Universidade Federal de Campina Grande - UFCG
Centro de Engenharia Elétrica e Informática - CEEI
Programa de Pós-Graduação em Engenharia Elétrica - PPgEE
Fernanda Bezerra Gómez Famá

Dissertação de Mestrado

Um Arcabouço para o Desenvolvimento e Simulação de Aplicações de Internet das Coisas em um Cenário de Computação na Borda

Campina Grande - PB

2020

Universidade Federal de Campina Grande - UFCG
Centro de Engenharia Elétrica e Informática - CEEI
Programa de Pós-Graduação em Engenharia Elétrica - PPgEE
Fernanda Bezerra Gómez Famá

Dissertação de Mestrado

Um Arcabouço para o Desenvolvimento e Simulação de Aplicações de Internet das Coisas em um Cenário de Computação na Borda

Dissertação apresentada ao Programa de Pós Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande - Campus de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento da Informação

Orientador: Angelo Perkusich

Coorientador: Danilo F. S. Santos

Campina Grande - PB

2020

F198a Famá, Fernanda Bezerra Gómez.

Um arcabouço para o desenvolvimento e simulação de aplicações de internet das coisas em um cenário de computação na borda / Fernanda Bezerra Gómez Famá. – Campina Grande, 2020.

83 f. : il. color.

Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2020.

"Orientação: Prof. Dr. Angelo Perkusich, Prof. Dr. Danilo Freire de Souza Santos".

Referências.

1. Processamento da Informação. 2. Computação na Borda. 3. Internet das Coisas. 4. Middleware. 5. Simulação. I. Perkusich, Angelo. II. Santos, Danilo Freire de Souza. III. Título.

CDU 621.391:004(043)

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECÁRIA SEVERINA SUELI DA SILVA OLIVEIRA CRB-15/225

Agradecimentos

Agradeço primeiramente a minha família, por todo o apoio durante todos esses anos, em especial a minha mãe, por sempre acreditar em mim e em meus sonhos.

Agradeço também a Pedro pelo companheirismo e suporte durante a graduação e o mestrado.

Aos meus amigos que escutaram minhas reclamações, pelas risadas compartilhadas e abraços carinhosos, em especial a Paulo, Raissa, Cleuves e Gabriel. As diversas mulheres que me apoiaram durante essa trajetória e me ensinaram de diversas formas possíveis, em especial as do Espaço Florescer e as da dança.

Aos professores do Embedded pelo apoio, paciência e ensinamentos, em especial aos meus orientadores Angelo e Danilo, a Kyller e Zé, e aos professores que compõem a banca, Jaidilson e Alexandre.

Por fim, aos colegas do Embedded que tornam meus dias muito mais leves e divertidos. E a CAPES pelo suporte financeiro.

"Quando uma pessoa vive de verdade,
outras também vivem. Esse é o principal
imperativo da mulher sábia. Viver para que os
outros também se inspirem. Viver do nosso próprio
jeito vibrante para que os outros aprendam conosco."
(Clarissa Pinkola Estés)

Resumo

O aumento da quantidade de dispositivos conectados à Internet potencializou o desenvolvimento de diversas aplicações e serviços em diferentes setores da indústria e engenharia. No entanto, a possibilidade de utilização desses dispositivos em diferentes aplicações ficam limitadas à capacidade da rede e seus componentes. Por exemplo, para aplicações que necessitam de um tempo de resposta curto, um cenário de rede onde a maior parte dos dados são processados na nuvem, pode não ser viável. Visando dispor de um ambiente que permita o desenvolvimento dessas aplicações e serviços, surge o paradigma de Computação na Borda, o qual possibilita o processamento de dados próximo da sua fonte geradora. Este novo paradigma permite unir o melhor dos dois mundos, o processamento de dados rápidos por uma rede distribuída de equipamentos na borda da rede, e a alta capacidade de armazenamento e processamento dos *data centers* na nuvem. Considerando esse contexto, um dos desafios inerentes a esse novo paradigma está em como desenvolver aplicações e validá-las em um ambiente integrado de desenvolvimento e simulação. Tendo em mente esse desafio, esse trabalho propõe um arcabouço que permite que aplicações para a Internet das Coisas sejam desenvolvidas utilizando um *Middleware* de aplicações IoT, e sejam validadas em um ambiente de simulação de rede de Computação na Borda. Para isso, é apresentada uma proposta de arcabouço de integração que visa unir duas ferramentas distintas, o *Middleware* Node-RED e o FogNetSim++, o qual é uma extensão do OMNeT++, através de um gerenciador que controla a troca de dados entre as mesmas. Detalhes de como foi realizada a escolha das ferramentas, e as adaptações realizadas que permitem a integração entre essas, são apresentadas neste trabalho. Além disso, são apresentados resultados experimentais do funcionamento dessa integração em um caso de uso de validação. Esses testes demonstram como o sistema reage ao se modificar parâmetros como intervalo de envio e recebimento de mensagens, tipo de mobilidade utilizada pelos dispositivos, quantidade de dispositivos na rede, entre outras modificações no cenário teste. Com isso, a depender dos requisitos da aplicação de cada desenvolvedor, é possível moldar o ambiente de simulação as suas necessidades, e avaliar o comportamento da aplicação antes do seu lançamento.

Palavras-chave: Computação na Borda, *Middleware*, Simulação, Internet das Coisas.

Abstract

The increase in the number of devices connected to the Internet has encouraged the development of several applications and services in different sectors of industry engineering. However, the possibility of using these devices in different applications is limited to the capacity of the network and its components. For instance, for applications that require a short response time, a network scenario where most of the data is processed in the cloud, may not be feasible. In order to have an environment that allows the development of these applications and services, the Edge Computing paradigm emerges, which enables data processing close to its generating source. This new paradigm enables to combine the best of both worlds, the fast data processing by a distributed network of equipment at Edge Computing, and the high storage and processing capacity of data centers in the cloud. Considering this context, one of the challenges inherent to this new paradigm is how to develop applications and validate them in an development and simulation environment. With this challenge in mind, this work proposes a framework that allows IoT applications to be developed using an IoT application middleware, and to validated in a Edge Computing network simulation environmet. For that, a proposal of integration framework is presented that aims to unite two distinct tools, Middleware Node-RED and FogNetSim ++, which is an extension of OMNeT ++, through a manager that controls the data exchange between them. Details of how the choice of tools was made, and the adaptations made that allow integration between the them, are presented in this work. In addition, experimental results of the operation of integration are presented in a validation use case. These tests demonstrate how the system reacts when modifying parameters such as interval of sending and receiving messages, type of mobility used by the devices, number of devices on the network, among other changes in the test scenario. Thus, depending on the application requirements of each developer, it is possible to tailor the simulation environment to their needs, and evaluate the behavior of the application before its launch.

Keywords: Edge Computing, Middleware, Simulation, Internet of Things.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
CoAP	Constrained Application Protocol
DEVS	Discret Event Sytems
D-NR	Distributed Node-RED
EC	Edge Computing
FIFO	First-in-First-Out
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Internet Protocol
IPC	Inter-Process Comunication
M2M	Machine-to-machine
MEC	Mobile Edge Computing
MPI	Message Passing Interface
MQTT	Message Queue Telemetry Transport
P2P	Peer-to-peer
PPP	Point-to-Point Protocol
QoS	Quality of Service
RACS	Radio Application Cloud Server
REST	Representational State Transfer
SDK	Software Development Kit

SMTP	Simple Mail Transfer Protocol
SO	Sistema Operacional
SOA	Service-Oriented Architecture
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UPnP	Universal Plug and Play
URI	Uniform Resource Identifier
XML	Extensible Markup Language
WLAN	Wireless LAN

Sumário

1 Introdução

O interesse por dispositivos que agregam serviços com o uso da Internet tornou-se algo comum no mundo moderno ao qual estamos inseridos. A Internet das Coisas (*do inglês: Internet of Things* - IoT) veio como uma forma de unir diversos setores e seus equipamentos a esse mundo virtualizado. Esse paradigma traz consigo uma infinidade de possibilidades de sistemas que podem ser monitorados, controlados e interligados entre si nos mais diversos setores, tais como, automobilístico, médico, industrial, residencial e alimentício. Esse conjunto de dispositivos melhora o controle e auxilia na tomada de decisões dessas áreas, porém, aumenta radicalmente a quantidade de dados que devem ser transmitidos via Internet, normalmente via rede sem fio, para serem analisados e tratados na Nuvem, para só então retornarem a seus dispositivos para que reajam conforme pré-definido pelo usuário [65].

Devido ao grande volume de dados gerados pela Internet das Coisas, as empresas vêm integrando essa tecnologia ao paradigma da Computação na Nuvem (do inglês: *Cloud Computing*) em razão dos recursos oferecidos. Esse tipo de computação oferece serviços que permitem acesso a infraestrutura, *software* e informações através de algum dispositivo conectado à Internet. Logo, uma tecnologia acaba por complementar a outra. No entanto, apesar do uso da nuvem contribuir para o avanço de alguns serviços que utilizam IoT, quando tratamos aplicações que demandam um tempo de resposta curto (por exemplo, em torno de uma dezena de milissegundos) essa tecnologia pode não ser satisfatória. A Computação na Borda (do inglês: *Edge Computing*) surge para solucionar esse problema, já que visa alocar alguns recursos armazenados na nuvem para a borda da rede. De forma mais simples, esse paradigma tem como objetivo trazer a computação para mais próximo de onde a informação é gerada. As duas possuem abordagens distintas, nos serviços de Computação na Nuvem a abordagem geralmente é armazenar todos os dados na nuvem e depois analisar o que é necessário ou não para uma dada implementação. Já na Computação na Borda temos uma abordagem diferente, com foco em captar apenas o que é necessário para só então enviar os dados para a nuvem [65].

Dessa forma, *Edge Computing* (EC) é um paradigma que visa processar parte dos dados na borda da rede, ou seja, mais próximo da fonte de dados, em vez de percorrer longos percursos pela rede até serem processados em um serviço em nuvem sendo executado em um *data center* distante. Com isso, é possível reduzir o consumo de energia da rede como um todo, reduzir a necessidade de largura de banda para *backbones*, diminuir a latência de resposta aos serviços para os dispositivos finais, e melhorar quesitos como segurança dos dados e privacidade. Atrelado ao conceito de Computação na Borda está o conceito de Computação na Névoa (em inglês, *Fog Computing*) [65]. Dentro do contexto

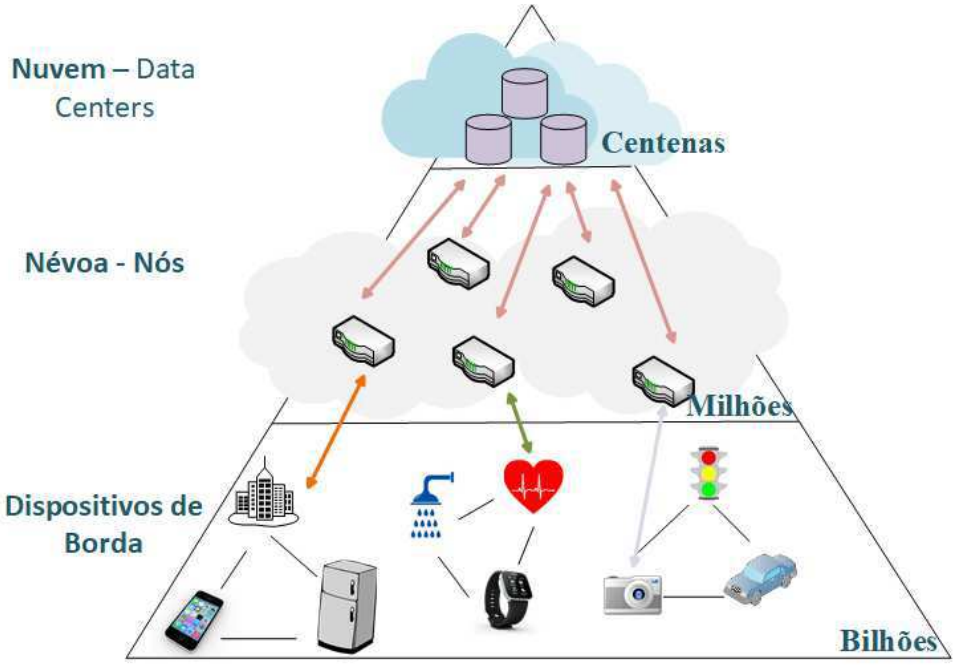

Figura 1 – Representação do novo cenário de rede

deste trabalho, os conceitos são considerados equivalentes, entretanto, de um ponto de vista mais amplo podemos considerar que a Computação na Borda é mais voltada para os desafios e soluções computacionais relacionados aos serviços e aplicações próximas aos dispositivos finais, enquanto o *Fog Computing* tem um foco maior em aspectos relacionados a infraestrutura de rede e computação distribuída entre equipamentos na borda, como ilustrado na Figura 1. Nessa Figura, é apresentada uma representação da quantidade de dispositivos em cada camada da rede. A medida que o processamento irá se distanciando dos dispositivos que geraram os dados, a quantidade de dispositivos reduz. O paradigma da *Fog Computing*, atua nos dispositivos computacionais e de infraestrutura, como roteadores, *switches* e estações rádio base, localizados na névoa (entre o dispositivo final e a nuvem), de modo que os mesmos possuam uma configuração adaptável às diferentes características dos nós e serviços na borda, acarretando, então, uma redução do gargalo na transferência de dados para a nuvem ou redes móveis. A utilização de rede em automóveis, é um exemplo de uso de Computação Móvel. Plataformas atuais já implementam e gerenciam serviços móveis interligados a nuvem, porém enfrentam desafios consideráveis em relação à flexibilidade, cobertura, confiabilidade e segurança [33]. Por exemplo, considerando um cenário onde um gigabyte de dados é gerado por um carro autônomo a cada segundo, como ilustrado no diagrama da Figura 2, é necessário que os dados sejam processados dentro de um determinado período de tempo para que o veículo execute uma determinada ação de forma correta dentro do seu prazo. Se todos esses dados forem enviados para a nuvem para serem processados, o tempo de resposta pode ser longo demais, considerando os prazos e requisitos temporais de resposta do sistema. Além disso, a quantidade de veículos gera

uma série de indagações ao que se refere a largura de banda e a confiabilidade da rede atual [65].

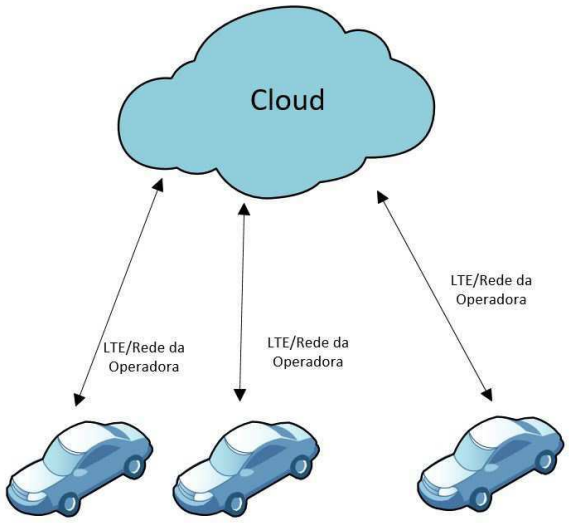

Figura 2 – Representação da implementação de serviços em automóveis utilizando unicamente Computação na Nuvem.

Apesar da base da *Edge Computing* ser datada por volta de 1990, o uso desse tipo de computação para serviços móveis surgiu há pouco tempo. Esse tipo de solução permite um acesso contínuo a um sistema com conexões de rede de alta largura de banda. A tecnologia *Mobile Edge Computing - MEC* é uma evolução dos serviços de *Cloudlets*, que é composto por um *data center* projetado para fornecer serviços rápidos de computação na nuvem a dispositivos móveis em uma pequena área utilizando rede sem fio com uma alta largura de banda [62]. Em virtude dessa tecnologia possuir recursos limitados, e um curto alcance oferecido pela conexão Wi-Fi, empresas como a Nokia, juntamente com a IBM introduziram o *Radio Applications Cloud Server - RACS* que é uma plataforma de EC para redes 4G/LTE no início de 2003 [62].

A tecnologia de *Mobile Edge Computing* tende a melhorar a eficiência do descarregamento dos dados da nuvem. Os recursos de Computação na Nuvem são fornecidos dentro da rede de acesso de rádio na proximidade dos dispositivos móveis, possibilitando o descarregamento de suas tarefas para os servidores MEC na borda da rede, em vez de usar servidores centralizados [78]. Ao permitirmos vários tipos de acesso a essa tecnologia, permite-se que as operadoras deixem suas redes disponíveis para um novo sistema, criando então uma rede de acessos múltiplos, chamada *Multi-Acess Edge Computing*. Segundo o artigo apresentado pelo SDxCentral em [63], "*O principal objetivo do acesso múltiplo é reduzir o congestionamento da rede e melhorar o desempenho das aplicações ao conseguir o processamento de tarefas relacionadas mais perto do usuário*". Algumas empresas multi-

nacionais de telecomunicações já oferecem plataformas de *Multi-access Edge Computing* que prometem um processamento rápido de conteúdo utilizando rede móvel, e oferece flexibilidade, escalabilidade e eficiência a vários locais de estações base utilizando uma infraestrutura já existente. Voltando a problemática dos automóveis conectados à rede, essa tecnologia propõe dispor de uma latência muito baixa (na faixa de 10ms), garantindo uma comunicação de boa qualidade. Os veículos conectados através dos *cloudlets* distribuídos com base em sua plataforma recebem informações, como avisos de outros veículos quase em tempo real. Na Figura 3 pode ser visualizado esse ambiente. Essa tecnologia é uma componente chave das redes 5G e da condução autônoma [49].

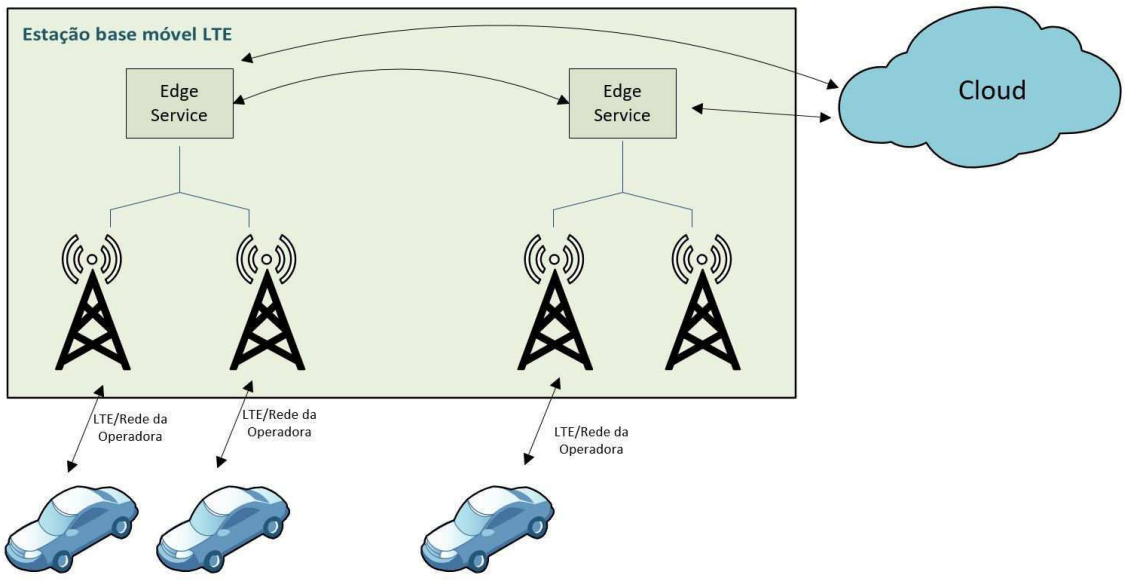

Figura 3 – Representação da implementação de serviços em automóveis utilizando Computação na Borda e Computação na Nuvem.

É nesse contexto, de *Edge/Fog Computing* e suas possibilidades, que se insere esse trabalho. Mais especificamente, propõe-se uma abordagem para o controle do fluxo de dados entre um *Middleware* de aplicação e um simulador de rede, onde as informações enviadas por aplicações são utilizadas e inseridas em um ambiente simulado. Esse ambiente dispõe de dispositivos e técnicas de comunicação que permitem simular esse novo paradigma da comunicação móvel.

De forma geral, neste trabalho é apresentada uma abordagem para possibilitar o desenvolvimento de aplicações no âmbito da Internet das Coisas em um cenário de EC através de um controle de fluxo de dados entre o *Middleware* de aplicações IoT e um

simulador de rede. Esse controle é realizado através de um controlador ou gerenciador que irá intermediar essa comunicação garantindo um ambiente confiável para desenvolvedores e pesquisadores validarem suas aplicações nesse novo paradigma.

1.1 Problemática e Justificativa

Baseado nesse novo cenário e visando o desenvolvimento rápido de novos serviços para EC e redes de próxima geração, é fundamental dispormos de mecanismos e ferramentas que permitam a análise da infraestrutura da rede de borda, desde a inserção das Coisas (dispositivos finais) na rede, até a avaliação de como os serviços de EC devem funcionar. Devido às caraterísticas específicas de rede de Computação na Borda, uma das possibilidades é o uso de ferramentas que permitam simular tais ambientes, com suas topologias de rede, gestão de recursos e serviços específicos. Além disso, a possibilidade de realizar simulações que possam validar e analisar o desempenho desses ambientes, de forma a mensurar os ganhos que podem ser obtidos, é relevante para empresas e desenvolvedores, além dos próprios usuários que receberão melhorias em termos de qualidade de serviços.

Nesses ambientes de simulação encontram-se diferentes tipos de dispositivos de redes, como roteadores, *gateways*, e dispositivos móveis que utilizam diferentes protocolos de comunicações como, UDP, TCP, MQTT, SMTP e outros, permitindo que desenvolvedores simulem diferentes tipos de redes e seus comportamentos. Dessa forma, cabe ao desenvolvedor a escolha do melhor ambiente de acordo com suas demandas. No entanto, apesar da diversidade que os simuladores propõem, os mesmos não dão flexibilidade para aplicações. Quando tratamos de sistemas ciber-físicos ou de aplicações mais complexas, um modelo que separa a comunicação e a computação pode fornecer uma solução mais flexível. Por exemplo, em um modelo de coordenação exógena, o processo de desenvolvimento de componentes de aplicações é completamente separado das comunicações, ou há cooperações entre eles [26]. Quando utilizamos um *Middleware* para o desenvolvimento de aplicações em alto nível, os usuários apenas precisam se preocupar com cada bloco de maneira independente e as conexões individuais de entradas e saídas. Isso faz com que um bloco não tenha conhecimento sobre o que outros blocos a sua volta estão realizando e nem com quais irão interagir. Tudo se resume a utilizar os dados que foram conectados a sua entrada, realizar alguma operação e colocar o resultado em suas saídas. Logo, um ambiente de simulação não tem como propósito principal o desenvolvimento de aplicações e de integração com dispositivos reais, como sensores e atuadores. Devido a isso, é inviável que essas ferramentas desenvolvam e simulem as aplicações IoT. Sendo assim, cabe ao desenvolvedor projetar formas que façam com que exista cooperação entre o desenvolvimento de aplicações e como as mesmas irão se comunicar em um determinado cenário de rede. Isso pode ser realizado através de um coordenador que utiliza os dados de saída de um determinado bloco ou atividade e os coloque como entrada em outros processos ou

atividades, conectando ou direcionado os dados entre as atividades de computação para que os requisitos da aplicação do desenvolvedor/cliente sejam atendidos [26].

Dessa forma, como pode ser visualizado no diagrama na Figura 1, na base da pirâmide estão dispostos os bilhões de dispositivos finais, o que acarreta vários desafios no que diz respeito ao desenvolvimento de aplicações IoT, como interoperabilidade, avaliabilidade, mobilidade, escalabilidade, desempenho, segurança e privacidade precisam ser superados [3], [34]. Muitos dispositivos com funções e protocolos de comunicação diferentes estão sendo desenvolvidos e isso é um grande desafio para a interoperabilidade. Portanto, é necessária uma padronização para tornar todos os objetos e dispositivos de sensoriamento acessíveis e interoperáveis [34]. Visando atender os desafios vistos anteriormente é necessário a utilização de uma plataforma de *software* denominada *Middleware*, que fornece uma abstração dos detalhes de implementação de baixo nível dos protocolos de rede e dos recursos de comunicação, além de oferecer vários serviços. A utilização de um *Middleware* é necessária porque ele atua como um elo que une dispositivos heterogêneos e aplicativos de IoT fornecendo uma API (*Application Programming Interface*) para comunicações de camada física e serviços para os aplicativos [47], [6].

No entanto, como foi explanado, as ferramentas para o desenvolvimento e simulação de serviços para EC focam em tornar possível a engenheiros e desenvolvedores adequar seus sistemas ao paradigma da Computação na Borda. Já os *Middleware* são ferramentas que possibilitam o desenvolvimento de aplicações IoT em alto nível, sendo assim, os mesmos não possuem mecanismos que realizem validação de suas aplicações e adéque-as ao cenário *Edge*. Devido a isso, é necessária uma ponte que permita interligar as ferramentas, possibilitando o desenvolvimento de aplicações IoT dentro de uma infraestrutura da Computação na Borda. Esta conexão se faz necessária aos desenvolvedores que precisam testar e validar suas aplicações antes de implantá-las, com o intuito de evitar que sejam implementados serviços e aplicações que não correspondam às suas necessidades. Quando for realizado o desenvolvimento dos serviços finais, é necessário integrá-los a diversos dispositivos de diferentes tipos, realizar a interligação dos mesmos utilizando diversos protocolos de comunicação e desenvolver os mais variados cenários de uso.

Portanto, só desenvolver as aplicações não é suficiente para garantir o funcionamento das mesmas no meio em que serão inseridas e, para tanto, é necessário simulá-las dentro do cenário desejado. Sendo assim, como o cenário ao qual deseja-se implantar as aplicações é o de *Edge/Fog Computing*, as mesmas devem ser validadas em um ambiente de rede como tal. Desta forma, o objetivo proposto com este trabalho é o de possibilitar a implementação e simulação de aplicações IoT em um ambiente de Computação na Borda (ou *Edge/Fog Computing*). Para tanto, almeja-se realizar a interligação entre ferramentas distintas, especificamente, um *Middleware* para o desenvolvimento de aplicações IoT e uma ferramenta de simulação do cenário de rede apresentado.

1.2 Objetivos

O principal objetivo é a investigação, projeto e implementação de um arcabouço de software que possibilite que desenvolvedores e engenheiros desenvolvam suas aplicações IoT, e realizem a validação delas em um ambiente simulado de Computação na Borda. Dentre os objetivos específicos alcançados destacam-se:

1. Pesquisa, avaliação e escolha de uma ferramenta para a implementação de aplicações IoT que utilize um modelo baseado em atores adequado a integração no ambiente simulado;

2. Pesquisa, avaliação e definição de uma ferramenta de simulação do cenário de rede de Computação na Borda, que contemple métricas de mobilidade e computação distribuída, e que permita a integração com ferramentas externas;

3. Projeto e implementação de um arcabouço de integração entre as ferramentas;

4. Validação do arcabouço proposto em relação a diferentes métricas e requisitos.

1.3 Contribuições

Durante o desenvolvimento desse trabalho de mestrado, atividades adicionais foram desenvolvidas na mesma linha de pesquisa. Um trabalho relacionado ao desenvolvimento de aplicações IoT utilizando um *Middleware* foi proposto e publicado. Neste trabalho, foi possível entender melhor como essas ferramentas funcionam e a possibilidade de controlar e monitorar de forma dinâmica dispositivos utilizando-as [20].

Em outra vertente, porém seguindo a mesma linha de pesquisa deste trabalho de mestrado, foi publicado um estudo inicial de ferramentas de simulação em um ambiente de *Edge Computing* [21]. Dessa forma, a pesquisa realizada contribui para a seleção do simulador utilizado nesse trabalho. Como contribuição direta dessa dissertação, um trabalho com a descrição do mecanismo de integração entre o *Middleware* IoT e o simulador de Computação na Borda foi submetido e aceito para publicação em conferência internacional[1].

