador para outro. Os emuladores limitam-se a simular um determinado hardware em outro, enquanto os simuladores possuem diferentes modelos de operação, tais como (i) simulação Monte Carlo, (ii) *trace-driven* e (iii) simulação por eventos discretos.

2.4.1 Simulação Monte Carlo

A simulação Monte Carlo utiliza um modelo probabilístico cujo fenômeno não se altera com o passar do tempo. Por isso, a simulação Monte Carlo também é descrita como "simulação estática"(Jai91). Ainda assim, este tipo de simulação requer a utilização de números pseudo-aleatórios e pode ser utilizada para avaliar expressões não probabilísticas com métodos probabilísticos.

2.4.2 Simulação *trace-driven*

A simulação *trace-drive* diferencia-se da simulação Monte Carlo por utilizar um conjunto de dados dos eventos ocorridos em um sistema real (rastro) como entrada para o modelo de simulação. Tais simuladores normalmente são utilizados para análise ou aperfeiçoamento (*tuning*) do sistema em questão a ser analisado, como por exemplo um algorítimo computacional (Jai91). As simulações *trace-driven* frequentemente analisam eventos de *caching*, pilha de processos, trocas de contexto da CPU e *deadlock* de recursos.

2.4.3 Simulação por eventos discretos

Uma simulação por eventos discretos utiliza um modelo de estados dos sistema. Essa condição a torna oposta às simulações de eventos contínuos, na qual o estado do sistema atinge valores contínuos. Um exemplo de sistema de valores contínuos é a análise de uma reação química de alguma substância, onde o valor desejado pode ser a concentração desta determinada substância em alguma ambiente específico. O termo "discreto"refere-se aos estados possíveis dos elementos do sistema, tais como a quantidade de tarefas em diversos dispositivos, e não quanto ao tempo de execução (Jai91). Em uma simulação por eventos discretos, o tempo pode ou não ser contínuo. Todas as simulações por eventos discretos possuem uma estrutura semelhante formada pelos seguintes componentes:

- **Agendador de tarefas:** mantém uma lista de eventos aguardando para serem disparados. O agendador permite que os eventos sejam manipulados de diversas maneiras.
- **Simulador de relógio e mecanismo de avanço do tempo:** todas as simulações discretas precisam de uma variável global que representa o tempo do sistema. Essa variável é controlada pelo agendador de tarefas ou pelo mecanismo de avanço do tempo. Existem duas abordagens principais: (i) avanço por uma unidade de tempo ou (ii) avanço pela ocorrência de um evento.
- **Variáveis de estado do sistema:** representam o estado global do sistema, como a quantidade de processos em uma CPU ou a quantidade total de memória sendo utilizada pelo sistema.
- **Rotinas de eventos:** cada evento é simulado pela sua própria rotina. Tais rotinas atualizam as variáveis de estado do sistema, o agendador de tarefas, e outras tarefas.
- **Rotinas de entrada:** representam os parâmetros de entrada do sistema, como demanda de CPU por processo. É comum que estes parâmetros sejam solicitados ao usuário no início da simulação e não durante a execução da mesma.

- **Gerador de relatórios:** são rotinas que geram dados de saída ao fim da simulação. Elas calculam os dados desejados e exportam os mesmos no formato configurado.
- **Rotinas de inicialização:** são responsáveis por definir o estado inicial da simulação, variáveis e números aleatórios a serem utilizados.
- **Rotinas de rastreamento:** são rotinas que geram dados de saída durante a execução da mesma, e servem para rastreamento e análise do sistema.
- **Gerenciamento dinâmico de memória:** durante uma simulação diversas entidades são criadas e destruídas. Portanto, é necessário um mecanismo de alocação e liberação de memória para execução eficiente da simulação.
- **Programa principal:** consiste no módulo responsável por orquestrar todas as rotinas juntas. Ele calcula as rotinas de inicialização, inicia a simulação, executa inúmeras interações e por fim, chama as rotinas de saída.

2.5 Modelos de mobilidade

Modelos de mobilidade são projetados para descrever o padrão de movimentação de usuários móveis, isto é, como sua localização, velocidade e aceleração modificam-se com o passar do tempo. Os padrões de mobilidade são relevantes para a determinação de desempenho, por isso, é desejado que modelos de mobilidade emulem o padrão de aplicações reais de maneira satisfatória, caso contrário, as observações e conclusões realizadas a partir das simulações podem ser equivocadas (BH04). Os modelos de mobilidade podem ser divididos em: aleatórios, com dependência temporal, com dependência espacial, e com restrições geográficas.

2.5.1 Modelos aleatórios

Os modelos de mobilidade aleatórios possuem como característica a movimentação libre e sem restrições dos nós. Parâmetros como destino, velocidade e direção são escolhidos aleatoriamente, e não dependem do destino, velocidade ou direção de outros nós. O modelo "*Random Waypoint*"proposto por Maltz (BMJ+98) é amplamente utilizado para simulações devido a sua simplicidade e disponibilidade (BEF+00).

Os principais parâmetros do modelo *Random Waypoint* são: (i) *VMax* e)ii) *TPause*. *VMax* é a velocidade máxima que um nó pode se movimentar, *TPause* é o tempo de espera a cada nova iteração. Se *VMax* for baixo e *TPause* for alto, a topologia tende a manter-se **estável**. Se *VMax* for alto e *TPause* for baixo, a topologia será altamente **dinâmica**.

O modelo aleatório *Random Waypoint* possui um problema, o *speed decay*. Estudos realizados em 2003 mostraram que este modelo é incapaz de atingir um estado fixo em termos de mobilidade, e com *TPause*=0 a velocidade média dos nós constantemente decai com o tempo (YLN03). O mesmo grupo de pesquisadores concluiu que este problema não é exclusivo do *Random Waypoint*, ocorrendo em todos os modelos aleatórios de mobilidade.

O modelo *Random Direction* foi proposto por Royer (BH04) para suplantar a distribuição espacial não uniforme dos nós. Ele se difere do *Random Waypoint* na seleção de destino. Ao invés de selecionar um destino aleatório a cada intervalo de tempo, o modelo define uma direção para o nó, que se moverá até que encontre uma fronteira. Após isso, o nó aguarda pelo tempo *TPause* e aleatoriamente seleciona outra direção.

2.5.2 Modelos com dependência temporal

Modelos com dependência temporal limitam o movimento dos nós. A velocidade atual de cada nó depende da sua velocidade anterior, e a velocidade de um único nó em diferentes espaços de tempo estão correlacionados. Os modelos aleatórios *Random Waypoint* e *Random Direction* são inadequados para reproduzir essa dependência temporal porque os nós não possuem informações históricas.

Gauss-Markov é um modelo amplamente utilizado. Neste modelo, a velocidade de um nó móvel é pressuposto como um valor co-relacionado ao longo do tempo e modelado como um processo estocástico Gauss-Markov. Quando um nó se movimenta para além dos limites do campo de simulação a direção, sua direção é rotacionada em 180 graus.

Smooth random é um modelo que implementa a dependência temporal através de funções de suavização, prevenindo assim movimentos não realistas. Mudanças bruscas de direção ou velocidade, e acelerações ou desacelerações súbitas passam a ser suavemente incrementadas, criando assim uma dependência com o estado anterior, portanto uma dependência temporal.

2.5.3 Modelos com dependência espacial

Em modelos com dependência espacial os comportamento dos nós dependem de outros nós, isto é, a velocidade de um nó pode influenciar na velocidade de outros nós. Nestes modelos é comum que ocorra uma colaboração entre nós ou que eles sigam um líder. Há uma correlação no espaço que estes nós ocupam, portanto há uma dependência espacial da velocidade.

Reference Point Group é um modelo que simula o comportamento de grupos onde o movimento de alguns nós tende a coordenar o movimento de outros. Este modelo pode ser usado para simular soldados em um grupo, por exemplo. Cada nó deriva o seu movimento do nó líder em algum nível.

Sets of correlated nodes foi proposto por Sanchez (SM01) e consiste em um grupo de nós móveis cuja movimentação é obtida colaborativamente. Para isso podem ser utilizadas técnicas de colunas, onde existe uma direção fixa de varredura, ou grupos nômades de movimentação.

2.5.4 Modelos com restrições geográficas

Modelos com restrições geográficas sujeitam os nós a regras do ambiente. Por exemplo, o movimento de veículos está sujeito as regras do ambiente de ruas e avenidas, assim como pedestres podem ter a sua passagem obstruída por prédios e outros obstáculos. Os modelos que vinculam a velocidade, direção ou destino a um mapa geográfico são chamados de modelos com restrições geográficas.

Pathway é um modelo que restringe o movimento dos nós a caminhos pré-definidos no mapa. Os caminhos possíveis podem ser descritos como um grafo, onde os vértices representam os prédios de uma cidade e as arestas representam as ruas e os possíveis caminhos entre os prédios.

O modelo de obstáculos não estabelece caminhos possíveis, mas sim define quais são os obstáculos de um mapa. Ao movimentar-se os nós evitam estes obstáculos, tendo a sua trajetória alterada quando se aproximam de algum obstáculo (BH04).

2.6 Considerações finais

Este capítulo apresentou questões relacionadas a Internet das Coisas, *Cloud Computing*, *Fog Computing*, simuladores e emuladores e modelos de mobilidade. É possível observar que a mobilidade é um aspecto fundamental da IoT e, por consequência, da *Fog Computing*. É esperado, portanto, que simuladores e emuladores voltados a este paradigma computacional contemplem esta característica. O Capítulo 3 apresenta trabalhos relacionados a emuladores e simuladores para *Fog Computing*.

3. TRABALHOS RELACIONADOS

Os trabalhos apresentados neste capítulo são emuladores e simuladores que se relacionam com a avaliação de aplicações para *Fog Computing*. São listadas ferramentas para avaliar ambientes de *Fog Computing* sob diferentes condições. Diversos autores (OBL15, CD16, JBC09, GDGB16, GNCG16, MGG+17, NKI+15, AaJD+16, SG16, ABJAA+17) propõem ferramentas de simulação ou utilizam ferramentas consolidadas para avaliação de ambientes de rede tradicionais, como *Cloud Computing*, para avaliar uma rede de *Fog Computing*. Apesar disso, a *Fog Computing* ainda carece de melhorias nas ferramentas utilizadas para a validação de ambientes e/ou aplicações (MKB18).

3.1 EmuFog

O *EmuFog* é um emulador de computação em névoa desenvolvido em Java, *Docker* e *Mininet*. Os autores (MGG+17) trazem a necessidade de um emulador específico para computação em névoa a partir da avaliação de dois emuladores de redes tradicionais, o *CORE Network Emulator* e o *MaxiNet*. O *MaxiNet* é uma extensão do *Mininet* com foco em simulação de uma rede distribuída. Segundo os autores, ambos emuladores deixam a desejar quanto aos recursos necessários pela *Fog Computing*.

O *EmuFog* é uma extensão do *MaxiNet* com foco em *Fog Computing*. As principais contribuições do trabalho são: o gerador de topologia de rede distribuída, as funções de transformação da topologia, as funções de otimização da topologia, e a possibilidade de *deploy* e execução de código real através do *Docker*. A Figura 3.1 apresenta a arquitetura interna do *EmuFog*.

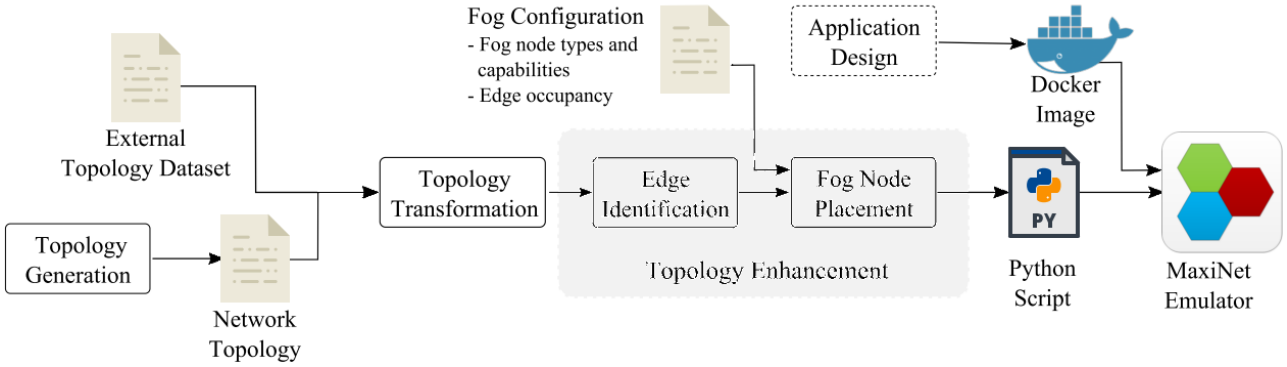


Figura 3.1: Arquitetura interna EmuFog (MGG+17)

Dentre as limitações apontadas pelos autores neste artigo está a ausência de um modelo de mobilidade. Segundo Mayer (MGG+17), a adição do modelo de mobilidade dará

mais utilidade ao emulador. O código dos autores está disponível na plataforma GitHub [1], mas não foram encontrados outros trabalhos acadêmicos que referenciem o uso dele para validação de algum ambiente ou para proposta de novas funções a *Fog Computing*.

3.2 SDMEC

SDMEC é um emulador para *Fog Computing* (ABJAA+17). Sua implementação utiliza os conceitos de SDS (*Software Defined Systems*) e NFV (*Network Functions Virtualization*) para criar um sistema baseado em comunicação *wireless* de *Edge Computing*.

Os autores utilizam uma pequena aplicação de gerenciamento de armazenamento distribuído para validação da ferramenta. Nesta aplicação, múltiplos clientes realizam 100 mil, 500 mil e um milhão de requisições ao ambiente de *Fog Computing*, que varia de dois a 10 nós. A Figura 3.2 apresenta a topologia da aplicação. O trabalho tem como foco a avaliação e otimização de sistemas de armazenamento distribuídos definidos por *software*.

O objetivo do *SDMEC* é prover a *Edge Computing* a nível de serviço, controlado pelo mesmo paradigma definido por *software* utilizado hoje para otimizar o provisionamento e gerenciamento de serviços de armazenamento em uma rede *wireless*. A tecnologia utilizada para implementação do *SDMEC* é o *Mininet-Wifi*, um emulador de WSDN (*Wireless Software Defined Networking*) que atua como uma extensão do *Mininet*. A implementação da *Edge Computing* contempla cenários de *Fog Computing* onde não há uma hierarquia definida em toda a rede.

3.3 SDFog

O *SDFog* é um *middleware* de *Fog Computing* baseado em SDN *Software Defined Networking*) com foco em serviços QoS (*Quality of Service*) e gerenciamento de dispositivos de borda. A motivação dos autores consiste na necessidade de banda que dispositivos IoT e de *Fog Computing* possuem, justificando o desenvolvimento dos serviços QoS (GNCG16).

De maneira similar, redes SDN resolvem de maneira elegante e eficiente os problemas relacionados a escalabilidade da rede distribuída. Portanto, a extensão da SDN em uma camada de aplicação voltada ao gerenciamento de dispositivos *Fog Computing* é, segundo os autores, uma boa abordagem.

O *SDFog* foi desenvolvido utilizando *Mininet*, *Network Namespaces* e *Open Vswitch* na plataforma *GNU/Linux*. As contribuições do trabalho são: desenvolvimento de um *middleware* orientado a serviço, extensão da SDN em uma camada de aplicação, e

[1] https://github.com/harshitgupta1337/emufog

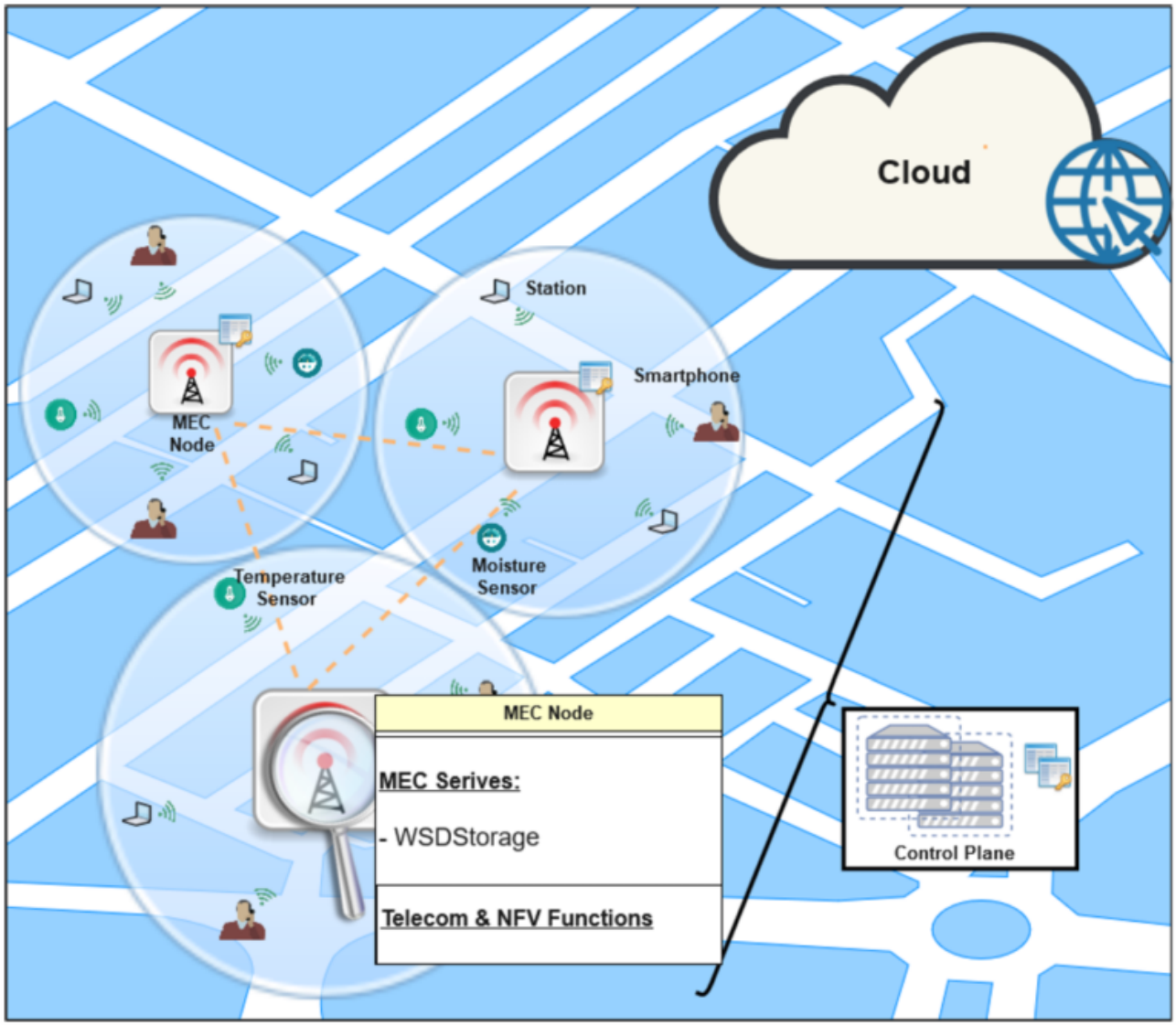


Figura 3.2: Arquitetura SDMEC (ABJAA+17)

um motor de orquestração de dispositivos distribuído. Os autores apontam algumas limitações do *middleware*, tais como a necessidade de extensão da *API southbound* para SDN, a necessidade de tornar a configuração de rede de alto nível para que aplicações na *API northbound* se conectem ao *SDFog* e solicitem tarefas com métricas simples de desempenho. A Figura 3.2 apresenta a topologia proposta pelo *SDFog*.