1.4 Organização do documento

Este documento está organizado da seguinte maneira:

[1] O artigo completo intitulado "*Integrating an IoT Application Middleware with a Fog and Edge Computing Simulator*"foi aceito para publicação no *28th International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2020)*

- no Capítulo 2 são apresentados conceitos relacionados de modo a facilitar a compreensão ao trabalho desenvolvido. Além disso são apresentados trabalhos relacionados as diversas áreas abordadas nesse trabalho;
- no Capítulo 3 são detalhadas algumas ferramentas e suas principais características. A análise dessas ferramentas é a base para o desenvolvimento do objetivo principal desse trabalho;
- no Capítulo 4 são apresentados detalhes da arquitetura base da integração entre o *Middleware* e o simulador de rede. Essa integração é a base para o desenvolvimento do objetivo principal desse trabalho;
- no Capítulo 5 é detalhada a implementação do arcabouço e as modificações realizadas no simulador que permitem a simulação das aplicações desenvolvidas no *Middleware* IoT. Além disso, é detalhada como foi realizada a validação da arquitetura proposta e a relevância do problema proposto nesse trabalho;
- no Capítulo 6 são apresentadas as conclusões sobre os resultados obtidos e as perspectivas de trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo são apresentados detalhes sobre a Internet das Coisas e as características que tornou esse paradigma tão difundido entre a comunidade acadêmica e os usuários finais no cenário da comunicação moderna sem fio. Em especial, serão apresentados os ambientes de aplicações IoT, os protocolos de comunicação mais utilizados e a inserção do novo cenário de rede da Computação na Borda. Com isso, por fim serão detalhados conceitos, os quais são considerados importantes para o entendimento da solução proposta nesse trabalho.

2.1 Internet das Coisas

O conceito de Internet das Coisas está relacionado a possibilidade de podermos conectar qualquer objeto com o uso da Internet. Devido a sua capacidade de coletar, analisar e distribuir dados que podemos transformar em informação e modificar o meio ao qual estamos inseridos nos mais diversos ambientes, em 2012 tinha-se entre 10 a 15 bilhões de dispositivos conectados, segundo a Cisco. De forma simples, de acordo com IBSG - *Cisco Internet Business Solutions Group*, a IoT é o momento em que mais coisas ou objetos foram conectados à Internet do que as pessoas. E esse momento aconteceu em 2010, cuja população mundial era de 6,8 bilhões de pessoas e tinham 12,5 bilhões de dispositivos conectados [19]. A IBM define a IoT como "o conceito de conectar qualquer dispositivo com a Internet e com outros dispositivos conectado". Apesar do termo IoT possuir diversas definições, segue um senso comum, sua derivação do termo *smart thing* que é um objeto que possui uma eletrônica inserida que o permite trocar dados através de uma rede, sem que exista interação humana. Dessa forma, a IoT possibilita transformar objetos simples em objetos inteligentes o que abre um leque para uma série de possibilidades em termos de comunicação, computação, aplicações e sensoriamento. Alguns outros benefícios da Internet das Coisas são [3]:

- Eficiência - Com mais informações sobre os processos e suas etapas permite que os sistemas funcionem de forma mais eficiente e inteligente;
- Automação e Controle - Permite o controle dos dispositivos físicos através da rede sem que exista interferência humana, o que torna o processo mais seguro e mais ágil;
- Monitoramento - Ao possibilitar o envio das informações dos dispositivos é possível realizar uma análise e monitoramento dessas informações, como por exemplo, a qualidade do ar em uma empresa ou em uma residência, que é uma informação difícil de ser obtida;

- Tempo - Com a integração da IoT os dispositivos e seus sistemas realizam funções que antes eram realizadas por nós, o que acarreta por possibilitar que esse tempo seja usado de outros formas;

- Segurança - Sensores de vigilância e GPS, são exemplos de dispositivos cujas funções permitem termos um ambiente mais seguro, ou nos transportarmos com mais segurança;

- Sensoriamento - No âmbito da saúde, dispositivos IoT podem ter suas funções direcionadas ao monitoramento de pacientes, fornecendo em tempo real indicadores que permitem avaliar o estado de saúde desses pacientes e alertar o mesmo ou equipe médica caso seja necessário;

- Conforto - Uma rede de sensores em um ambiente empresarial ou residencial pode atuar de diversas formas, seja na adaptação da temperatura do ambiente de acordo com o ambiente externo, como decidir entre a escolha da intensidade da luz de LED de acordo com a luminosidade externa, ligar ou desligar o sistema de alarme ou de resfriamento/aquecimento antes de chegar ou quando sair, são ações que promovem mais conforto aos usuários.

Assim como o conceito de Internet das Coisas, sua arquitetura não possui uma única definição, a mesma pode ser baseada em três camadas; em cinco camadas; baseada em *Middleware*; que é um modelo que permite mais abstração para a arquitetura IoT e uma arquitetura orientada a serviços (*Service-Oriented Architecture - SOA*) [3]. Dessa forma, o fato de não possuir uma padronização na arquitetura IoT, faz com que alguns trabalhos utilize uma abordagem integrada entre a arquitetura *Middleware* e a arquitetura orientada a serviços. Em [5] é referenciada essa abordagem. A utilização dessa arquitetura possibilita o desenvolvimento de aplicações IoT de forma ágil, simples e versátil, permitindo ocultar detalhes que simplifique a implementação das aplicações, decompondo sistemas complexos através da interligação de blocos simples que podem ser readaptados e modificados, o que favorece que os clientes readéquem esses sistemas para as novas demandas e tecnologias que possa vir a surgir. Na Figura 4 é apresentada essa arquitetura SOA que possui as seguintes camadas:

- Objetos - Na primeira camada tem-se a representação das “Coisas”, todos os dispositivos físicos, sensores e atuadores IoT. Esses dispositivos podem ser sensores de temperatura, presença, umidade, pressão, lâmpadas, semáforos, câmeras de vigilância ou dispositivos de saúdes. Esses dispositivos através de diversos protocolos de comunicação têm seus dados enviados para a camada superior;

- Abstração dos Objetos - Os dispositivos da camada anterior possuem características próprias que podem fornecer imagens, mensagens, estados e conjuntos de informações,

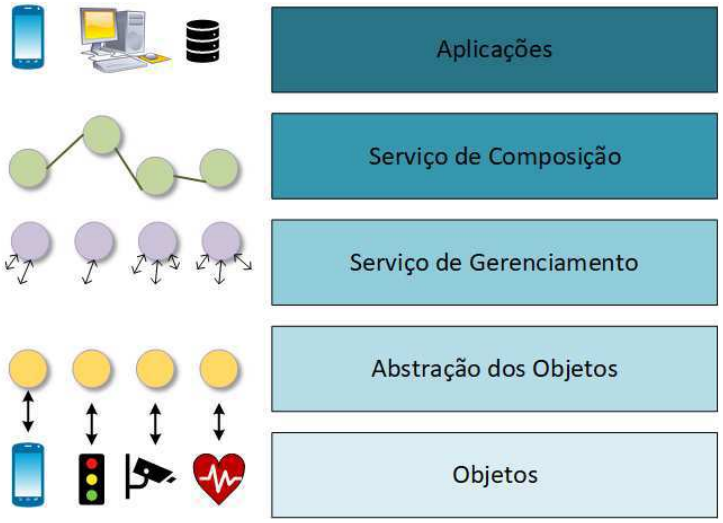

Figura 4 – Representação da Arquitetura *Middleware* Orientada a Serviços

por exemplo. Portanto, é necessária uma camada capaz de obter todos esses dados e transformar em uma linguagem comum. Ao menos que o dispositivo utilize um modo de comunicação UPnP (*Universal Plug and Play*), essa camada de abstração, se faz necessária, para fornecer uma interface de serviço web que será responsável pelo gerenciamento das entradas e saídas dos dispositivos, bem como a implementação da lógica que converte esses métodos em comandos que os objetos compreendam. De maneira geral, o *proxy* tem como função abrir um soquete de comunicação com o console do dispositivo e enviar através de *bluetooth*, *WiFi*, 4G ou ZigBee por exemplo, todos os comandos através de uma função de empacotamento [3];

- Serviço de Gerenciamento - Nessa camada têm os *Middleware* em si, esses fornecem através de atores/blocos a representação dos dispositivos físicos, com suas entradas e saídas. Além disso, desses frameworks fornecem funções que permitem avaliar, monitorar e modificar as informações enviadas e recebidas pelos dispositivos;

- Serviço de Composição - Agora com a representação dos dispositivos físicos é possível interligar esses blocos com outros, que podem ser funções, serviços, outros dispositivos e/ou protocolos de comunicações para desenvolver então a aplicação desejada;

- Aplicações - Na camada superior têm as aplicações desenvolvidas de acordo com a funcionalidade especificada pelo usuário. Essa etapa não é mais considerada parte do *Middleware*, mas sim a representação de tudo que foi desenvolvido nessa ferramenta.

Em relação a comunicação desses dispositivos com a Internet, fatores como consumo de energia ou de recursos computacionais, devem ser levados em consideração. Em linha geral, a Internet das Coisas também é a comunicação entre dispositivos, que ocorre ao atingir a camada de composição, quando através de *Middleware* é possível uma comunicação

em larga escala entre múltiplos dispositivos. Logo, há uma correlação entre IoT e a comunicação *Machine-to-Machine* (M2M) que é uma rede de comunicação entre dispositivos através de qualquer canal que diferente da IoT não necessita de uma conexão com a Internet. Considerando esses fatores, alguns protocolos foram desenvolvidos para facilitar a comunicação entre dispositivos IoT, tomando como base a comunicação M2M e destacando-se pelo baixo consumo de energia e de recurso. Dentre os protocolos disponíveis que consideram as limitações dos dispositivos IoT, tem-se o MQTT e o CoAP.

2.1.1 MQTT

O protocolo *Message Queue Telemetry Transport* (MQTT) é um protocolo da camada de aplicação que utiliza o protocolo TCP/IP como meio de transporte. O MQTT surgiu como uma proposta de centralizar o envio e recebimento de dados de aplicações que utilizam uma rede de sensores sem fio. Essa centralização, possibilita a comunicação entre dispositivos que possuem como fonte de energia baterias, fator limitante para a capacidade de processamento e armazenamento de dados. Essa arquitetura é baseada em um sistema *Publisher/Subscriber*, que baseia sua arquitetura em dispositivos que estão interessados em consumir dados mensurados ou monitorados por sensores. De forma sucinta, os *Publishers* são entidades responsáveis pelo envio dos dados coletados através de sensores, e os *Subscribers* são aqueles interessados em consumir os dados que foram coletados, geralmente atuadores [31]. Porém, sem uma entidade intermediária para coordenar o envio e recebimento de dados, a comunicação não será estabelecida de forma confiável. O *Broker*, é a entidade responsável por receber, em forma de tópicos, os dados captados e repassar a informação para os dispositivos interessados em um tópico específico, assegurando então, que os dados enviados serão recebidos apenas pelos *Subscribers* interessados. Na Figura 5 foi ilustrada uma representação de como funciona a troca de mensagens entre os componentes. Para garantir uma comunicação de dados segura e de qualidade, entre os clientes que irão publicar informações, ou se inscrever para obter informações, o MQTT possibilita opções que vão do uso de criptografia e login até a determinação de níveis de qualidade de serviço (QoS) [39].

Além da arquitetura desse protocolo possibilitar a comunicação necessária para o desenvolvimento de aplicações de Internet das Coisas (IoT), ainda propicia a criação de um ambiente de desenvolvimento para essas aplicações. Isso devido a sua alta capacidade de integração com várias plataformas de desenvolvimento IoT, já que, possibilita o desenvolvimento de serviços utilizando várias linguagens de programação, como Python, Java, JavaScript, PHP, Ruby e C; e a possibilidade de estender esses serviços a plataformas móveis, como iOS e Android [16].

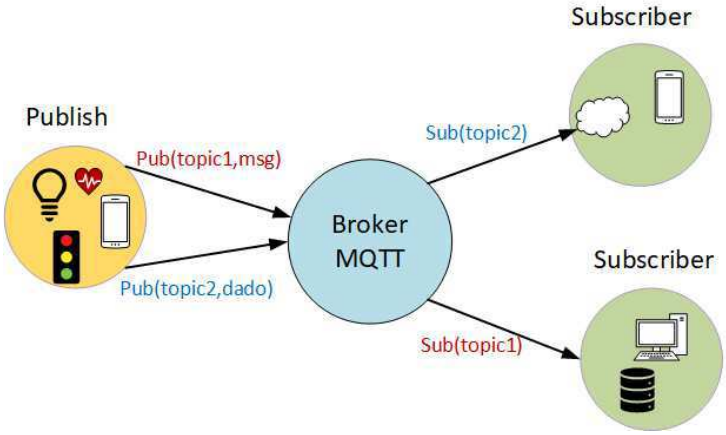

Figura 5 – Representação do funcionamento do protocolo MQTT

2.1.2 *Constrained Application Protocol*

O protocolo da camada de aplicações Constrained Application Protocol (CoAP) foi desenvolvido pelo grupo *Internet Engineering Task Force - IETF* para aplicações IoT que possuem poucos recursos disponíveis. Esse protocolo é baseado na arquitetura REST - *Representational State Transfer* que representa uma forma simples de troca de dados entre cliente e servidor utilizando HTTP (*Hypertext Transfer Protocol*), e utiliza o protocolo UDP como meio de transporte. Devido à falta de confiabilidade do protocolo UDP, o CoAP possui seu próprio mecanismo para garantir confiabilidade na comunicação. Esse mecanismo é implementado através do uso de mensagens com confirmação de entrega, além disso, o CoAP suporta um modelo Cliente/Servidor que garante características semelhantes ao HTTP. No entanto, se for utilizado um método de GET estendido, o CoAP suporta também o modelo Publisher/Subscriber em sua composição. Diferentemente do MQTT que utiliza um modelo baseado em tópicos que irá definir o envio/recebimento de mensagens, nesse protocolo é utilizado o URI - *Universal Resource Identified*, além disso, o CoAP não possui níveis diferentes de QoS [3].

2.1.3 Ambiente de aplicação IoT

Quando trata-se de aplicações IoT podemos separá-las em duas categorias, as aplicações que tem como principal foco fornecer um ambiente para coletar e analisar os dados, ou as aplicações que reagem em tempo real. Para o desenvolvimento do segundo tipo de aplicações é preciso que os dados sejam enviados, processados, analisados e sirvam como entrada para outro processo ou dispositivo, como por exemplo, processos de fabricação em que os sistemas toma decisões em tempo real com base nos valores observados dos sensores. Os *Middlewares*, como foi mencionado, possibilitam o desenvolvimento de aplicações IoT que permitem integrar os dispositivos físicos, coletar esses dados através de outros blocos

para realizar funções ou disponibilizar esses dados para outras finalidades.

Os *Middleware* IoT existentes são divididos em três tipos de classes. A primeira é uma solução baseada em serviços, como por exemplo a arquitetura representada na Figura 4. A segunda é conhecida como solução *cloud-based*, que diferente da primeira, é mais restrita. Pois, demanda que os casos de usos sejam determinados e especificados o seu funcionamento antes de serem implementados. Apesar disso, permite a conexão dos usuários, a realização de coleta e análise dos dados. Por fim, temos a abordagem de um *framework* baseado em atores (*actor-based*), que permite a conexão de vários dispositivos em uma rede distribuída reutilizável.

A arquitetura baseada em serviços possui três planos, o físico, o virtualizado e o da aplicação. O plano virtualizado é o diferencial dessa arquitetura, nele fica localizado as unidades computacionais que irão realizar o controle de acesso, o gerenciamento de armazenamento e o processamento dos eventos. Esse tipo de *Middleware* possui um alto desempenho devido a quantidade de nós executados na nuvem ou nos *gateways*. Esses, no entanto, não foram projetados para funcionar na borda da rede, onde os recursos dos dispositivos são limitados. Já a arquitetura baseada em nuvem possui uma limitação na quantidade de atores/blocos que podem ser utilizados, já que a utilização dos blocos muda de acordo com o que tem disponível nesse ambiente que variam de plataforma para plataforma. Dessa forma, os serviços de dispositivos IoT disponíveis na nuvem só podem ser acessados e/ou controlados se a plataforma que utiliza o serviço em nuvem suportar uma API RESTful ou através de aplicativo fornecido pelo próprio fornecedor do dispositivo IoT [47].

Por fim, temos a arquitetura baseada em atores que também é dividida em três partes, a primeira é o conjunto dos dispositivos IoT, seguidamente tem-se os serviços que podem ficar na nuvem ou serem acessados através de *hosts*. Posteriormente, têm a camada de acesso móvel onde fica armazenado o *Middleware* em si, que pode ser em uma Raspberry Pi, em um notebook, *gateway* ou em ambientes locais, como um desktop. Nessa abordagem, o *Middleware* pode ser incorporado em qualquer um dos ambientes, caso seja utilizado *Middleware* como *Distributed* Node-RED ou Calvin, como serão apresentados posteriormente, cujas unidades computacionais podem ser distribuídas na rede, fornecendo um poder computacional semelhante ao que pode ser utilizado na Nuvem. Essa por sua vez, pode armazenar um determinado repositório que só será consultado caso tenha necessidade. Dessa forma, essa abordagem abre espaço para uma maior diversidade de aplicações e versatilidade na hora de incorporar um novo serviço ou dispositivo. Por conseguinte, essa arquitetura destaca-se por apresentar um ambiente com maior escalabilidade e menor latência quando se trata de dispositivos IoT em larga escala [47].

2.2 Computação na Borda

Em 2018 tinham-se por volta de 18,4 bilhões de dispositivos conectados através de uma rede IP, no entanto, apenas cerca de 6 bilhões utilizavam uma conexão *Machite-to-Machine* [15]. Dessa forma, ao levarmos em consideração essa grande quantidade de dados, seria mais eficiente se parte dos dados fossem processados na mesma camada em que foram gerados, na borda da rede, visando isso, surge o paradigma da Computação na Borda. A fim de possibilitar o processamento de dados próximo da fonte, alguns trabalhos foram propostos, como micro *data center* e *cloudlet*. Em consequência da redução da distância entre a geração e processamento dos dados é possível avaliar diversos benefícios em comparação ao paradigma da Computação na Nuvem. Pensando nisso, várias pesquisas foram realizadas a fim de demonstrar os ganhos com a inserção desse novo paradigma. Como mencionado por Shi Weisong at al em [65], pesquisadores provaram uma redução significativa no tempo de resposta em uma aplicação ao movê-la da nuvem para a borda e ao descarregar tarefas de computação nesse mesmo ambiente. Além disso, foi verificado uma redução no consumo de energia e no tempo de execução das aplicações.

Sendo assim, a Computação na Borda, ou *Edge Computing*, se refere a possibilidade de transferir parte da computação da Nuvem para a borda da rede, sendo a borda, qualquer recurso de computação e de rede entre os *data centers* da Computação na Nuvem e a fonte de dados, que engloba o ambiente de *Edge Computing* e de *Fog Computing*. A diferença entre esses dois últimos conceitos, é que um tem um foco maior nos dispositivos finais, as "Coisas" e outro mais na infraestrutura, nos elementos de rede. Quando falamos na computação, trata-se na possibilidade de descarregamento e armazenamento de dados, processamento e armazenamento de cache, além de gerenciar as solicitações dos usuários aos seus respectivos clientes.

Dado o cenário da Internet das Coisas apresentado é de extrema importância analisar alguns benefícios que esse novo paradigma visa promover. Abaixo serão listadas algumas das vantagens de integrar EC com IoT [77]:

- Transmissão - Como os dados irão ser processados na mesma camada da fonte, a melhora no desempenho da rede pode ser analisada com a diminuição da latência, da largura de banda, da perda de pacotes e outros fatores relacionado a transmissão de dados. Com o descarregamento e armazenamento de dados próximo aos usuários de forma distribuída, o tempo de resposta e do fluxo do tráfego irão reduzir significativamente, o que permitirá o desenvolvimento de aplicações que demandam baixa latência, que realizam, por exemplo, análise de vídeos ao vivo para a identificação e classificação de objetos e ações humanas. Nesse mesmo exemplo de análise de vídeo, que são aplicações que demandam uma grande quantidade de armazenamento, deve ser analisada uma forma de maior otimização dos recursos que serão alocados. Dessa

forma, por exemplo, pode ser escolhido realizar o descarregamento dos objetos a ser comparados, e caso exista semelhança, os *frames* serão analisados na Nuvem para que não exista comprometimento do desempenho de outros serviços. De todo modo, apenas parte dos dados serão enviados para nuvem, o que diminuirá a média do *delay* da tarefa em questão;

- Energia - Quando se trata de dispositivos IoT, a utilização de energia é um fator importante, já que esses dispositivos possuem bateria limitada. Levando isso em consideração na otimização dos recursos mencionados anteriormente, pode ser levado em consideração também a otimização dos recursos energéticos de cada dispositivo. Como mostrado em [77], foram considerados quesitos como associação de estações rádio base de computação e de comunicação, distribuição de tarefas e algoritmos visando reduzir o consumo de energia em aplicações IoT-Edge;

- Largura de Banda - Devido ao aumento da quantidade de dispositivos IoT, o envio dos dados gerados por esses dispositivos para os *data centers* é extremamente inviável, já que isso demandaria um alto consumo da largura de banda da rede. Sendo assim, várias pesquisas foram realizadas com o intuito de reduzir o consumo de banda, como por exemplo, a diminuição da distância em que a largura de banda será alta, em aplicações residências por exemplo, os dados podem ser manipulados em *gateways* domésticos sem que seja necessário de enviá-los a nuvem. Essa solução ocupar a banda até os *gateways*, deixando o resto do percurso livre, além disso, ainda traz benefícios como confiabilidade, em aplicações em que essa solução não atende, isso é, que ainda precisam que os dados sejam processados da nuvem, esses podem ser pré-processados antes de serem enviados para a nuvem, o que reduz o tempo de *upload* já que o tamanho dos dados também foi reduzido [65];

- Armazenamento - No paradigma de EC, o armazenamento dos dados segue um sistema de rede distribuída diferentemente do modo centralizado da Computação na Nuvem. Dessa forma, para evitar obstrução da rede devido à grande quantidade de dados que os dispositivos IoT geram, esses dados podem ser enviados para diferentes dispositivos espalhados na rede em vez de serem enviados para a Nuvem, o que manterá os dados próximos para serem processados, analisados ou até mesmo armazenados. Para isso, também deve ser realizado uma ampla investigação de técnicas de balanceamento de armazenamento que os aloque da melhor forma a depender da aplicação;

- Computação - Como os dados irão ser processados em dispositivos que possuem menor poder computacional do que os *data centers* da Nuvem, as tarefas de computação precisam ser distribuídas entre vários nós da borda para possuírem a mesma capacidade. Dessa forma, nesse tipo de computação há vários tipos de métodos de

escalonamento de tarefas diferentes que vão depender das aplicações para serem implantados. Pode-se variar entre tarefas de computação local, na borda, em *cloudlet* ou na nuvem. Logo, com o aumento da capacidade de computação dos dispositivos finais, algumas tarefas podem ser executadas em uma rede M2M com diversos dispositivos que irão prover juntos capacidade de processamento em um tempo de resposta pequeno [77].

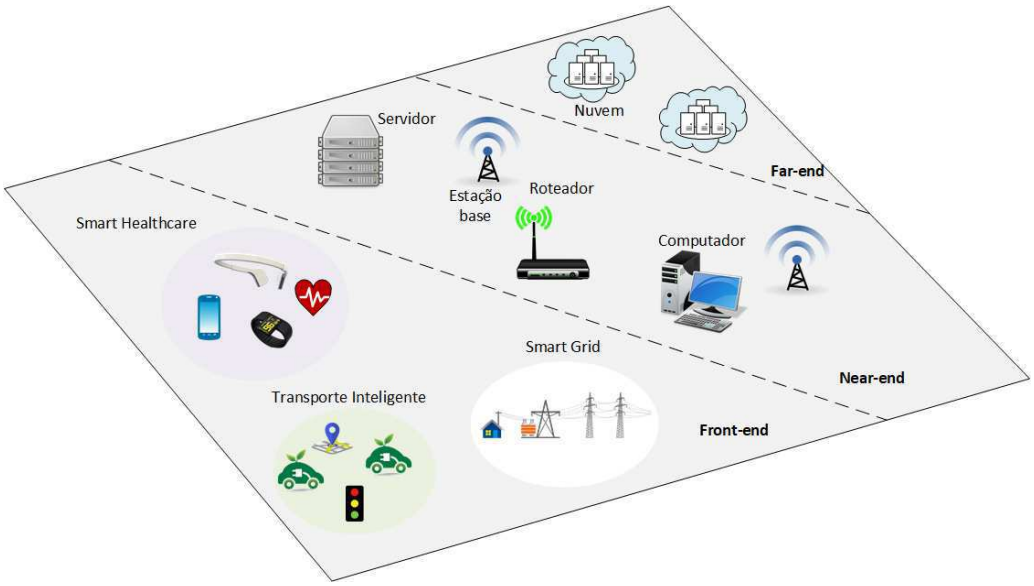

Figura 6 – Representação das camadas de *Edge Computing*

Ao definir a *Edge Computing* como a possibilidade de desenvolver tarefas de computação em dispositivos finais, gateways, rádio bases, micro *data centers* até mesmo na nuvem, tem-se uma ampla área que engloba essas entidades, sendo assim a estrutura de *Edge Computing* pode ser dividida em três aspectos, *front-end*, *near-end* e *far-end*. Como pode ser visualizado na Figura 6, o *front-end* é onde fica os sensores e atuadores, nesse meio estão localizados a maior quantidade de dispositivos, no enquanto, são os que possuem menor capacidade de armazenamento. Seguidamente, tem-se o *near-end*, os dispositivos de redes como gateways, micro *data centers* e estações bases estão localizados nesse meio intermediário, cujo o poder computacional é maior que a camada interior, no entanto menor do que a superior, o inverso ocorre em relação a quantidade de disponíveis presentes. Por fim, tem-se o *Far-end*. Nesse ambiente os servidores da Nuvem estão localizados bem distante dos dispositivos finais.

2.3 Revisão Bibliográfica

Nesta Seção são apresentados trabalhos relacionados ao tema desta dissertação. A fim de seguir um padrão na busca por trabalhos relacionados na literatura, foi realizada

uma revisão sistemática seguindo um protocolo que estabelece critérios, estratégias e métodos de buscas. Para a avaliação desses trabalhos, foram divididos grupos tomando como base as características comuns presentes nos trabalhos. Dessa forma, são avaliados trabalhos que incorporam o EC em um ambiente de IoT que utilize um modelo baseado em atores ou similar. Inicialmente são analisadas ferramentas que tem como foco a análise de recursos. Em seguida, são apresentados trabalhos que tem como foco o gerenciamento de serviços e recursos, posteriormente são detalhados trabalhos que no mesmo contexto da IoT e EC apresentam definições de plataformas e ferramentas ou novos padrões. Além dessas categorias, são apresentados trabalhos que desenvolveram ou utilizaram *frameworks* e modelos de programação, ferramentas de software e, por fim, *testbeds* e experimentos. Com isso, nessa seção são detalhados trabalhos com o objetivo de realizar uma análise comparativa com a solução proposta apresentada nesta dissertação.

2.3.1 Análise de Recursos

Esta categoria é caracterizada pelos trabalhos que tem como foco principal a análise dos recursos em seus trabalhos. O trabalho [76] tem como foco analisar os dados das aplicações IoT com o uso do método de *Machine Learning*. Para isso, propõe o desenvolvimento de uma arquitetura que coordene os dados entre a nuvem e a borda, utilizando um modelo de mineração de dados que irá distribuir os dados entre os dispositivos *Edge*. Nesse trabalho, os autores utilizam a tecnologia *Apache NiFi* como servidor central para simular o ambiente da nuvem e o sub projeto MiNiFi que possui um poder de processamento menor para simular dispositivos *Edge*. Como caso de uso, o trabalho usa um cenário com veículos conectados, no entanto, não apresenta um ambiente versátil e adaptável para o desenvolvimento de aplicações IoT.

Em [35] temos a descrição de uma abordagem que aplica um algoritmo de mineração de dados em um modelo baseado em atores, com o intuito de melhorar o desempenho e o tráfego da rede em um ambiente *Fog Computing*. Para realizar esse trabalho, foi utilizado o Apache Spark MLlib, biblioteca da Spark que utiliza aprendizado de máquina, para o fornecimento de uma abordagem centralizada, e para a abordagem distribuída foram utilizados blocos com um algoritmo específico para simular os nós *Fog*. Assim, como o trabalho anterior, não foi fornecido um ambiente para o desenvolvimento de aplicações IoT. Já [4], apresenta um *framework* chamado ActorEdge que tem como proposta integrar tecnologias existentes e facilitar o desenvolvimento e implementação de aplicações em um sistema heterogêneo. Isso por sua vez, se tornou possível através de micro serviços fornecidos na *edge*, o que acarretou uma latência 10x menor, 30% menos jitter e maior largura de banda em comparação com as opções de Computação na Nuvem. Esse framework utiliza uma linguagem de programação orientada a objeto denominada Objetive C, que faz com que a aplicação só funcione em MacOs e em dispositivos iOS.