Além dos problemas apontados pelos próprios autores (GNCG16), o código da aplicação não está disponível para a comunidade. Portanto não há nenhum outro trabalho que utilize ou avalie o *SDFog* como plataforma de desenvolvimento ou simulação de um ambiente de *Fog Computing*, o que limita a sua aplicabilidade no mundo real.

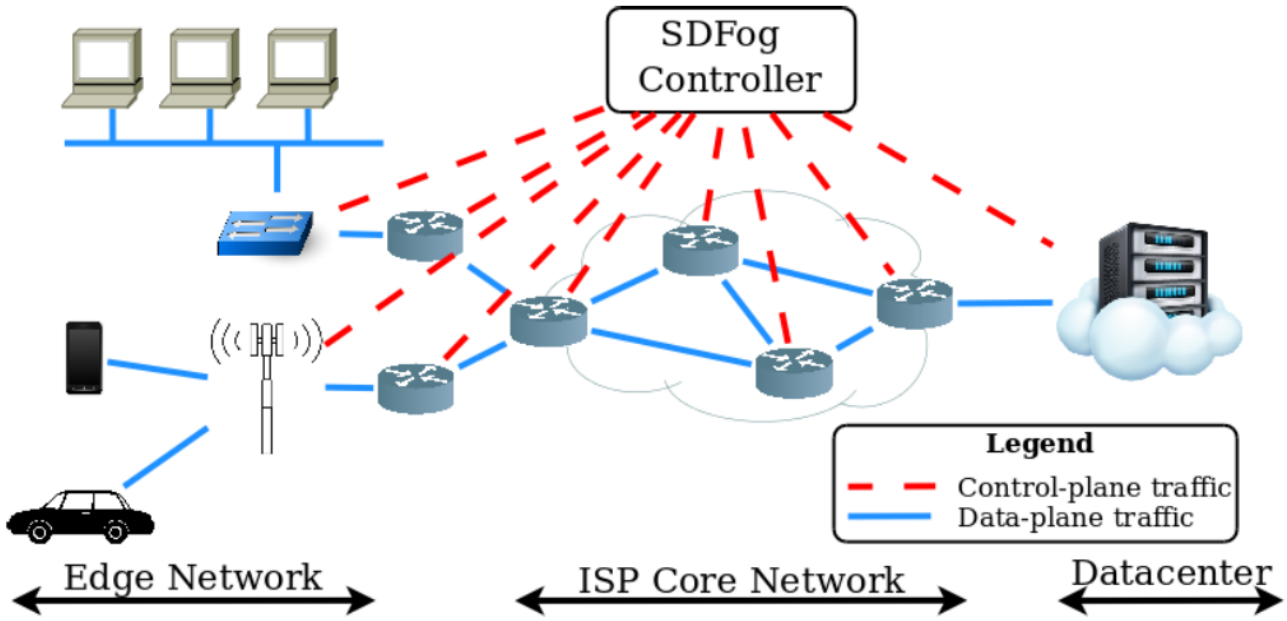


Figura 3.3: Arquitetura interna SDFog (GNCG16)

3.4 NS-3

O NS-3 (*Network Simulator* 3) é um simulador de eventos discretos que foi desenvolvido como um substituto ao simulador NS-2. Ele possui como principais recursos, em relação ao seu antecessor, a ênfase em modelos que recriam cenários mais realísticos e validações de modelos. O NS-3 é um projeto de código-aberto e é *software* livre, portanto está em constante desenvolvimento e já recebeu diversas versões desde o seu lançamento em 2008. O NS-3 é usado com frequência em pesquisas que envolvem emulação de redes, e os módulos de redes sem fio são parte integral do núcleo do arcabouço, ao contrário do NS-2 (RH10).

O módulo de redes sem fio divide-se em duas camadas: (i) MAC e (ii) PHY. Cada uma destas camadas divide-se em diversas outras, todas com encapsulamento C++. Tais camadas se assemelham a estrutura de rede IEEE 802.11 com um alto nível de detalhe, podendo configurar questões como QoS, redes Ad-hoc ou via *Access Points*, MacRXMiddle e DcaTXOp (sequenciamento de *frames*, retransmissão, filtragem de recepção duplicada), RTS/CTS, interferência de sinal conforme posicionamento e natureza dos *frames* e modelos de erros (RH10). Uma visão global de alto nível é apresentada na Tabela 3.1

Alguns modelos do NS-3 não são aplicáveis a cenários de *Edge Fog Computing*, pois aplicações não podem ser implementadas simultaneamente em *Edge* e *Cloud* (CGP17). Outra limitação existente é a ausência de um arcabouço específico para a implementação de *Fog Computing*, tais como dispositivos sensores e atuadores. Tais ausências tornam o tempo e o esforço exigido para implementação significativamente maior que em outras ferramentas (SOE17).

Test			
Helper			
Routing	Internet-stack	Devices	Applications
Node		Mobility	
Common	Simulator		
Core			

Tabela 3.1: Diagrama de módulos internos do NS-3 (elaborado pelo autor)

3.5 SimGrid

O *SimGrid* não é um simulador de redes para computação em névoa, porém a sua ampla adoção pela comunidade acadêmica o torna uma das soluções mais completas (LMC03, AGC17). Existem alguns poucos trabalhos que o utilizam para avaliar um ambiente de MEC, como (AGC17), que utiliza um simulador privado desenvolvido pelos autores (ComBoS) como ponto de partida e a MSG API do *SimGrid* como plataforma para validação de um sistema de transferência de video.

Uma das grandes limitações do *SimGrid* para uso como simulador de ambientes *Edge Computing* é a ausência de um modelo de conexão sem fio (*wireless*). Consequentemente, também não há um modelo de mobilidade aplicável a dispositivos, especialmente quando se trata de interferência de rede. Outros aspectos, no entanto, são facilmente alcançáveis, tais como heterogeneidade, possibilidade de simulação de um grande número de nós e ampla distribuição geográfica dos mesmos. A Figura 3.4 apresenta a arquitetura interna do simulador *SimGrid*.

O *SimGrid* é um framework de desenvolvimento baseado na extensão e implementação de classes de programação para a geração de uma simulação de eventos discretos, conforme apresentado na Seção 2.4.3. Legrand etal. (LMC03) avaliam que, apesar de idealmente as simulações ocorrerem em ambientes reais, na maioria das aplicações modernas isso não é possível devido a limitada oferta de recursos. Outro problema apontado pelos autores é o pequeno número de configurações da plataforma que podem ser explorados. Tipicamente, a simulação por eventos discretos é utilizada para validar estratégias de agendamento que podem ser mensuradas, comparadas e repetidas. O *SimGrid* não possui mobilidade de dispositivos ou modelos de mobilidade.

O *SimGrid* é composto pelos seguintes componentes básicos:

- **Agent:** entidade que faz as decisões de agendamento. É definido por um código, dados privados e a localização na qual executa.

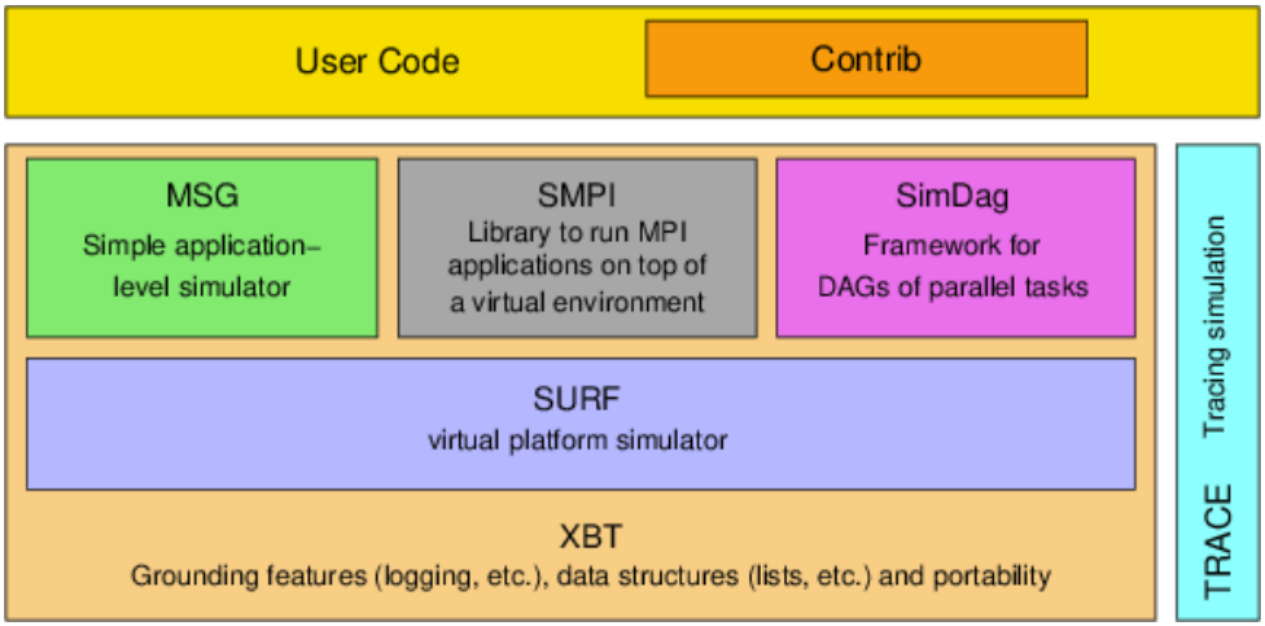


Figura 3.4: Arquitetura interna SimGrid (SLQ19)

- **Location:** a localização também pode ser entendida como um *host*. É um local onde o *agent* é executado. É definido por um recurso computacional, um número de caixas de entrada que permitem a comunicação com outros agentes e dados privados que podem ser acessados por agentes na mesma *location*.
- **Task:** é a atividade da aplicação simulada, podendo ser uma computação a ser executada ou transferência de dados. Uma *task* é definida por uma quantidade de computação, um tamanho de dados e dados privados.
- **Path:** é um aglomerado de recursos de comunicação que representam um conjunto de conexões físicas. *Locations* são interconectadas através de *paths*, mas a aplicação simulada não tem acesso direto as conexões.
- **Channel:** é a abstração da comunicação entre agentes. *Channels* incorporam o conceito de portas de comunicação abertas pelos agentes nas localizações.

3.6 EdgeCloudSim

EdgeCloudSim é um simulador proposto por Sonmez, Ozgovde e Ersoy em 2017 (SOE17). O *EdgeCloudSim* é composto por uma arquitetura modular, onde a comunicação entre os módulos da arquitetura segue a implementação de uma interface. Sendo assim, os módulos podem ser atualizados e modificados independentemente uns dos outros, minimizando o impacto das alterações e facilitando o desenvolvimento da ferramenta.

Atualmente, a implementação do *EdgeCloudSim* é composta por cinco principais módulos (Figura 3.5). Cada um dos módulos pode receber parâmetros independentemente e é otimizado individualmente. Os módulos do *EdgeCloudSim* são:

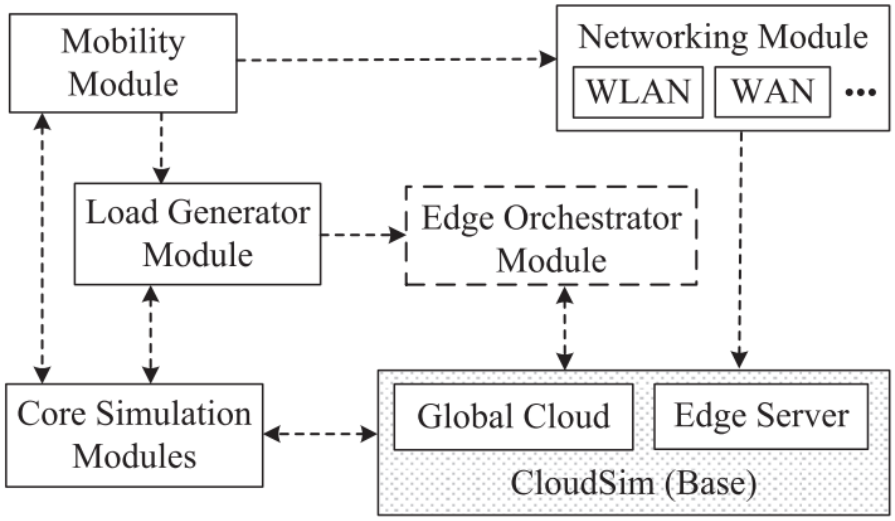


Figura 3.5: Arquitetura interna EdgeCloudSim (SOE17)

- **Core Simulation Module:** módulo responsável pelo carregamento e execução de diferentes cenários de simulação. Este módulo também disponibiliza ferramentas de depuração e mecanismos de *log* da execução. Por padrão, os valores exportados são arquivos no formato CSV (*Comma-Separated-Values*).
- **Networking Module:** módulo responsável pelo tratamento de eventos de rede, tais como atrasos de transmissão entre LAN e WLAN com base nas taxas de *upload* e *download*. Grandes cenários podem ser simulados através do uso da extensão MAN (*Metropolitan Area Network*), que é processada por uma fila única de um único servidor. Atualmente, os desenvolvedores trabalham na expansão deste módulo para obtenção de modelos de atrasos mais realísticos.
- **Edge Orchestrator Module:** módulo responsável pelas decisões a nível de sistema. Este módulo coleta informações dos outros módulos e, com base nelas, é capaz de tomar decisões que impactam os demais módulos, tais como controlar o número de aplicações que pode ser executado em cada *host* devido as suas limitações de memória e CPU.
- **Mobility Module:** módulo responsável por determinar a localização dos dispositivos móveis. No *EdgeCloudSim* cada dispositivo possui coordenadas X e Y, que são atualizadas dinamicamente e gerenciadas através de uma tabela *hash*. A conexão sem fio também possui uma posição, porém ela é fixa.

- **Load Module Generator:** módulo responsável por gerar tarefas (*tasks*) para uma determinada configuração. Juntamente com o Mobility Module, o Load Module Generator é responsável pela entrada de dados no sistema de simulação. É possível modificar parâmetros no Load Module Generator, alterando a quantidade, tamanho e distribuição das tarefas entre os dispositivos do sistema.

Quanto a escalabilidade, os autores do *EdgeCloudSim* informam que é possível simular 75 minutos de cenários com 100 dispositivos e aproximadamente 100 mil tarefas em 10 minutos de processamento em um computador Linux com um processador Intel Core I7-5600U(SOE17).

3.7 iFogSim

O *iFogSim* é um simulador desenvolvido por (GDGB16) para modelagem e simulação de gerenciamento de recursos para Internet das Coisas, *Edge Computing* e *Fog Computing*. Sua arquitetura interna e suas funcionalidades estão fortemente atreladas ao simulador *CloudSim*, que foi utilizado como base para o desenvolvimento. A Figura 3.6 apresenta as principais classes do *iFogSim*.

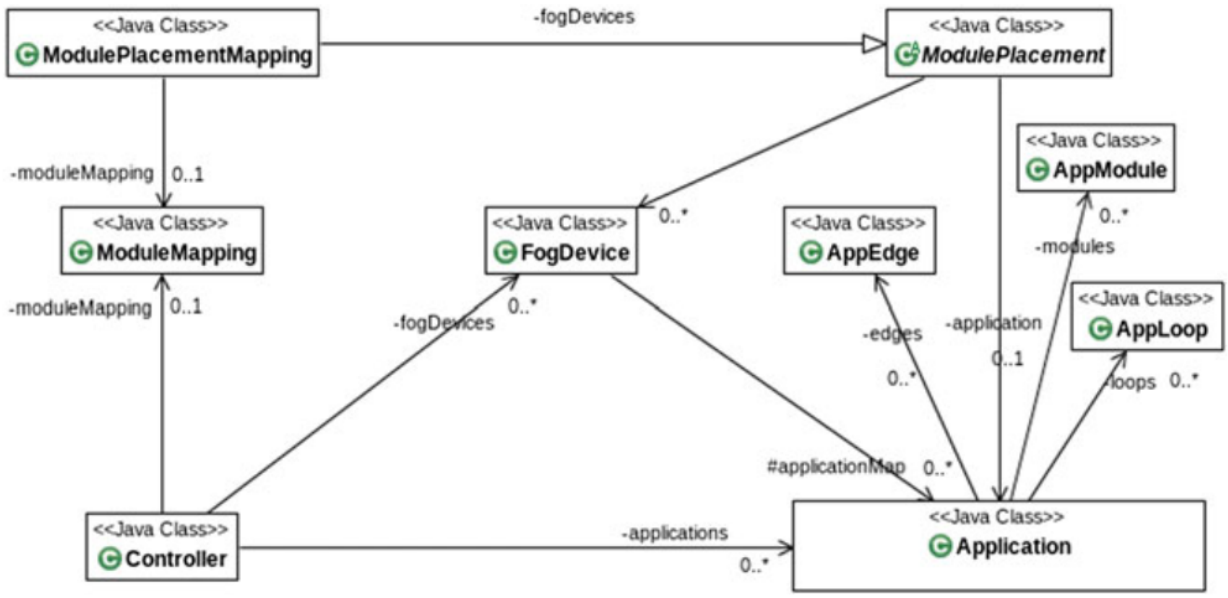


Figura 3.6: Principais classes Java iFogSim (GDGB16)

No simulador *CloudSim* todas as entidades do sistema (como *datacenters*, por exemplo) comunicam-se através da passagem de mensagens. Assim, o *CloudSim* é responsável por administrar os eventos entre as entidades *Fog* no *iFogSim*. A implementação do *iFogSim* é constituida por entidades e serviços (GDGB16).