No trabalho desenvolvido em [44] é apresentado uma API denominada PiCo (*Pipeline Composition*) que visa facilitar a programação de aplicativos de análise de *Big Data* mantendo ou aprimorando o desempenho delas. Isso é feito através de um modelo de programação baseado em *pipelines*. Em [60] visa solucionar problemas de fluxos de dados com o uso de um algoritmo que irá aproveitar os núcleos de CPU adicionais. E por fim, [71] propõe o desenvolvimento de uma plataforma de processamento dinâmico de fluxo de dados com o intuito de modificar dinamicamente a estrutura desse fluxo através do método *Publisher/Subscriber* basedo em tópicos (TBPS) com o uso de um algoritmo de roteamento P2P, chamado LASR (*Locality-Aware Stream Routing*).

2.3.2 Gerenciamento

Nesta categoria foram englobados dois tipos de gerenciamento, o de serviços e o de recursos. Em [13] foi desenvolvido um *framework* de código aberto com o intuito de gerenciar serviços distribuídos geograficamente em grande escala. No trabalho são analisadas várias características inerentes a EC, como baixa latência, conhecimento da localização dos dispositivos, ampla distribuição geográfica, mobilidade e heterogeneidade. Este trabalho foi desenvolvido para um hospital de pesquisa com um controle distribuído de dispositivos, coleta de dados e medição.

Já em [55] foi desenvolvido um projeto que ajuda a fornecer uma plataforma *Cloud* composta com *Software Delevopment Kit - SDK* e um ambiente de execução com o intuito de facilitar o desenvolvimento de aplicações com dados intensivos em um meio *Fog* e melhorar a execução das aplicações desenvolvidas, permitindo a movimentação e a computação de dados para que os dados e as tarefas possam migrar de recursos na nuvem para aqueles na borda ou vice-versa, com isso satisfazia algumas restrições colocadas pelo desenvolvedor em termos de desempenho, segurança e privacidade.

2.3.3 Definições e Padrões

Nesta categoria foram classificados os trabalhos que focam em definir outras plataformas e ferramentas, ou adotar novos padrões. Na Europa foi criado um regulamento com o intuito de proteger os dados gerados por dispositivos IoT, móveis e serviços na nuvem. Devido a isso, em [41] foi detalhado as modificações realizadas na plataforma *DataBox* que visa desenvolver aplicativos domésticos. Já [70], tem como foco apresentar o conceito da transformação de fluxo de dados que permitia o processamento de parte desses dados na EC em dispositivos que possuem limitações de recursos, a possibilidade de usar o framework *uFlow* nesses dispositivos e adequar um caso de uso que utiliza o framework para o *Middleware* NodeRED.

2.3.4 *Framework* e Modelos de Programação

Esta categoria é destinada aos modelos de programação e a *Frameworks*, no entanto na análise de recursos também possuem trabalhos que desenvolvem *frameworks*, porém esses são destinadas a análise de dados. Nesta tem-se ferramentas e modelos que foram desenvolvidas para outros propósitos.

Em [24] é apresentado um *framework* distribuído baseado no *Middleware* Node-RED, o (*Distributed Node-RED* ou D-NR). Neste trabalho são identificadas características importantes a uma aplicação que irá ser desenvolvida na nuvem e na névoa, sendo essas a necessidade de um sistema heterogêneo, que suporte diferentes percepções de ações de ciclo, mobilidade e escalabilidade. Posteriormente, detalha como a ferramenta resolve cada uma dessas características. Por fim desenvolve uma aplicação com o intuito de validar sua ferramenta, este trabalho é o segundo mais citado pela comunidade acadêmica, pode ser facilmente replicado pois, a ferramenta é open-source e está disponível na plataforma GitHub [1], que também possui o link direto através do site do próprio Node-RED.

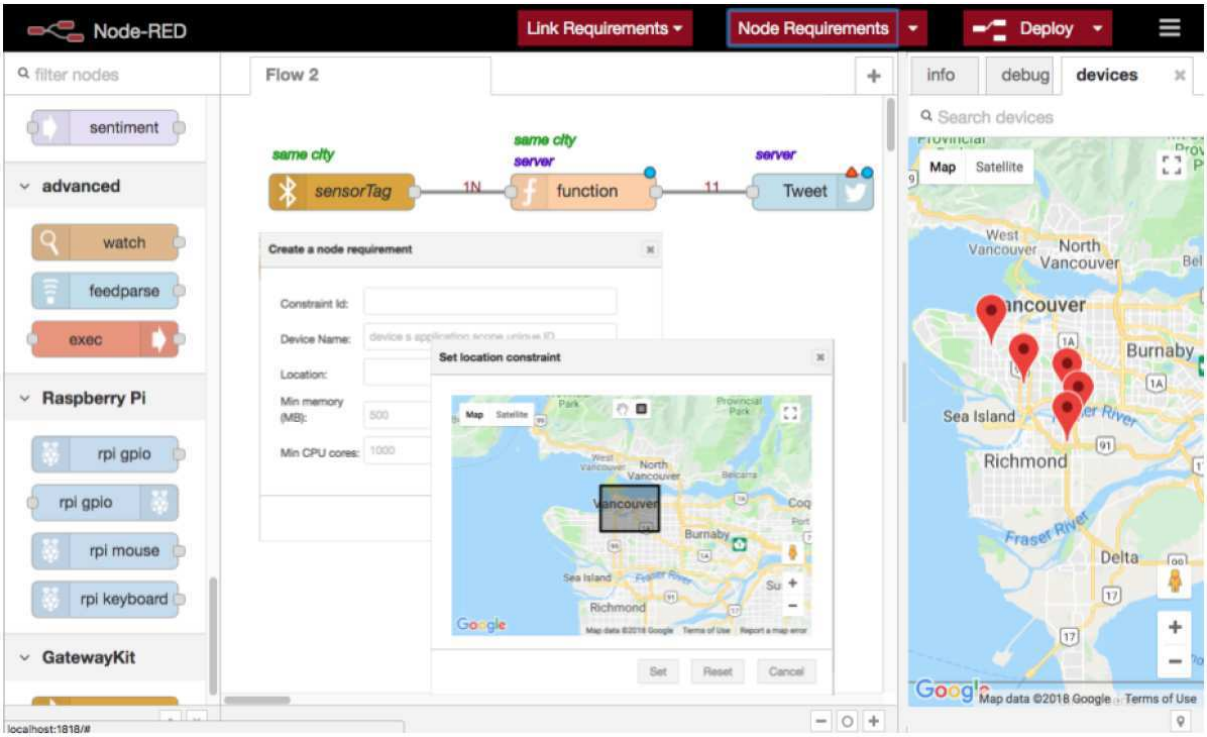

Figura 7 – Representação do desenvolvimento de aplicações CPSCN usando o framework D-NR apresentado por Giang et al [26].

Em [26] é apresentado um modelo voltado para sistemas ciber-físicos de computação e redes, chamado CPSCN (em inglês *Cyber Physical Social Computing and Netwirking Systems*). Esse trabalho visa desenvolver um modelo de coordenação exógena que realize a

[1] D-RN, disponível em: `https://github.com/namgk/node-red-contrib-dnr`

separação entre computação e atividades de comunicação, e ajude a desenvolver alguns desafios a respeito da dinâmica e os sistemas CPSCN em larga escala. Neste trabalho os CPSCN são caracterizados por serem sistemas das redes sociais da próxima geração. Para o desenvolvimento desse modelo foram utilizados o *framework* D-NR e a ferramenta de simulação da OMNET++. Para a validação do modelo foi configurada uma área de simulação com $1000m^2$, quatro componentes de softwares conectados e restrito o processamento de dados aos dispositivos *Fog*. Além disso é estabelecido que a fonte de dados não possam ser localizada nos dispositivos responsáveis por salvar os dados e reduzido a 300 metros o fluxo de aplicação entre os dispositivos. Na Figura 7 é possível visualizar o desenvolvimento da aplicação utilizando a plataforma D-RN. Nela é demonstrada a utilização de dispositivos em diferentes localizações que permitem o desenvolvimento de sistemas distribuídos.

Em [32] é desenvolvido um modelo que permite que aplicativos IoT possam ser modelados em um único local. Logo depois, o aplicativo é divido entre os dispositivos da *Edge*, que tem as informações a respeito de suas localizações anotadas. Com essas informações, são gerados códigos automáticos que serão responsáveis por conectar as partes da aplicação que se encontram distribuídas independente das mesmas estarem localizadas na borda ou na nuvem. Por fim, em [59] é desenvolvida uma arquitetura para detecção de anomalias em tempo real a partir de fluxo de dados provenientes de sensores conectados a pacientes, logo o foco do trabalho é o desenvolvimento de aplicações para a chamada IoTM -*Internet of Medical Things*.

2.3.5 Software e Ferramentas

Em [41] temos um trabalho que visa adequar as ferramentas ao novo padrão europeu GDPR como foi visto, entretanto em [45] é apresentado a mesma problemática, porém, este trabalho tem um viés maior para o modelo de DataBox, especificamente, essa ferramenta irá se adequar a este novo padrão. O DataBox é um dispositivo na borda que se destina a estar nas residências, e permite agrupar dados de dispositivos IoT de forma direta ou via APIs, com isso possibilita que aplicativos os utilizem para atuação e processamento de dados. Na Figura 8 é possível visualizar a arquitetura proposta por esse trabalho.

Em [7] é desenvolvido uma nova abordagem que tem como finalidade integrar serviços de plataformas IoT, para isso, a ferramenta utiliza o *Middleware* Node-RED e o projeto Inter-IOT, esse desenvolve um sistema a fim de garantir a interoperabilidade entre plataformas. Na arquitetura desenvolvida nesta pesquisa são levados em consideração requisitos que facilitem o intercâmbio de informações entre diversos serviços, mecanismos que possibilitem que a saída de informações de um serviço possa ser entrada em outro serviço e o gerenciamento da execução dos serviços. [10] analisa diferentes abordagens de fluxo de dados para torná-lo mais eficiente e expressivo, tendo como foco plataformas de

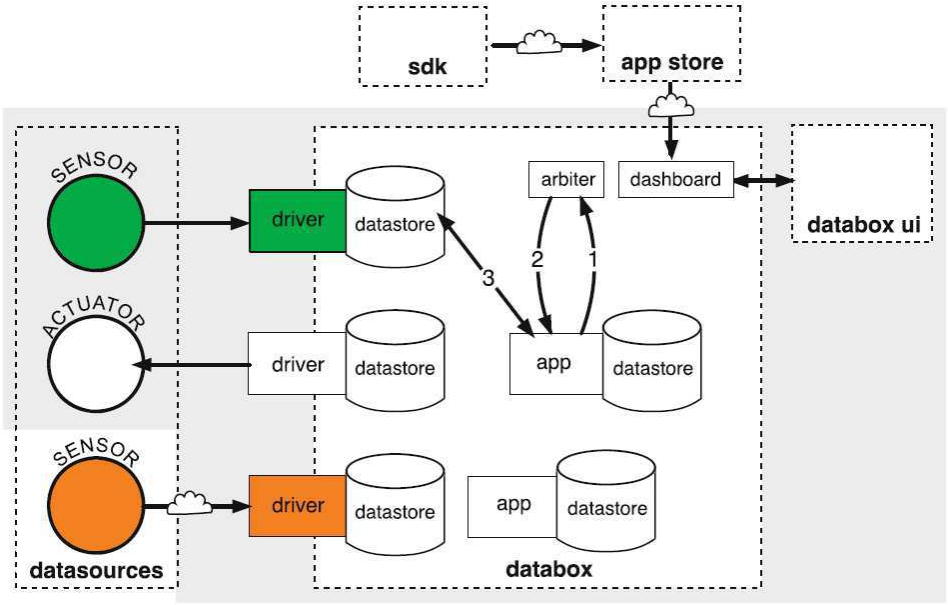

Figura 8 – Diagrama com a arquitetura do DataBox aprsentado por Lodge et al [45].

computação heterogêneas e formas de economizar energia no processamento de aplicativos. Neste trabalho é utilizado experimentos utilizando *Adaptive Deep Neural Network* em duas plataformas, um Soc de multicore móvel e uma estação de trabalho equipada com GPU. Já a plataforma ThingNet desenvolvida em [58] é baseada em micro serviços com o intuito de permitir roteamento e processamento de dados distribuídos. Esta plataforma possui o código aberto e inclui ferramentas como Kubernetes e MiNiFi. Com o ThingNet é possível implantar, modificar, migrar e dimensionar a lógica do aplicativo. Por fim, temos o iFogSim ferramenta que já foi apresentada anteriormente.

2.3.6 Testbeds e Experimentos

Em [25] temos outro trabalho voltado para a ferramenta D-NR, no entanto, nesta abordagem é apresentado alguns desafios que foram enfrentados ao logo do desenvolvimento das versões da plataforma. Para isso, são apresentados experimentos realizados, e métodos de resolução adotados para resolver problemas que foram descobertos. [66] é apresentado um Testbed com o intuito de estudar o controle de missão crítica sobre a *edge cloud* distribuída. Logo, neste trabalho é mostrado o potencial da ferramenta ao fundir IoT, 5G e a nuvem.

A plataforma de orquestração ECHO é proposto em [64], está é comercial e oferece uma composição híbrida de fluxo de dados que pode operar em diversos modelos de dados. Esta ferramenta gerencia o ciclo de vida do aplicativo, incluindo *contêiner* e possui um registro para gerenciamento de estados. Além disso, pode executar migração dinâmica de tarefas entre os recursos. Por fim, tem-se [51], que propõe um sistema chamado LiReD

(*light-weight real-time fault detection system*) para manipular robôs industriais e um modelo para detectar falhas baseado em LSTM. Logo, são utilizados dispositivos *edge* que irão coletar, processar, armazenar e analisar os dados com o uso de um computador e um sensor.

2.3.7 Considerações Finais do Capítulo

Com isso, dentre os trabalhos pesquisados na literatura, não foi encontrado uma proposta que apresente uma abordagem que possibilite o desenvolvimento de aplicações IoT no contexto da Computação na Borda e sua simulação em um ambiente de rede específico. Apesar do trabalho [26] desenvolver uma aplicação utilizando um *Middleware* e simular essa aplicação em um simulador de redes, não foi realizada a integração entre as duas ferramentas, além de não demonstrar um ambiente que possa ser utilizado por outras aplicações. Nesse trabalho foi desenvolvido um ambiente de simulação apenas para sua solução específica.

3 Revisão Tecnológica

Neste capítulo é apresentada uma revisão tecnológica dos principais componentes e conceitos utilizados no desenvolvimento desta dissertação. Inicialmente é apresentado o conceito de modelos baseado em atores, os quais são utilizados pelos principais *Middlewares* IoT disponíveis. Em seguida são detalhados alguns *Middlewares* que utilizam esse modelo no desenvolvimento de suas ferramentas. Posteriormente, são apresentadas ferramentas que foram desenvolvidas com o intuito de possibilitar a engenheiros e desenvolvedores simular cenários de aplicação e de rede no novo paradigma de Computação na Borda. Ao final desse capítulo é apresentada uma comparação entre as ferramentas, a fim de selecionar a que mais se adéqua ao cenário proposto nessa dissertação.

3.1 Modelo baseado em atores

Um modelo orientado a atores, ou simplesmente, modelo de atores é uma metodologia de componentes no qual, os componentes são chamados de atores. Esses estabelecem comunicação e executam determinadas operações que são definidas através de parâmetros utilizados em sua configuração. Além dos parâmetros, cada componente de interface abstrai o estado interno de cada ator, seu comportamento e a forma que realizam suas interações. Desta forma, as interfaces que cada ator possui são destinadas para que sejam estabelecidas a comunicação entre eles. Os modelos, assim como os atores, podem realizar conexões através de portas e parâmetros externos, que permitem a sua interação com outros modelos ou com um ator do seu modelo. Este termo, "ator", é utilizado em vários contextos e foi introduzido no contexto da modelagem orientada a atores nos anos 70, por Carl Hewitt em [30] para descrever o conceito de agentes de raciocínios autônomos. Posteriormente foi utilizado na descrição da formalização de modelos concorrentes em [2], além do uso em modelos de fluxo de dados em [18], [56]. Sendo assim, um sistema de tempo contínuo pode ser modelado utilizando um ator, como a representação apresentada na Figura 9. Nessa

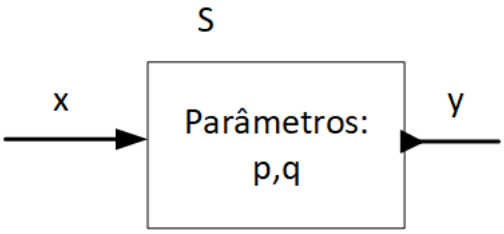

Figura 9 – Representação de um ator.

representação, o sinal de entrada x e o de saída y são funções da forma:

$$x : \mathbb{R} \to \mathbb{R} \qquad y : \mathbb{R} \to \mathbb{R}.$$

Nessa função, o domínio é representado pelo tempo e o contradomínio pelos valores dos sinais esse tempo específico. Dessa forma, o modelo do sistema é função da forma:

$$S : X \to Y, \qquad onde \quad X = Y = \mathbb{R}^{\mathbb{R}}$$

A função S pode depender de parâmetros do sistema. Na Figura 9 esses parâmetros são representados por p e q. Dessa forma, a função do sistema pode ser representada em função da forma acima ou com base nesses parâmetros, *S(p,q)*. Quando temos um sistema que possui funções de entrada e de saída, podemos representá-lo através de uma caixa, denominada ator [38].

A sintaxe desse tipo de modelo é determinada por seu modelo de computação. A forma como esse irá ser empregado em cada ferramenta varia de ferramenta para ferramenta. Adiante serão apresentados alguns modelos e o seu modelo ou modelos de computação. A especificação do modelo a ser adotado determina as regras de execução, atualização e computação dos atores, bem como, a natureza de suas comunicações. Quando se define o modelo de computação a ser empregado, está sendo feita a definição das propriedades de modelagem do modelo/sistema. Alguns dos modelos de computação que são úteis para o design baseado em modelos de hardware e software [37]:

- Síncronos/reativos

  Esse modelo é um sistema discreto onde os sinais estão ausentes todo o tempo exceto nos tiques de um relógio global. De forma conceitual, Lee et al em [38] afirma que a execução de um modelo é uma sequência de reações globais que ocorrem em momentos discretos, e a cada reação, a reação de todos os atores é simultânea e instantânea. Modelos com realimentação é um tipo de modelo reativo síncrono.

- Fluxo de dados

  Nesse modelo, temos um sistema assíncrono, cuja ocorrência de um evento está totalmente relacionada a ocorrência ou não de um evento em outro ator. Isso é, para que um sistema A execute qualquer tarefa, é requerido um dado produzido por uma reação de um sistema B, dessa forma, A só pode ocorrer depois que B ocorrer. Sendo assim, tem-se um modelo de computação baseado em fluxo de dados quando a principal restrição do sistema é essa dependência de dados. Na comunicação entre atores, as sequências de mensagens são os sinais que fornecem a comunicação. Nesse modelo tem-se fluxo de dados síncronos, dinâmicos e estruturados. Dessa forma, quando temos um *Middleware* por exemplo, tem-se um modelo baseado em atores que utiliza um modelo de computação de fluxo de dados, ao realizar-se a composição entre atores do desenvolvimento das aplicações.

- Temporizado

  Esse tipo de modelo de computação refere-se a software, por exemplo, que possuem um tempo diferente de execução que deve ser levado em consideração quando esses interagirem com processos físicos. Quando for utilizado um modelo temporizado, podem ser de vários tipos, como por exemplo o modelo acionado por tempo, que em sistemas distribuídos possuem mecanismos para acionar periodicamente cálculos distribuídos de acordo com um relógio também distribuído que mede a passagem do tempo. O sistema de modelagem Simulink suporta esse tipo de modelo de computação. Outro tipo de modelo temporizado é o de sistemas de eventos discretos, que é bastante usado em vários tipos de aplicações, como redes digitais, sistemas econômicos, hospitalares e políticos. Esse sistema é comumente conhecido como DEVS, abreviação do inglês *discrete event system*. Se tratando de uma rede de atores, um modelo desse tipo é quando cada ator reage aos eventos de entradas seguindo a ordem que ocorreu de acordo com registro de data e hora, e produzem eventos de saída seguindo essa mesma ordem. Sendo assim, o estado do sistema só muda quando ocorre um evento. Além desse modelo tem-se os sistemas de tempo contínuo. Que diferente do de tempo discreto, não se baseia na ocorrência de eventos para modificar seu estado, mas nas mudanças que ocorreram em pequenos intervalos de tempo [38].

A seguir são apresentadas algumas ferramentas que utilizam um modelo baseado em atores.

3.1.1 Capecode

O módulo CapeCode[1] é uma configuração do Ptolomy II[2] que permite a composição, execução e implementação de acessores (do inglês: *accessors*) compostos através de uma GUI - Interface Gráfica do Usuário. O projeto Ptolemy permite a implementação de aplicações utilizando vários modelos de computação com o uso de um componente chamado *Director*. Sendo o Ptolemy II um *framework* de código aberto, desenvolvido em Java pelo professor E. Lee na Universidade da Califórnia, Berkeley [47]. Um *acessor* segundo [12] serve como um proxy para uma Coisa ou serviço que pode ser local ou remoto. Isso é, como um proxy de um serviço em um navegador web, onde o proxy é executado em um hospedeiro local, mas interage com um serviço remoto, também são projetados para serem executados em diversos hospedeiros e possuem uma padronização que permite ser executado em qualquer navegador. A composição de atores conectados por fluxo é chamada de *swarmlet*, quando se tem um serviço de implementação que é executado em uma Coisa, um servidor local ou na nuvem, temos um *swarm service*, a compactação é feita em um *acessor* que pode ser

[1] CapeCode, disponível em: `https://wiki.eecs.berkeley.edu/accessors/Main/CapeCodeHost`
[2] Ptolemy II, disponível em: `http://ptolemy.berkeley.edu/ptolemyII/`

conectado a outros atores que são executados em um *host acessor*. Sendo assim, um *acessor* é uma classe de componentes que um *swarmlet* instancia para a acessar um dispositivo ou serviço [36].

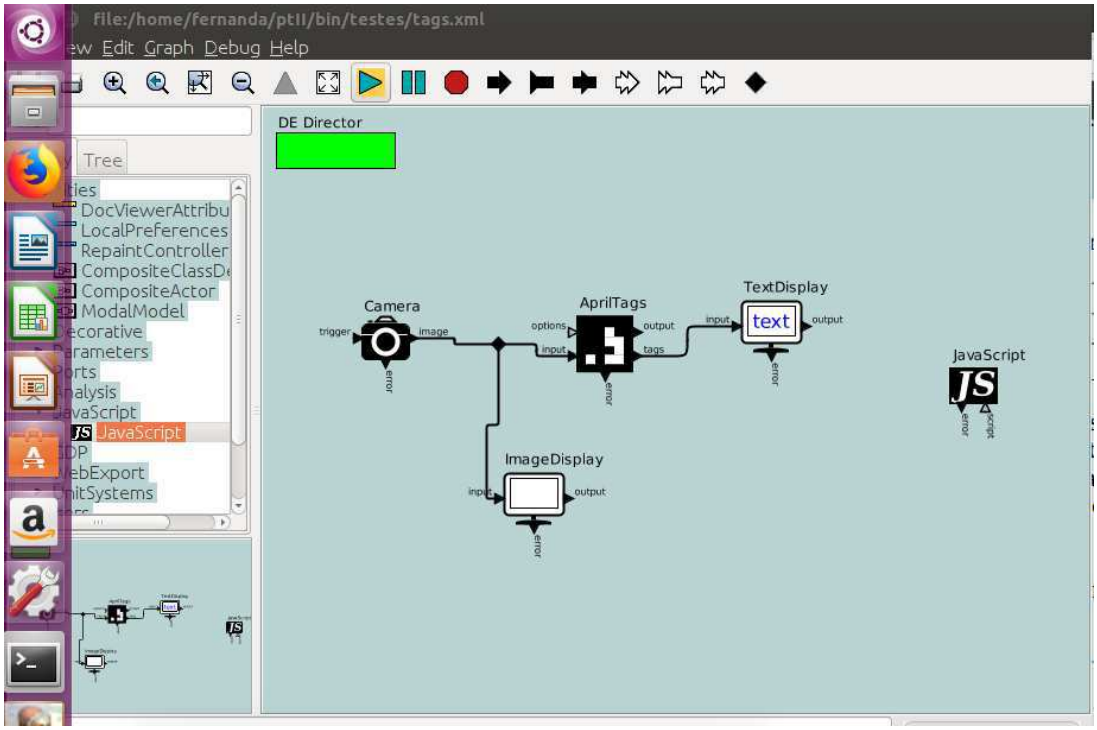

Figura 10 – Interface gráfica do Capecode.

Logo, o Capecode permite a composição de acessores em um único ambiente, e o *Director* é o orquestrador do fluxo de trabalho, como pode ser visualizado na Figura 10. Um componente de acessor possui uma interface e um script, na Figura 10 tem-se a imagem da interface do acessor AprilTags, por exemplo, já o script é desenvolvido em JavaScript e define uma ou mais funções que irão variar de acordo com o acessor. No caso do AprilTags, são utilizadas as funções *setup*, *initialize* e *wrapup*. Essas funções irão invocar funções de entrada, saída e parâmetros, por exemplo; marcar o início da execução; e finalização da execução, respectivamente. Desta forma, no padrão de acessores, tem-se dois tipos de interações, uma interação horizontal e outra vertical. A horizontal possibilita a composição de vários servos e Coisas, já a vertical possibilita a interação entre dispositivos e host específicos através de protocolos de comunicação como, HTTP (Protocolo de Transferência de Hipertexto), CoAP (Protocolo de Aplicação Restrita), TCP (Protocolo de Controle de Transmissão) *sockets* e WebSockets [36].

3.1.2 Node-RED

A plataforma de *Middleware* Node-RED possui código aberto e foi desenvolvida pela IBM baseada no *framework* Node.js, sendo implementada em JavaScript. Essa

ferramenta permite a conexão de dispositivos e APIs, assim como as outras ferramentas, possui atores, denominados *nodes*, onde uma aplicação pode ser desenvolvida ao arrastar, soltar e interligar blocos na tela de fluxo, tornando isso ainda mais simples se os blocos já tiverem desenvolvidos. Além dessa forma de operação, também é possível importar códigos JavaScript, isso permite uma resposta rápida, já que a ferramenta possui um modelo de computação orientada a eventos para interação em tempo real; possibilitando o desenvolvimento de aplicações IoT cujo o tempo de resposta é um fator crucial em seu funcionamento; através de protótipos rápidos desenvolvidos nesse formato [8], [47]. Na Figura 11 é apresentada a interface gráfica do Node-RED.

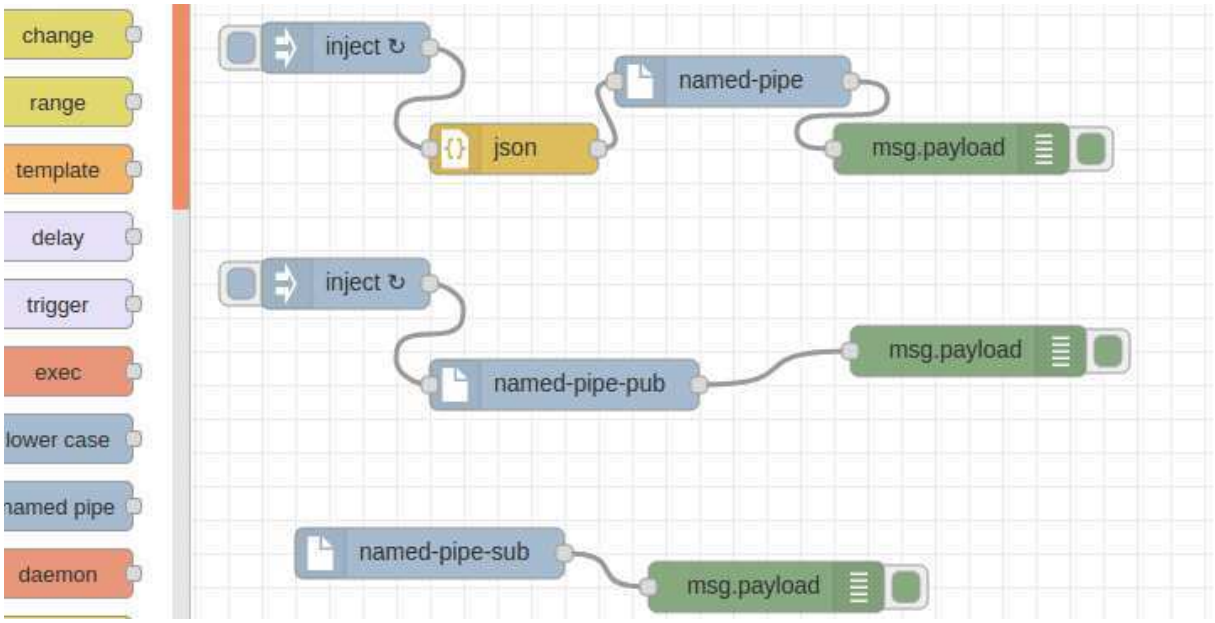

Figura 11 – Interface gráfica do Node-RED.

A ferramenta permite assim como as demais, criar fluxos ao conectar nós de entrada e saída de dados para processar dados e controlar Coisas, por exemplo. Isso é possível através dos seus três componentes básicos, painel de nó, painel de fluxo e painel de informações e depuração [40]. O Node-Red possui uma vasta comunidade de colaboradores, o que acarretou a disponibilidade de vários nós que outros podem utilizar-los em seus projetos. Eles variam de nós que facilitam a conexão aos pinos da Raspberry Pi, sistemas de mensagens instantâneas e redes sociais. A difusão da ferramenta em Raspberry levou aos desenvolvedores do sistema operacional Raspbian a inclui-la em seu sistema. Além disso, alguns projetos surgiram com o intuito de melhorar esse *framework*, como por exemplo o sistema FRED, um *front-end* para Node-RED, que tem o intuito de fornecer um sistema NR hospedado na nuvem com vários inquilinos para rápida integração e desenvolvimento de aplicativos hospedados em nuvem; e o *Distributed Node-RED (D-NR)*, que tem com

intuito tornar a ferramenta distribuída para permitir o desenvolvimento de aplicações IoT em um cenário de *Fog Computing* [9], [24].

3.1.3 Calvin

O Calvin é um framework *open-source* desenvolvido por um grupo de pesquisa na Ericsson com um intuito de facilitar o desenvolvimento de aplicações IoT em sistemas distribuídos, através de uma metodologia de programação de fluxo de dados por meio de um modelo de atores [48], [46]. O paradigma da programação de fluxo de dados está presente em modelos de atores, pois nesse modelo, um ator é um componente de software que representa funções, dispositivos, serviços ou algum tipo de computação na forma de código objeto, comumente chamado de "caixa preta". No caso do Calvin, para estabelecer comunicação entre os atores, são necessários *tokens*. Esses são adquiridos quando os atores de entrada, responsáveis por captar os dados, processa-os e transforma em recursos, que nesse ambiente são considerados, *tokens*, que serão consumidos por outros atores [53].

O *framework* já possui vários atores padrões, como módulos de entrada/saída, mídia, de acesso a rede, como HTTP, TCP e MQTT, processamento de texto, operações e alguns sensores [3]. Todavia, novos atores podem ser implementados em Python, e o fluxo de dados entre eles expresso usando uma linguagem puramente declarativa, CalvinScript [54]. Devido a facilidade de desenvolver atores, o Calvin simplifica a implementação de diversos novos serviços baseado nesse tipo de modelo.

Para realizar o desenvolvimento de uma aplicação no Calvin é necessário seguir um ciclo composto por quatro fases: *Describe*, *Connect*, *Deploy* e *Manage*. Ao determinar os módulos de entrada/saída através dos atores padrões ou de novos atores que foram desenvolvidos, no que se refere ao ciclo de uma aplicação, está sendo realizado a descrição de como suas tarefas serão executadas. Em seguida, é necessário realizar as conexões entre elas, que são descritas como mencionado através dos CalvinScripts. As conexões entre os atores, podem ser feitas através de uma interface gráfica, o Calvin GUI. Essa ferramenta lista os atores disponíveis e permite que seus blocos sejam deslocados e as conexões estabelecidas através de suas entradas e saídas. Ao realizar isso, o CalvinScript é automaticamente gerado. Na Figura 12 pode ser visualizado o detalhamento da interface gráfica do Calvin. Feito os passos anteriores, a aplicação já está pronta para ser implantada, que ocorre quando o aplicativo é instanciado de acordo com fluxo de dados do sistema que está sendo desenvolvido. Por fim, temos a última fase, a de gerenciamento, no Calvin podem ser realizadas alterações no aplicativo, logo, como o desenvolvedor definiu sua implementação pode ser modificada, e os atores podem ser movidos para outros dispositivos diferente do que foi instanciado inicialmente [48], [46].

[3] Calvin, disponível em: `https://github.com/EricssonResearch/calvin`

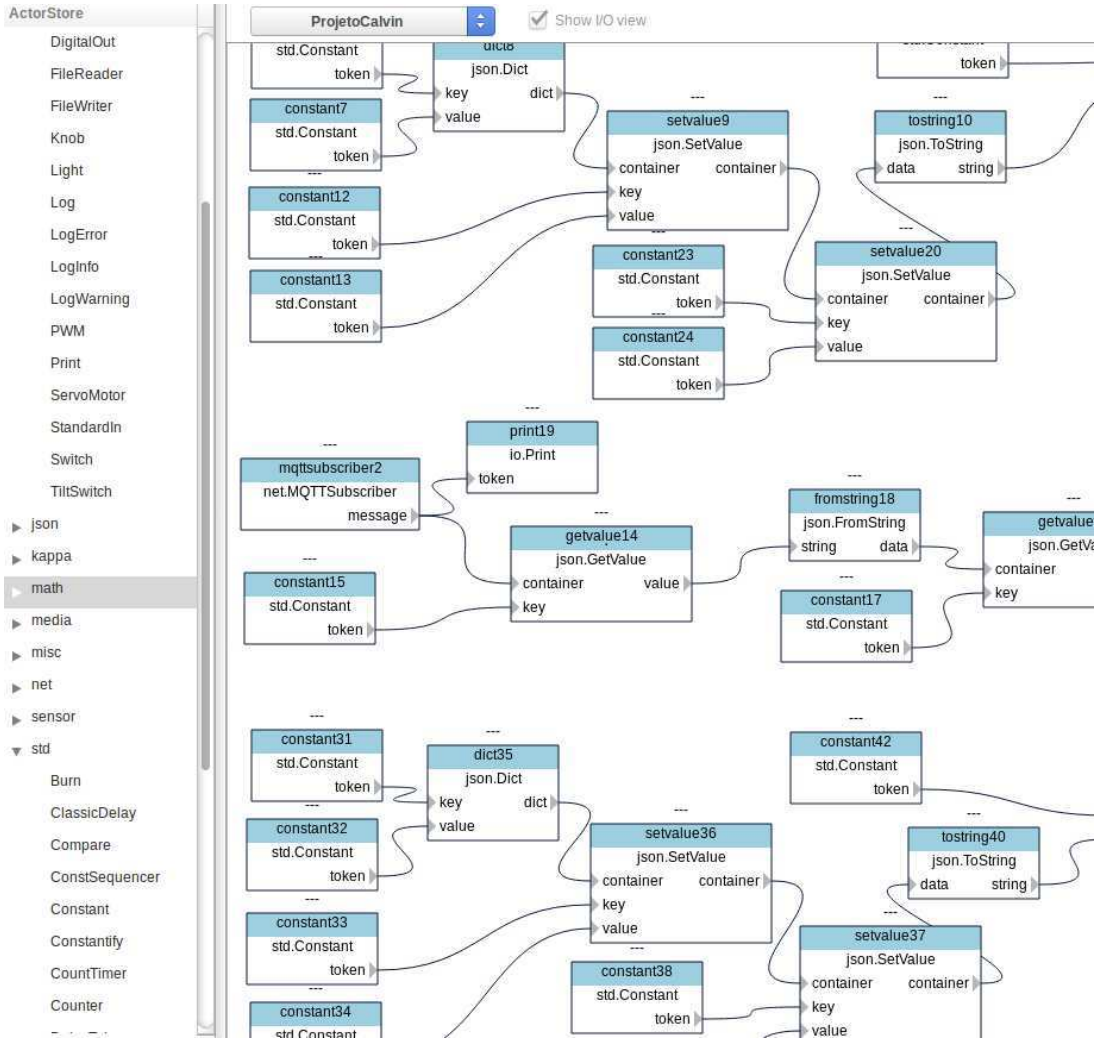

Figura 12 – Interface do Usuário do Calvin.

Na arquitetura do Calvin tem-se uma camada denominada *runtime* que é onde os atores são executados, a mesma possui duas subcamadas, a *Plataform dependent* e a *Plataform Independent*. A primeira subcamada possibilita a comunicação entre diversas *runtimes* distintas por meio de plug-in que permite a comunicação entre os mais variados protocolos de comunicação, como Wi-Fi e Bluetooth. Já a segunda, possibilita a execução de diversos atores em um *runtime* só, o Calvin GUI funciona nessa camada.

3.2 Ferramentas de Computação na Borda

No que se refere a implementação de uma nova arquitetura, infraestrutura ou serviço, é imprescindível a realização de testes, onde serão analisados a viabilidade, desempenho e configurações que serão adotadas. Visando isso, o uso de ferramentas que simulam condições semelhantes às desejáveis se torna uma prática. Dessa forma, o objetivo desta seção é apresentar as principais ferramentas de simulação identificadas e disponíveis para ambientes envolvendo *Fog* e *Edge Computing*, obtidas através de um processo de revisão

bibliográfica.

3.2.1 EmuFog

O EmuFog é um emulador open-source da MaxiNet[4] - *Distributed Emulation of Software-Defined Networks*, desenvolvido em Java que permite controle e repetibilidade dos experimentos. O MaxiNet, adveio do MiniNet, que possibilita a simulação de redes complexas, devido a sua infraestrutura distribuída [75], [27]. O EmuFog tem sua estrutura dividida em: geração de topologia; transformação da topologia; aprimoramento; desenvolvimento e execução.

A ferramenta suporta que a rede seja gerada usando o BRITE[5] ou importada do CAIDA[6]. Na segunda etapa, configurações como latência e a taxa de transferências de dados dos equipamentos são aplicadas; a rede é representada por um grafo unidirecional. Na etapa de aprimoramento é onde são usados os Nós *Fog*, que possuem configurações diferenciadas como, conexões máximas e custo. Fica a cargo do desenvolvedor especificar a capacidade computacional dos Nós e a quantidade de clientes que estarão conectados. Os componentes de aplicações são obtidos através de contêineres de aplicação Docker [7] e implantados nos Nós, o que permite a execução em várias plataformas, suporte à limitação de recursos de hardware, além de ser leve e seguro [43].

3.2.2 EdgeCloudSim

Esse simulador é uma extensão do CloudSim, framework open-source desenvolvido em Java que foi projetado para cenários envolvendo Computação na Borda. Dessa forma, o principal foco é a simulação de rede e recursos computacionais. O CloudSim é uma ferramenta que analisa o desempenho no que se refere a Computação na Nuvem. O EdgeCloudSim tem sua arquitetura dividida em: *Core Simulation, Networking, Load Generator, Mobility e Edge Orchestrator*. A simulação de dispositivos móveis é um dos pontos fortes dessa ferramenta, já que possibilita simular redes WLAN [67].

3.2.3 FogTorchπ

Essa ferramenta foi desenvolvida para dar suporte a ambientes envolvendo a nuvem, a borda e IoT. Esse protótipo é uma extensão do FogTorch, ferramenta essa open-source desenvolvida em Java. A mesma, é voltada para análise de recursos de processamento, latência, largura de banda, requisitos de processamento e a qualidade de serviço (QoS) de um aplicativo. Isso é feito através de modelagem de links de comunicação. Quando

[4] Disponível em `https://maxinet.github.io/`
[5] BRITE, `https://www.cs.bu.edu/brite/`
[6] CAIDA, `http://www.caida.org/home/`
[7] Docker, `http://www.docker.com`

entramos no âmbito de hardware, é levado em consideração, memória, armazenamento e núcleos de CPU. Já em software, o sistema operacional, linguagem de programação e estruturas de dados, são informações descritas em listas [11].

3.2.4 iFogSim

O conjunto de ferramentas de simulação iFogSim é outra extensão do framework CloudSim. Essa, assim como a EdgeCloudSim é open-source e desenvolvida em Java. No entanto, foi projetada visando simular ambientes que integram dispositivos IoT, como sensores e atuadores; sendo capaz de localizar-se na nuvem, na névoa ou na borda da rede [67]. Esse simulador através da inserção desses dispositivos, permite a criação de topologia, análise de políticas de gerenciamento de recursos, consumo de energia, congestionamento na rede, custo e latência [22].

Figura 13 – Diagrama da arquitetura do iFogSim proposta por Gupta et al [29].

O iFogSim tem sua arquitetura dividida em várias camadas, como pode ser visto na Figura 13, de baixo para cima temos: os dispositivos IoT, dispositivos *Fog*, geração de dados, monitoramento da infraestrutura, gerenciamento de recursos, modelos de aplicações e aplicações IoT. Na camada mais baixa, é onde estão situados os dispositivos IoT, que através dos seus sensores e atuadores, irão captar dados do ambiente e integrar com o meio a sua volta. Os dispositivos *Fog* são organizados em ordem hierárquica, oferecem recursos de memória, rede e computação; quando criados devem ser associados a uma taxa de

processamento de instruções e consumo de energia. Na geração de dados, o fluxo de dados pode advir de sensores ou de módulos de aplicativos. Quando gerado pelos dispositivos *Fog*, serão monitoradas pela camada seguinte. Nessa camada, a de monitoramento de infraestrutura é onde é feita a análise do desempenho dos dispositivos. Posteriormente, na camada de gerenciamento de recursos, tem-se uma previsão do uso de recursos que são administrados através de políticas de gerenciamento que controlam e gerenciam os serviços oferecidos pelos dispositivos [29]. A última é dedicada ao desenvolvimento de serviços e aplicações de Internet das Coisas, utilizando um cenário com Computação na Borda.

3.2.5 FogNetSim++

A ferramenta é um simulador de rede que inclui várias características de um ambiente de *Fog Computing*. Entre as que foram apresentadas é a única que possui ambiente de simulação de EC utilizando ferramentas como o NS-3 ou OMNeT++[8]. O FogNetSim++[9] foi desenvolvida como uma extensão do OMNeT++ que é uma ferramenta open-source comumente utilizada entre a comunidade acadêmica e desenvolvedores para simular características de rede e seus dispositivos que agem como os de uma rede real [57]. Além disso, o modelo de simulação presente no framework INET é empregado pelo toolkit para simular conexão cabeada, *wireless* e redes móveis, para isso, a ferramenta possui uma variedade de protocolos de implementação, como o INET que permite simular diversos protocolos em seu ambiente, como TCP, UDP, ICMP, IP, PPP, Ethernet e algumas rotinas de protocolos. [57]

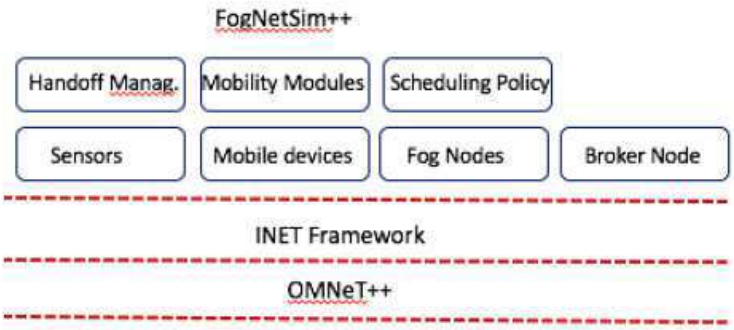

Figura 14 – Diagrama da arquitetura proposta do FogNetSim++ desenvolvida por Qayyum et al [57].

Se tratando de redes móveis, a extensão fornece uma referência quando se trata de parâmetros de rede como atraso na execução, taxa de perca de pacote, delay e latência. Além

[8] OMNeT++, `https://omnetpp.org/`
[9] FogNetSim++, `https://github.com/rtqayyum/fognetsimpp`

disso, devido a seu enforque em *Fog Computing*, mecanismo de gerenciamento de *handover*, algoritmos de escalonamento dos nós Fog além de apresentar modelos de mobilidade configuráveis a tornam uma ferramenta bastante atrativa em termos de simulação nesse tipo de ambiente [69]. Na Figura 14 é ilustrada a arquitetura da ferramenta e em blocos a representação dos dispositivos e algumas de suas configurações.

3.2.6 FogBed

O FogBed é um framework e uma ferramenta de integração que visa possibilitar uma rápida prototipagem de componentes fog em um ambiente virtualizado. Em termos gerais, a ferramenta é um emulador que utiliza o framework Mininet e contêineres Docker no ambiente virtualizado como forma de simular nós *Fog*, *Cloud* e *Edge*. Dessa forma, com o uso dessas ferramentas, foi desenvolvido um testbeds para um ambiente de *Cloud Computing* e *Edge/Fog Computing*. Essa ferramenta analisa quesitos como latência, atentando as limitações que os equipamentos Fog/Edge possuem, dessa forma leva em consideração requisitos não-funcionais como baixo custo, configuração flexível e compatibilidade com tecnologias do mundo real [17].

3.2.7 Outras ferramentas

Outras ferramentas foram identificadas, as quais abrangem apenas alguns aspectos do EC. O SherlockFog é voltado para aplicações MPI (*Message Passing Interface*), padrão que especifica um modelo para comunicação de dados em computação paralela. O que garante diferentes topologias, características de desempenho e capacidade computacional [23].

RECAP é um *framework* que visa implementar uma nova arquitetura na qual possa ser realizado o gerenciamento de recursos e análise de dados. Isso é proposto através de um simulador de cenários, em nuvem, borda e Computação na Névoa. Essa ferramenta, dispõe-se a realizar simulações de aplicações, de recursos de infraestrutura, de gerenciamento de recursos para então, realizar experimentação e validação dos resultados. No entanto, a ferramenta Recap está em desenvolvimento, e o projeto RECAP[10] é financiado pela Horizon 2020 [50].

De acordo com Massimo et al [22] várias soluções open-source foram propostas para dar suporte a arquitetura de Nuvem, de borda e névoa, entre elas os testbeds FIT IoT-Lab e o SmartSantander, e o simulador SimpleIoTSimulator. O FIT IoT-Lab[11] é uma plataforma de larga escala real distribuída na França que permite a construção, comparação e otimização de protocolos, serviços e aplicativos para o novo paradigma IoT. Essa plataforma possui mais de 2000 nós de sensores sem fio fixos ou móveis espalhados em

[10] RECAP, `https://recap-project.eu/`
[11] FIT IoT-Lab, `https://www.iot-lab.info/`

seis locais diferentes na França [1]. O testbed SmartSantander[12] adveio de um projeto que prevê a implantação de 20.000 sensores, que tem como objetivo permitir uma avaliação experimental em escala urbana de novas tecnologias IoT [61]. Por fim, o SimpleIoTSimulator, ferramenta comercial que permite simular cenários na borda de IoT em grande escala em dispositivos como sensores e gateways.

3.2.8 Comparativo entre ferramentas de Computação na Borda

Levando em consideração as ferramentas apresentadas anteriormente, foram analisadas as principais características de cada uma delas, a fim de avaliar em que ambiente cada uma se adéqua. Primeiramente, no EmuFog avaliou-se como objetivo principal o aprimoramento das características do framework da MaxiNet para um contexto de Fog Computing. Isso, através de uma nova política de posicionamento dos *nodes* Fog, o que em [43] é apresentado como uma melhoria na topologia da rede devido a escalabilidade dos algoritmos criados. O uso dessa ferramenta deve ser considerado quando se necessita de um cenário de teste mais realista e dinâmico onde não há possibilidade ou disponibilidade de criar e simular uma rede manualmente, apesar da plataforma também disponibilizar esse uso.

Tratando-se do EdgeCloudSim foi considerado como característica principal desse framework a simulação dos recursos computacionais e de rede inerentes a Computação na Borda. Diferente do anterior que tem como foco o ambiente Fog, com seus nós, comutadores e links; esse trata de serviços que podem ser desenvolvidos na borda da rede. Logo, o uso desse framework deve ser considerado para modelar e simular serviços móveis que sejam necessários transferir tarefas para a borda da rede. Já o iFogSim tem como principal objetivo fornecer uma ferramenta que simule ambientes IoT e Fog em grande escala, permitindo a modelagem de componentes como sensores, atuadores, dispositivos Fog e Cloud. O ponto forte desse framework é o fato de possibilitar aos usuários a implementação de soluções próprias para o gerenciamento de recursos, através da modelagem e verificação de políticas de alocação de serviços. Devido a isso, possibilitou que vários estudos fossem realizados com o uso dessa ferramenta.

O FogTorchπ dentre as ferramentas analisadas tem uma característica própria, fornece um meio que possibilite a validação de serviços e aplicativos voltados para o paradigma de Computação na Borda. Dessa forma, esse protótipo deve ser utilizado por exemplo, quando deseja-se determinar quais implementações de um aplicativo em uma infraestrutura Fog atende a todos os requisitos de processamento, IoT e QoS. Já o FogBed, assim como o EmuFog é um emulador, mas diferente desse, não permite carregar topologias existentes de rede, no entanto, com o uso do Docker nesse caso, é possível emular infraestruturas de Nuvem e *Fog* do mundo real, inicializando ou finalizando instâncias. A

[12] SmartSantander, `http://www.smartsantander.eu/`

Atributos	**EmuFog**	**FogBed**	**iFogSim**	**EdgeCloudSim**	**FogTorchπ**	**FogNetSim++**
Linguagem de Programação	Python	Python	Java	Java	Java	C++
Comunicação de rede	Sim	Sim	Sim	Sim	Não	Sim
Heterogeneidade	Não	Não	Não	Não	Não	Sim
Escalabilidade	Não	Não	Não	Não	Não	Sim
Mobilidade	Não	Não	Não	Sim	Não	Sim
Paradigma de Computação	Fog	Fog	Fog	Edge	Fog	Fog
Tipo	Emulador	Emulador	Simulador	Simulador	Modelo de Programação	Simulador
Gerenciamento de recursos	Carga de trabalho	Carga de Trabalho Consumo de Recursos Largura de banda	Consumo de Recursos Consumo de Energia Política de alocação	Consumo de Recursos Política de Mobilidade	Consumo de Recursos QoS	Consumo de Recursos
Configuração de rede	Sim	Sim	Não	Não	Não	Sim

Tabela 1 – Comparativo entre ferramentas selecionadas que suportam Computação na Borda.

mesma possui uma API de instância que fornece uma semântica de infraestrutura como serviço (IaaS) para gerenciar nós virtuais que permite integrar e testar sistemas de terceiros ou desenvolver sua própria interface de gerenciamento. Isso garante flexibilidade ao que se refere em estratégias de gerenciamento das instâncias. Deve ser usada quando diferente da EmuFog visa adotar diferentes formas de gerenciamento dos nós e avaliar métricas como RAM, CPU e largura de banda [17] [69].

Por fim, temos o FogNetSim++ que ao ser desenvolvido no topo do OMNeT++ permite simular redes Fog/Edge em larga escala, levando em consideração requisitos como mobilidade, assim como a EdgeCloudSim, mas o mesmo permite ir além, com a possibilidade de implementar diferentes modelos de mobilidade, lidar com *handover* e algoritmos de gerenciamento dos nós Fog. Aliás, o mesmo possui características comuns com o iFogSim, como, a possibilidade de inserir sensores em sua simulação, no entanto, diferente dessa outra ferramenta, utiliza um rede distribuída de dados. Por utilizar um simulador amplamente difundido pela comunidade acadêmica, possui uma biblioteca extensa para simular características de rede usando uma simulação baseada em eventos discretos, além disso, dispõe de interface gráfica que permite simular sua rede de forma simples, visualizar a interação entre módulos que é registrado em um arquivo de log que pode ser processado durante ou após a execução da simulação. De resto, a ferramenta permite analisar métricas como latência, delay, perda de pacotes e outros requisitos através de gráficos, histogramas na própria ferramenta ou em outras caso deseje exportar os dados. De forma sucinta, na Tabela 1 são apresentados alguns atributos comuns entre as ferramentas a fim de gerar um comparativo entre suas principais características.

3.3 Conclusões Finais do Capítulo

Com base na análise realizada neste trabalho, a arquitetura baseada em atores ou componentes fornece uma melhor forma de lidar com dispositivos IoT conectados em larga escala, ou seja, melhor escalabilidade, pois o *Middleware* pode ser implementado em todas as camadas da arquitetura. Por conseguinte, *Middlewares*, como Calvin e Node-RED, são uma boa escolha em aplicações que envolvam uma grande quantidade de "coisas". Levando em consideração o cenário em questão, a ferramenta Node-RED possui características importantes para a implementação do arcabouço proposto nesse trabalho. Dessa forma, a ferramenta será utilizada no desenvolvimento de aplicações IoT no contexto do trabalho aqui proposto. Detalhes sobre essa implementação serão discutidas no capítulo seguinte.

Seguidamente, para atingir os objetivos desse trabalho, é necessário simular aplicações IoT em um cenário de rede de Computação na Borda, incluindo também o paradigma de *Fog Computing*. Com base nisso, foi realizado uma ampla pesquisa por ferramentas que simule ou emule esse cenário. Realizado um comparativo entre essas ferramentas, foi identificado a ferramenta FogNetSim++, a qual permite simulações utilizando ferramentas como o NS-3 ou OMNeT++. Com isso, no Capítulo 4 serão utilizadas como base o *Middleware* Node-RED e simulador FogNetSim++ para o desenvolvimento da arquitetura proposta neste trabalho.

4 Arquitetura da Solução

O paradigma da Computação na Borda (EC) traz consigo uma variedade de possibilidades de serviços e aplicações, como já foi visto no decorrer deste trabalho. Desta forma, várias pesquisas estão sendo desenvolvidas na área a fim de viabilizar que esse novo cenário se torne uma possibilidade de implementação nos dispositivos atuais. Para isso, é necessário o desenvolvimento de ferramentas que possibilitem a desenvolvedores e engenheiros simularem o uso desse novo paradigma e compará-lo com as tecnologias desenvolvidas até então, seja Nuvem, *cloudlets* ou serviços de virtualização. Visando inserir o EC na rede atual, a comunidade tem aperfeiçoado ou desenvolvido ferramentas para que cada vez mais novos serviços sejam implementados.

No entanto, a quantidade de dispositivos conectados à Internet tem aumentado, e acaba por acarretar diversos problemas, como interoperabilidade entre dispositivos heterogêneos, gerenciamento de dispositivos, escalabilidade, gerenciamento de grandes volumes de dados, aspectos de segurança e privacidade. Desta forma, os *Middleware* são ferramentas que propõe lidar com essas lacunas trazendo consigo a possibilidade de desenvolver aplicações IoT em alto nível [6]. Todavia, apesar de terem sido desenvolvidas ferramentas como D-NR[1] e DataBox[2] por exemplo, que possibilitam desenvolver alguns tipos de aplicações IoT na borda, não foi encontrada uma ferramenta que possibilite aos projetistas desenvolverem e validarem suas aplicações em um ambiente simulado ao longo da revisão sistemática realizada.

Para os desenvolvedores se faz necessário testar e validar suas aplicações antes de implementá-las, com o intuito de evitar que não correspondam com suas necessidades e/ou de seus clientes. Quando realizamos o desenvolvimento de serviços em geral, é necessário integrá-los a diversos dispositivos de diferentes tipos, realizar a interligação dos mesmos utilizando diversos protocolos de comunicação e desenvolver os mais variados cenários de uso. Ao realizar esses testes, o desenvolvedor terá uma visão de como sua aplicação deve funcionar. No entanto, só desenvolver as aplicações não é suficiente para garantir o funcionamento delas no meio em que serão inseridas, para isso, é necessário simular este cenário. Desta forma, com o propósito de possibilitar a implementação e simulação de aplicações IoT em um ambiente de Computação na Borda, nesse capítulo é apresentada uma arquitetura que estabelece um elo entre um *Middleware* e um simulador.

[1] D-NR, `https://github.com/namgk/dnr-editor`
[2] DataBox, `https://www.databoxproject.uk/code/`

4.1 Visão Geral

Como introduzido nos capítulos anteriores, os *midllewares* facilitam a implementação, modificação e desenvolvimento de aplicações IoT em alto nível. Além disso, a possibilidade de utilizar componentes pré-definidos ou desenvolver novos, juntamente com a utilização de um modelo baseado em atores que permita a integração da ferramenta com diversos dispositivos IoT, foram fatores decisivos para a escolha dessas ferramentas no cenário proposto. Dessa forma, considerando as ferramentas apresentadas no Capítulo 3 e a revisão da literatura realizada, a ferramenta Node-RED é o ambiente de desenvolvimento de aplicações IoT que mais se adéqua ao cenário em questão. Além de possuir as características elencadas acima, dispõe de uma interface visual desenvolvida em um ambiente de computação distribuída. Isso permite então a prototipagem de aplicativos IoT codificadas de forma simples e executadas com um modelo de computação orientada a eventos que integram em tempo real dispositivos físicos. Isso se dá, em grande parte devido a sua vasta comunidade de colaboradores que permite desenvolver aplicações avançadas de forma rápida em diversos cenários como transporte, aquisição de dados e monitoramento, usando apenas o navegador [14], [40].