A utilização do *iFogSim* se dá através da implementação e extensão de classes e interfaces Java. Assim é feita a construção do ambiente simulado. As principais classes a serem implementadas são:

- **FogDevice:** especifica as características de *hardware* do dispositivo de névoa e as suas conexões com outros dispositivos, sensores e atuadores. Como é uma extensão da classe *PowerDatacenter* no *CloudSim*, os principais atributos são memória acessível, processador, espaço de armazenamento e *links* de *download* e *upload*.
- **Sensor:** instâncias da classe sensor são entidades que atuam como sensores IoT. A classe contém atributos que representam as características internas do sensor e uma referência a um objeto que contém as conexões com outros dispositivos.
- **Tuple:** classe essencial para a comunicação entre entidades *fog*. As tuplas representam instâncias da classe *Cloudlet* do simulador *CloudSim*. Caracterizam-se por especificar a origem, o destino e a quantidade de requisições de processamento na ordem de milhões de instruções.
- **Actuator:** classe modelo que define as propriedades de conexão dos atuadores. Os atributos desta classe referem-se ao *gateway* com o qual o atuador está conectado e a latência da conexão.
- **Application:** a aplicação é modelada como um grafo direcionado onde os vértices representam os módulos que realizam o processamento nos dados de entrada e as arestas como dependências entre os módulos. Estas entidades são definidas pelas seguintes classes:
  - **AppModule:** é a classe que processa elementos, sendo uma extensão da classe *PowerVM* do *CloudSim*
  - **AppEdge:** é a classe que representa a dependência de dados entre um par de módulos de aplicações.
  - **AppLoop:** é uma classe adicional utilizada especificamente para o controle de *loops* conforme o interesse do usuário. No *iFogSim* ela pode ser utilizada para medir a latência ponta-a-ponta de uma conexão.

O simulador *iFogSim* traz uma visualização gráfica da topologia de rede (GUI) para facilitar a compreensão da mesma, conforme apresentado na Figura 3.7. As interfaces gráficas podem ser salvas em um arquivo JSON e recriadas posteriormente. A topologia de rede pode ser criada tanto através da interface gráfica quanto através das APIs de programação Java (GDGB16). Ele está disponível para *download* na plataforma GitHub [2]. Em (GDGB16) os autores apontam como uma das principais falhas do *iFogSim* a ausência de modelos de

[2]https://github.com/harshitgupta1337/fogsim

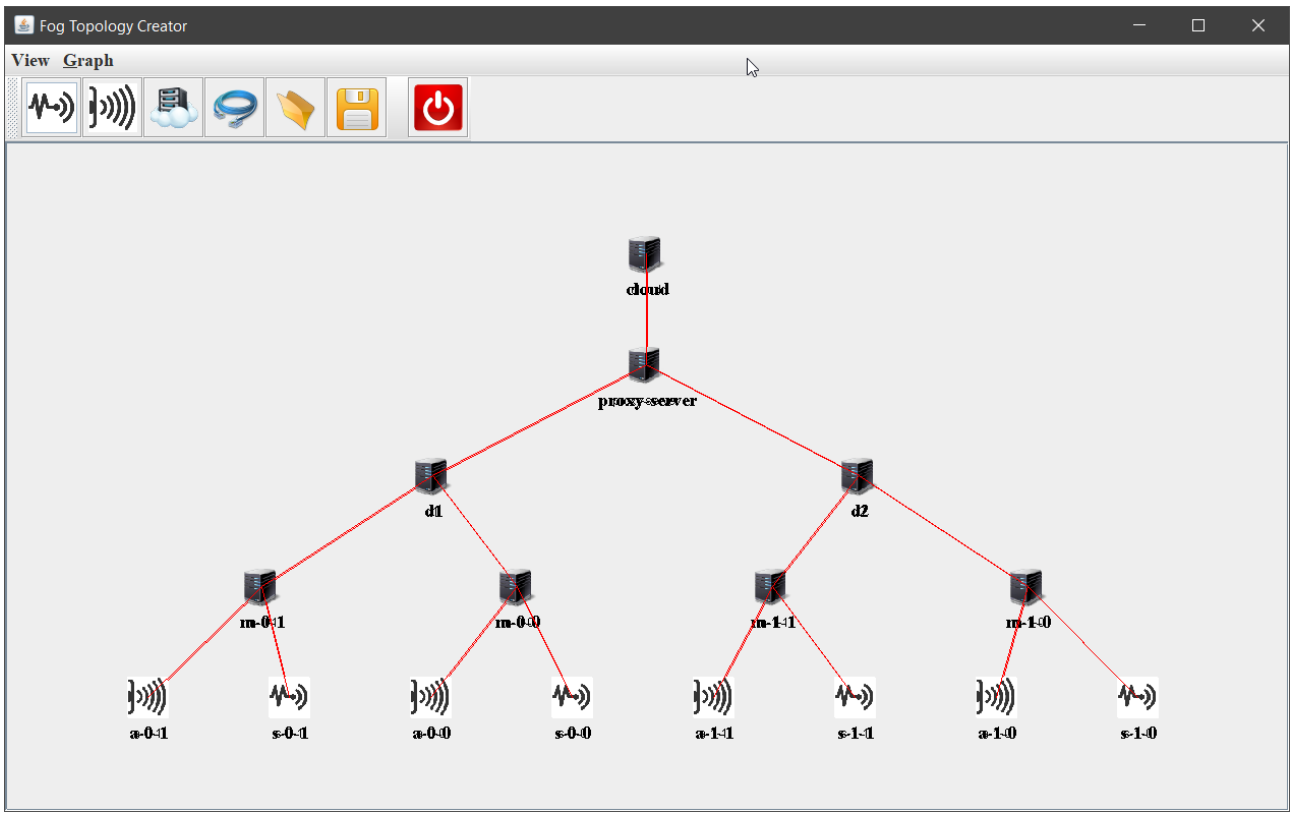


Figura 3.7: Representação gráfica da topologia no iFogSim (elaborado pelo autor)

mobilidade, essenciais para a computação de névoa. Outros problemas identificados são a falta de politicas de gerenciamento de energia, ausência de modelos de falhas para dispositivos *fog*, ausência de priorização de recursos em ambientes com recursos compartilhados (*multi-tenant*) e ausência de agendamento dinâmico de fluxo para redes *Edge Computing* adjuntas.

3.8 MyiFogSim

O *MyiFogSim* é uma extensão do *iFogSim*. Ele implementa um escalonador de aplicações que movimenta as aplicações das *cloudlets* entre diferentes pontos de acesso conforme a movimentação dos nós clientes da aplicação. Essa movimentação acontece com direção e velocidade fixas. O *MyiFogSim* estende as classes *Sensor*, *Actuator* e *FogDevice*, além de introduzir as classes *MobileDevice*, *Coordinate*, *ApDevice*, necessárias para controle da conexão e movimentação dos nós de *fog computing*. Ele foi proposto para a simulação de um balanceador de carga com diferentes políticas de migração de aplicações (BDB+17). A Figura 3.8 apresenta o diagrama das classes que compõem a arquitetura interna do simulador.

As classes FogDevice, Sensor, Actuator, Datacenter, PowerDatacenter e SimEntity são originais do *iFogSim*, portanto sem alterações. As classes MobileSensor e MobileAc-

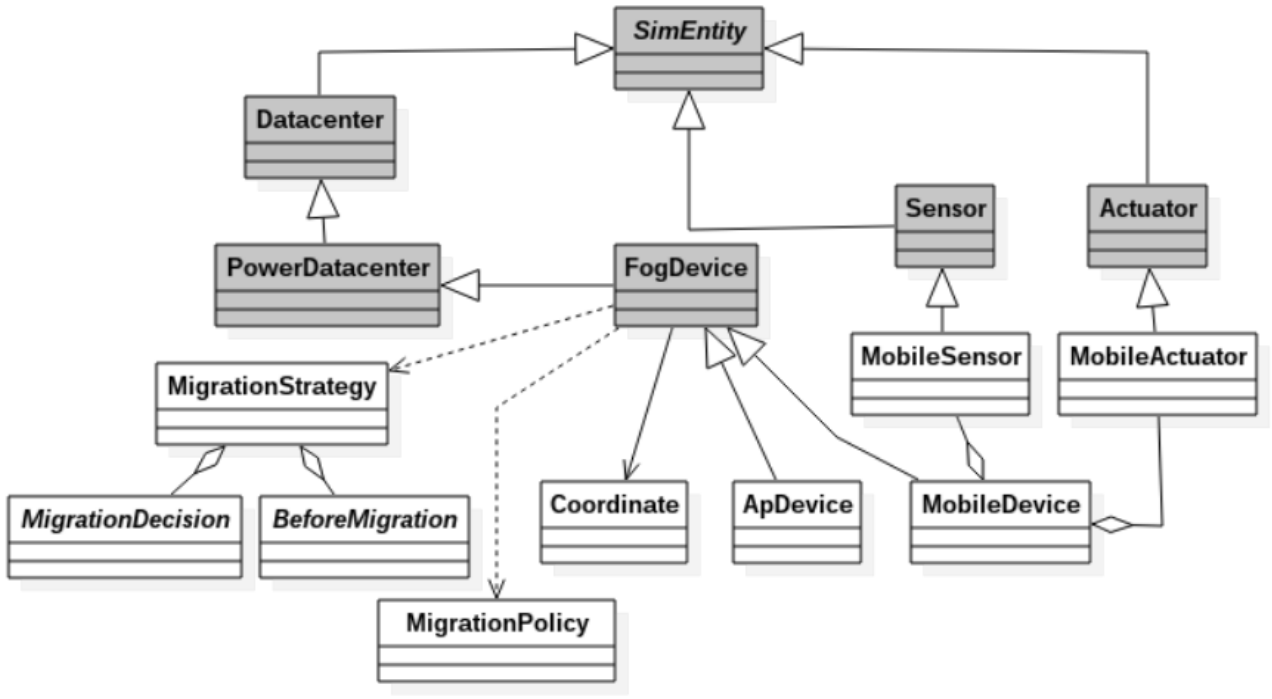


Figura 3.8: Principais classes *MyiFogSim* (GDGB16)

tuators são especializações das classes Sensor e Actuator, respectivamente, que por sua vez são especializações da classe MobileDevice. Esta última herda as propriedades de FogDevice, adicionando novos atributos, como Coordinate, que representa as coordenadas na qual o dispositivo se encontra, e ApDevice, que representa a qual *Access Point* sem fio aquele dispositivo está conectado.

As classes MigrationStrategy, MigrationDecision, BeforeMigration e MigrationPolicy dizem respeito as políticas de migração das máquinas virtuais de aplicação entre dois FogDevices distintos. Essa política pode ser preditiva, que antecipa a movimentação dos dispositivos e realiza a migração pouco antes da mudança de conexão de um dispositivo, balanceada, onde a migração é preparada antes da mudança de conexão mas só é efetuada quando ocorre a mudança de conexão entre dispositivos, ou reativa, com a migração sendo preparada e efetuada somente após a mudança de conexão entre dispositivos. Os resultados das diferentes políticas de migração podem ser verificados em (BDB+17).

O *MyiFogSim* implementa apenas a movimentação de dispositivos com direção e mobilidade fixas. Esse modelo pode descrever alguns cenários, como o trânsito de um avião comercial em uma determinada região, de maneira satisfatória. Porém, diversos outros cenários, como movimentações de grupos militares, perseguições de agentes de segurança a alvos e trânsito de pessoas em ambientes com áreas restritas não são contemplados. A arquitetura do *MyiFogSim* não permite a inclusão de outros modelos que não o de direção e velocidades fixas.

3.9 Considerações finais

A *Fog computing* é um paradigma computacional recente que necessita da utilização de ferramentas de emulação e simulação para a aceleração do desenvolvimento da área. Com o surgimento de mais dispositivos móveis surgiu também a necessidade de mais testes destas plataformas, porém o acesso limitado aos ambientes reais leva a utilização de ferramentas de emulação e simulação.

Podemos classificar estas ferramentas quanto ao ambiente alvo do simulador, tecnologia de implementação, característica de simulação ou emulação, escalabilidade, mobilidade de dispositivos e modelos de mobilidade. Estas informações são descritas na Tabela 3.2.

Ferramenta	Ambiente alvo	Implementação	Característica	Escalabilidade	Mobilidade de dispositivos	Modelos de mobilidade
Emufog	Fog	Maxinet	Emulador	3200 dispositivos em 12 servidores	Não	Não
SDMEC	Edge	Mininet-Wifi	Emulador	4,5 segundos para simulação de acesso a 10 dispositivos	Não	Não
SDFog	Fog	Mininet-Wifi	Emulador	Não especicada	Não	Não
NS-3	Genérico	C++	Simulador	Não especicada	Sim	Sim
Simgrid	Genérico	C++, Java	Simulador	Não especicada	Não	Não
EdgeCloudSim	Edge	Java	Simulador	75 minutos, 100 dispositivos e 100 mil tarefas simuladas em 10 minutos	Sim	Não
iFogSim	Fog e edge	Java	Simulador	25 segundos para simulação de cenários com 64 dispositivos	Não	Não
MyiFogSim	Fog e edge	Java	Simulador	Não especicada	Sim	Não

Tabela 3.2: Comparativo de simuladores e emuladores para *Edge* e *Fog Computing* (elaborado pelo autor)

O primeiro critério é **ferramenta**, que representa o nome da ferramenta ou do conjunto de tecnologias, arcabouços e/ou bibliotecas que descreve um projeto com a proposta de simulação ou emulação de redes em um determinado escopo. São descritos em ferramentas os seguintes itens: *Emufog*, *SDMEC*, *SDFog*, *NS-3*, *Simgrid*, *EdgeCloudSim*, *iFogSim* e *MyiFogSim*. Todos os demais critérios são relacionados as ferramentas.

O critério **ambiente alvo** representa a característica de rede predominante nos artigos e pesquisas que propuseram e/ou utilizaram a ferramenta. As ferramentas *Emufog*, *SDFog*, *iFogSim* e *MyiFogSim* possuem como característica predominante a topologia de rede de *Fog computing*. As ferramentas *SDMEC*, *iFogSim* e *MyiFogSim* possuem referências a topologia de *Edge Computing* e as ferramentas *NS-3* e *Simgrid* referenciam arquiteturas de rede genéricas, indicando a possibilidade de utilização em ambientes de *Edge* e/ou *Fog Computing*, ainda que este não seja o foco dos projetos.

A coluna **implementação** descreve as tecnologias de computação utilizadas na elaboração e desenvolvimento das ferramentas. Aqui é possível verificar que *Emufog*, *SDMEC* e *SDFog* utilizam o *Mininet-Wifi* como base, pois *Maxinet* é uma extensão desenvolvida para o Mininet-Wifi. Os simuladores *NS-3* e *SimGrid* utilizam C++ como linguagem básica de desenvolvimento, sendo que este último também aceita o desenvolvimento na linguagem Java. Além do *Simgrid*, a linguagem Java também é utilizada para as ferramentas *EdgeCloudSim*, *iFogSim* e *MyiFogSim*, o que a caracteriza como a linguagem de desenvolvimento mais presente.

O critério **característica** representa a natureza de operação da ferramenta, isto é, se a ferramenta possui características de simulação ou de emulação. Nesta comparação, a característica de emulador indica que o tempo de execução da simulação é idêntica ao tempo real, ou seja, não é possível simular diferentes períodos de tempo (horas, dias, meses) em um tempo diferente do real. A característica de emulação também está presente quando a ferramenta simula o funcionamento de *hardwares* de rede. São ferramentas de emulação *Emufog*, *SDMEC* e *SDFog*. A característica de simulador é aquela em que o comportamento de dispositivos e subredes é obtido não pela simulação do *hardware* de cada um dos componentes envolvidos, mas sim de modelos e abstrações que simplificam o funcionamento destes dispositivos. Os simuladores podem simular diferentes períodos de tempo em tempo real, e em especial os simuladores de eventos discretos possuem uma alta reprodutibilidade dos resultados. São ferramentas de simulação *NS-3*, *Simgrid*, *EdgeCloudSim*, *iFogSim* e *MyiFogSim*.

O critério **escalabilidade** indica a capacidade de cada ferramenta em lidar com volumes de dados ou ambientes complexos crescentes. Em termos gerais, a escalabilidade é uma característica desejável em um sistema, pois demonstra que uma determinada ferramenta está preparada para lidar com um volume maior de dados e não será um empecilho para o crescimento do sistema.