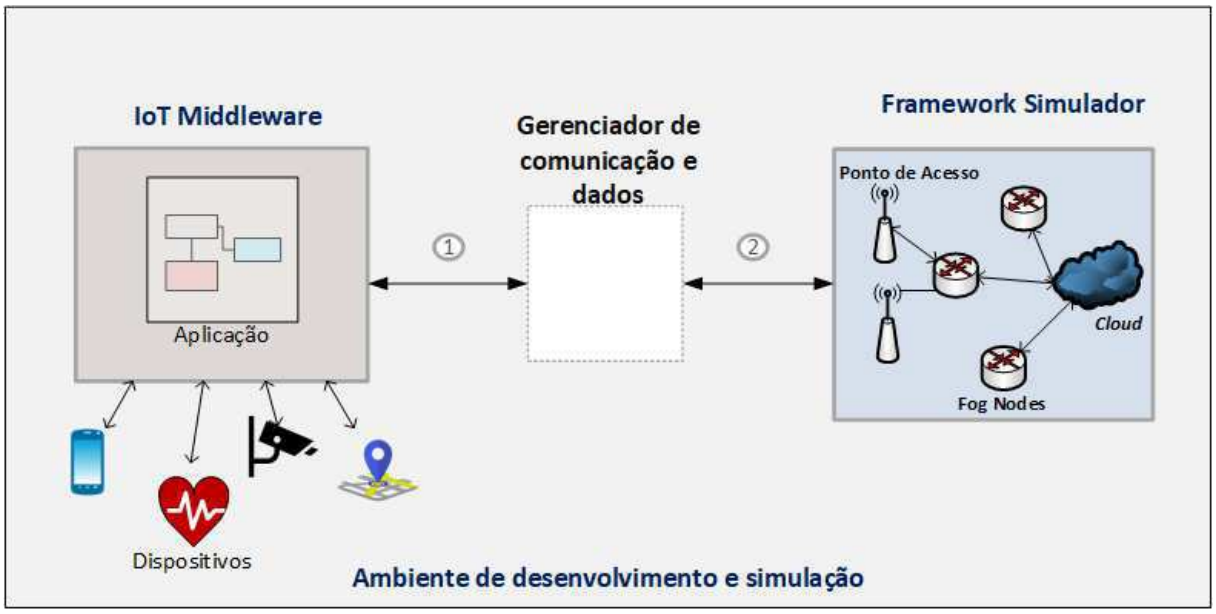

Figura 15 – Diagrama da Arquitetura Geral

Na outra ponta temos o FogNetSim++, um simulador de Computação na Borda que permite que usuários possam simular dispositivos IoT em seu ambiente. Além disso, permite que requisitos não funcionais sejam avaliados, como por exemplo, mobilidade e a customização de parâmetros de rede, como algoritmos de escalonamento e mecanismos de *handover*. O fator determinante para a escolha dessa ferramenta foi a utilização do

simulador OMNeT++, por ser uma ferramenta de código fonte aberto, e reconhecida pela comunidade acadêmica. O detalhamento dessa ferramenta será realizado a posteriori.

Isto posto, foi observado a necessidade de desenvolver um novo componente que se encarregue de estabelecer a integração entre as ferramentas. Com o entendimento que teremos um *Middleware*, um simulador e um componente que realizará o gerenciamento da comunicação entre essas ferramentas, foi definido então uma arquitetura geral para o cenário proposto. Na Figura 15 tem-se a representação de tal arquitetura. Primeiramente temos o *Middleware* IoT onde será desenvolvida as aplicações que podem ser conectadas a dispositivos como Smarthphone, GPS, câmeras ou equipamentos de monitoramento da saúde, a depender da aplicação desenvolvida pelo usuário. Dessa ferramenta deve ser implementada uma conexão bidirecional, a conexão 1, que deve interligá-la com o gerenciador, esse componente deve receber os dados e adequá-los para que sejam enviados e recebidos pelo simulador, através de outra conexão bidirecional, a conexão 2. Com isso, o simulador irá receber esses dados e inseri-los no ambiente simulado para então enviar novamente para o gerenciador que irá adequá-los novamente para que seja recebido pelo *Middleware*.

4.2 Comunicação entre as ferramentas

Para realizar a integração entre as ferramentas é necessário investigar os mecanismos e modelos de comunicação existentes nas mesmas. De acordo com Wasserman et al [74], há cinco formas de integrar ferramentas em um ambiente de engenharia de software: (i) integração de apresentação, (ii) de plataforma, (iii) de dados, (iv) de controle e (v) integração de processos. No entanto, antes de analisar qual tipo de integração será implementada, é indispensável analisar as ferramentas e as possíveis formas de integração entre as mesmas. O *framework* FogNetSim++, como extensão do OMNeT++, pode integrar ao seu código outros módulos disponíveis no OMNeT++, logo, como passo inicial, o entendimento dessa ferramenta, que segue como premissa para vários desenvolvedores, será o próximo passo a ser esclarecido.

4.2.1 OMNeT++

OMNeT++[3] (acrônimo para *Objective Modular Network Testbed in C++*) é um simulador usado para modelagem de redes de comunicação orientado à objetos, que dispõe de mecanismos e ferramentas para o desenvolvimento de simulações em diversos âmbitos. De acordo com a necessidade do desenvolvedor ou pesquisador, *frameworks* podem ser incorporados a ferramenta a fim de agregar outros tipos de parâmetros, dispositivos e configurações de outros cenários de uso. Como por exemplo, no caso do FogNetSim++

[3] OMNeT++, `https://omnetpp.org/`

é utilizado o INET[4], que utilizando as ferramentas disponibilizadas pelo OMNeT++ desenvolveu novos mecanismos para possibilitar a simulação de dispositivos móveis e suas configurações especificas. Devido a consolidação da ferramenta, companhias como IBM, Intel e Cisco a utilizam em projetos comerciais ou para pesquisas internas [57]. Dentre as principais características do OMNeT++ pode-se destacar a possibilidade de fornecer um Ambiente de Desenvolvimento Integrado (*Integrated Development Environment-IDE*) que facilita a simulação de rede em larga escala, o desenvolvimento de modelos e análise de seus resultados, e o fato de ser um ambiente customizável que permite incorporar simulações em aplicativos maiores, como software de planejamento de rede [73].

A ferramenta possui um modelo de estrutura baseado em blocos que podem ser agrupados e que se comunicam através de mensagem passante via *gates*, sejam eles de entrada ou saída. Algumas propriedades como taxa de dados, taxa de erro de bit e propagação de atraso podem ser acrescentadas entre as conexões dos módulos. Os usuários devem utilizar a linguagem de descrição de topologia, NED para definir a estrutura dos módulos. Essa linguagem possui uma representação equivalente a XML, dessa forma, arquivos NED podem ser convertidos em XML. Além disso, como mencionado, o OMNeT++ possui uma IDE que contém um editor gráfico que usa NED como formato de arquivo nativo, no entanto, isso não proíbe o desenvolvedor de escrever seu próprio código NED. Dessa forma, o usuário pode editar sua topografia de rede graficamente, utilizando o editor ou através de seu código NED. De forma sucinta, a ferramenta faz uma separação entre o comportamento do modelo que será empregado na simulação, que utiliza arquivos em C++ e o modelo da topologia que é definida em arquivos NED. Isso permite que um mesmo modelo de comportamento seja simulado de diversas formas, de acordo com diversos modelos de topologia que o usuário desejar simular. Sendo assim, os valores de uma topologia são armazenados em um arquivo INI, o que permite que um modelo de comportamento, tenha vários arquivos INI e NED atrelados. Na Figura 16 pode ser visualizada a interface do OMNeT++, a direita tem-se o editor gráfico, e a esquerda alguns projetos atrelados a ferramentas e exemplos de arquivos, NED e INI.

4.2.2 Adaptação do Node-RED para a arquitetura proposta

Ao analisar o OMNeT++/INET, tem-se com primeiro passo, verificar se a ferramenta possui mecanismos de interação externa com outros ambientes. No INET foi desenvolvida uma interface de conexão externa que permite a comunicação de nós simulados com nós externos utilizando uma pilha IP padrão em tempo real. Essa interface de conexão foi detalhada no trabalho de Tüxen et al. [72]. No entanto, como foi mencionado nesse trabalho, quando um pacote IP é recebido pelo *host* que está executando a simulação para um nó que está sendo simulado, ele deve ser transformado em um objeto

[4] INET, `https://inet.omnetpp.org/`

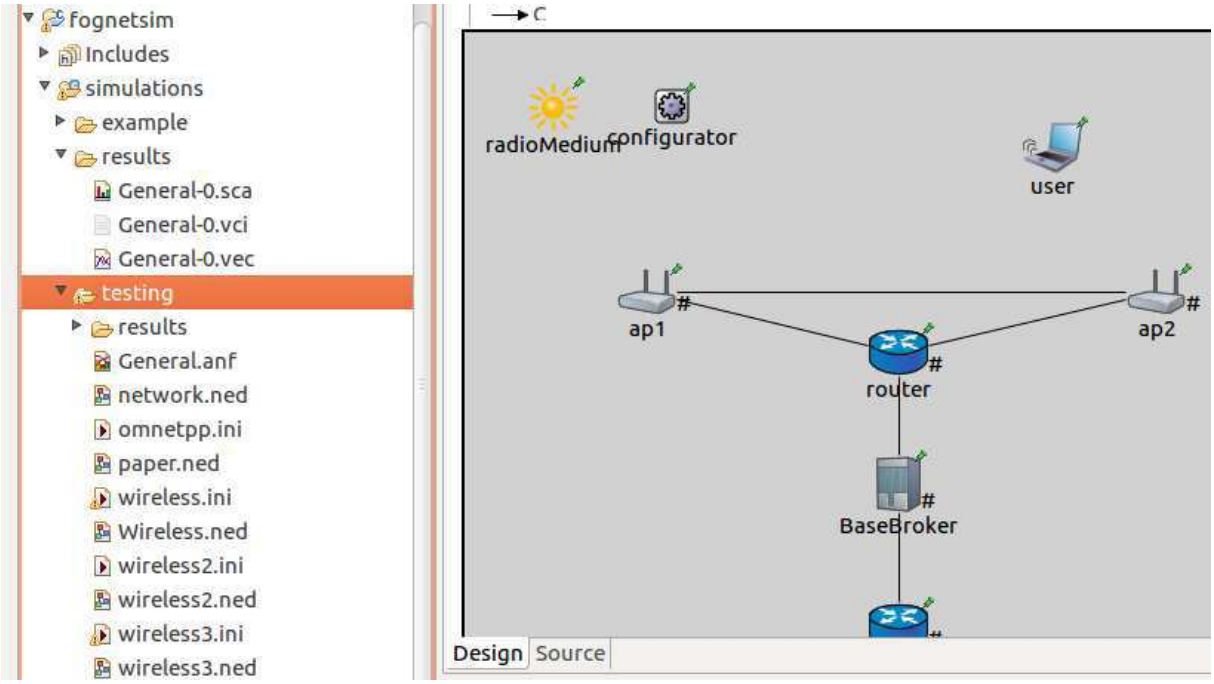

Figura 16 – Interface do OMNeT++.

OMNeT++ e inserido na rede simulada. Isso significa que, o uso de *raw socket* não é um mecanismo apropriado para receber esses pacotes, já que permite que a aplicação construa seus próprios cabeçalhos, diferentemente dos cabeçalhos do protocolo IP que é criado automaticamente. Nesse trabalho é recomendado que para receber pacotes de outros simuladores, deve ser usado o pacote *libpcap*[5] utilizando sistemas do tipo Unix, que pode capturar pacotes Ethernet. Posteriormente, ao capturar esses pacotes utilizando a ferramenta apropriada, o sistema dispõe um método responsável por analisar o pacote e transformar o pacote do formato de rede no formato interno da simulação.

No entanto, na versão 4.6 do OMNeT++ e 3.3.0 do INET ao qual a extensão FogNetSim++ foi desenvolvida não foram encontradas formas de desencapsular o pacote e extrair os dados enviados exteriormente. Dessa forma, a ferramenta possibilita o envio de pacotes do simulador para o mundo real através de *Raw Packet*, mas não possibilita que os dados sejam recebidos dessa forma, ou extraídas as informações quando utilizado o PCAP (API para capturar o tráfego de rede) disponível da biblioteca *libcap*. Sendo assim, essa interface não permite a modificação e manipulação da troca de mensagens entre a interface OMNeT++ e o ambiente externo, mas possibilita que desenvolvedores agreguem suas redes reais ao ambiente simulado.

Dessa forma, já que a interface OMNeT++ não dispõe em sua composição de meios que nos permita integrar as ferramentas, uma proposta de solução é deixar o sistema

[5] libpcap, `https://www.tcpdump.org/`

operacional (SO) realizar essa integração utilizado um mecanismo de Comunicação entre Processos, em inglês *Inter-Process Communication (IPC)*. O SO fornece vários mecanismos para essa comunicação e sua sincronização, dentre eles destacam-se, compartilhamento de memória, mensagem passante, semáforos, sinais e *pipe* [68].

Pipe é um buffer circular que permite que dois processos se comuniquem seguindo o modelo produtor/consumidor. Portanto, é uma fila *first-in-first-out*(FIFO), onde um processo escreve e o outro lê. O *pipe* possui um tamanho fixo quando criado, e o sistema operacional impõe um sistema de exclusão mútua no canal, apenas um processo pode acessar o canal por vez. Sendo assim, quando um processo tenta escrever no canal, essa escrita é executada imediatamente se houver espaço suficiente, caso contrário, o processo será bloqueado. De mesmo modo, um processo de leitura é bloqueado se tentar ler mais bytes do que tem disponível, caso contrário, a solicitação de leitura é executada imediatamente. O *pipe* pode ser de dois tipos, nomeado e não nomeado. No caso não nomeados, somente processos relacionados entre si podem compartilhar *pipes*. Todavia, quando utilizado o primeiro tipo de *pipe*, processos relacionados e não relacionados podem compartilhar *pipes* nomeados [68]. No cenário proposto, o *pipe* nomeado é o mais adequado, já que permite a comunicação entre processos não relacionados, OMNeT++ e Node-RED.

Com isso em mente, foi avaliado se o *Middleware* Node-Red possuía algum ator desenvolvido utilizando IPC. Foi encontrado o node *node-red-contrib-ipc*[6], que cria um servidor em um arquivo especificado pelo desenvolver. Dessa forma, outros processos podem se conectar e enviar mensagens através do arquivo que foi criado. No entanto, esse bloco, não possibilita a conexão bidirecional entre o Node-RED e outro processo, já que o mesmo só recebe mensagens.

4.2.2.1 Novo Componente para o Node-RED

Sendo assim, foi optado por desenvolver um novo componente que permita enviar e receber mensagens do OMNeT++. Dessa forma, essa nova contribuição desenvolvida no decorrer desse trabalho possui essa comunicação bidirecional necessária. Na Figura 17 tem-se uma representação do detalhamento do novo componente, *Named-Pipe* e as conexões entre esse bloco, outros blocos no *host* do Node-RED e do arquivo *pipe* que permite estabelecer a comunicação entre a ferramenta e qualquer outro processo, no cenário em questão o OMNeT++. Especificando cada uma das conexões:

- as conexões 1, 4 e 6 são para enfatizar que cada um desses blocos estão disponíveis no *host* Node.js, inclusive o novo bloco;
- na conexão 2, temos o bloco *inject* representado nesse diagrama pelo bloco Mensagem, pois ele enviará mensagens para o bloco *Named-Pipe*, seja no formato *string* ou JSON.

[6] node-red-contrib-ipc, `https://flows.nodered.org/node/node-red-contrib-ipc`

Essas mensagens também podem ser enviadas repetitivamente em um intervalo pré definido pelo usuário;

- na conexão 3, tem a conexão estabelecida entre o componente *Named-Pipe* e o arquivo *pipe*. Assim que esse arquivo é criado, na sintaxe do bloco *Named-Pipe* é aberta essa conexão, o que possibilita o envio e recebimento de dados por esse canal. Logo, essa conexão é estabelecida implicitamente pelo SO;
- por fim, temos a conexão 5, que é uma conexão que pode ser realizada caso o usuário deseje conectar a saída do *Named-Pipe* a outro bloco quaisquer do *host* Node.js, como por exemplo, o bloco função, que pode acrescentar alguma outra informação, modificá-la ou usar como condição para a realização ou não de outra funcionalidade.

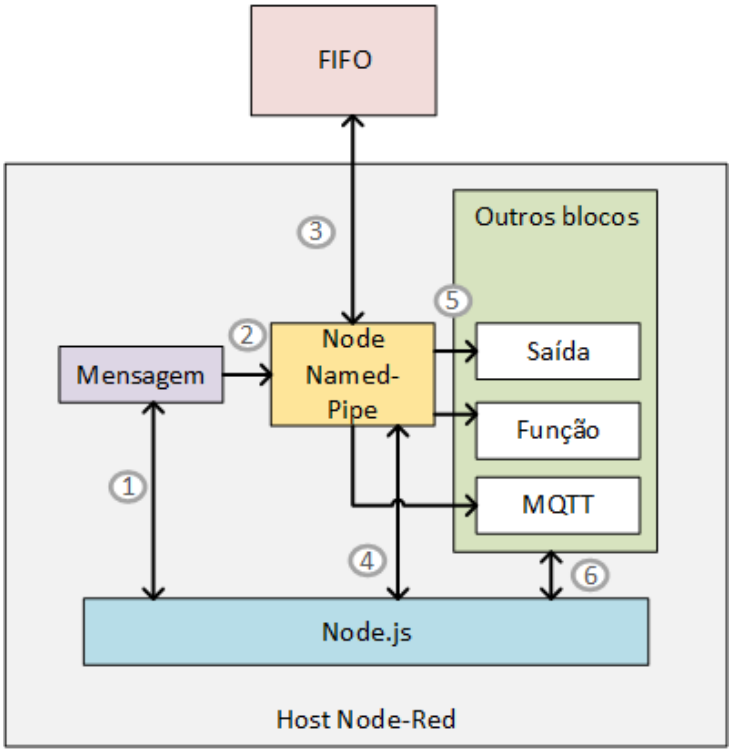

Figura 17 – Diagrama com visão geral do componente *Named-pipe* no Host Node-RED

Para o melhor entendimento de como será o fluxo de dados entre os componentes apresentados no diagrama da Figura 17, no diagrama de sequência apresentado na Figura 18 pode ser visualizada como será a troca de mensagens entre o componente *Named-Pipe* e demais entidades. Inicialmente, um servidor deve ser desenvolvido, o qual será responsável por estabelecer a comunicação entre o OMNeT++ e o Node-RED. No exemplo apresentado no diagrama da Figura 15, esse servidor, chamado de gerenciador, é executado primeiro e fica em modo de espera, no entanto, normalmente fica a cargo do cliente iniciar a comunicação.

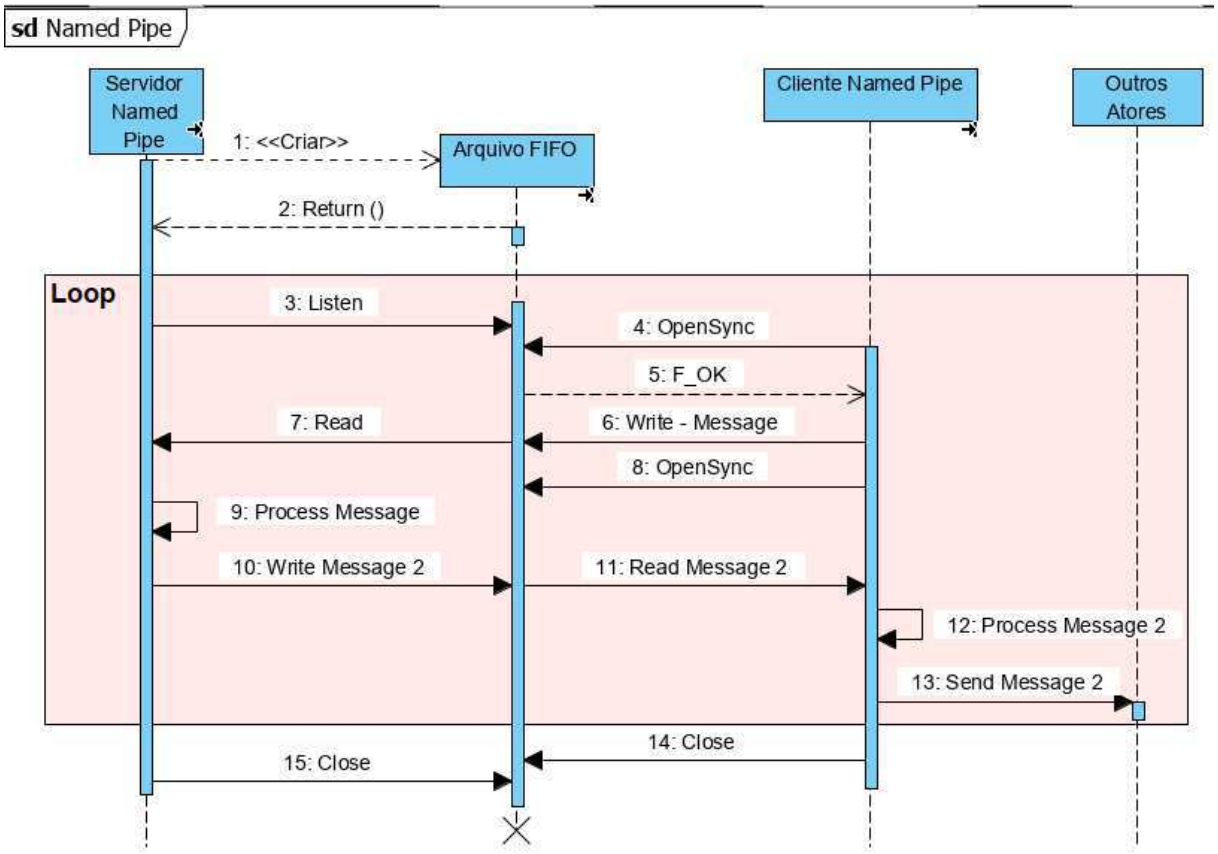

Figura 18 – Diagrama de sequência do novo componente proposto

Sendo assim, para a transmissão de dados do Cliente Node-RED, mensagens são trocadas entre esse componente, o gerenciador (Servidor *Named-Pipe*) e demais atores, através de conexões diretas com ele ou do arquivo FIFO. Logo, o fluxo de mensagens segue o passo a passo como descrito no diagrama da Figura 18:

1. O Servidor *Named-Pipe* inicia o fluxo com a criação do arquivo FIFO através do Servidor;

2. Caso o arquivo não seja criado, ou no caso da arquitetura proposta, não seja possível para o servidor escrever e lê nesse arquivo, uma mensagem de erro é retornada;

3. Se o arquivo foi criado, o servidor entrará no loop e começará a esperar pelo recebimento de uma mensagem;

4. Após o usuário inserir os blocos e conexões apresentadas na Figura 17, deve ser habilitada a implementação. Ao realizar isso, o *Named-Pipe* será executado, o que possibilitará a comunicação entre o cliente e o arquivo FIFO. Permitindo então, o envio e recebimento de dados;

5. Assim como no servidor, a troca de mensagens só será possível caso o cliente *Named-Pipe* possa escrever e lê no arquivo FIFO;
6. Após a confirmação de que a comunicação pode ser estabelecida, o cliente escreve a mensagem no arquivo FIFO;
7. Essa mensagem será lida pelo servidor;
8. Após o envio da mensagem, o cliente irá se preparar para lê uma mensagem, habilitando então a conexão para esse novo estado;
9. Nesse passo, ocorre o processamento da mensagem, que varia de acordo com a necessidade do desenvolvedor, mas a mesma não pode deixar de ser readequada para o novo processo que irá recebe-lá;
10. Com o processamento da mensagem, que pode ser a modificação da mensagem recebida ou uma nova mensagem enviada por outro processo, o servidor escreverá essa segunda mensagem no arquivo (*Message 2*);
11. O cliente *Named-Pipe* em seu novo estado, fará a leitura da nova mensagem (*Message 2*);
12. O novo componente irá readequar a mensagem e permitir que novos atores a utilizem, no passo a seguir;
13. Por fim, na saída do bloco *Named Pipe*, podem ser conectados diferentes blocos, como por exemplo, o bloco *function*, representado por função, o *mqtt in*, representado por MQTT ou *debug* presentado por Saída que apenas mostrará a mensagem recebida.

Por fim, o fluxo de funcionamento típico do componente *Named-Pipe* baseado na troca de mensagens com o arquivo FIFO e demais componente, é apresentado no fluxograma da Figura 19. Em uma comunicação entre ferramentas distintas, que utilizam linguagens de programação diferentes, é indispensável a readequação de parâmetros para uma comunicação ser estabelecida corretamente. Além disso, no fluxograma da Figura 19 é apresentada uma lógica para o envio de mensagem para o *host* Node-RED, com o intuito de garantir que a mensagem recebida seja diferente da mensagem enviada pelo *Named-Pipe*.

4.3 Conclusões Finais do Capítulo

Neste capítulo foi apresentada uma arquitetura base para contemplar o objetivo deste trabalho. Na definição dessa arquitetura, foram definidas as ferramentas que serão utilizadas, o FogNetSim++ e o Node-RED. Seguidamente, foram esclarecidos alguns conceitos necessários para o entendimento da implementação e comunicação entre elas.

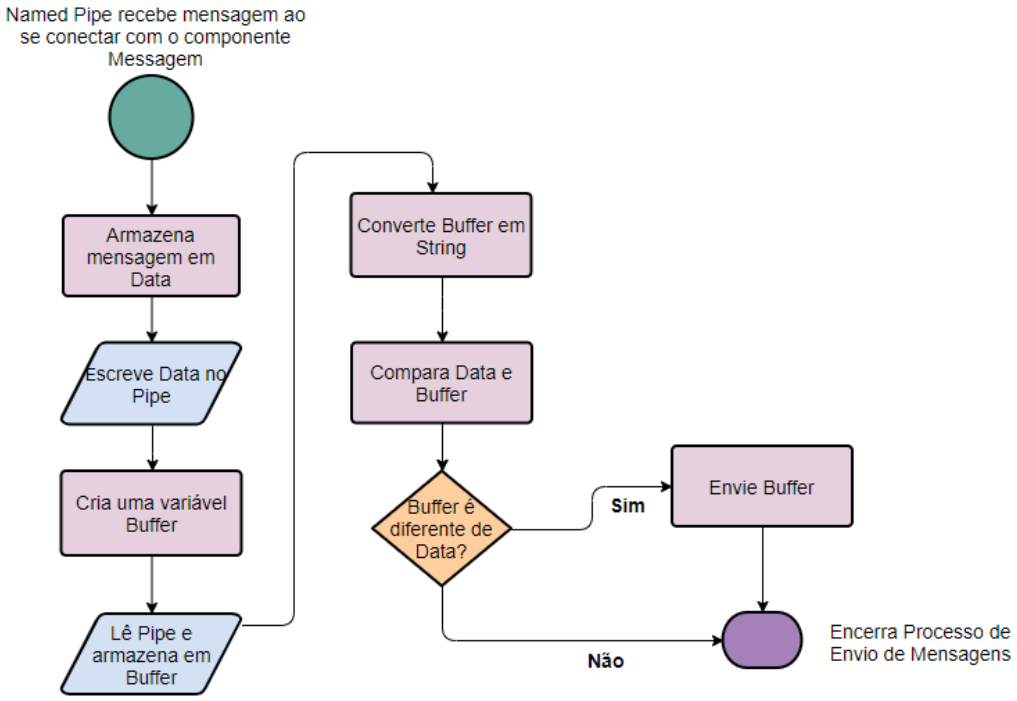

Figura 19 – Fluxo da lógica implementada no *Named Pipe*

Além disso, foi apresentada uma nova contribuição, o desenvolvimento de um componente que foi implementado no Node-RED a fim de possibilitar a comunicação IPC utilizando *pipes* entre essa ferramenta, o gerenciador, e consequentemente, o OMNeT++. Através do diagrama apresentado na Figura 17, foi exemplificado como o mesmo pode ser utilizado no *Middleware* Node-RED, e como tal componente pode ser conectado a outros blocos que a ferramenta já disponibiliza. Além disso, no diagrama da Figura 18 pode ser visualizado o detalhamento de como ocorre o fluxo de dados na comunicação IPC. Por fim, tem-se a definição do fluxo da lógica implementada no *Named-Pipe* para desde a chegada dos dados até o envio pelo nó de saída de dados.