Há diferentes medições quanto a escalabilidade dos diferentes simuladores e emuladores. Enquanto *Emufog*, *SDMEC*, *EdgeCloudSim* e *iFogSim* indicam o número de dispositivos emulados/simulados, apenas *SDMEC*, *EdgeCloudSim* e *iFogSim* indicam o tempo de simulação/emulação. A ferramenta *Emufog*, no entanto, indica quantos computadores foram utilizados para a realização da emulação. A quantidade de tarefas simuladas, a complexidade das tarefas, o tempo total emulado/simulado e o *hardware* utilizado para a simu-

lação também varia entre todas as ferramentas, o que dificulta a comparação direta entre elas. Aqui existe uma lacuna de pesquisa, onde as ferramentas deveriam ser comparadas com uma única aplicação, mesmo número de clientes e tempo de simulação idêntico, afim de obter um comparativo mais relevante quanto a escalabilidade. *SDFog*, *NS-3*, *Simgrid* e *MyiFogSim* não apresentaram dados quanto a escalabilidade.

O critério **mobilidade de dispositivos** indica a capacidade de cada ferramenta em alterar a posição dos dispositivos emulados/simulados em um determinado plano. Este plano pode ter um representação geográfica com pontos correspondentes a realidade, como coordenadas de latitude e longitude, por exemplo, ou ser um plano abstrato que é indicado apenas por dois eixos em um plano cartesiano. As ferramentas *NS-3*, *EdgeCloudSim* e *MyiFogSim* possuem este recurso, todas as demais ferramentas analisadas não possuem o conceito de mobilidade de dispositivos.

A última coluna apresenta o critério de comparação **modelos de mobilidade**. Por modelos de mobilidade entende-se modelos matemáticos que descrevem o movimento de um ou mais dispositivos em um determinado plano. Estes modelos normalmente são classificados entre: (i) modelos totalmente aleatórios e (ii) modelos com alguma restrição. Dentre os modelos aleatórios, os mais frequentemente utilizados são o *Random Waypoint* e o *Random Walk Model*. Já os modelos com restrições compreendem três principais categorias: (i) modelos com restrição temporal (ex.: *Gauss-Markov*), (ii) modelos com restrição geográfica (ex.: grupos de nós co-relacionados) e (iii) modelos com obstáculos (ex.: *Pathway*) (BH04). Apenas a ferramenta *NS-3* possui modelos de mobilidade implementados, todas as demais ferramentas não possuem modelos de mobilidade.

É possível observar que as ferramentas que possuem a característica de **emulação** são implementadas utilizando o *Mininet* e algumas extensões do *Mininet*, em especial o *Mininet-Wifi*. Nenhum deles possui mobilidade de dispositivos ou modelos de mobilidade, isto pode estar relacionado a natureza de emulação presente nelas.

Com relação ao simuladores, é possível observar que a linguagem de programação Java é a mais utilizada dentre todas as ferramentas analisadas. A ferramenta *MyiFogSim* complementa a ferramenta *iFogSim*, adicionando a capacidade de lidar com mobilidade de dispositivos. No entanto o *MyiFogSim* não implementa modelos de mobilidade, recurso que, dentre todas as ferramentas analisadas, somente o *NS-3* possui.

Esta seção apresentou ferramentas de emulação e simulação para *Cloud computing*. As três primeiras ferramentas, *Emufog*, *SDMEC* e *SDFog* são emuladores, enquanto *NS-3*, *Simgrid*, *EdgeCloudSim*, *iFogSim* e *MyiFogSim* são ferramentas de simulação por eventos discretos. Cada um dos trabalhos relacionados foi explorado quanto as suas características individuais, tais como arquitetura interna, diagrama de classes, disponibilidade da aplicação, recursos de integração, mobilidade de dispositivos e presença de modelos de mobilidade. Ao final, os trabalhos foram comparados entre si e suas características foram descritas e analisadas. É possível ressaltar que nenhuma das ferramentas atende os cri-

térios de escalabilidade, ambiente-alvo de *Fog Computing* e disponibilidade de modelos de mobilidade simultaneamente.

4. IMPLEMENTAÇÃO DOS MODELOS DE MOBILIDADE NO SIMULADOR MYIFOGSIM

Este Capítulo apresenta a contribuição deste trabalho, isto é, a implementação de modelos de mobilidade no simulador MyiFogSim e a ferramenta de visualização de mobilidade. São descritos detalhes quanto aos parâmetros de cada modelo, casos de uso aplicados ao contexto de *Fog Computing* e algoritmos de movimentação. A escolha da ferramenta MyiFogSim ocorre pela combinação de características em relação as demais ferramentas. Ao mesmo tempo em que é um simulador de eventos discretos e com escalabilidade avaliada em sua base (iFogSim), o MyiFogSim também possui recursos de posicionamento de dispositivos em um domínio (mapa ou plano cartesiano) e um modelo de movimentação de direções e velocidades fixas. Adicionalmente, o MyiFogSim é uma ferramenta desenvolvida especificamente para realizar simulações de cenários de *Fog Computing*.

Em comparação as ferramentas de emulação, o MyiFogSim possui uma maior escalabilidade e possibilidade de execução de diferentes cenários em tempos diferentes do tempo-real. A característica de emulação das ferramentas *EmuFog*, *SDMEC* e *SDFog* limitam as suas aplicações e a sua execução para cenários que façam uso de simulações diferentes do tempo-real, como simulações de dias, meses ou anos. Quando comparado com os simuladores *Simgrid* e *iFogSim*, o *MyiFogSim* é capaz de lidar com cenários de mobilidade de dispositivos, isto é, é possível obter a posição de cada dispositivo, e também modificá-la em tempo de execução, que não são possíveis nos simuladores anteriores.

Quanto ao *NS-3*, o MyiFogSim tem a desvantagem de não possuir múltiplos modelos de mobilidade de dispositivos, tendo a sua capacidade de simulação reduzida a cenários onde os dispositivos possuem direções e velocidades fixas. Este modelo pode ser aplicado, por exemplo, para simular o comportamento de um veículo em uma rodovia, mas fica aquém da realidade de outros cenários mais complexos, como o trânsito de pessoas em um campus de universidade com diferentes prédios e áreas de circulação limitadas. Neste cenário, a aplicação do modelo *Obstacles* é mais apropriada. O *NS-3*, no entanto, é um simulador de propósito genérico, cuja adaptação a contextos de *Fog Computing* é complexa e trabalhosa (SOE17).

Outro aspecto negativo é a ausência de uma ferramenta para visualização dos dados da movimentação dos dispositivos. A visualização da movimentação dos dispositivos facilita a compreensão do cenário simulado, pois dá pistas visuais quanto ao posicionamento dos dispositivos. Esta visualização, juntamente com os dados de latência, consumo energético e posição de cada dispositivo e/ou antena de comunicação, pode auxiliar o pesquisador a identificar gargalos e/ou a realizar otimizações.

Este trabalho avança na criação de uma ferramenta de simulação de *Fog Computing* mais completa, escalável, que possua diferentes modelos de mobilidade para disposi-

tivos. A arquitetura da ferramenta permite também que outros novos modelos possam ser adicionados futuramente, sem impacto no restante da arquitetura e sem a necessidade de reescrita das aplicações já compatíveis com o MyiFogSim. Adicionalmente, também há o desenvolvimento de uma ferramenta de visualização da mobilidade dos dispositivos, que representa a movimentação dos mesmos em todo o período simulado.

4.1 Modelos de mobilidade

Modelos de mobilidade descrevem o padrão de movimentação de usuários e/ou dispositivos. Tais dispositivos, portanto, possuem uma localização que é alterada com o passar do tempo, isto é, são dispositivos móveis. A *Fog Computing* é uma área fortemente relacionada a mobilidade de dispositivos, tendo diversas aplicações projetadas para cenários onde os dispositivos se movimentam. O comportamento móvel dos dispositivos pode influenciar o desempenho destas aplicações, do gerenciamento da rede e da sobrecarga de utilização de recursos compartilhados.

É necessário, portanto, levar em consideração a movimentação dos dispositivos na execução das simulações de *Fog Computing*. Entretanto, os simuladores de *Fog Computing* não possuem modelos de mobilidade implementados, com exceção do MyiFogSim, que implementa somente o modelo de direção e velocidade fixas. A adição de outros modelos de mobilidade torna possível a simulação de cenários mais precisos quando comparados a realidade, e a mudança no padrão de movimentação pode trazer impacto sobre os balanceadores de carga de máquinas virtuais, por exemplo.

Um modelo de mobilidade é composto por características como: domínio espacial, posição inicial dos nós, rota/trajetória, velocidade, transição entre rotas, e regras de borda (San12). O domínio espacial, aqui referenciado como mapa, é um domínio de duas ou mais dimensões, que determina quais são os limites espaciais dos dispositivos. A posição inicial é o valor de cada dispositivo dentro do domínio (mapa) ao iniciar a simulação. A rota, ou trajetória, é o caminho percorrido pelo dispositivo desde a sua posição inicial até a sua posição final, e a velocidade é em quanto tempo (ou ciclos de simulação) essa trajetória ocorre. De maneira semelhante, a transição entre rotas determina as regras após o término de uma rota e início de outra. Por fim, as regras de borda determinam o comportamento dos nós, sendo possível que em alguns modelos os nós temporariamente não estejam dentro do domínio espacial (mapa).

Foram escolhidos cinco modelos de mobilidade para serem implementados no simulador MyiFogSim: *Random Waypoint*, *Reference Point Group*, *Nomadic Community*, *Pursue* e *Obstacles*. A escolha destes modelos se dá a partir da representatividade de cada um deles quanto as características de restrições de modelos de mobilidade. A seguir, cada

modelo é detalhado quanto a descrição geral do modelo, parâmetros de cada modelo, quais os seus impactos na movimentação e algoritmos implementados.

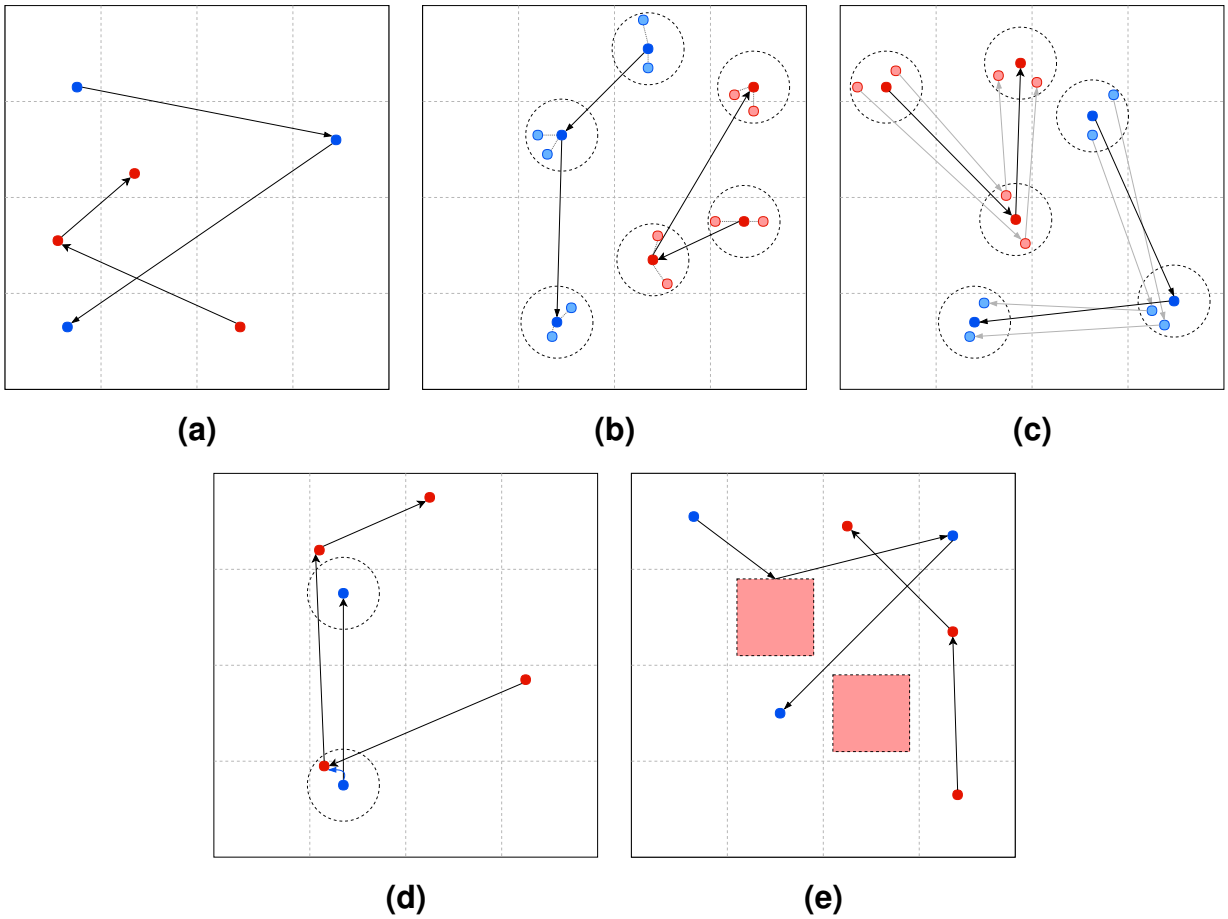


Figura 4.1: Representação da movimentação dos modelos de mobilidade: **(a)** *Random Waypoint*, **(b)** *Reference Point Group*, **(c)** *Nomadic Community*, **(d)** *Pursue*, **(e)** *Obstacles* (elaborado pelo autor).

4.1.1 *Random Waypoint*

O modelo *Random Waypoint*, proposto por Johnson (BH04) é utilizado para avaliar protocolos de roteamento MANET (*Mobile Ad hoc Network*), e consiste na determinação aleatória de posições, destinos e de uma velocidade a cada novo trecho, isto é, a cada vez que o destino é alcançado. A implementação do modelo *Random Waypoint* é realizada através do seguinte fluxo: ao início da simulação, cada nó escolhe aleatoriamente um dentre os destinos possíveis no mapa e o define como destino. Em seguida, cada nó se movimenta em direção ao destino com uma velocidade uniforme, definida de maneira aleatória no intervalo [0, VMax], onde VMax é a velocidade máxima permitida para cada nó. Tanto a velocidade quanto o destino de cada nó são definidos independentemente dos demais nós. Quando o nó alcança o destino, ele permanece no mesmo ponto por Tpause ciclos de

simulação. Se TPause = 0, então o movimento é contínuo e não há parada após a chegada a um destino. Após aguardar o número de ciclos definidos em Tpause, um novo destino e uma nova velocidade são definidos aleatoriamente para o nó. A Figura 4.1 (a) representa a movimentação dos nós deste modelo.

O modelo *Random Waypoint* possui dois parâmetros chave: VMax e TPause. VMax refere-se a velocidade máxima permitida a cada nó, e TPause é o número de ciclos que devem ser aguardados por cada nó após este chegar ao seu destino. Quando VMax é alto e TPause é baixo, a topologia ou mobilidade é altamente dinâmica, e de maneira oposta, quando VMax é baixo e TPause é alto, a topologia é relativamente estável. Ao variar os parâmetros do *Random Waypoint* é possível obter diferentes comportamentos, sendo o parâmetro VMax o mais expressivo deles no que se refere a velocidade média dos nós. A medida de mobilidade de um sistema, proposta por Johansson, Larsson e Hedman (JLH+99) quantifica a média da velocidade de todos os nós ao longo do tempo da simulação, onde |i,j| é um par distinto de nós, n é o número total de nós na simulação e T é o tempo da simulação.

Algoritmo 4.1: calculateNextDevicePositions (*Random Waypoint*)

Result: Calcula a próxima posição dos dispositivos
Input : none
Output: none

```
1  foreach smartthing ∈ smartthings do
2      if smartthing.tpause = 0 then
3          if smartthing.coordinate = smartthing.destination then
4              smartthing.tpause ← defaulttpause
5              destinationGenerator(smartthing)
6          else
7              move(smartthing)
8          end
9      else
10         smartthing.decreaseTPause()
11     end
12 end
13 exportData()
```

Dois algoritmos compõem a movimentação básica do modelo *Random Waypoint*: *(i) calculateNextDevicePositivon* e *(ii) move*. O Algoritmo 4.1 (*calculateNextDevicePositivon*) verifica se os dispositivos estão no período de espera de TPause ou se necessitam se movimentar através de um *loop* na lista de todos os dispositivos da simulação.

O Algoritmo 4.2 (*move*) calcula, de fato, a distância de cada nó para o seu respectivo destino, qual trajetória para se mover em direção ao destino, e qual a distância que o nó pode percorrer no ciclo, finalizando então com a definição de uma nova posição do nó.

Algoritmo 4.2: move (*Random Waypoint*)

Result: Move o dispostivo *smartthing* no plano simulado.
Input : smartthing
Output: none

```
14 steps ← smartthing.speed
15 while steps > 0 do
16 |   CalculateDistances()                    ▷ Calcula a distancia do dispositivo até o destino
17 |   if ratioY * |distanceX| >= ratioX * |distanceY| then
18 |   |   if −1 <= distanceX <= 1 then
19 |   |   |   currentX = destX                                  ▷ Chegou ao destino em X
20 |   |   else
21 |   |   |   if distanceX > 0 then
22 |   |   |   |   currentX ← currentX + 1                       ▷ Distancia X maior, incrementa
23 |   |   |   else
24 |   |   |   |   currentX ← currentX − 1                       ▷ Distancia X maior, decrementa
25 |   |   |   end
26 |   |   end
27 |   else
28 |   |   if −1 <= distanceY <= 1 then
29 |   |   |   currentY ← destY                                  ▷ Chegou ao destino em Y
30 |   |   else
31 |   |   |   if distanceY > 0 then
32 |   |   |   |   currentY ← currentY + 1                       ▷ Distancia Y maior, incrementa
33 |   |   |   else
34 |   |   |   |   currentY ← currentY − 1                       ▷ Distancia Y maior, decrementa
35 |   |   |   end
36 |   |   end
37 |   end
38 |   steps − −
39 end
40 smartthing.coordinate ← newCoordinate(currentX, currentY)
```

4.1.2 *Reference Point Group*

O modelo *Reference Point Group* se diferencia dos modelos puramente aleatórios, como por exemplo o modelo *Random Waypoint*, por introduzir uma dependência espacial na co-relação entre os nós. Existem dois tipos de nós neste modelo: (i) o líder do grupo, que determina a movimentação do grupo, e (ii) membros do grupo, que se movimentam de acordo com o seu respectivo líder. A dependência acontece entre a posição de todos os membros de um grupo com relação a posição do líder deste mesmo grupo, onde a direção e a velocidade do líder influenciam a direção e a velocidade dos demais membros do grupo. No modelo *Reference Point Group* cada grupo possui um ponto central, que é a posição do líder do grupo. A cada ciclo de simulação os demais membros deste mesmo grupo variam a sua posição em relação a esta posição central conforme um fator definido aleatoriamente a cada novo ciclo de simulação (*clock tick*). Os membros do grupo não possuem destino, apenas movimentam-se de acordo com o líder do seu grupo. Este modelo pode ser mais interessante para simulação da movimentação de equipes onde os membros dela seguem o comportamento do líder do seu grupo, como bombeiros ou soldados. A movimentação dos líderes é feita com o modelo *Random Waypoint*. A Figura 4.1 (b) representa a movimentação dos nós deste modelo.