5 Implementação e Validação do Arcabouço

Como detalhado no Capítulo 4, a arquitetura apresentada consiste na integração da ferramenta OMNeT++ e o *middleware* Node-RED. Para isso, foi desenvolvido um novo componente no Node-RED que permite a comunicação entre essa ferramenta e outros processos. A fim de viabilizar a integração, este capítulo apresenta detalhes sobre como foi introduzido esse novo componente na concepção do arcabouço, e o que foi implementado de modo a viabilizar o desenvolvimento e simulação de aplicações IoT em um cenário de Computação na Borda. Após a implementação, é apresentado um caso de uso que possibilitou identificar requisitos funcionais e não funcionais do sistema, onde foi possível compreender de forma objetiva a interação entre os componentes desse arcabouço. Visando validar tais requisitos, foram descritos procedimentos experimentais realizados para a avaliação do arcabouço em diferentes cenários de rede. Na execução desses experimentos, foi possível identificar o funcionamento e as limitações do sistema proposto, e propor soluções através de ajuste em alguns parâmetros.

5.1 Implementação

Para o desenvolvimento do arcabouço, foi utilizada como base a arquitetura proposta e apresentada no Capítulo 4, a qual utiliza como comunicação entre o OMNeT++ e o Node-RED um mecanismo de comunicação entre processos. O trabalho apresentado em [42] descreve três técnicas de integração da ferramenta OMNeT++ com outras aplicações fora do ambiente simulado:

- comunicação via *socket*;
- integração através de código fonte;
- compartilhamento de bibliotecas.

Dentre essas técnicas destaca-se o compartilhamento de bibliotecas, que pode utilizar ao mesmo tempo a comunicação via *socket* e a integração através de código fonte. A sua utilização assemelha-se a de incorporar o código fonte da ferramenta a ser integrada sem necessitar que essa seja readaptada para o ambiente de simulação. Essa forma de integração acaba por não necessitar de grandes mudanças em nenhuma das ferramentas, pois, se faz necessário apenas a inclusão dessas bibliotecas e adaptações nos ambientes de compilação, como por exemplo, vinculando tais bibliotecas ao compilar, ou alterando o seu arquivo de construção (exemplo, Makefile). Dessa forma, como o ambiente de simulação do OMNeT++ fornece um kernel com bibliotecas de classes escritas em C++, a melhor

solução encontrada foi a utilização do mecanismo de comunicação entre processos (IPC), mais especificamente, a utilização de *pipe* nomeado já que possibilita a comunicação entre duas aplicações distintas em um mesmo ambiente. Dessa forma, como será utilizada o IPC, tem-se uma integração que irá compartilhar dados entre os processos envolvidos. Agora que foi definido o tipo de integração e o mecanismo que será utilizado, se faz necessário o entendimento de quais modificações foram realizadas no OMNeT++, para que possa enviar e receber dados externos.

5.1.1 Adaptação do FogNetSim++

Na seção 3.2.5 foi introduzida algumas das funcionalidades da ferramenta FogNetSim++. No entanto, para o entendimento de como devemos implementar a comunicação nesse simulador, é necessário conhecer o funcionamento dessa extensão. Essa ferramenta possui três componentes diferentes, os *Fog Nodes*, o módulo *Fog Broker* e os dispositivos finais. Os dispositivos finais juntamente com o *Broker* estabelecem comunicação utilizando mensagens MQTT. Dessa forma, esses dispositivos podem ser *Publisher*, *Subscriber* ou ambos, cabendo ao *Broker* intermediar essa conexão, e encaminhar os dados para os dispositivos que subscreveram para recebê-los. Todavia, entre o envio e o recebimento dos dados, pode ocorrer que o *Broker* envie os dados para dispositivos intermediários, os *Fog Nodes*, que também são gerenciados por ele. Esses dispositivos oferecem serviços de computação, que receberá os dados vindo dos *Publisher* e irá processá-los e encaminhar a resposta/resultado caso tenha algum *Subscriber*. Na Figura 20, pode ser visualizada uma representação desses componentes, e equipamentos intermediários necessários para que a conexão seja estabelecida.

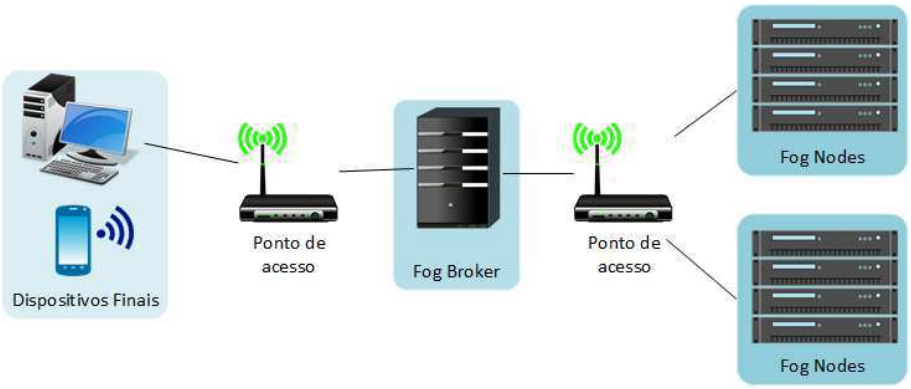

Figura 20 – Representação dos dispositivos do FogNetSim++.

Primeiramente, têm os dispositivos finais, que podem ser sem fio ou possuir uma conexão cabeada até o ponto de acesso, esse irá encaminhar as requisições vindas desses dispositivos para o *Broker*. Sendo esse, responsável por registrar e gerenciar os dispositivos dessa ponta, como da outra, os *Fog Nodes*, realizando a organização das solicitações

dos dispositivos, gerenciando seus recursos, possíveis *handovers* e estabelecendo um link com *data centers* para que seja possível a entrega de dados para esses equipamentos, caso os nós não possam atender tais demandas [57]. Sendo assim, após o *Broker* receber as solicitações dos dispositivos finais, poderá tratá-las ou encaminhar os dados para os *Fog Nodes*, seguindo os critérios de proximidade entre os dispositivos. Na Figura 21, é apresentado um diagrama de fluxo que representa como a ferramenta estabelece conexão com seus componentes. Nesse diagrama, tanto os clientes (dispositivos finais) como o *Fog Node* solicitam a conexão com o *Broker*, após a conexão ter sido estabelecida, o *Broker* fica no aguardo para que as requisições sejam feitas. O cliente *Publisher* então publica os dados no *Broker*, que encaminha para serem processados no *Fog Node*, o resultado desse processamento será enviado para os *Subscribers* que solicitaram, através de um *Publisher*.

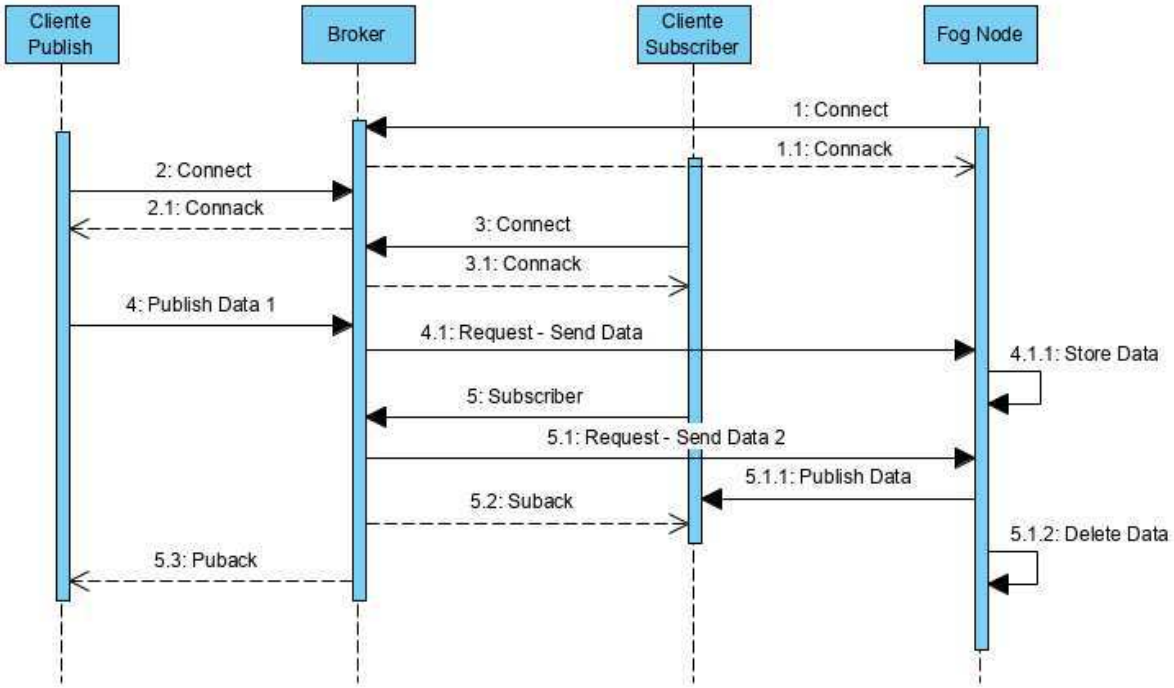

Figura 21 – Representação do fluxo de mensagens no FogNetSim++.

No passo seguinte foi analisado como a ferramenta foi implementada e seus dispositivos e conexões. Analisando uma conexão IPC, temos o *Publisher* como Produtor dos dados e o *Subscriber* como consumidor, dessa forma, a conexão do simulador com o gerenciador será realizada utilizando esses dois componentes. O *Publisher* nesse caso, ficará responsável por realizar a leitura dos dados que serão enviadas pelo gerenciador. Após isso, irá passar o tamanho da *string* de dados recebidas para ser publicada. Já o *Subscriber*, assim que receber os dados requeridos irá comunicar ao gerenciador o recebimento. Para que a conexão *Named Pipe* seja estabelecida, foi desenvolvido o Algoritmo 1 que será acrescentado

no código do *Publisher* a função Read_Data, e no *Subscriber* a função Write_Data. Nesse Algoritmo, a função Read_Data apresentada na linha 4, irá abrir o FIFO e verificar se no mesmo possui dados para serem lidos, caso sim, a função irá fazer a leitura desses dados e irá enviar o seu tamanho para a função MQTTPUBLISHER_SETBYTELENGTH, presente na linha 8. Já a função Write_Data presente na linha 9, irá abrir o FIFO e verificar se tem algum dado a ser enviado pelo canal, caso exista, os dados são escritos nesse FIFO.

Como mencionado no capítulo anterior, a ferramenta OMNeT++ possui blocos, esses são associados a um arquivo C++ que descreve suas funcionalidades, na maioria das vezes também possui um cabeçalho .h onde são definidas as classes pertencentes a associação de vários módulos. Entre essas, a ferramenta possui uma classe base para todos

Algorithm 1 Named Pipe

```
1: include <Communication_Libraries>
2: define Fifo_File[2]
3: int declareSomeVariables;
4: function Read_Data(void)
5:     fd ← open Fifo_File[0]
6:     if fd > 0 then
7:         read data
8:         function mqttPublish_setByteLength(data)
9: function Write_Data(void)
10:    fd ← open Fifo_File[1]
11:    if fd > 0 then
12:        write data
```

os módulos simples, a *cSimpleModule*[1]. Nela são apresentadas três funções-membro:

- void initialize() - Chamada uma única vez depois dos modelos serem criados, neste método que as variáveis são inicializadas;
- void handleMessage(cMessage *msg) - Contém a lógica interna do módulo, esse método é chamado pelo kernel do simulador quando o módulo recebe uma mensagem;
- void finish() - Essa função é chamada quando o simulador termina sua execução.

Sendo assim, quando uma mensagem chega em um dispositivo final, é a função handleMessage que irá determinar que tipo de ação será realizada, no caso do *Publisher*, irá chamar uma função que determinará qual mensagem será enviada e para onde, assim como no caso do *Subscriber*, quando uma mensagem enviada pelo *Publisher* chega, esse módulo encaminha para outra função que realizará seu processamento. Assim então, que as

[1] cSimpleModule, `https://doc.omnetpp.org/omnetpp/api/classomnetpp_1_1cSimpleModule.html`

funções presentes no Algoritmo 1 ao ser acrescentadas nessas funções do handleMesssage, serão executadas.

Por fim, com o detalhamento das modificações realizadas no FogNetSim++ e no Node-RED, falta compreender como foi implementado o gerenciador.

5.1.2 Desenvolvimento do Gerenciador

Como mencionado nos capítulos anteriores, o gerenciador tem como função permitir que seja estabelecida conexão entre o FogNetSim++ e o Node-RED, organizar e gerenciar os dados. Dessa forma, esse módulo tem como funcionalidade:

- Criar os *pipes*;
- Iniciar comunicação;
- Receber as mensagens proveniente do Node-RED;
- Enviar mensagens para o OMNET++;
- Receber mensagens provenientes do OMNET++;
- Enviar mensagens para o Node-RED;
- Encerrar comunicação.

Logo, a comunicação entre os componentes deve seguir a ordem apresentada na Figura 22. Nessa Figura, temos em 3 e 4 apenas o envio do tamanho da mensagem. Isso se dá pois, por mais que a mensagem chegue no OMNeT++, a mesma não tem como trafegar pelo simulador, pelos motivos elencados na seção 4.2.2, mas pode-se enviar o tamanho da mensagem e esse valor será encaminhado via MQTT para o *Broker*, que por sua vez será recebido pelo *Subscriber* que enviará novamente para o gerenciador. Porém, essa limitação do simulador acaba por demandar que o gerenciador salve as mensagens recebidas pelo *Named-Pipe* e posteriormente, quando receber do FogNetSim++ a confirmação do fim da simulação, acesse tal mensagem e encaminhe-a para o *Named-Pipe* novamente. Isso levou a implementação de uma tabela de dispersão, em inglês *hash table*.

Essa tabela tem uma estrutura de dado que relaciona chaves de busca a valores, dessa forma, eventos são nomeados com chaves o que faz com que o armazenamento de tais eventos bem como a recuperação sejam realizados usando essas chaves. Na Figura 23 é apresentada uma representação desse tipo de tabela. Nesse tipo de estrutura, tem-se uma função *Hash* ou função de dispersão que será responsável por realizar a associação das chaves com o *index*, armazenando o valor associado a chave em cada *index*, que no caso, serão as mensagens recebidas do *Named-Pipe* através do *Pipe*. O mesmo ocorrerá quando

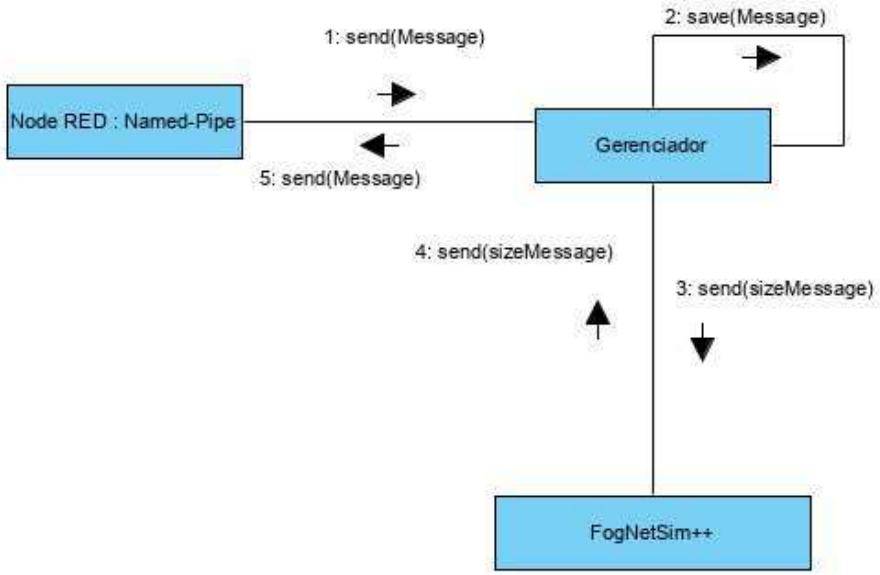

Figura 22 – Diagrama de comunicação entre os componentes.

o *Subscriber* enviar a confirmação do recebimento dos dados para o gerenciador. Com essa chave, a função *hash* associa novamente o *index* ao local onde a mensagem foi armazenada, resgata essa informação e manda pelo *Pipe* a mesma mensagem de volta ao *Named-Pipe*.

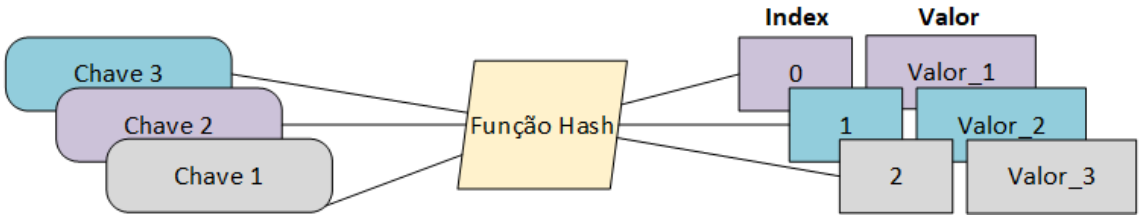

Figura 23 – Representação de uma *Hash Table*.

Posto isto, este componente tem funções bastante similares a um *Broker* em uma conexão MQTT. Para o entendimento de seu funcionamento, pode-se observar o pseudo código presente no Algoritmo 2. Nesse Algoritmo, a função PIPE_NODE_OMNET presente na linha 6, irá criar os *pipes* nomeados 1 e 2 e abri-los, caso tenha dados a serem lidas no *pipe* 1, a função irá salvar esses dados na tabela e apenas enviar o tamanho desses dados através do *pipe* 2. Já na função PIPE_OMNET_NODE, serão criados os *pipes* 3 e 4. Após a abertura da conexão, será verificado se há dados a serem lidos no terceiro *pipe* aberto, caso sim, serão resgatados os dados salvos na tabela pela função PIPE_NODE_OMNET para serem enviados através do quarto *pipe*.

Algorithm 2 Gerenciador

```
1: include <Communication_Libraries>
2: include <Thread_Library>
3: define Fifo_File[4]
4: int declareSomeVariables
5: Create Hash_Table
6: function *PIPE_NODE_OMNET(void* fd)
7:     Create Fifo_File[1]
8:     Create Fifo_File[2]
9:     file1 ← open Fifo_File[1]
10:    file2 ← open Fifo_File[2]
11:    while 1 do
12:        if (read_file1 data)>0 then
13:            save data in the table
14:            write_file2 length_data
15: function *PIPE_OMNET_NODE(void* fd)
16:    Create Fifo_File[3]
17:    Create Fifo_File[4]
18:    file3 ← open Fifo_File[3]
19:    file4 ← open Fifo_File[4]
20:    while 1 do
21:        if (read_file3 data)>0 then
22:            get data from the table
23:            write_file4 data
24:            delete data from the table
```

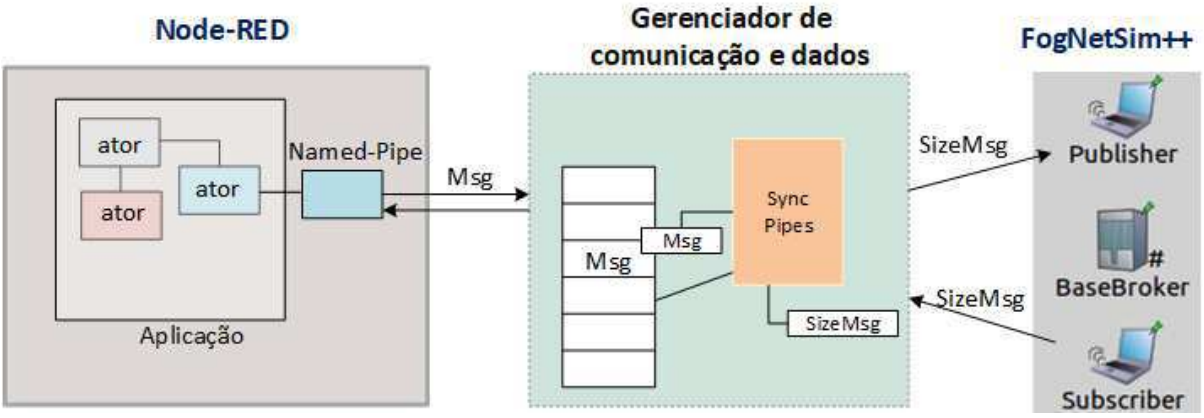

Figura 24 – Representação Final da Arquitetura Implementada

Sendo assim, foi elaborada uma tabela de dispersão e duas funções, uma destinada a conexão do Node-RED para o OMNeT++ e outra para gerenciar a conexão do OMNeT++ para o Node-RED. Dessa forma, a primeira função irá enviar os dados recebidos para a tabela, ficando a segunda função, após a confirmação do recebimento dos dados enviados pelo OMNeT++, encarregada de resgatar tais dados, enviar para o Node-RED e deletá-

lo da tabela. Como apresentado no Algoritmo 2, dois *pipes* terão acesso a tabela de dispersão, para garantir o acesso de forma concorrente, foi utilizado a biblioteca *threads*, representada na Figura 24, pela função *Sync Pipes*. Por fim, na Figura 24 é apresentada uma representação da arquitetura final implementada.

5.1.3 Padronização das Mensagens

Como mencionado anteriormente, a comunicação estabelecida entre as ferramentas tem um processo de envio e recebimento de mensagens bastante parecida com a base do protocolo de comunicação MQTT, *Publisher*, *Subscriber* e *Broker*. No entanto, quando nos referimos a esse protocolo e seus componentes, para que os dados publicados por um componente cheguem a aqueles que desejam essa informação, é necessário o envio da mensagem e o tópico ao qual essa mensagem pertence, por parte do *Publisher*, e cabe ao *Subscriber*, informar quais dados deseja receber por meio de tópicos específicos. Pensando nisso, foi optado pelo envio de mensagens seguindo o formato JSON. O objeto JSON é uma sequência de chaves/valores, cuja chave deve ser uma *string* e o valor pode ser número ou uma *string*, por exemplo. Como esse formato é um subconjunto da linguagem JavaScript é conveniente implementá-lo no Node-RED. Sendo assim, ao adequar a *string* de envio para o formato JSON, na informação a ser publicada, foi utilizado o componente *json* do Node-RED que irá converter JSON *String* e Objeto antes de enviar a informação pelo *named-pipe*. A mensagem então será recebida pelo gerenciador através de uma *String* JSON. A coleta das chaves e valores presentes nessa *string* foi realizada ao utilizar a biblioteca *json-c*[2], que teve o *header json.h* adicionado nos seus *includes*. Isso foi possível através de uma análise sintática da *string* JSON ao utilizar um método *parse* disponível nesta biblioteca.

Com a utilização do método *parse*, foi desenvolvido um mecanismo que utiliza os próprios tópicos como chaves para armazenar as mensagens na *hash table*. Após isso, foi utilizado outro método da mesma biblioteca *json-c* para enviar o tópico e o tamanho da mensagem através de uma *string* JSON para o OMNeT++. No entanto, apesar de enviarmos o tópico para o OMNeT++, ao definir o ambiente de simulação que será executado, deve-se determinar quais tópicos deseja-se publicar e/ou subscrever, dessa forma, os mesmos devem ser pré-definidos antes, no *.ini* da simulação correspondente no FogNetSim++. Seguindo o mesmo padrão de mensagens, foi acrescentado ao FogNetSim++ nas definições que serão executadas pelo *Publisher* e *Subscriber* na simulação, a biblioteca *RapidJSON*[3] que é um gerador de JSON rápido para C++. Com isso, fecha-se o ciclo ao enviar uma *string* JSON quando o *Subscriber* na simulação receber os dados desejados.

[2] json-c, `https://github.com/json-c/json-c`
[3] *RapidJSON*, `http://rapidjson.org/`

5.2 Validação

Após detalhamento de como foi implementado o arcabouço proposto nesse trabalho, nessa etapa serão descritos os procedimentos realizados para a avaliação e validação da integração realizada através de um cenário de uso, o que facilita a compreensão do que foi desenvolvido neste trabalho.

Para a verificação do arcabouço foi proposto uma aplicação em prova de conceito de monitoramento de ambulâncias. Essa aplicação exemplo tem como propósito interligar as ambulâncias da cidade ou de uma região, a centrais de monitoramento. Ao detectar alguma ocorrência próxima, informações podem ser encaminhadas para essas ambulâncias, como a localização de suplementos, de hospitais ou profissionais especializados em um determinado caso. As ambulâncias, durante a prestação de socorro a um determinado paciente, podem necessitar de respostas rápidas, que serão determinantes na sobrevivência ou redução de dados ao paciente, como por exemplo, procedimentos rápidos que podem ser realizados em caso de superdosagem de um medicamento ou quadro alérgico grave. Na Figura 25 é apresentado um diagrama desse caso de uso, onde temos a integração entre computação e comunicação para monitorar e controlar a rede de ambulâncias. Nesse sistema um usuário irá passar as informações para Central a respeito da ocorrência, a gravidade do paciente e a localização, por exemplo. Após adquirir essas informações, a Central irá passar as informações para as ambulâncias que estão próximas do paciente. Caso os socorristas necessitem de um dado rápido para o atendimento do paciente, irá adquirir essa informação através dos *Fog Nodes* espalhados pela região, e realizará os procedimentos. Posteriormente, atualizará a Central a respeito do atual estado do paciente e suas necessidades.

Para o desenvolvimento do cenário foi escolhido o sistema operacional Linux, com uma distribuição Ubuntu 16.04 LTS. As ferramentas foram instaladas nesse ambiente, o OMNeT++ 4.6 e adicionados na mesma, o framework INET 3.3.0 e a extensão FogNetSim++, além do *Middleware* Node-RED. Nesse ambiente, foi desenvolvido o gerenciador que intermediou a troca de dados entre as ferramentas.

5.2.1 Requisitos de Validação do Arcabouço

Para a validação do arcabouço, foi realizado o levantamento dos requisitos funcionais e não funcionais usando como exemplo a aplicação da central de monitoramento de ambulâncias. Os requisitos do sistema a serem validados são:

1. **ID**: FR1
   Título: Estabelecer comunicação entre diferentes aplicações.
   Descrição: O sistema deve ser capaz de enviar e receber mensagens de diferentes aplicações. Por exemplo, entre a aplicação ambulância e a aplicação central.

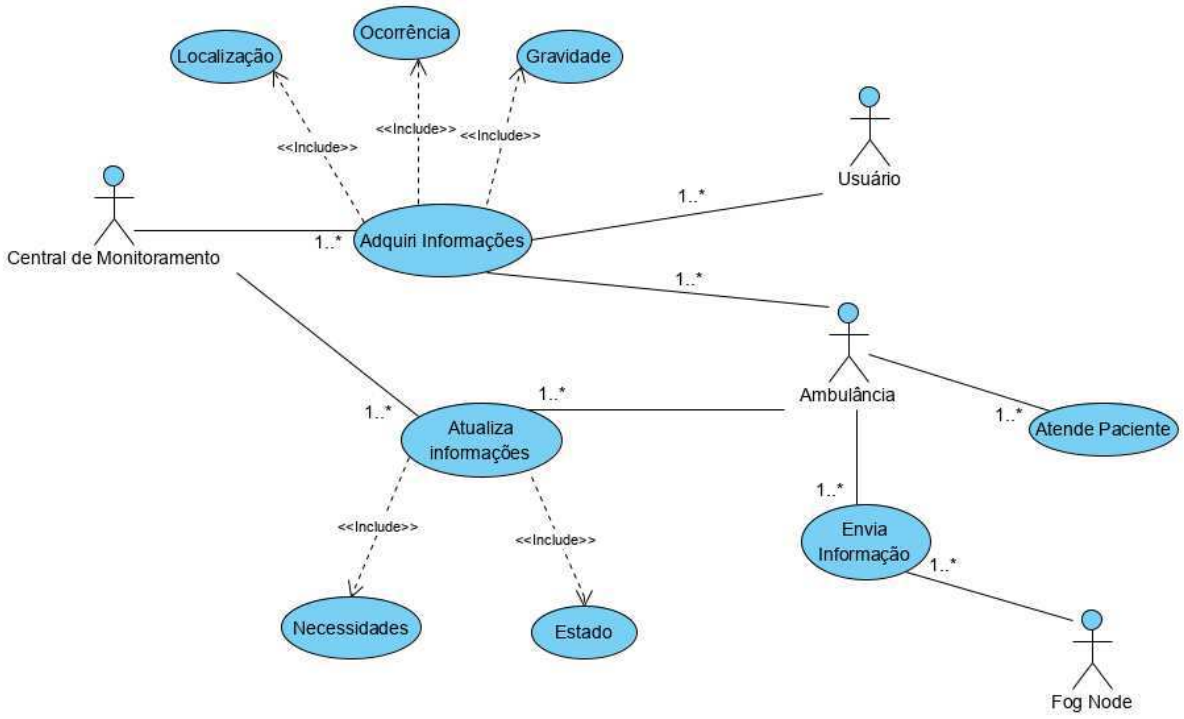

Figura 25 – Diagrama de Caso de Uso.