O modelo *Reference Point Group* possui três parâmetros chave: VMax, TPause e split. VMax e TPause possuem as mesmas propriedades e características do modelo *Random Waypoint*, e *split* refere-se ao quantidade de membros em cada grupo, incluindo o líder.

Este modelo se diferencia do *Random Waypoint* na implementação dos algoritmos de cálculo de novas posições dos nós e no gerador de posições dos nós dependentes dos líderes. Enquanto o primeiro, o Algoritmo 4.3 passa a ter iterações por grupos, o segundo recebe uma coordenada como parâmetro de alvo, e a partir deste valor deriva uma nova posição dentro de uma tolerância (em percentual), definida nas configurações do modelo no início da simulação. O Algoritmo 4.4 detalha a implementação deste comportamento.

Algoritmo 4.3: calculateNextDevicePositions (*Reference Point Group*)

Result: Calcula a próxima posição dos dispositivos
Input : none
Output: none

```
41 foreach group ∈ groups do
42     foreach smartthing ∈ group do
43         if isLeader(smartthing) then
44             if smartthing.tpause = 0 then
45                 if smartthing.coordinate = smartthing.destination then
46                     smartthing.tpause ← defaulttpause
47                     destinationGenerator(smartthing)
48                 else
49                     move(smartthing)
50                 end
51             else
52                 smartthing.decreaseTPause()
53             end
54         else
55             generatePosition(group.smartthing, group.leader.coordinate)
56         end
57     end
58 end
59 exportData()
```

Algoritmo 4.4: generatePosition

Result: Calcula o posicionamento do dispositivo *smartthing* utilizando um fator e a posição do dispositivo alvo (líder do grupo)
Input : smartthing, target
Output: none

```
60 low ← factor ∗ −1
61 high ← factor
62 altX ← random(high + 1 − low) + low
63 altY ← random(high + 1 − low) + low
64 smartThing.coordinate ← newCoordinate(target.x + altX, target.y + altY)
```

4.1.3 *Nomadic Community*

Assim como o *Reference Point Group*, o modelo *Nomadic Community* também descreve cenários de movimentação onde os nós se movimentam em grupo. Ele se diferencia, no entanto, pela existência de uma velocidade e um destino distinta para cada nó do grupo ao invés da posição ser recalculada com um novo parâmetro aleatório a cada ciclo de simulação com relação a posição do líder grupo. Assim, individualmente a movimentação dos nós é mais previsível e constante que o modelo *Reference Point Group*, sendo comparável a modelos de grid ou de varredura por colunas, que descrevem um movimento ainda menos esporádico e mais previsível. No modelo *Nomadic Community* também existem as figuras de líderes, que determinam a velocidade e o destino do grupo. Porém, cada nó varia a sua velocidade e o seu destino com relação ao que o líder estabeleceu, e, a partir disso, descreve o seu movimento de maneira independente. Há situações em que os membros chegam antes do líder ao destino, e nestes casos permanecem parados até a chegada do líder e do estabelecimento do novo destino do grupo.

O modelo *Nomadic Community* recebe os mesmos parâmetros de configuração que o modelo *Reference Point Group*: VMax, TPause e split. A Figura 4.1 (c) representa a movimentação dos nós deste modelo. O Algoritmo 4.5 detalha o construtor do modelo *Nomadic*, com dois *loops* aninhados que dividem a lista de dispositivos em grupos e posteriormente outros dois *loops* que geram as posições iniciais para cada dispositivo, inclusive dos líderes, dos grupos.

Algoritmo 4.5: NomadicCommunity construtor

Result: Cria um objeto-modelo *Nomadic Community*
Input : Smartthings list, TPause
Output: Objeto NomadicModel

```
65 foreach smartthing ∈ smartthings/split do
66     group ← newGroup()
67     foreach smartthing ∈ split do
68         group ← smartthing
69     end
70     groups.add(group)
71 end
72 foreach group ∈ groups do
73     foreach smartthing ∈ group do
74         if isLeader(smartthing) then
75             speedGenerator(group.smartthing)
76             destinationGenerator(group.smartthing)
77         else
78             destinationGenerator(leader.destination)
79             speedGenerator(leader.speed)
80             positionGenerator(leader.coordinate)
81         end
82     end
83 end
```

Os algoritmos do modelo *Nomadic* se diferenciam do *Reference Point Group* pelo estabelecimento individual de destinos para todos os nós, mas diferentemente do modelo *Random Wayipont*, os destinos de um grupo estão atrelados ao destino do líder do grupo. A velocidade também é relativa ao líder, e o fator de relacionamento é definido nas configurações do modelo quando este é instanciado no início da simulação.

4.1.4 *Pursue*

O modelo *Pursue* consiste em emular cenários em que diversos nós tentam capturar um único nó. Este nó se move livremente conforme o modelo *Random Waypoint*, ou seja, sem restrição quanto a direção e velocidade. Os demais nós permanecem parados em suas posições iniciais até que o nó alvo se aproxime deles. Neste momento, os nós próximos obtém uma velocidade aleatória que varia de um fator máximo e mínimo configurado previamente nos parâmetros de simulação, e iniciam o seu movimento tendo como destino a

posição do nó fugitivo. Os agentes perseguidores podem ter o seu movimento interrompido se, a qualquer momento, o fugitivo sair do alcance dos agentes perseguidores. O raio de alcance também é um parâmetro configurável da simulação. Este modelo pode ser utilizado para simulações de rastreamento e perseguição como, por exemplo, perseguições policiais em ambientes públicos ou segurança de ambientes limitados. A Figura 4.1 (d) representa a movimentação dos nós deste modelo.

Algoritmo 4.6: calculateNextDevicePositions (*Pursue*)

Result: Calcula as próximas posições dos dispositivos

```
84  target = smartthings.get(0)
85  foreach smartthing ∈ smartthings do
86      if smartthing.tpause == 0 then
87          if smartthing.Coordenate == smartthing.destination then
88              smartthing.setTPause(this.defaulttpause)
89              destinationGenerator(smartthing)
90          else
91              if smartthing == target||isTargetInRange(target, smartthing) then
92                  if smartthing! = target then
93                      copyDestination(target, smartthing)
94                      speedGenerator(smartthing, target.speed)
95                  end
96                  move(smartThing)
97              end
98          end
99      else
100         smartthing.decreaseTPause()
101     end
102 end
```

Os algoritmos implementados se diferenciam no cálculo de novas posições, levando em consideração a proximidade dos dispositivos do dispositivo alvo (*target*). O algoritmo 4.6 apresenta as mudanças, sendo as linhas que envolvem comparações com a posição de *target* as mudanças mais significativas. O algoritmo 4.7 detalha a implementação do método que determina se o *target* está dentro do alcance da *smartthing*.

Algoritmo 4.7: isTargetInRange

Result: Calcula as próximas posições dos dispositivos
Input : target, smartthing
Output: result (boolean)

```
103 result ← false
104 x ← smartThing.coordinate.X
105 y ← smartThing.coordinate.Y
106 range ← constants.range/2
107 xtarget ← target.coordinate.X
108 ytarget ← target.coordinate.Y
109 if x − range <= xtarget & xtarget <= x + range & y - range <= ytarget & ytarget <= y +
    range then
110 |   result ←true
111 end
```

O modelo *Pursue* recebe os seguintes parâmetros de configuração: VMax, TPause, alcance, fator de velocidade mínima e fator de velocidade máxima. VMax e TPause são idênticos aos parâmetros do modelo *Random Waypoint*. Além destes, o raio de alcance dos agentes perseguidores define a sensibilidade dos nós ao nó alvo. Por fim, os fatores de velocidade mínima e máxima referem-se ao percentual de velocidade que pode ser adquirido aleatoriamente pelos agentes perseguidores em relação ao nó alvo.

4.1.5 *Obstacles*

O modelo *Obstacles* representa cenários onde a movimentação dos nós é influenciada pelo mapa do ambiente simulado, isto é, existem obstáculos que modificam as interações dos dispositivos. Este cenário leva os nós a evitarem determinadas áreas, o que pode incorrer no surgimento de corredores de movimentação. São exemplos de cenários com obstáculos, conferências e apresentações, onde o acesso ao palco é restrito, mapas de movimentações por prédios nos quais os nós não possuam acesso, e áreas públicas que estejam passando por manutenção e possuem áreas em reforma. Estes cenários podem ser simulados através da criação de caixas retangulares que são adicionadas a lista de obstáculos do mapa. Todos os obstáculos são evitados pelos nós pois representam uma área proibida aos nós. Esta modelagem não possui níveis de acesso, isto é, a possibilidade de que somente um subgrupo de nós tenha acesso a determinadas áreas do mapa. Em áreas livres de obstáculos, os movimentos dos nós são descritos conforme o modelo *Random Waypoint*. O algoritmo 4.8 é chamado a cada movimentação de dispositivos e verifica se a tentativa de movimentação é válida. Se não for, a *smartthing* recalcula o destino e

verifica novamente a sua viabilidade. Cenários com muitos obstáculos geram mais eventos de cálculo.

Algoritmo 4.8: isInObstacle

Result: Verifica se a smartthing está tentando entrar em uma região restrita
Input : x, y
Output: result (boolean)

```
112 result ← false
113 foreach coordinatesList ∈ obstaclesList do
114     xMin ← coordinatesList.getStartCoordinate.X
115     yMin ← coordinatesList.getStartCoordinate.Y
116     xMax ← coordinatesList.getFinishCoordinate.X
117     yMax ← coordinatesList.getFinishCoordinate.Y
118     if x >= xMin  & x <= xMax & y >= yMin & y <= yMax then
119         result ← true
120     end
121 end
```

O modelo *Obstacles* recebe como parâmetro, além dos parâmetros VMax e TPause do modelo *Random Waypoint*, as dimensões e posição de pelo menos um obstáculo a ser incluído no mapa. Posteriormente outros objetos podem ser adicionados ao modelo, tendo efeito imediato após a inserção. Se um nó estiver em uma posição onde foi inserido um obstáculo ele poderá se movimentar livremente até que alcance uma área livre, e a partir deste momento não poderá mais retornar as áreas marcadas como obstáculos. A Figura 4.1 (e) representa a movimentação dos nós deste modelo.

4.2 Prototipação no simulador MyiFogSim

A implementação do protótipo foi realizada como uma extensão do simulador *MyiFogSim*. As aplicações originais do *MyiFogSim* e iFogSim podem ser executadas sob esta extensão, ou seja, há retrocompatibilidade com as aplicações já existentes. A implementação consiste em três etapas distintas: (i) adaptação de classes existentes do *MyiFogSim*, (ii) criação de uma interface para implementação de modelos de mobilidade, e (iii) implementação dos modelos de mobilidade conforme a interface.

As classes que foram adaptadas são mostradas na Figura 4.2. As classes em verde são classes que foram adicionadas ao simulador, as classes em amarelo são as classes que sofreram alterações, e as classes em cinza são classes relacionadas à mobilidade que não foram alteradas.

As seguintes alterações foram realizadas nas classes já existentes:

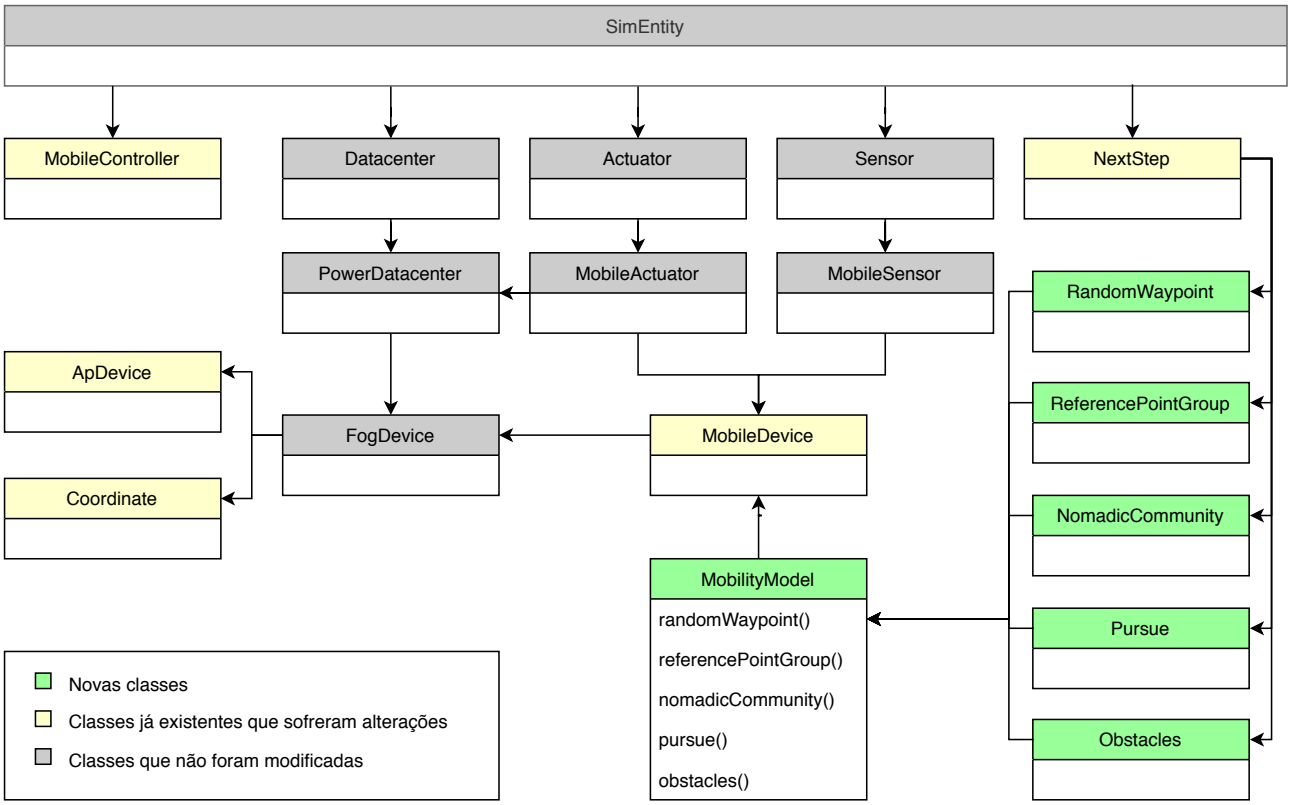


Figura 4.2: Diagrama de classes modificadas (elaborado pelo autor).

- **fog/entities/MobileDevice:** inclusão dos atributos privados Coordinate destination, Coordinate ratio e int TPause e dos seus respectivos métodos públicos de get/set. Destination refere-se a coordenada de destino do MobileDevice, ratio representa a razão entre o eixo X e Y inicial, necessários ao cálculo da melhor trajetória mantendo a direção originalmente definida pelo modelo, e TPause é o parâmetro que se refere ao tempo (em ciclos de simulação) em que o dispositivo permanece na mesma posição após alcançar o destino. TPause é um dos parâmetros iniciais fundamentais para os modelos de mobilidade, juntamente com a velocidade.

- **fog/localization/Coordinate:** adição dos atributos inteiros X e Y e do método público isEqual(Coordinate c), que compara os valores privados dos inteiros X e Y do objeto Coordinate (this) com os valores do objeto Coordinate c recebido por parâmetro. Retorna um *boolean* verdadeiro se ambos valores forem idênticos, caso contrário retorna falso.

- **fog/localization/ApDevice:** sobrecarga de métodos para adicionar parâmetros de coordenadas, que são recebidos como parâmetros e gravados em variável interna.

- **fog/placement/MobileController:** inclusão da biblioteca MobilityModel (implementada posteriormente), criação do atributo estático MobilityModel mobilitymodel, que é o objeto do modelo de mobilidade, adição de novo método construtor que recebe como parâmetro extra um objeto MobilityModel, alteração na chamada do método

NextStep.nextStep para incluir o objeto mobilityModel e os métodos de get/set do objeto mobilityModel.

- **fog/vmmigration/NextStep:** inclusão da biblioteca MobilityModel (implementada posteriormente), adição do método construtor nextStep que recebe um objeto MobilityModel model como argumento e somente executa o método model.calculateNextDevicesPositions(), que fará o cálculo das novas posições dos dispositivos conforme o modelo que for instanciado implementando a interface MobilityModel.

A segunda etapa contempla a criação de interface MobilityModel no diretório /src/org/fog/**mobilitymodel**, que hospeda tanto a interface dos modelos de mobilidade quanto os diferentes algoritmos de implementação da interface. A classe MobilityModel é uma classe abstrata que facilita a adaptação do código de diferentes modelos de mobilidade através do uso de herança e polimorfismo. Ela possui os seguintes componentes:

- **public abstract void addDevice(MobileDevice device):** método que adiciona um MobileDevice a lista de dispositivos cuja posição deve ser controlada pelo modelo de mobilidade.
- **public abstract void removeDevice(MobileDevice device):** método que remove um MobileDevice da lista de dispositivos cuja posição deve ser controlada pelo modelo de mobilidade.
- **public abstract void removeDevice(int index):** método que remove um MobileDevice existente através do índice do mesmo em uma lista de dispositivos cuja posição deve ser controlada pelo modelo de mobilidade.
- **public abstract void calculateNextDevicesPosition():** método que calcula as novas posições de todos os dispositivos. É invocado pelo método nextStep, que por sua vez é controlado pela classe MobilityController.

4.2.1 Execução de simulações na ferramenta MyiFogSim

O *MyiFogSim* é uma ferramenta de simulação por eventos discretos desenvolvida especificamente para cenários de *Fog Computing*, com atenção especial a cenários com mobilidade de dispositivos. Sua utilização se dá através da programação de código Java e da manipulação de bibliotecas de arcabouços de simulação, tais como iFogSim e CloudSim.

Dada a frequência de utilização de certos componentes, faz-se necessário uma breve explicação a cerca de sua nomenclatura, já que estes serão referenciados a seguir. *Policies* são as políticas de migração de máquinas virtuais, isto é, quais as regras

e sob quais circunstâncias estas migrações devem ocorrer. *Access Points* são os pontos de acesso sem fio, ou seja, as antenas às quais os dispositivos sem fio se conectam. *Smartthings* são dispositivos de *Fog Computing* móveis que executam uma aplicação. *Brokers* são dispositivos que recebem e encaminham as mensagens encaminhadas pelas *Smartthings* dentro da estrutura interna do *MyiFogSim*. *Cloudlets* são pequenos *datacenters* que executam aplicações de maneira autocontida, muitas vezes lidando com dados temporários e não permanentes.

Para executar uma simulação de um cenário de *Fog Computing* na ferramenta *MyiFogSim* deve-se seguir as seguintes etapas:

1. Inicialização dos pacotes de simulação CloudSim;
2. Definição das *Policies* de migração de máquinas virtuais entre os *Cloudlets*;
3. Criação de todos os dispositivos, incluindo *Access Points* e *Smartthings*;
4. Estabelecimento de conexão entre os sensores e os *brokers*, dispositivos que recebem as informações do sensores e as repassam a rede;
5. Criação do *Controller*, objeto que controla/orquestra a simulação através da manipulação de *Cloudlets*, *Access-points*, *Smartthings*, mapeamento de módulos da aplicação, parâmetros de estratégia de migração de máquinas virtuais e coordenadas de dispositivos;
6. Configuração dos parâmetros de log da simulação;
7. Início da simulação através dos pacotes CloudSim.

A primeira etapa consiste na parametrização dos pacotes *CloudSim*, cujas bibliotecas são a base para o desenvolvimento do iFogSim e, posteriormente, do MyiFog. São parâmetros necessários a necessidade de log, o número de *clouds* a serem simuladas, uma instância de calendário, um objeto de *trace* de eventos a inicialização do pacote CloudSim.

A segunda etapa consiste na definição das políticas de migração de máquinas virtuais entre diferentes *Access Points*, tamanho de cada passo de simulação, possibilidade das máquinas virtuais de migrarem de *Cloudlet*, estratégia de migração dentre os três estratégias apresentadas em (BDB+17), número máximo de dispositivos *Smartthings*, banda de rede disponível por dispositivo e metodologia disponível de réplica de máquinas virtuais (premeditada, em tempo real ou com atraso).

A terceira etapa é a criação de objetos que representam os *Access Points*, *Cloudlets* e sua rede, *Smartthings*, *Broker* (intermediário das mensagens enviadas pelas *Smartthings*), máquinas virtuais e aplicação a ser executada pelas *Smartthings*. Com os objetos criados, são feitas as conexões entre eles que ainda não necessárias, como a conexão entre sensores e atuadores e *brokers*, finalizando a quarta etapa.

A quinta etapa consiste na criação do *Controller*. O *Controller* é o objeto que unifica o mapeamento de modulos, a lista de aplicações a serem executadas, *Cloudlets*, *Access Points*, *Brokers*, parâmetros e estratégias de migração de máquinas virtuais. A sexta etapa consiste na configuração dos parâmetros de log e a última etapa é, enfim, a inicialização da simulação.

A contribuição deste trabalho, a extensão do *MyiFogSim*, se dá através da criação de uma nova etapa, opcional, entre a definição das conexões entre sensores e *brokers* e a criação do objeto *Controller*. Esta etapa é a criação do modelo de mobilidade. O modelo de mobilidade é um objeto de qualquer classe que implemente a classe abstrata (interface) MobilityModel, e os seus respectivos métodos *addDevice*, *removeDevice* e *calculateNextDevicesPosition*. A classe RandomWaypoint implementa a interface MobilityModel, portanto o único código necessário para a utilização do modelo *Random Waypoint* como modelo de simulação em uma aplicação já desenvolvida para o *MyiFogSim* é apresentada em *Listing* 4.1:

Listing 4.1: Utilização do modelo Random Waypoint como modelo de simulação

```
MobilityModel mobilitymodel = new RandomWaypoint(smartThingsList, 2);
```

O objeto *MobilityModel* é posteriormente incorporado ao objeto *Controller* na quinta etapa. Outros exemplos inicialização de modelos de mobilidade são apresentados em *Listing* 4.2

Listing 4.2: Utilização de diferentes modelo de simulação

```
MobilityModel mobilitymodel = new RandomGroupPoint(smartThingsList, 3, 1);
MobilityModel mobilitymodel = new NomadicCommunity(smartThingsList, 2, 10);
MobilityModel mobilitymodel = new Pursuit(smartThingsList, 0);
MobilityModel mobilitymodel = new Obstacles(smartThingsList, 1, coordinateTopLeft,
    coordinateBottomRight);
```

O impacto no desenvolvimento de aplicações é, portanto, de apenas uma linha. Se não há a necessidade da utilização de um modelo de mobilidade não há necessidade de alteração do código, portanto a extensão do *MyiFogSim* proposta neste trabalho mantém retrocompatibilidade com as aplicações já existentes para este simulador.

A última etapa é a inicialização da simulação. O seguinte trecho de código inicializa o pacote CloudSim e invoca o método *run()*. Este método dá início ao processamento dos eventos de simulação configurados no *Controller*. Mais especificamente, o *Listing* 4.3 exemplifica um trecho que processa todos os eventos enquanto o estado da simulação for *RUNNABLE*:

Listing 4.3: Inicialização do *CloudSim* e execução do método *run()*

```
public void run() {
        SimEvent ev = evbuf != null ? evbuf : getNextEvent();
        while (ev != null) {
                processEvent(ev);
                if (state != RUNNABLE) {
                        break;
                }
                ev = getNextEvent();
        }
        evbuf = null;
}
```

A chamada de *getNextEvent* aciona o método *processEvent* do *Controller*. O *Controller*, por sua vez, invoca o método *NextStep* da classe *NextStep* do namespace *Vmmigration*. Uma vez que o método *NextStep* da classe *NextStep* foi acionado, o modelo de mobilidade tem o seu método de cálculo de posições chamado. Na proposta original do *MyiFogSim* o cálculo de posições é feito diretamente aqui, porém este trabalho propõe a substituição deste método por uma chamada única. Dadas as propriedades de herança e polimorfismo com as quais os modelos foram organizados, não é necessário informar qual o modelo implementado, visto que existe a certeza de que ele implementa o método *calculateNextDevicesPositions*. O *Listing* 4.4 apresenta um trecho que substitui o modelo de mobilidade de direção e velocidade fixas propostos no *MyiFogSim*, reduzindo as 76 linhas da implementação original para apenas uma, independente do modelo de mobilidade instanciado anteriormente.

Listing 4.4: Substituição do modelo de direção e velocidades fixas pela instância do objeto *model*

```
public static void nextStep(List<FogDevice> serverCloudlets, List<ApDevice>
    apDevices, List<MobileDevice> smartThings,
                Coordinate coordDevices, int stepPolicy, int seed,
                    MobilityModel model) {


        model.calculateNextDevicesPosition();
}
```

4.2.2 Ferramenta de visualização

Em adição a implementação dos modelos de mobilidade *Random Waypoint*, *Reference Point Group*, *Nomadic Community*, *Pursue* e *Obstacles*, este trabalho apresenta também uma ferramenta de visualização da mobilidade. A visualização gráfica da mobilidade dos dispositivos pode auxiliar a compreensão do cenário simulado, especialmente quando este é mais complexo. A partir das pistas gráficas é possível elaborar hipóteses que estejam relacionadas ao posicionamento dos dispositivos, como, por exemplo, relacionar o aumento na latência das aplicações a partir da visualização de uma concentração de dispositivos em uma mesma região. A Figura 4.3 mostra a ferramenta de visualização de dois modelos diferentes.

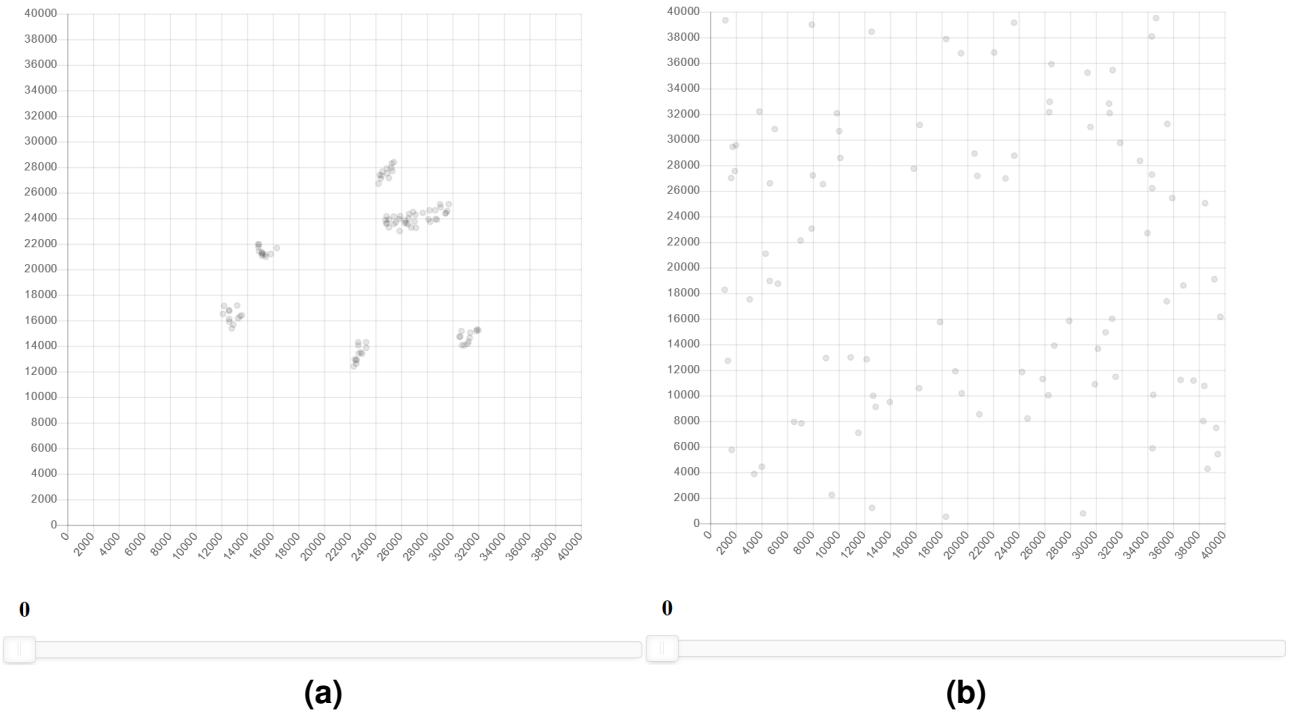


Figura 4.3: Representação da posição dos dispositivos *Smartthings* na ferramenta de visualização dos modelos: **(a)** *Nomadic Community*, **(b)** *Obstacles* (elaborado pelo autor).

A ferramenta de visualização se encontra disponível no diretório log na raiz do *MyiFogSim*. Neste diretório são coletadas as coordenadas de cada dispositivo *Smartthing* e o seu consumo energético a cada instante de simulação. Estes dados são exportados pelos modelos de mobilidade no formato JSON em tempo de execução da simulação. Essa metodologia permite que os dados seja analisados durante a execução da simulação, e não apenas após a conclusão da mesma. Outro aspecto positivo desta abordagem é a manutenção dos dados de log caso a simulação venha a falhar em tempo de execução, seja por falta de memória ou erro de execução da aplicação que está sendo executada pelas *Smartthings*. Em contrapartida, a gravação dos dados em disco durante a execução pode

representar um problema de desempenho em arquiteturas síncronas, onde o computador não consegue gravar os dados no sistema de armazenamento simultaneamente a execução dos passos de simulação. Isto, entretanto, não afeta o resultado da simulação, mas sim o tempo necessário para a execução da mesma, e só deve configurar um problema em computadores com apenas um núcleo de processamento.

A visualização consiste em um arquivo HTML que utiliza bibliotecas javascript para a criação de um gráfico de dispersão. As dimensões X e Y do gráfico representam as coordenadas X e Y do domínio de localização dos dispositivos. O gráfico carrega os dados exportados pelos modelos de mobilidade de todas as *Smartthings* e de todos os instantes de tempo simultaneamente. Em seguida, estes dados são carregados como objetos javascript e categorizados por dispositivo, criando assim uma série de coordenadas para cada dispositivo. Nesta versão, os dados de energia não são carregados para a ferramenta de visualização.

As series de coordenadas são posteriormente filtradas por outro componente, a barra de seleção de tempo. Por padrão, a posição inicial do filtro é o primeiro instante da simulação (t=0). A barra de seleção pode ser ajustada para qualquer instante de tempo simulado, e a alteração no gráfico de dispersão ocorre na mudança de tempo. É possível obter um efeito de continuidade através da mudança periódica do tempo marcado na barra de seleção, cuja velocidade depende apenas da frequência de troca de valor.

4.3 Considerações finais

Este Capítulo apresentou a contribuição deste trabalha à área. Foram detalhados os modelos de mobilidade, como se deu a implementação de cada um deles, incluindo algoritimos, parâmetros de configurações e seus impactos na mobilidade, prototipação e mudanças no simulador *MyiFogSim*. Também foram apresentadas as alterações no fluxo de execuções de simulações, comparações entre trechos do código anterior e a nova implementação com uma arquitetura diferente e também a apresentação da ferramenta de visualização. O Capítulo seguinte descreve a avaliação de desempenho destas contribuições.

5. AVALIAÇÃO DE DESEMPENHO

Este capítulo descreve a metodologia utilizada para a validação da modelagem proposta no Capítulo 4 e analisa os resultados obtidos, comparando-os quanto ao impacto dos modelos na latência das aplicações e escalabilidade dos mesmos quanto ao número de dispositivos simulados.

5.1 Metodologia

A validação da implementação dos modelos de mobilidade no *MyiFogSim* é realizada através da execução de uma aplicação de *Fog Computing* em diferentes cenários de mobilidade, isto é, utilizando diferentes modelos de mobilidade. A aplicação utilizada é a *EEG Tractor Beam Game* (ZGY+14), um jogo onde usuários podem atrair objetos virtuais. A definição da aplicação *EEG Tractor Beam Game* justifica-se pelo uso desta mesma aplicação na proposta do simulador *MyiFogSim* (BDB+17), portanto ao utilizar a mesma aplicação cria-se uma base comparativa entre os trabalhos. A hipótese é de que os modelos de mobilidade podem alterar o desempenho da aplicação, e é através da avaliação de desempenho que a implementação de modelos de mobilidade no simulador *MyiFogSim* é justificada. Ao todo, o jogo *EEG Tractor Beam Game* foi simulado com seis diferentes modelos de mobilidade:

- **Direção e velocidade fixas:** modelo de mobilidade originalmente utilizado na proposta do simulador *MyiFogSim*, sem mudança de direção ou velocidade.
- **Random Waypoint:** modelo de mobilidade aleatório onde não há restrições temporais, espaciais ou geográficas.
- **Reference Point Group:** modelo de mobilidade que introduz uma co-relação entre a posição dos dispositivos.
- **Nomadic Community:** modelo de mobilidade de grupos de dispositivos com velocidade e destino distintos para cada nó.
- **Pursue:** modelo de mobilidade de cenários em que diversos nós tentam capturar um único nó.
- **Obstacles:** modelo onde a movimentação dos nós é influenciada pelo mapa do ambiente simulado, tendo áreas inacessíveis aos nós.

A modelagem original segue os parâmetros descritos no estudo por Lopes, Capretz e Bittencourt (BDB+17). Esta execução é a base de comparação para todos os demais

modelos. Conforme os resultados apresentados no artigo original, a estratégia de migração de máquinas virtuais que apresentou o melhor desempenho foi a “estratégia 3”, que refere-se a migração da máquina virtual completa utilizando como critério a latência mais baixa. Portanto, a “estratégia 3” é utilizada em todos os cenários com migração.

As configurações da aplicação *EEG Tractor Beam Game* e os parâmetros dos modelos de mobilidade também são compartilhados por todos os cenários de simulação. Tais configurações e parâmetros são idênticos aos apresentados pelo trabalho que propôs o *MyiFogSim*, quando existentes. A Figura 5.1 apresenta o fluxo de simulação, onde o simulador *MyiFogSim* recebe como entradas os parâmetros de configuração da simulação, tais como tempo máximo simulado, dimensões do mapa, estratégia de migração e os parâmetros de configuração dos modelos de mobilidade, tais como *TPause*, velocidade máxima de cada dispositivo e tamanho dos grupos de dispositivos, quando aplicável. Os resultados exportados do simulador *MyiFogSim* são o arquivo data.log, contendo o log de posição e energia de cada dispositivo, e a árvore de diretórios *outputLatencies*, contendo a informação detalhada de latência de cada dispositivo a cada ciclo de tempo simulado.