2. **ID**:FR2
   Titulo: Comunicação simultânea com várias aplicações.
   Descrição: O sistema deve ser capaz de enviar dados concorrentemente entre várias aplicações. Por exemplo, permitir que várias aplicações ambulâncias enviem ao mesmo tempo no ambiente simulado.

3. **ID**: FR3
   Titulo: Simulação de diferentes cenários de Computação na Borda.
   Descrição: O sistema deve ser capaz de viabilizar a simulação da aplicação proposta em diferentes ambientes de teste em um cenário de Computação na Borda.

4. **ID**:FR4
   Título: Alteração de configurações computacionais e quantidade de aplicações.
   Descrição: O sistema deve ser capaz de permitir a expansão da capacidade de processamento dos nós no ambiente simulado, de modo a acomodar diferentes cenários de uso e um maior número de aplicação.

5. **ID**:FR5
   Título: Alterações de configurações na aplicação
   Descrição: O sistema deve ser capaz de permitir alterações no comportamento da

aplicação. Por exemplo, alterações configurações que modifique o tempo de resposta da aplicação.

Dependência: FR3.

6. **ID**:NFR1
   Título: Garantir confiabilidade no envio e recebimento de mensagens.
   Descrição: O sistema deve ser capaz de garantir que todas as mensagens da aplicação sejam enviadas e recebidas entre o *Middleware* e simulador. Por exemplo, todas as mensagens enviadas pela aplicação central devem chegar na aplicação ambulância, e vice-versa.

7. **ID**:NFR2
   Título: Flexibilidade e garantia na mudança de parâmetros.
   Descrição: O sistema deve ser capaz de garantir que alterações na aplicação tenham impacto na simulação.

Após o levantamento dos requisitos é necessário definir casos de testes que serão executados a fim de validá-los. Nesses casos de testes serão detalhados, seu objetivo, metodologia aplicada e os resultados obtidos.

5.2.2 Experimento de Validação do Envio de Mensagens entre Aplicações

O principal objetivo desse experimento é avaliar se a comunicação ocorre da forma esperada entre a aplicação no Node-RED e o simulador. Com isso podemos validar parcialmente os seguintes requisitos:

- FR1;
- FR3.

Para isso foram utilizadas as seguintes configurações:

- O componente *Named-Pipe* deve enviar mensagens a cada x segundos;
- FogNetSim++ deve possuir um ambiente com um *Publisher* e um *Subscriber*.

Processo Experimental

A fim de validar os elementos do sistema, foram definidos alguns passos para viabilizar o objetivo do experimento:

1. Realizar a conexão individual entre o Node-RED e o gerenciador;

2. Realizar a conexão individual do gerenciador e o FogNetSim++;
3. Realizar a conexão entre os três componentes;
4. Avaliar se os dados seguem a sequência apresentada na Figura 22.

O procedimento funcionou conforme o esperado em todas as execuções.

Adequações e Discussão

Foram realizados testes com o componente *named-pipe* a fim de validar a comunicação entre o ele e o gerenciador. Durante esse processo, foi analisado que, no caso de uso proposto, em determinados momentos pode surgir a necessidade de que a aplicação apenas publique informações, isso é, que a ambulância apenas publique informações para a central de atendimento, ou que apenas se subscreva para receber informações. Dessa forma, alterações foram realizadas para adequar o *framework* a este cenário. Em consequência disso, foi optado por separar o componente *named-pipe* desenvolvido no Node-RED em dois, um *Publisher* e um *Subscriber*, permitindo mais flexibilidade no envio e recebimento de informações. Sendo assim, foram acrescentados ao ambiente os componentes, *named-pipe-sub* e o *named-pipe-pub*, como apresentado no diagrama da Figura 26. Após isso, foram realizados testes entre o gerenciador e o FogNetSim++, nos quais pôde-se verificar como o método *handleMessage* funciona com a chegada de mensagens.

Sequencialmente, foi realizado um teste de integração entre os componentes para verificar a troca de mensagens entre eles. Nesse teste, a aplicação envia mensagens através do *named-pipe-pub* para o gerenciador de forma contínua, que por sua vez identifica cada mensagem e salva na tabela de forma sequencial. Após salvar uma mensagem na tabela, o gerenciador envia outra mensagem teste para o *Publisher* do simulador. Esse processo se repete a cada mensagem recebida. Assim que o método *handleMessage* permitir que o *Publisher* envie informações, o *pipe* é aberto, então a mensagem enviada pelo gerenciador é identificada e o tamanho da mensagem recebida é publicado utilizando um tópico definido previamente. Assim que o *Subscriber* daquele tópico recebe a mensagem publicada, uma mensagem é enviada para o gerenciador confirmando o seu recebimento. Por fim, o gerenciador ao receber a confirmação, resgata a mensagem salva na tabela e a encaminha para o *named-pipe-pub*.

Após o teste de integração, foi realizado um teste de componente. Nesse teste foi verificado que, seguindo a ordem da comunicação entre os componentes apresentada na seção anterior, o gerenciador inicia a comunicação, a aplicação desenvolvida no Node-RED e por fim, a simulação, como apresentado na Figura 22. Caso a ordem de execução seja diferente, há opção da simulação ou da aplicação serem executadas antes do gerenciador. É indicado que o padrão seja seguido, contudo não ocorrerá falha caso não seja empregado,

já que a comunicação entre as ferramentas só é estabelecida caso os três componentes já tenham sido iniciados.

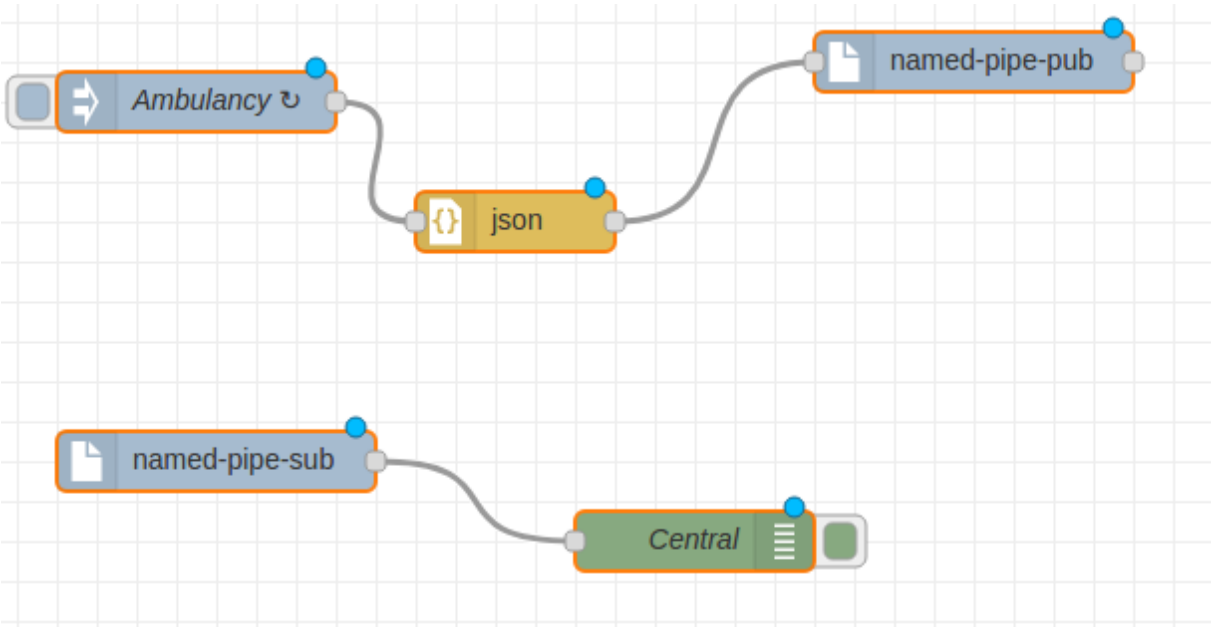

Figura 26 – Representação da Individualização do Subcriber e Publisher.

Resultados

Por meio das configurações de cenário de rede apresentados nas Figuras 26 e 27, foi possível validar o requisito FR1, que consistia em estabelecer comunicação entre a aplicação (central) e a aplicação (ambulância). Nesse teste, foi considerado que a aplicação (ambulância) envia informações para a aplicação (central) através do *named-pipe-pub*, que receberá os dados através do *named-pipe-sub*. A mensagem só será recebida pela aplicação (central) quando a troca de dados ocorrer entre o *Publisher* (ambulância) e o *Subscriber* (central) presente no ambiente simulado. Nesse processo, quando implementadas as configurações de redes apresentadas na Figura 27, foi possível validar o requisito FR3, que consiste em viabilizar a simulação da aplicação em um cenário de Computação na Borda. Nesse teste foram utilizados componentes de redes que permitem o gerenciamento e processamento de dados em dispositivos na borda, representados no diagrama da Figura 27 por meio do *Base Broker*, *ComputeBroker1* e *ComputeBroker2*. Sendo assim, nesse primeiro teste foi possível validar a integração entre as ferramentas, e analisar a troca de mensagens entre os componentes. Com isso, foi verificada a comunicação estabelecida, o que possibilita a realização de outros experimentos a fim de analisar parâmetros e configurações do arcabouço desenvolvido.

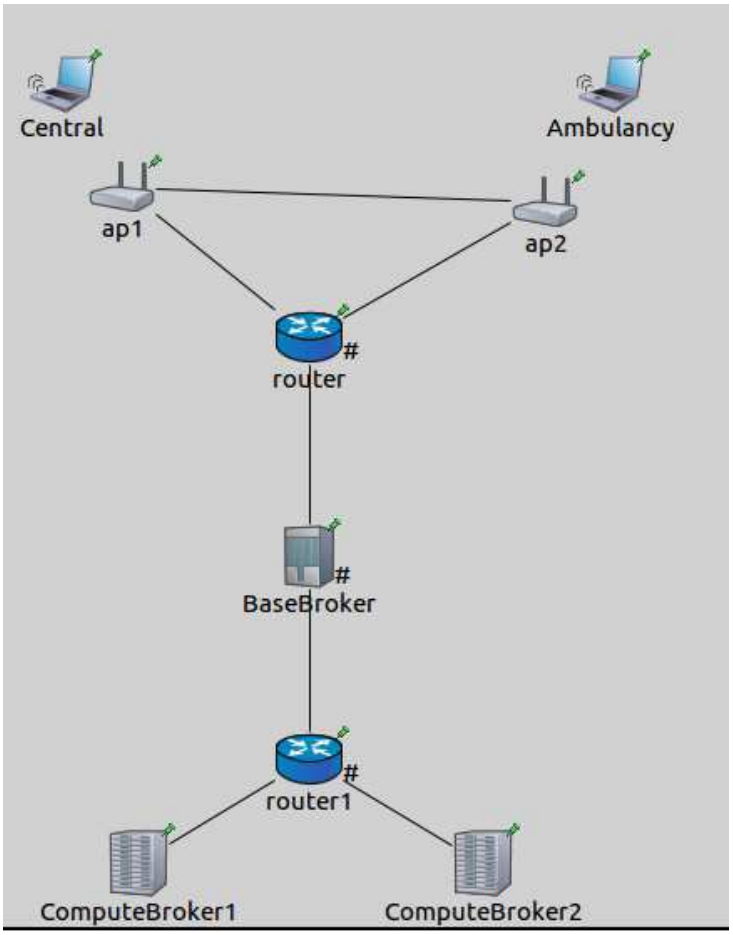

Figura 27 – Representação da aplicação no ambiente simulado.

5.2.3 Experimento de Validação da Comunicação entre Múltiplas Aplicações

O principal objetivo desse experimento foi avaliar como o sistema se comporta ao inserir mais de um *Publisher* e *Subscriber* na aplicação. Além disso, foi possível validar os seguintes requisitos:

- FR2;
- NFR1.

Para isso foram utilizadas as seguintes configurações do arcabouço desenvolvido:

- Dois *Publishers* que devem enviar mensagens a cada x segundos e dois *Subscribers* para o recebimento;
- Duplicação da quantidade de *pipes* em relação ao experimento anterior;
- Dois gerenciadores;

Processo Experimental

Para a avaliação do objetivo do experimento, foi definido o seguinte procedimento:

- Enviar mensagens distintas por meio dos *Publishers*;
- Avaliar se cada *Subscriber* recebe apenas a mensagem que solicitou;
- Avaliar se no ambiente simulado as mensagens serão recebidas apenas a quem solicitou;

Adequações e Discussão

Para possibilitar a utilização de múltiplas aplicações foi necessário duplicar a quantidade de componentes no Node-RED, acrescentando outro *named-pipe-pub* e outro *named-pipe-sub*. Como para estabelecer novas comunicações precisamos utilizar a mesma quantidade de *pipes* utilizados até então, foi preciso acrescentar mais dois novos blocos na aplicação. Apesar da abordagem não ser a mais intuitiva, é preciso apenas copiar os blocos existentes, modificar suas identificações e definir outros *pipes* para a comunicação, como apresentado nas Figuras 28 e 29. Para o entendimento da troca de mensagens entre os componentes, na Tabela 2 é apresentado o padrão utilizado para a troca de mensagens. A aplicação (central) envia mensagens por meio do tópico atendimento, e a aplicação (ambulância) envia outras, utilizando o tópico disponibilidade. Os *Subscribers*, se inscrevem em um dos tópicos disponíveis. Na aplicação apresentada na Figura 28, a *Ambulancy2* recebe mensagens enviadas pela aplicação (central) por meio do tópico atendimento, já na Figura 29, a aplicação central, irá receber as mensagens enviadas pela aplicação (*Ambulancy*) por meio do tópico disponibilidade.

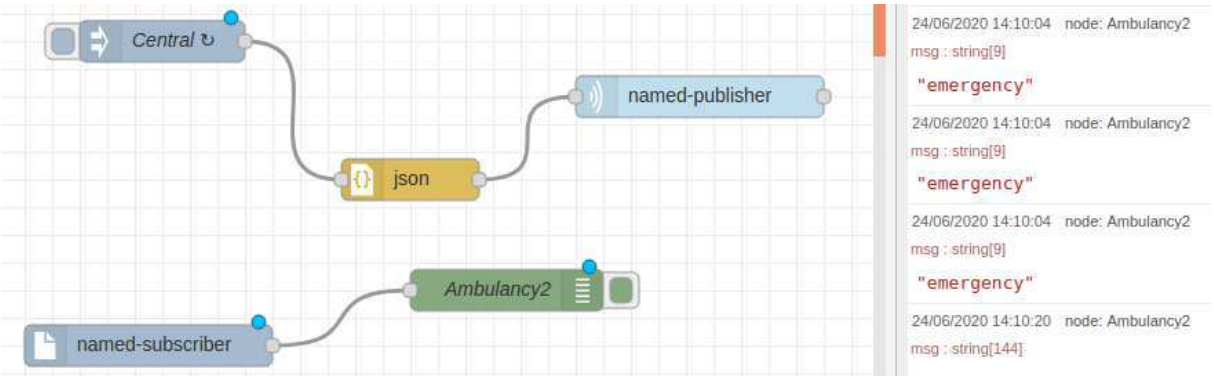

Figura 28 – Representação dos primeiros *Publisher* e *Subscriber*.

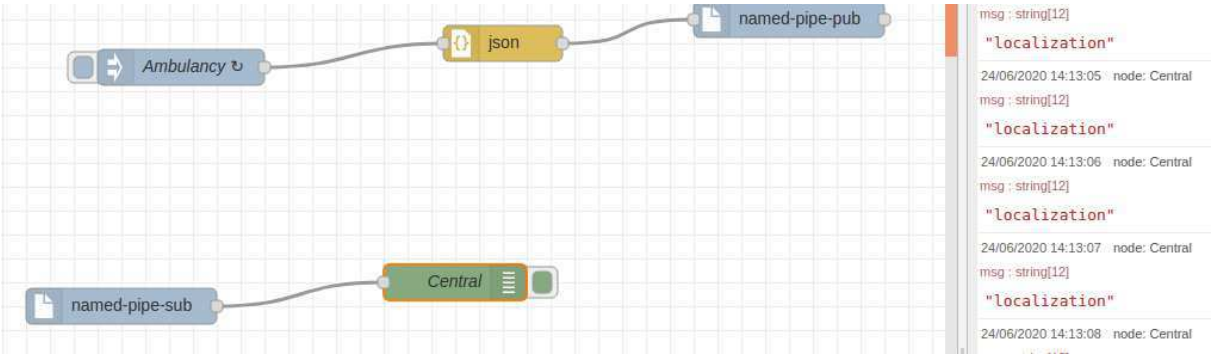

Figura 29 – Representação dos segundos *Publisher* e *Subscriber*.

Aplicação	JSON String	Subscriber
Central	{"topic: "atendimento";"message": emergency"}	Ambulancy2
Ambulância	{"topic:"disponibilidade";"message":"localization"}	Central

Tabela 2 – Padronização de mensagens.

Resultados

Com a definição do padrão JSON para mensagens e o envio utilizando tópicos, foi possível validar o cumprimento do requisito NFR1, que visa garantir que o sistema seja capaz de enviar mensagens por meio de múltiplas aplicações (central/ambulância) de forma confiável. No diagrama da Figura 28 é possível visualizar o recebimento das mensagens esperadas pelo node *Ambulancy* com a mensagem "emergency". O mesmo se repete no diagrama da Figura 29, no qual o nó Central recebe as mensagens esperadas com a mensagem "localization". Por fim, foi possível validar o requisito FR2, que visa possibilitar o envio de dados concorrentemente entre várias aplicações. Apesar da ampliação da aplicação não ter se dado de forma dinâmica, o sistema foi capaz de permitir o envio de mensagens através das aplicações de forma contínua e simultânea.

5.2.4 Experimento de Validação do Uso de Aplicações em Diferentes Cenários de Rede

O principal objetivo desse experimento foi avaliar como o sistema se comporta ao variar os parâmetros de rede, como mobilidade, desempenho dos *Fog Nodes* e a mudança (inserção ou remoção) de equipamentos intermediários de rede. Além disso, foi possível avaliar os seguintes questionamentos e requisitos:

- Como o simulador irá reagir ao utilizarmos algoritmos de mobilidade diferentes?
- Como o sistema irá reagir ao utilizarmos mais roteadores, pontos de acesso e *Fog Nodes*?

- Como o sistema irá reagir ao modificarmos o desempenho dos *Fog Nodes*?
- FR4.

Para isso, foram utilizadas as seguintes configurações do arcabouço desenvolvido:

- Algoritmos de mobilidade Linear e Circular;
- Vários *Publishers* no ambiente simulado;
- Modificação da quantidade de equipamentos intermediários de rede;
- Valores de MIPS = 1000 e 3000 nos *Computers Brokers*.

Processo Experimental

Para a avaliação do objetivo do experimento, foi definido o seguinte procedimento:

- Realizar a conexão dos *Publishers* e definir o intervalo de envio de mensagens;
- Modificar a mobilidade linear que está sendo usada para a circular;
- Simular com um número variado de *Publishers* e *Subscribers*;
- Acrescentar mais equipamentos de rede;
- Modificar desempenho dos *Computer Brokers*;
- Avaliar as métricas em cada um dos casos anteriores.

Adequações e Discussão

O primeiro experimento realizado avaliou a aplicação de dois tipos de mobilidades distintas implementadas nos *Publishers* e *Subscribers* no ambiente simulado. Nesse teste, foi possível variar parâmetros de mobilidade e observar que a aplicação desenvolvida se comportou melhor quando foi utilizado um modelo de mobilidade linear, se comparado com o modelo de mobilidade circular. Além desses modelos de mobilidade, o simulador possui outros modelos definidos, como *GaussMarkovMobility*, que baseia-se no teorema de Gauss-Markov, *MassMobility* que é um modelo de mobilidade aleatória empregado em [52] e o *TractorMobility*, que tem o intuito de simular a movimentação de um trator por um campo com uma quantidade de linhas definidas. A escolha do modelo linear e circular, se deu por serem alguns dos modelos empregados no simulador FogNetSim++. Contudo, como esses modelos são definidos no INET, podem ser substituídos facilmente nas configurações da simulação. Como o objetivo do experimento foi avaliar como o sistema

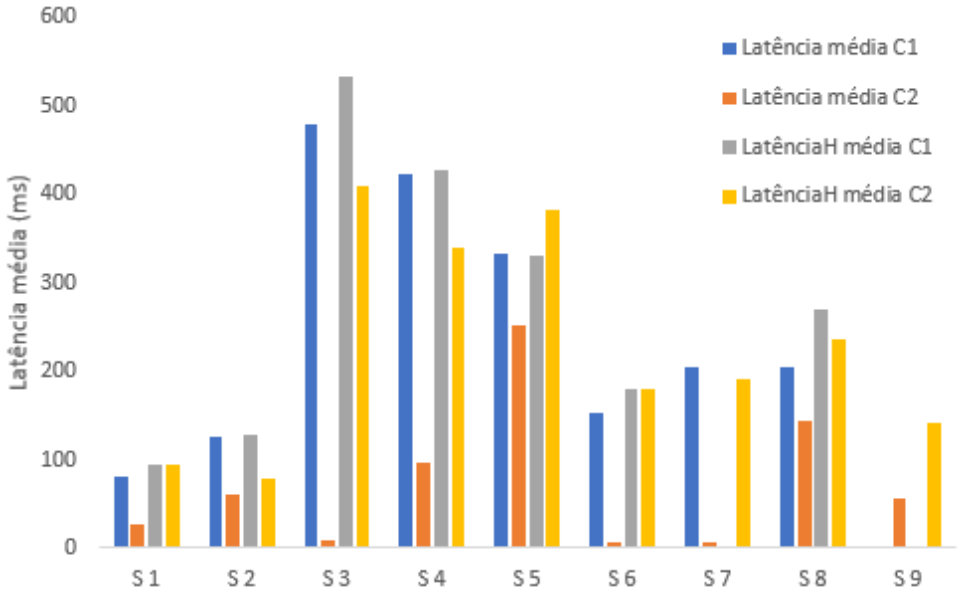

Figura 30 – Gráfico da relação entre a latência média e dois cenários diferentes de teste.

se comportar ao variar o modelo de mobilidade, poderiam ter sido escolhido quaisquer dois modelos na avaliação.

Em um segundo momento, foi realizado outro tipo de experimento. Nesse teste, foi modificado o ambiente de teste e os equipamentos de rede que estavam sendo utilizados. No gráfico da Figura 30, são apresentados os dados obtidos nas simulações (representadas pela letra S da Figura) realizadas nesse teste. Nessa Figura são ilustrados dois cenários diferentes, C1 e C2. Em C1, são utilizados dois pontos de acesso e dois roteadores, de forma que a diferença entre os dois cenários foram os pontos de acesso (AP) e a localização dos *Publishers*. Em C1 são utilizados pontos de acessos que possuem comunicação sem fio e utilizam o padrão IEEE 802.11. Além disso, tanto os APs como os usuários foram introduzidos ao sistema simulado de forma aleatória. Já no segundo cenário, foram utilizados pontos de acesso estacionários previamente escolhidos, e todos os *Publishers* iniciaram sua movimentação do mesmo ponto. Como pode ser observado, o valor da latência média foi reduzido consideravelmente na maioria das simulações. No mesmo gráfico, também é analisada a latência superior (*higher latency*). Nesse caso, os *Computer Broker* não executam as solicitações imediatamente, sendo então encaminhadas para uma fila, pois o componente está executando outra solicitação. Essa latência é o tempo que o *Compute Broker* leva para receber a solicitação e encaminhar a tarefa para a fila de execução. Dessa forma, pode ser observado que o valor da latênciaH média também foi reduzida na maioria das simulações. Após isso, uma confirmação foi enviada para o *Publisher* que recebeu a solicitação. Com isso, pode ser concluído que utilizar pontos de acesso estacionários e definir o mesmo ponto de partida para todos os *Publishers* faz com que o sistema tenha um desempenho melhor, independente das simulações realizadas.

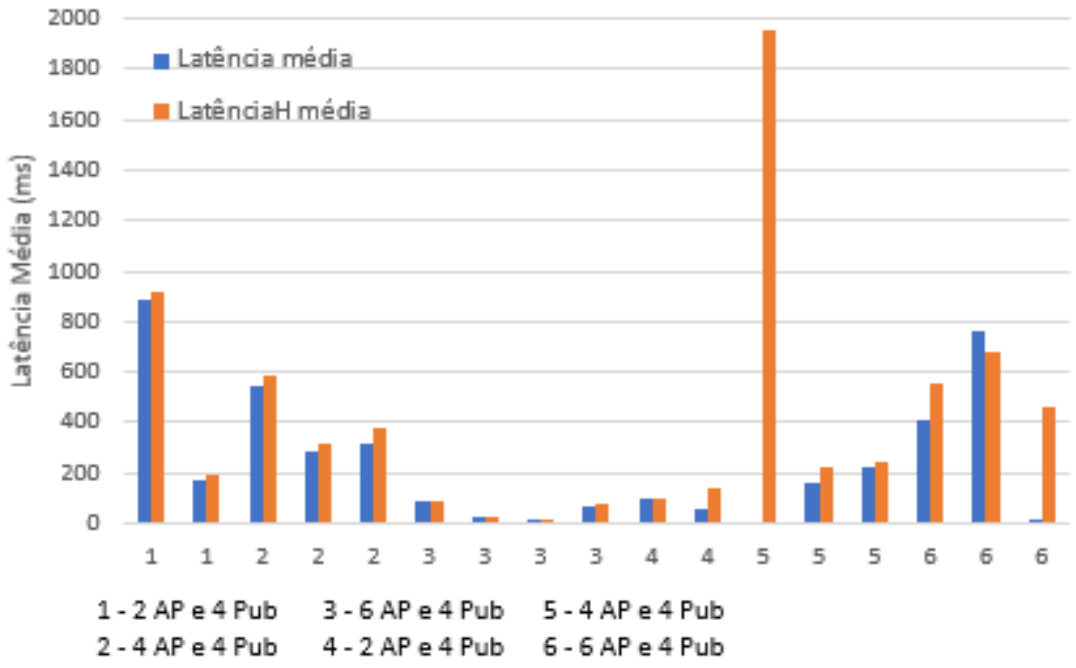

Figura 31 – Gráfico da relação entre a latência média e modificações nos componentes do cenário.

O próximo teste, além da mobilidade, é avaliada a influência de outros equipamentos em diferentes cenários de rede. Isso é, foram avaliados os possíveis impactos no sistema ao introduzir outros equipamentos de rede intermediários. No gráfico da Figura 31, pode ser visualizada a influência de modificações, como o número de pontos de acesso (AP), na latência média. Além disso, têm em 2 e 5 e 3 e 6 respectivamente, as mesmas quantidades de dispositivos, porém com valores diferentes de latência. Ao serem utilizadas localizações aleatórias para os dispositivos, a distância para os equipamentos de rede irá também influenciar no tempo de resposta das aplicações e no envio e recebimento de dados. Isso pode ser notado ao verificar que apesar de terem sido utilizados 4 *Publishers*, em apenas uma das 6 simulações foi possível obter o valor de latência para todos eles.