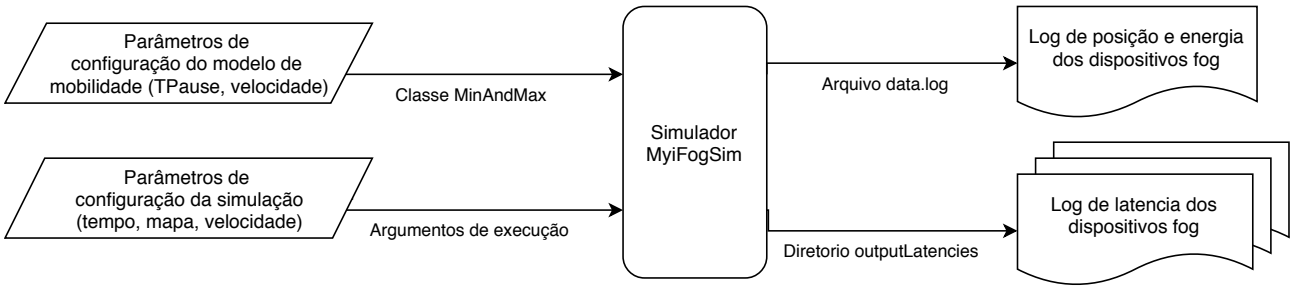


Figura 5.1: Representação das entradas e saídas do MyiFogSim (elaborado pelo autor).

O mapa de simulação utilizado é uma área retangular de 40km x 40km, onde os *cloudlets* estão distribuídos a cada 10km no eixo X e 6.5km no eixo Y. Os *access points*, pontos de conexão sem fio aos dispositivos móveis, estão distribuídos a cada 5km em ambos os eixos. O *handoff*, transição de um dispositivo entre células de cobertura dos *access points*, foi configurado como 1000m, que também é a velocidade máxima de um dispositivo (em m/s). Os *links* de conexão das *smartthings* são de 1Mbps, e o tamanho das máquinas virtuais tem entre 100 e 200MB. A conexão de rede tem entre 10 e 20Mbps. A Figura 5.2 representa o mapa de simulação e a distribuição de *cloudlets* e *access points*, e a Tabela 5.1 detalha as configurações da simulação.

Quanto aos modelos, o valor de *TPause* é de dois ciclos de relógio. O número de dispositivos, originalmente proposto como um único dispositivo na proposta do *MyiFogSim*, foi ampliado para 12, a fim de contemplar as movimentações de modelos de grupo, tais como *Pursue*, *Reference Point Group* e *Nomadic Community*. Por fim, o tempo de execução foi estendido de 1800 segundos para 18000 segundos, totalizando até 5h de tempo simulado em cada cenário. Cada cenário é executado cinco vezes, totalizando 30 execuções independentes, isto é, que não tem relação entre si e não se influenciam de qualquer

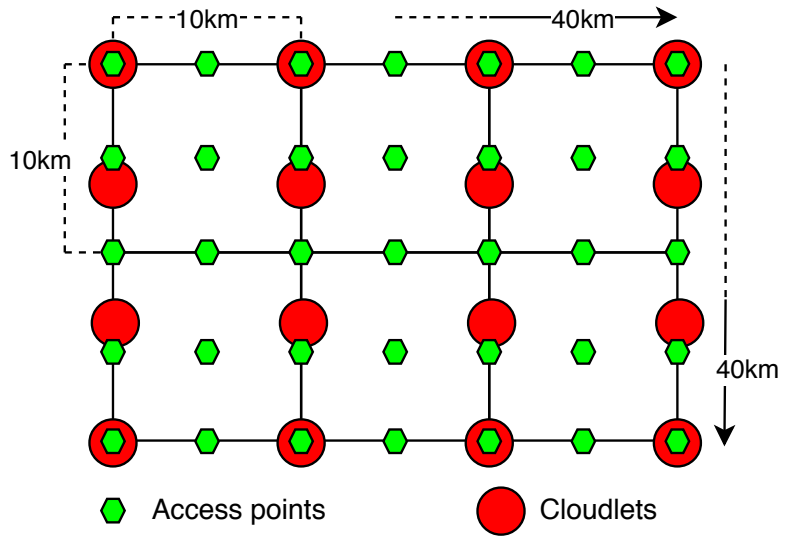


Figura 5.2: Representação do mapa de simulação (elaborado pelu autor).

Descrição da configuração	**Valor utilizado nas simulações**
Mapa	40 x 40 km
Distribuição de Cloudlets	A cada 10 e 6.5 km nos eixos X e Y, respectivamente
Distribuição access points	A cada 5km em ambos eixos
Links de conexão dos dispositivos	1 Mbps
Tamanho da máquina virtual (VM)	100-200 MB
Rede de comunicação	10-20 Mbps
Handoff	1 km
Tempo simulado	18000s (5 horas)
Aplicação fog	*EEG Tractor Beam Game*
Quantidade de dispositivos *Smartthings*	12

Tabela 5.1: Configurações do MyiFogSim para a execução das simulações

maneira, com a simulação de 360 dispositivos e até 150 horas ao todo. Os dados coletados são de posição (coordenada X e Y), energia consumida, latência de cada dispositivo em cada ciclo e tempo total de execução da simulação.

A coleta da posição dos dispositivos se faz necessária para avaliação da ferramenta de visualização. Este dado pode auxiliar na identificação de gargalos ou cenários de migração de máquinas virtuais entre *cloudlets*. Para cada ciclo de simulação são coletadas as coordenadas (posição X e Y no domínio) de todos os dispositivos presentes na simulação. Adicionalmente, os dados de consumo energético de cada dispositivo são exportados juntamente com as coordenadas. O *MyiFogSim* possui ferramentas que permitem ao desenvolvedor a simulação do consumo energético de cada dispositivo de *Fog Computing*. Este dado pode ser relacionado a mobilidade dos dispositivos, por isso justifica-se a sua exportação aos dados de log da simulação. Estes dados, no entanto, não são analisados neste trabalho pois a aplicação *EEG Tractor Beam Game* não faz uso dos recursos de consumo energético.

A latência de comunicação de cada dispositivo é coletada para avaliação do desempenho da aplicação nos diferentes cenários de movimentação. A latência é o atraso na

comunicação entre o dispositivo e o *cloudlet*. Esta é a única métrica utilizada no simulador *MyiFogSim* para avaliação do escalonador de máquinas virtuais, que decide quando estas devem migrar de *cloudlet* conforme o trânsito do seu respectivo dispositivo entre diferentes *access points*, ou seja, a máquina virtual tenta acompanhar a movimentação do dispositivo migrando para outro *cloutlet* mais próximo. O escalonador é quem decide quando esta migração deve ocorrer, e ele o faz utilizando somente a métrica de latência do usuário. Portanto, a coleta desta métrica permite a comparação dos resultados apresentados na implementação original do *MyiFogSim* em relação a esta versão extendida do *MyiFogSim*. A latência é calculada pelas próprias bibliotecas de simulação do pacote *CloudSim* através dos atrasos de rede configurados entre sensores, atuadores, *brokers*, *access points* e *cloudlets*. Em termos gerais, é esperado que quanto maior o caminho a ser percorrido, isto é, quanto mais passos necessários entre a origem e o destino, maior a latência. Isto pode não ser verdadeiro em cenários com redes assíncronas e/ou diferentes larguras de banda e tráfego.

O tempo total de execução, ou seja, o tempo real que o computador precisou para executar a simulação de 18000 segundos, também foi coletado afim de determinar a escalabilidade dos modelos. A escalabilidade se refere a capacidade do sistema de crescimento da complexidade do cenário simulado através do aumento do número de dispositivos presentes na simulação. As configurações dos modelos de mobilidade são detalhados na Tabela 5.2.

Configuração	**Random Waypoint**	**Reference Point Group**	**Nomadic Community**	**Pursue**	**Obstacles**	**MyiFogSim original**
Velocidade máxima	1000 m/s	1000 m/s	1000 m/s	1000 m/s	1000 m/s	1000 m/s
TPause	2	2	2	2	2	N/A
Número de dispositivos	12	12	12	12	12	12
Número de dispositivos por grupo	N/A	3	3	N/A	N/A	N/A
Numero de obstáculos	N/A	N/A	N/A	N/A	1 (26-10km)	N/A

Tabela 5.2: Parâmetros dos modelos de mobilidade para a execução das simulações

Todos os modelos utilizam uma velocidade constante e *TPause* = 2. O valor de *TPause* = 2 consiste em uma pausa nos dispositivos após este ter alcançado o destino, neste caso, de dois ciclos de simulação. Um valor menor torna o cenário mais dinâmico, e, de maneira contrária, um valor mais alto aumenta a previsibilidade e constância do cenário. A velocidade se refere a velocidade dos dispositivos, podendo ser ajustada conforme a natureza do dispositivo simulado, o valor de 1000m/s escolhido justifica-se pela maior incidência de eventos de migração, já que em um mesmo ciclo o dispositivo pode percorrer uma maior distância.

Quanto ao *hardware*, todos os cenários e simulações ocorrem em um computador com processador Intel Core i3 6100, 8GB de memóra RAM DDR4 2133Mhz com armaze-

namento de estado sólido (SSD). O sistema operacional corrente é Windows 10 Pro, build 1803, e a plataforma de execução é o Oracle Java 8, versão 181, build 13.

O trabalho original apresenta como métrica a medida da latência de comunicação da aplicação da perspectiva dos usuários, sendo executada com apenas um usuário. A velocidade e a direção são fixas, e tais características são mantidas na Modelagem Original do *MyiFogSim*, utilizada aqui como base de comparação. Modelagem Original *MyiFogSim* é o nome designado neste trabalho ao modelo de direção e velocidades fixas utilizado na proposta original. Um diferença entre a execução deste trabalho com relação as simulações apresentadas em (BDB+17) é a quantidade de usuários - alterada de 1 para 12 dispositivos - pois este número permite a avaliação de modelos de comportamento em grupo, tais como *Reference Point Group*, *Nomadic Community* e *Pursue*. A latência de cada usuário foi coletada individualmente através do diretório *outputLatencies*, conforme indicado na Figura 5.1.

5.2 Análise dos resultados

Esta seção analisa os dados coletados através da experimentação detalhada na Seção 5.1. As principais métricas são: (i) a latência das aplicações de cada usuário e, (ii) o tempo total de execução de cada modelo de mobilidade. As latências coletadas foram agrupadas para análise dos dados da seguinte forma: para cada modelo de mobilidade foram executadas 5 simulações distintas. Cada simulação foi executada com 12 *smartthings*. Cada simulação teve a duração máxima de 18000s em tempo simulado, e a cada segundo simulado a latência de todos os dispositivos *smartthings* foi registrada.

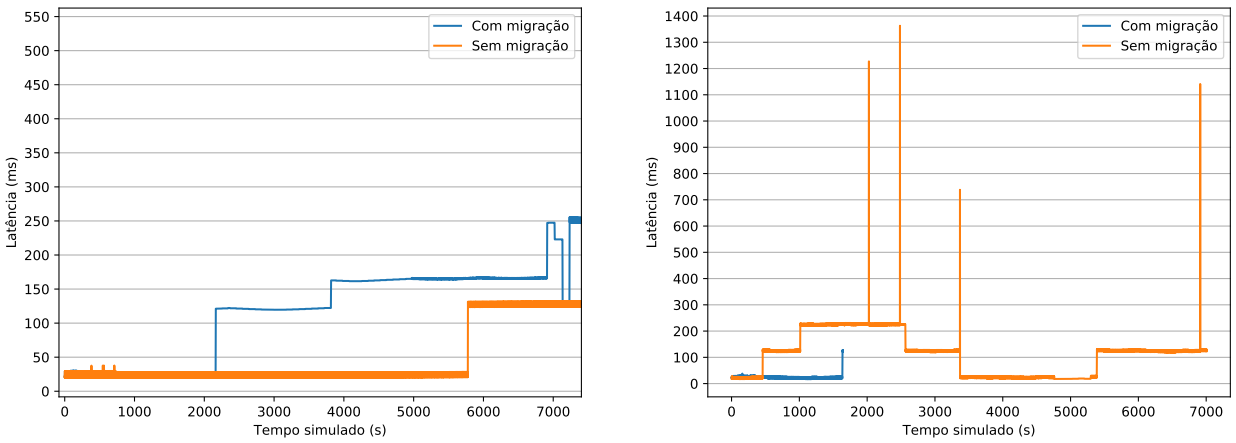
O agrupamento das latências se deu de acordo com o tempo, isto é, as latências do instante 1 dos 12 usuários foram somadas e divididas pelo número de usuários, resultando na latência média do instante 1 da primeira execução do primeiro modelo de mobilidade. Para facilitar a compreensão, pela palavra *combinada* a partir daqui, entenda-se *fazer a média dos resultados A e B*. O mesmo cálculo de média foi realizado para o instante 2, e assim por diante, resultando em cerca de 16 mil resultados médios, cada um correspondente a um instante de tempo naquela execução. O processo foi repetido para as 4 execuções restantes, resultando em cinco vezes de 16 mil latências médias para cada modelo de mobilidade. As 5 latências médias de cada instante de simulação foram, então, combinadas, resultando em uma série de 16 mil latências para cada modelo de mobilidade. Cada uma destas séries representa a média das execuções das médias dos usuários para cada modelo de mobilidade. Estas séries finais são chamadas neste trabalho de tendência, pois representam a tendência de latência de cada modelo.

Quanto aos usuários em uma mesma execução, é possível observar que as latências possuem uma grande variação. O modelo *Random Waypoint*, por exemplo, possui

picos máximos que variam de 1096ms a 9122ms. Porém, após o primeiro agrupamento, os picos de latência são absorvidos pela média e obtém-se a tendência do modelo naquela execução específica, pois a media das latências converge com o incremento do tempo. A variação média (desvio padrão médio dentro de todos os instantes de simulação) do modelo *Random Waypoint* foi de 81,25ms, com desvio padrão de aproximadamente 41ms. Os picos de latência estão relacionados com eventos de migração de máquinas virtuais entre diferentes *cloudlets*.

A latência dos usuários, critério utilizado originalmente na proposta de escalonamento de máquinas virtuais no *MyiFogSim*, pode ser avaliada quanto aos modelos de mobilidade, isto é, quais impactos os modelos de mobilidade podem causar à latência dos usuários. O escalonador proposto por (BDB+17) se mostrou adequado em cenários que utilizam o modelo de mobilidade *Reference Point Group* e 12 dispositivos, porém o mesmo escalonador obteve uma latência maior em um cenário com o modelo *Random Waypoint* quando comparado ao mesmo cenário sem a utilização do escalonador. A Figura 5.3 apresenta os gráficos **(a)** e **(b)**, onde a série "com migração" representa a latência em um cenário com o uso do escalonador de máquinas virtuais e a série "sem migração" representa a latência no mesmo cenário sem o uso do escalonador de máquinas virtuais.

De maneira semelhante ao modelo *Reference Point Group*, os modelos *Nomadic Community*, *Pursue* e *Obstacles* obtiveram menor latência com o uso do escalonador proposto por (BDB+17). A Figura 5.4 e a Figura 5.5 apresentam a latência de dispositivos em cenários com e sem a migração de máquinas virtuais. Um aspecto que chama atenção é o fato de que frequentemente as aplicações são finalizadas antecipadamente nos cenários com migração de máquinas virtuais, especialmente nos modelos *Reference Point Group* e *Nomadic Community*. Estes eventos podem estar ligados a finalização da aplicação do jogo, onde o resultado é a terminação da aplicação antes do prazo máximo da simulação.



(a) **(b)**

Figura 5.3: Latência da aplicação do usuário em diferentes cenários de migração de máquinas virtuais: **(a)** *Random Waypoint*, **(b)** *Reference Point Group* (elaborado pelo autor).

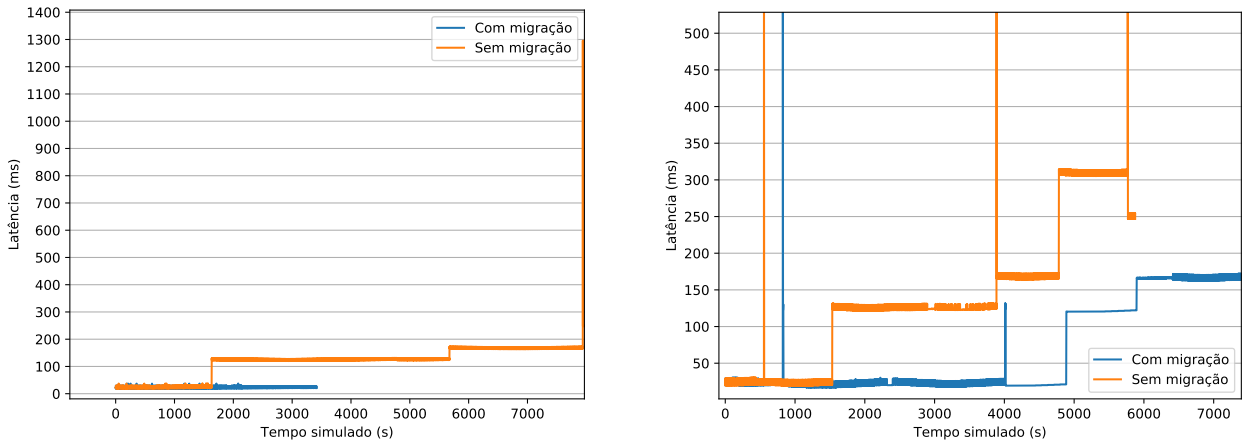


Figura 5.4: Latência da aplicação do usuário em diferentes cenários de migração de máquinas virtuais: **(c)** Nomadic Community, **(d)** Pursue (elaborado pelo autor).

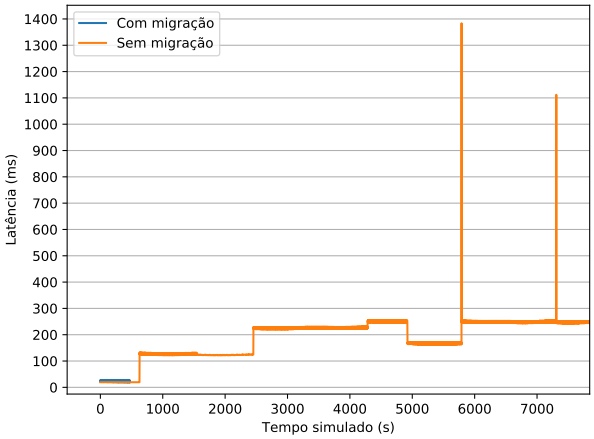


Figura 5.5: Latência da aplicação do usuário em diferentes cenários de migração de máquinas virtuais com o modelo Obstacles (elaborado pelo autor).

A segunda métrica, o tempo de execução de cada simulação, foi coletada afim da obtenção da escalabilidade de cada modelo. A escalabilidade é a capacidade de um determinado sistema de lidar com um crescimento de dados ou de requisições, sendo assim, uma característica desejável a grande maioria dos sistemas. Pretende-se avaliar aqui se a adição dos modelos de mobilidade ou as mudanças propostas na arquitetura do *MyiFogSim* impactaram a escalabilidade do sistema.

A Figura 5.6 apresenta a média do tempo de execução de cada modelo com 12 dispositivos simulados e os respectivos desvios. Todos os modelos de mobilidade implementados, exceto o modelo *Random Waypoint*, obtiveram um tempo de execução inferior ao modelo de direção e velocidade fixas utilizado na proposta original de escalonador do simulador *MyiFogSim* (BDB+17). O modelo *Random Waypoint* teve um impacto médio de -3,67%, enquanto o modelo *Pursue* obteve um ganho de 7,14% de desempenho em relação ao modelo de direção e velocidade fixas. Os modelos *Reference Point Group* e *Nomadic*

Community obtiveram ganhos de 103,87% e 118,83% respectivamente, e o modelo *Obstacles* obteve 477,69%, o maior ganho de desempenho observado.

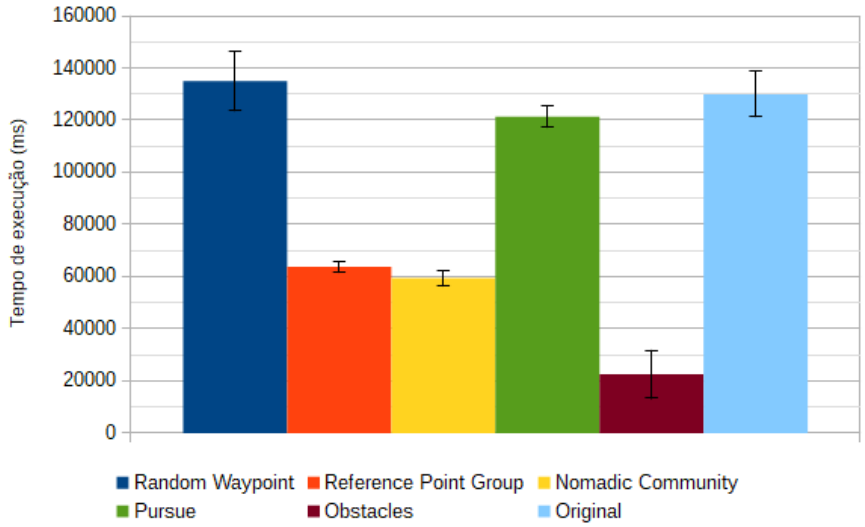


Figura 5.6: Comparação dos modelos quanto ao tempo médio de execução com 12 dispositivos (elaborado pelo autor).

Este ganho de desempenho esta relacionado a finalização antecipada das aplicações. A presença de mais usuários e a proximidade deles acelerou o processamento da aplicação, que finalizou antecipadamente. Então a mudança de modelos de mobilidade não só influenciou a latência da aplicação, como também a sua duração total.

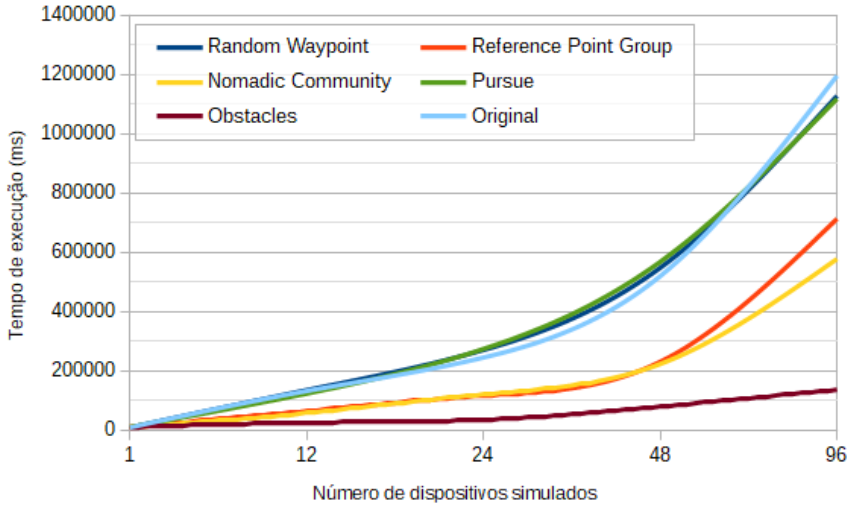


Figura 5.7: Comparação dos modelos quanto a escalabilidade (elaborado pelo autor).

Quanto a escalabilidade, o modelo *Obstacles* se mostra mais escalável que os demais, seguido pelos modelos *Nomadic Community* e *Reference Point Group*. Por fim, os modelos *Pursue*, *Random Waypoint* e o modelo de direção e velocidades fixas possuem escalabilidade similares, conforme indicado na Figura 5.7 (b). O número de dispositivos avaliados foi de 1 a 96 dispositivos, em cinco execuções para cada modelo em cada cenário.

Portanto é possível observar que os modelos de mobilidade podem impactar na latência das aplicações dos usuários. O escalonador de máquinas virtuais proposto com o *MyiFogSim* obteve um desempenho inferior em um cenários de mobilidade diferentes do modelo de direção e velocidade fixas. A implementação dos modelos de mobilidade propostos neste trabalho aproxima as simulações de *Fog Computing* de cenários reais e permite o desenvolvimento de ferramentas de escalonamento e aplicações mais eficientes.

6. CONCLUSÃO

Neste trabalho são descritas a implementação e a avaliação dos modelos de mobilidade *Random Waypoint*, *Reference Point Group*, *Nomadic Community*, *Pursue* e *Obstacles* no simulador de ambiente de *Fog Computing MyiFogSim*. Quanto à avaliação dos modelos de mobilidade, esta foi feita utilizando algoritmos para escalonamento de serviços em ambientes de *Fog Computing*, sendo analisado o impacto que diferentes modelos de mobilidade têm sobre a escalabilidade e latência das aplicações e também sobre o escalonador de máquinas virtuais do *MyiFogSim*. Os resultados obtidos demonstram que os modelos de mobilidade podem impactar na latência das aplicações dos usuários, sendo assim, a implementação dos modelos de mobilidade propostos neste trabalho aproxima as simulações de *Fog Computing* de cenários reais e permite o desenvolvimento de ferramentas de escalonamento e aplicações mais eficientes.

Este trabalho também propõe uma ferramenta de visualização de dados de mobilidade. Estes dados são coletados em tempo-real e consistem nas posições dos dispositivos em um domínio durante a simulação. Através da análise visual desta ferramenta é possível obter uma compreensão mais rápida do cenário simulado. Este recurso não está presente nos trabalhos relacionados da mesma categoria de simulação na qual se encontra o *MyiFogSim*.

A implementação dos modelos de mobilidade permite a simulação de cenários mais próximos aos reais no que tange a mobilidade dos dispositivos. Cenários como campus de universidades com obstáculos, praças públicas de livre circulação, conferências e situações de guerra em que a movimentação ocorra por grupos também podem ser simulados com mais fidelidade dentro da ferramenta, o que abre possibilidade para o desenvolvimento de soluções mais precisas e eficientes.

A continuação deste trabalho será na expansão dos modelos de mobilidade implementados, englobando também modelos com dependência temporal, tais como os de funções de suavização. A ferramenta de mobilidade pode ser aprimorada no que tange a seleção de fonte de dados, opções para visualização dos *access-points* e *cloudlets* e mudança de cor dos dispositivos de acordo com o consumo energético. A expansão de testes e a otimização do escalonador de máquinas virtuais também se mostra uma área interessante a ser explorada, dadas as possibilidades de otimização que são possíveis a partir da implementação destes modelos de mobilidade no *MyiFogSim*.

REFERÊNCIAS BIBLIOGRÁFICAS

[AaJD+16] Al-ayyoub, M.; Jararweh, Y.; Doulat, A.; Alqudah, O.; Ahmed, E.; Al-ayyoub, M. “The Future of Mobile Cloud Computing : Integrating Cloudlets and Mobile Edge Computing”. In: 23rd International Conference on Telecommunications, 2016, pp. 760–764.

[ABJAA+17] Al-Badarneh, J.; Jararweh, Y.; Al-Ayyoub, M.; Al-Smadi, M.; Fontes, R. “Software Defined Storage for Cooperative Mobile Edge Computing Systems”. In: 4th International Conference on Software Defined Systems, 2017, pp. 174–179.

[AGC17] Alonso-Monsalve, S.; García-Carballeira, F.; Calderón, A. “Fog Computing Through Public-Resource Computing and Storage”. In: 2nd International Conference on Fog and Mobile Edge Computing, 2017, pp. 81–87.

[AIM10] Atzori, L.; Iera, A.; Morabito, G. “The Internet of Things: A Survey”, *Computer Networks*, vol. 54–15, Outubro 2010, pp. 2787 – 2805.

[BDB+17] Bittencourt, L. F.; Diaz-Montes, J.; Buyya, R.; Rana, O. F.; Parashar, M. “Mobility-Aware Application Scheduling in Fog Computing”, *IEEE Cloud Computing*, vol. 4–2, Março 2017, pp. 26–35.

[BEF+00] Breslau, L.; Estrin, D.; Fall, K.; Floyd, S.; Heidemann, J.; Helmy, A.; Huang, P.; McCanne, S.; Varadhan, K.; Ya Xu; Haobo Yu. “Advances in Network Simulation”, *Computer*, vol. 33–5, Maio 2000, pp. 59–67.

[BH04] Bai, F.; Helmy, A. “A Survey of Mobility Models in Wireless Adhoc Networks”. University of Southern California, 2004, cap. 1, pp. 1–30.

[BMJ+98] Broch, J.; Maltz, D. A.; Johnson, D. B.; Hu, Y.-C.; Jetcheva, J. “A Performance Comparison of Multi-hop Wireless Ad Hoc Network Routing Protocols”. In: 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking, 1998, pp. 85–97.

[BMZA12] Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. “Fog Computing and Its Role in the Internet of Things”. In: 1st MCC Workshop on Mobile Cloud Computing, 2012, pp. 13–16.

[CD16] Corcoran, P.; Datta, S. K. “Mobile-Edge Computing and the Internet of Things for Consumers: Extending Cloud Computing and Services to the Edge of the Network.”, *IEEE Consumer Electronics Magazine*, vol. 5–4, Setembro 2016, pp. 73–74.

[CGP17] Coutinho, A. A. T. R.; Greve, F.; Prazeres, C. “An Architecture for Fog Computing Emulation”. In: Anais do XV Workshop em Clouds e Aplicações, 2017, pp. 101–114.

[Con17a] Consortium, O. “OpenFog Reference Architecture for Fog Computing”, Relatório Técnico, OpenFog Consortium, 2017, 162p.

[Con17b] Consortium, O. “The OpenFog Consortium Reference Architecture : Executive Summary”, Relatório Técnico, OpenFog Consortium, 2017, 10p.

[CZ16] Chiang, M.; Zhang, T. “Fog and IoT: An Overview of Research Opportunities”, *IEEE Internet of Things Journal*, vol. 3–6, Dezembro 2016, pp. 854–864.

[GDGB16] Gupta, H.; Dastjerdi, A. V.; Ghosh, S. K.; Buyya, R. “iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments”, *Computing Research Repository*, vol. 1606.02007, Agosto 2016, pp. 1–22.

[GNCG16] Gupta, H.; Nath, S. B.; Chakraborty, S.; Ghosh, S. K. “SDFog: A Software Defined Computing Architecture for QoS Aware Service Orchestration over Edge Devices”, *Computing Research Repository*, vol. 1609.01190, Setembro 2016, pp. 1–8.

[GRdC14] Gomes, M. M.; Righi, R. d. R.; da Costa, C. A. “Future Directions for Providing Better IoT Infrastructure”. In: ACM International Joint Conference on Pervasive and Ubiquitous Computing, 2014, pp. 51–54.

[HPS+15] Hu, Y. C.; Patel, M.; Sabella, D.; Sprecher, N.; Young, V. “Mobile Edge Computing A Key Technology Towards 5G”, Relatório técnico, European Telecommunications Standards Institute, 2015, 16p.

[Jai91] Jain, R. “The Art of Computer Systems Performance Analysis - Techniques for Experimental Design, Measurement, Simulation and Modeling.” Wiley, 1991, 720p.

[JBC09] Jouve, W.; Bruneau, J.; Consel, C. “DiaSim: A Parameterized Simulator for Pervasive Computing Applications”. In: 7th Annual IEEE International Conference on Pervasive Computing and Communications, 2009, pp. 1–3.

[JLH+99] Johansson, P.; Larsson, T.; Hedman, N.; Mielczarek, B.; Degermark, M. “Scenario-based Performance Analysis of Routing Protocols for Mobile Ad-hoc Networks”. In: 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking, 1999, pp. 195–206.

[LGL+15] Luan, T. H.; Gao, L.; Li, Z.; Xiang, Y.; Sun, L. “Fog Computing: Focusing on Mobile Users at the Edge”, *Computing Research Repository*, vol. 1502.01815, Agosto 2015, pp. 1–11.

[LHCB17] Lopes, M. M.; Higashino, W. A.; Capretz, M. A.; Bittencourt, L. F. “MyiFogSim: A Simulator for Virtual Machine Migration in Fog Computing”. In: 10th International Conference on Utility and Cloud Computing, 2017, pp. 47–52.

[LMC03] Legrand, A.; Marchal, L.; Casanova, H. “Scheduling Distributed Applications: the SimGrid Simulation Framework”. In: 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003, pp. 1–9.

[MG11] Mell, P.; Grance, T. “The NIST Definition of Cloud Computing”, Relatório técnico, National Institute of Standards and Technology, 2011, 7p.

[MGG+17] Mayer, R.; Graser, L.; Gupta, H.; Saurez, E.; Ramachandran, U. “EmuFog: Extensible and Scalable Emulation of Large-scale Fog Computing Infrastructures”. In: IEEE Fog World Congress, 2017, pp. 1–6.

[MKB18] Mahmud, R.; Kotagiri, R.; Buyya, R. “Fog Computing: A Taxonomy, Survey and Future Directions”. Springer Singapore, 2018, cap. 4, pp. 103–130.

[NKI+15] Nunna, S.; Kousaridas, A.; Ibrahim, M.; Dillinger, M.; Thuemmler, C.; Feussner, H.; Schneider, A. “Enabling Real-Time Context-Aware Collaboration through 5G and Mobile Edge Computing”. In: 12th International Conference on Information Technology: New Generations, 2015, pp. 601–605.

[OBL15] Orsini, G.; Bade, D.; Lamersdorf, W. “Computing at the Mobile Edge: Designing Elastic Android Applications for Computation Offloading”. In: 8th Wireless and Mobile Networking Conference, 2015, pp. 112–119.

[OSMF17] Oliveira, R. R.; Schirmer, N. M.; Machry, M.; Ferreto, T. C. “A Transparent Code Offloading Technique for Android Devices”. In: 13th International Wireless Communications and Mobile Computing Conference, 2017, pp. 1078–1083.

[RH10] Riley, G. F.; Henderson, T. R. “Modeling and Tools for Network Simulation”. Springer Berlin Heidelberg, 2010, 545p.

[San12] Santi, P. “Mobility Models for Next Generation Wireless Networks: AD HOC, Vehicular and Mesh Networks”. Wiley, 2012, 374p.

[SG16] Slabicki, M.; Grochla, K. “Performance Evaluation of CoAP, SNMP and NETCONF Protocols in Fog Computing Architecture”. In: IEEE/IFIP Network Operations and Management Symposium, 2016, pp. 1315–1319.

[SHFW10] Sundmaeker, H.; Huillemin, P.; Friess, P.; Woelfflé, S. “IoT Vision and Challenges for Realising the Internet of Things”. Luxembourg Publications Office of the European Union, 2010, 229p.

[SLQ19] Suter, F.; Legrand, A.; Quinson, M. “Simgrid Documentation”. Capturado em: http://simgrid.gforge.inria.fr/simgrid/3.20/doc/uhood_arch.html, Fevereiro 2019.

[SM01] Sánchez, M.; Manzoni, P. “ANEJOS: a Java Based Simulator for Ad Hoc Networks”, *Future Generation Computer Systems*, vol. 17, Março 2001, pp. 573–583.

[SOE17] Sonmez, C.; Ozgovde, A.; Ersoy, C. “EdgeCloudSim: An Environment for Performance Evaluation of Edge Computing Systems”. In: 2nd International Conference on Fog and Mobile Edge Computing, 2017, pp. 39–44.

[WV16] Wang, G.; Venkatasubramanian, N. “Network-aware Edge Service for Resilient Cloud-based Internet-of-Things Applications”, Dissertação de Mestrado, University of California, 2016, 48p.

[YFN+18] Yousefpour, A.; Fung, C.; Nguyen, T.; Kadiyala, K.; Jalali, F.; Niakanlahiji, A.; Kong, J.; Jue, J. P. “All One Needs to Know about Fog Computing and Related Edge Computing Paradigms: A Complete Survey”, *Computing Research Repository*, vol. 1808.05283, Agosto 2018, pp. 1–48.

[YLN03] Yoon, J.; Liu, M.; Noble, B. “Sound Mobility Models”. In: 9th Annual International Conference on Mobile Computing and Networking, 2003, pp. 205–2016.

[ZGY+14] Zao, J. K.; Gan, T. T.; You, C. K.; Méndez, S. J. R.; Chung, C. E.; Wang, Y. T.; Mullen, T.; Jung, T. P. “Augmented Brain Computer Interaction Based on Fog Computing and Linked Data”. In: International Conference on Intelligent Environments, 2014, pp. 374–377.