No último teste realizado, foi avaliado o desempenho dos *Computers Brokers* em um cenário de rede específico. Foram utilizados MIPS de 1000 e 3000, tendo como base valores definidos nos exemplos disponíveis no FogNetSim++. Nesse experimento, 1000 MIPS é considerado um baixo poder de processamento, já 3000 é considerado um alto poder de processamento. Na Figura 32 é apresentado um gráfico com os dados obtidos nesse teste. Na primeira simulação foi utilizado um *Computer Broker* com 1000 MIPS (*Millions of Instructions Per Second*), que significa milhões de instruções por segundo de capacidade de processamento. Nesse primeiro experimento, pode ser visualizado que apenas uma tarefa foi executada com um valor de latência elevado. No segundo experimento, foram utilizados 2 *Computers Brokers*, cada um com 1000 MIPS de capacidade. No entanto, algo bastante semelhante ao primeiro caso ocorreu: apenas uma tarefa foi executada divergindo pelo

fato de que a latência foi cerca de 15 vezes menor. Por fim, no terceiro cenário foram utilizados 2 *Computers Brokers*, cada um com 3000 MIPS de capacidade. Observou-se no gráfico da Figura 32 que esses equipamentos foram capazes de processar 112 tarefas com um valor de latência inferior. Nas colunas cinzas são representadas o tempo da tarefa, isso é, o *Puback* (reconhecimento da publicação). Nesse experimento, foi configurado o envio de mensagens a cada 50ms e o período geral da simulação de 2,8 segundos, portanto, deveriam ser enviadas um total de 112 mensagens. Dado esse fato, foi analisado o primeiro cenário. Nele, foi possível notar que o sistema não estava respondendo como esperado, então foi acrescentado mais 2000 MIPS de poder de processamento em cada *Computer Broker*, com isso, foi obtido um resultado satisfatório. Dessa forma, o desenvolvedor pode acrescentar e remover *Computers Brokers* a depender das demandas apresentadas em sua aplicação.

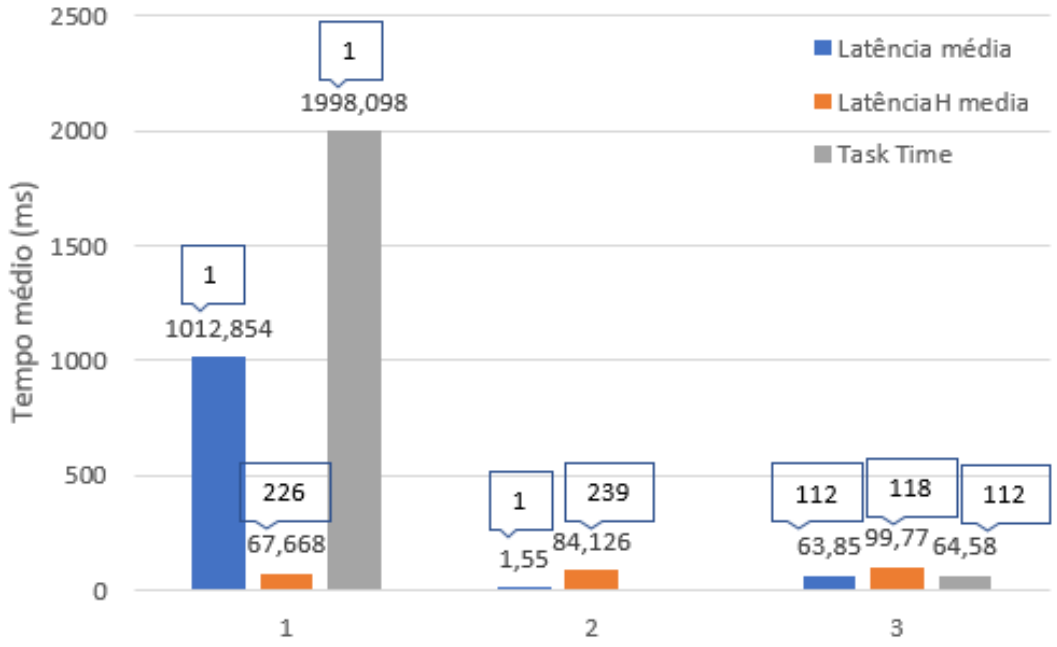

Figura 32 – Gráfico da relação entre *Computers Brokers* e cenário de rede.

Resultados

Com os experimentos realizados nessa seção, foi possível validar o cenário de uso no qual o desenvolvedor pode variar diferentes parâmetros de configuração da rede e avaliar como sua aplicação irá se comportar. Por exemplo, o desenvolvedor pode avaliar quanto poder computacional em MIPS deve ser implantado para que sua aplicação funcione de modo adequado em diferentes cenários de rede. Ademais, pode ser avaliado qual o impacto do uso de diferentes tipos de mobilidades nos dispositivos de sua aplicação, além de poder analisar como estes dispositivos irão reagir em diferentes situações. Por fim, foi possível validar o requisito FR4, que visa possibilitar que sejam realizadas alterações de

configurações computacionais e variada a quantidade de aplicações, já que foram avaliados os impactos ao variar a quantidade de aplicações, no segundo experimento, e o poder computacional dos *Computers Brokers*, no último experimento, no ambiente simulado.

5.2.5 Experimento de Validação da Alteração de Aplicações para Diferentes Cenários de Rede

O principal objetivo desse experimento foi adequar a aplicação para um cenário de rede específico. Além disso, com isso foi possível avaliar os seguintes questionamentos e requisitos:

- Como o ambiente simulado irá reagir ao enviarmos mensagens de diferentes tamanhos?
- É possível proporcionar melhorias ao ambiente simulado ao realizar modificações na aplicação?
- NFR2.

Para isso foram utilizadas as seguintes configurações do arcabouço desenvolvido:

- O envio de mensagens de tamanho x;
- O envio de mensagens de tamanho 10x;
- Definição da relação entre tamanho da mensagem e MIPS, MIPS=5x.

Processo Experimental

Para a avaliação do objetivo do experimento, foi definido o seguinte procedimento:

- Enviar mensagens através dos *Publishers* no Node-RED com tamanho 10x;
- Avaliar métricas do cenário de teste;
- Modificar a aplicação através do tamanho das mensagens;
- Realizar comunicação mantendo o mesmo cenário de testes;
- Avaliar métricas e identificar possíveis melhorias.

Adequações e Discussão

No primeiro teste desse experimento foram enviadas mensagens grandes de 200 Bytes, que necessitam de um tempo médio de 1 segundo de processamento em uma máquina com 1000 MIPS. Nesse teste foi identificado o recebimento de apenas uma mensagem no simulador com latência igual a 1,2723ms. No entanto, o valor do tempo da tarefa foi igual a 1001,59ms, bem superior a latência da tarefa. Isso significa que, por mais que a tarefa tenha sido executada rapidamente, o sistema demorou para reconhecer a publicação. Com isso, observou-se que o primeiro cenário de teste não apresentou um desempenho

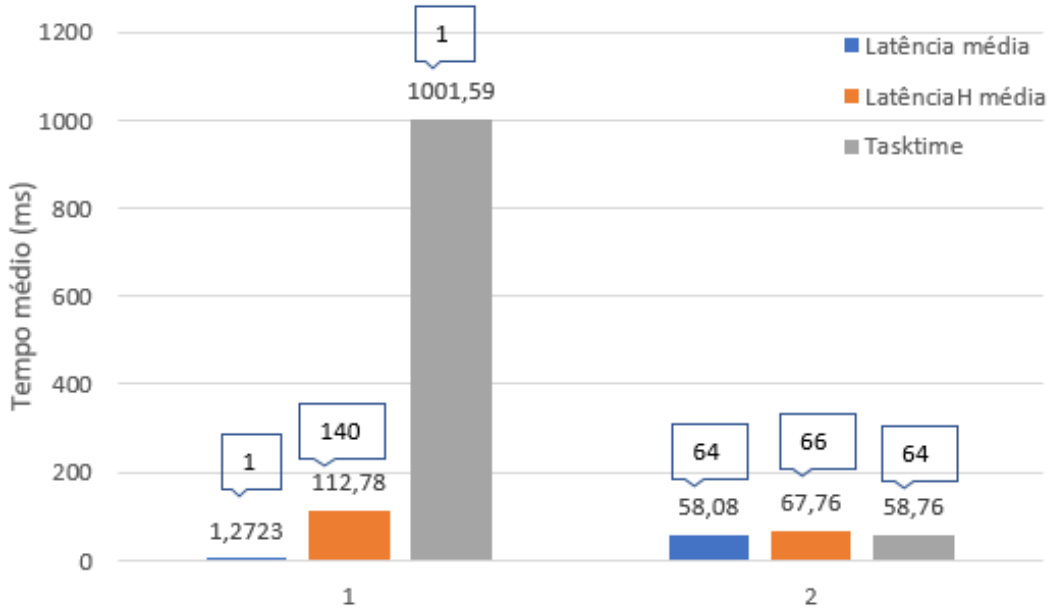

Figura 33 – Gráfico da relação entre a latência, tempo da tarefa e quantidade de mensagens enviadas.

satisfatório. Então, o tamanho da mensagem na aplicação foi reduzida de modo que seu processamento em 1 segundo fosse o equivalente a 100 MIPS, ou seja 10 vezes menor do que a enviada no primeiro cenário de teste. Nesse segundo cenário, foi identificado o recebimento de 64 mensagens, com valores de latência abaixo de 60ms. Nesse teste, todas as mensagens foram publicadas, pois foi utilizado um tempo de simulação de 3,2s, e as mensagens deveriam ser publicadas a cada 10ms, diferente do primeiro cenário. Além da diferença entre a quantidade de mensagens recebidas, observou-se valores semelhantes entre a latência média e o tempo de tarefa. Para compreender a relação entre latência média e o tempo de tarefa é preciso entender que a latência média é calculada quando a tarefa é atribuída pelo *Computer Broker*, dessa forma, o tempo da tarefa é o tempo que o sistema levou para reconhecer a publicação. No gráfico da Figura 33, os resultados obtidos com esse experimento são apresentados, além dos valores de latência média, tempo de tarefa e quantidade de mensagem reconhecida. No gráfico também é possível observar uma diminuição em cerca de 60% do valor da latênciaH na segunda simulação em relação a

primeira, fato que evidência a melhoria do desempenho na rede ao reduzir o tamanho da mensagem enviada.

Resultados

Com os testes realizados foi possível responder aos questionamentos levantados nesse experimento. O sistema obteve uma melhora significativa ao reduzir o tamanho da mensagem enviada pela aplicação para ambiente simulado. Com isso, o tamanho da mensagem enviada pode aumentar ou reduzir o desempenho de um cenário de rede. Nesse experimento, foi validado o requisito NFR2 já que o sistema possibilitou que alterações na aplicação impactassem no ambiente simulado. Sendo assim, esse experimento validou um dos principais objetivos deste trabalho, que consiste em permitir que o desenvolvedor ajuste sua aplicação a depender do cenário de rede em questão.

5.2.6 Experimento de Avaliação do Sistema de Temporização entre as Ferramentas

O principal objetivo desse experimento foi avaliar a sincronização de relógio entre as ferramentas. Além disso, foi possível avaliar os seguintes questionamentos e requisitos:

- Como o simulador irá reagir ao variar a frequência de envio de mensagens por parte do Node-RED ?
- Como o sistema irá reagir ao variar a frequência que as mensagens serão processadas pelo FogNetSim++ ?
- Avaliar o requisito FR5.

Para isso, foram utilizadas as seguintes configurações:

- Modificação do intervalo de temporização no FogNetSim++;
- Modificações dos intervalos de envio de mensagens no Node-RED.

Processo Experimental

Para a avaliação do objetivo do experimento, foi definido o seguinte procedimento:

- Realizar a conexão de *Publisher* e definir intervalo de envio de mensagens;
- Variar o intervalo de envio de mensagens no simulador de 10ms para 50ms;
- Realizar o mesmo procedimento variando o tempo de envio de mensagens no Node-RED;

- Realizar os mesmos procedimentos com mais de um *Publisher*;
- Avaliar métricas a cada variação realizada;
- Avaliar se o simulador se comporta da mesma forma ao acelerar o tempo de execução.

Adequações e Discussão

Devido ao modelo de integração proposto, o simulador é responsável por definir o tempo de envio de mensagens. Nessa ferramenta o método *handleMessage* determina o tempo de execução dos eventos (*tick*) no nó que executa a aplicação no ambiente simulado. Dessa forma, toda vez que chega uma mensagem no módulo MqttApp, esse método é executado. Entre essas mensagens tem-se as *self-message*, que são programadas por meio do método *scheduleAt(simtime_t t,cMessage* msg)* que permite o envio de mensagens de forma temporalizada. Sendo assim, ao ser definido o ambiente de simulação, pode-se ajustar o instante de tempo que chega mensagem no *Publisher*, já que é utilizado o módulo *MqttApp*. Dessa forma, a temporização das mensagens pode ser definida no *.ini*, em milissegundos.

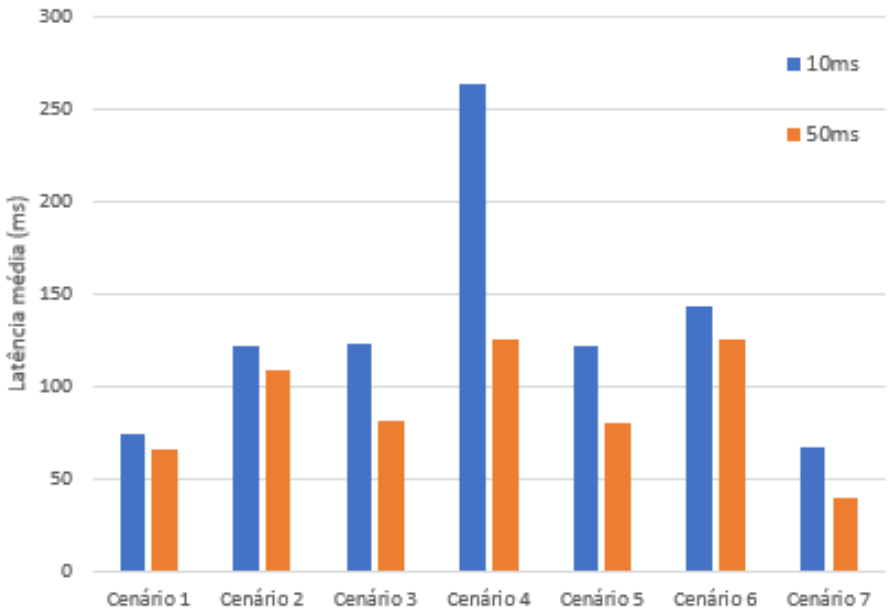

Figura 34 – Gráfico da relação entre a latência e a variação do intervalo de mensagens no Node-RED.

Nesse experimento, foi modificado o intervalo de tempo de envio de mensagens no simulador utilizando diferentes configurações de rede. Na Figura 34 são apresentados os resultados obtidos com esses testes. Nesse gráfico é possível visualizar que uma latência menor foi obtida quando o envio de mensagens foi realizado a cada 50ms em relação ao envio a cada 10ms. Dessa forma, é preciso ter cautela ao definir o intervalo de tempo

de envio de mensagens no simulador, pois, se o intervalo for curto demais, pode acabar resultando em uma latência fim a fim maior.

Por fim, foi realizado um último teste, no qual o simulador utilizando o mesmo cenário, estabeleceu conexão com o gerenciador e Node-RED em seu tempo normal por cerca de 1 hora. Posteriormente, foram realizadas outras simulações, cujo tempo de execução foi de cerca de 10 minutos, porém teve seu passo acelerado. Nesse experimento, o tempo simulado nos cenários foi exatamente o mesmo, 2,1 segundos. Ao analisar métricas como latência média, quantidade de mensagens recebidas, enviadas e o tempo das tarefas, foi observado que os valores foram os mesmos em ambas as simulações. No entanto, quando o desenvolvedor desejar acelerar a simulação, deve ser ajustado, o tempo de envio de mensagens pelo Node-RED para que não ocorra de o simulador executar suas funções antes do Node-RED.

Resultados

Como apresentados no gráfico da Figura 34, a latência média no ambiente simulado foi inferior a 150ms no intervalo de envio de mensagens de 50ms, esse valor de 150ms é inferior aos requisitos de QoS necessários para atender a aplicações que possuem maior prioridade entre dispositivos médicos, como alarmes e alertas posicionais em tempo real, que requerem um tempo inferior a 3s de latência na comunicação [28].

Nesse experimento foi possível avaliar os questionamentos levantados anteriormente. Ao variar o intervalo de envio de mensagens no Node-RED não foram obtidas alterações no cenário de rede simulado devido ao controle do tempo de envio de mensagens ser realizado no simulador. Sendo assim, no ambiente integrado, o controle do intervalo de envio de mensagens da aplicação deve ser configurado em sua parte simulada. Dessa forma, foi possível validar o requisito FR5, já que é possível que o desenvolvedor altere a configuração da aplicação de modo a reduzir o tempo de resposta. Todavia, só é possível temporizar o envio de mensagens na visão da aplicação do lado do simulador. No entanto, essa limitação poderá ser resolvida no futuro ao se implementar uma integração de temporização no ambiente da aplicação.

5.3 Conclusões Finais do Capítulo

Nesse capítulo foram apresentados detalhes a respeito da implementação e validação do arcabouço proposto. Foram apresentadas as adaptações realizadas no FogNetSim++ que permitiram a integração com o Node-RED. Também foram descritos detalhes sobre o desenvolvimento do gerenciador.

Com base em uma aplicação de exemplo desenvolvida como caso de uso, foi possível elencar requisitos necessários para o funcionamento esperado do arcabouço implementado.

Por fim, foram apresentados resultados obtidos ao realizar testes experimentais que permitiram validar os requisitos do sistema e suas limitações.

6 Conclusão

Os dispositivos, os meios de comunicação e os usuários vêm a cada dia se adaptando e se reinventando devido às vantagens promovidas pela Internet das Coisas (IoT). A demanda para atender a novos serviços e aplicações em diversas áreas faz gerar novas tecnologias e novas possibilidades. Visando prover meios de melhorar o desempenho da rede para possibilitar o desenvolvimento de aplicações que necessitam de um tempo de resposta curto, e viabilizar o processamento massivo de dados, surgiram diversas novas técnicas. Entre elas, se encontra o paradigma da Computação na Borda que permite a distribuição e processamento de dados em várias camadas da rede mais próximas do usuário, de modo a permitir que os dados sejam tratados próximos dos dispositivos que os geraram, e dos que almejam recebê-los.

Considerando um cenário de rede que acrescenta essa nova forma de computação, novas aplicações, que tem a latência de rede como um fator determinante, podem então serem desenvolvidas. No entanto, para verificar as adaptações da rede necessária para atender a essas aplicações, o desenvolvedor necessita de um meio para simular tal ambiente. Considerando o cenário descrito, algumas características precisam ser avaliadas, como quantos equipamentos para a comunicação de borda são necessários, quantos pontos de acesso, roteadores e que tipo de conexão será utilizada.

Este trabalho, portanto, apresentou um arcabouço que permite a validação de aplicações IoT no cenário de rede descrito através de simulações. Para possibilitar isso, foi desenvolvida uma arquitetura que, com o suporte de um gerenciador, permitiu a integração de duas ferramentas para o desenvolvimento de aplicações IoT e a simulação de cenários de rede de Computação na Borda. Para o desenvolvimento das aplicações foi utilizado o *Middleware* Node-RED, que permite a implementação da aplicação utilizando atores. O uso dessa ferramenta, além de vários outros fatores, se deu pela facilidade de desenvolver novos atores. Neste trabalho foi realizado o desenvolvimento dos componentes *named-pipe-sub* e *named-pipe-pub*. Esses blocos foram utilizados para permitir a comunicação entre os processos das ferramentas.

Para a validação das aplicações foi utilizado o simulador OMNeT++, mais especificamente uma extensão desenvolvida que permite inserir equipamentos na borda, denominada FogNetSim++. Nesta ferramenta, foram utilizados *pipes* nomeados para o envio de mensagens para o gerenciador. Este, por sua vez, administrou os *pipes*, os dados enviados e recebidos pelas ferramentas, permitindo assim a integração entre o *Middleware* e simulador.

A partir da utilização de uma aplicação de exemplo desenvolvida, foram levantados

requisitos que permitiram a validação do arcabouço desenvolvido. Para isso, foram realizados testes experimentais por meio de diversas simulações, que permitiram a avaliação de métricas como latência, tipo de mobilidade, tempo de envio e recebimento de mensagens em diferentes cenários de rede. Dessa forma, durante os experimentos, foi possível demonstrar as limitações do sistema, e como, ao realizar ajustes nas ferramentas, algumas dessas limitações podem ser controladas.

Com isso, foi possível atingir os objetivos propostos, já que após uma ampla pesquisa foram definidas as ferramentas Node-RED, que permite a implementação de aplicações IoT utilizando um modelo baseado em atores, e o simulador FogNetSim++ que permite a simulação de cenário de rede de Computação na Borda, e que permite avaliar métricas de mobilidade e computação distribuída. Com o desenvolvimento das adaptações necessárias nessas ferramentas e o gerenciador de comunicação e dados, foi implementado o arcabouço proposto. Posteriormente, foram realizados experimentos que permitiram validar o arcabouço e analisar diversas métricas e requisitos.

Para trabalhos futuros, os componentes desenvolvidos no Node-RED devem permitir que seja definido o *pipe* que será utilizado no ambiente de aplicação. Isso irá permitir utilizar os mesmos blocos sem a necessidade de criar novos blocos caso o desenvolver deseje acrescentar mais *Publishers* e *Subscribers* na aplicação. Com isso, novas simulações podem ser realizadas utilizando outros modelos de mobilidade. Por fim, o ambiente de aplicação poderá realizar a temporização do envio de mensagens, o que permitirá adequar a aplicação a um cenário de rede específico.

Referências

[1] Cedric Adjih, Emmanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noel, Roger Pissard-Gibollet, Frederic Saint-Marcel, Guillaume Schreiner, Julien Vandaele, et al. Fit iot-lab: A large scale open experimental iot testbed. In *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*, pages 459–464. IEEE, 2015. Citado na página 35.

[2] Gul A Agha. A model of concurrent computation in distributed systems. *the MIT Press*, 1986. Citado na página 24.

[3] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376, 2015. Citado 5 vezes nas páginas 6, 9, 10, 11 e 13.

[4] Austin Aske and Xinghui Zhao. An actor-based framework for edge computing. In *Proceedings of the10th International Conference on Utility and Cloud Computing*, pages 199–200. ACM, 2017. Citado na página 18.

[5] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010. Citado na página 10.

[6] Soma Bandyopadhyay, Munmun Sengupta, Souvik Maiti, and Subhajit Dutta. Role of middleware for internet of things: A study. *International Journal of Computer Science and Engineering Survey*, 2(3):94–105, 2011. Citado 2 vezes nas páginas 6 e 38.

[7] Andreu Belsa, David Sarabia-Jacome, Carlos E Palau, and Manuel Esteve. Flow-based programming interoperability solution for iot platform applications. In *Cloud Engineering (IC2E), 2018 IEEE International Conference on*, pages 304–309. IEEE, 2018. Citado na página 21.

[8] Michael Blackstock and Rodger Lea. Toward a distributed data flow platform for the web of things (distributed node-red). In *Proceedings of the 5th International Workshop on Web of Things*, pages 34–39. ACM, 2014. Citado na página 28.

[9] Michael Blackstock and Rodger Lea. Fred: A hosted data flow platform for the iot. In *Proceedings of the 1st International Workshop on Mashups of Things and APIs*, page 2. ACM, 2016. Citado na página 29.

[10] Jani Boutellier and Shuvra S Bhattacharyya. Low-power heterogeneous computing via adaptive execution of dataflow actors. In *Signal Processing Systems (SiPS), 2017 IEEE International Workshop on*, pages 1–6. IEEE, 2017. Citado na página 21.

[11] Antonio Brogi, Stefano Forti, and Ahmad Ibrahim. How to best deploy your fog applications, probably. In *Fog and Edge Computing (ICFEC), 2017 IEEE 1st International Conference on*, pages 105–114. IEEE, 2017. Citado na página 32.

[12] Christopher Brooks, Chadlia Jerad, Hokeun Kim, Edward A Lee, Marten Lohstroh, Victor Nouvellet, Beth Osyk, and Matt Weber. A component architecture for the internet of things. *Proceedings of the IEEE*, 2018. Citado na página 26.

[13] VK Cody Bumgardner, Victor W Marek, and Caylin D Hickey. Cresco: A distributed agent-based edge computing framework. In *Network and Service Management (CNSM), 2016 12th International Conference on*, pages 400–405. IEEE, 2016. Citado na página 19.

[14] Zenon Chaczko and Robin Braun. Learning data engineering: Creating iot apps using the node-red and the rpi technologies. In *2017 16th International Conference on Information Technology Based Higher Education and Training (ITHET)*, pages 1–8. IEEE, 2017. Citado na página 39.

[15] Cisco. Cisco anual internet report (2018-2023). *CISCO white paper*, 1(2020):1–35, 2020. Citado na página 15.

[16] Matteo Collina, Giovanni Emanuele Corazza, and Alessandro Vanelli-Coralli. Introducing the qest broker: Scaling the iot by bridging mqtt and rest. In *2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications-(PIMRC)*, pages 36–41. IEEE, 2012. Citado na página 12.

[17] Antonio Coutinho, Fabiola Greve, Cassio Prazeres, and Joao Cardoso. Fogbed: A rapid-prototyping emulation environment for fog computing. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2018. Citado 2 vezes nas páginas 34 e 36.

[18] Jack B Dennis. First version of a data flow procedure language. In *Programming Symposium*, pages 362–376. Springer, 1974. Citado na página 24.

[19] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1(2011):1–11, 2011. Citado na página 9.

[20] Fernanda Famá, Cleuves de Carvalho, Danilo Santos, Angelo Perkusich, and Kyller Gorgônio. Dynamic and interoperable control of iot devices and applications based on calvin framework. In *The 31st International Conference on Software Engineering*

and Knowledge Engineering, pages 221–226. Pittsburgh: KSI Research Inc & KSI, 2019. Citado na página 7.

[21] Fernanda B. G Famá, Danilo Freire de Souza Santos, and Angelo Perkusich. Edge computing: Um estudo sobre ferramentas de simulação e suas características. *XXXVI Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*, 2018. Citado na página 7.

[22] Massimo Ficco, Christian Esposito, Yang Xiang, and Francesco Palmieri. Pseudo-dynamic testing of realistic edge-fog cloud ecosystems. *IEEE Communications Magazine*, 55(11):98–104, 2017. Citado 2 vezes nas páginas 32 e 34.

[23] Maximiliano Geier and Esteban Mocskos. Sherlockfog: Finding opportunities for mpi applications in fog and edge computing. In *Latin American High Performance Computing Conference*, pages 185–199. Springer, 2017. Citado na página 34.

[24] Nam Ky Giang, Michael Blackstock, Rodger Lea, and Victor CM Leung. Developing iot applications in the fog: a distributed dataflow approach. In *Internet of Things (IOT), 2015 5th International Conference on the*, pages 155–162. IEEE, 2015. Citado 2 vezes nas páginas 20 e 29.

[25] Nam Ky Giang, Rodger Lea, Michael Blackstock, and Victor CM Leung. Fog at the edge: Experiences building an edge computing platform. In *2018 IEEE International Conference on Edge Computing (EDGE)*, pages 9–16. IEEE, 2018. Citado na página 22.

[26] Nam Ky Giang, Rodger Lea, and Victor CM Leung. Exogenous coordination for building fog-based cyber physical social computing and networking systems. In *IEEE Access*, volume 6, pages 31740–31749. IEEE, 2018. Citado 5 vezes nas páginas iii, 5, 6, 20 e 23.

[27] Leon Graser. Design and implementation of an evaluation testbed for fog computing infrastructure and applications. Master's thesis, 2017. Citado na página 31.

[28] RF Wireless Working Group et al. Guide for health informatics—point-of-care medical device communication: Technical report (guidelines for the use of rf wireless technology), j. morrissey, ed., ieee, piscataway. Technical report, NJ, Tech. Rep. Citado na página 72.

[29] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296, 2017. Citado 3 vezes nas páginas iii, 32 e 33.

[30] Carl Hewitt. Viewing control structures as patterns of passing messages. *Artificial intelligence*, 8(3):323–364, 1977. Citado na página 24.

[31] Urs Hunkeler, Hong Linh Truong, and Andy Stanford-Clark. Mqtt-s—a publish/subscribe protocol for wireless sensor networks. In *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)*, pages 791–798. IEEE, 2008. Citado na página 12.

[32] Rakesh Jain and Samir Tata. Cloud to edge: distributed deployment of process-aware iot applications. In *2017 IEEE International Conference on Edge Computing (EDGE)*, pages 182–189. IEEE, 2017. Citado na página 21.

[33] Yaser Jararweh, Ahmad Doulat, Omar AlQudah, Ejaz Ahmed, Mahmoud Al-Ayyoub, and Elhadj Benkhelifa. The future of mobile cloud computing: Integrating cloudlets and mobile edge computing. In *Telecommunications (ICT), 2016 23rd International Conference on*, pages 1–5. IEEE, 2016. Citado na página 2.

[34] Rafiullah Khan, Sarmad Ullah Khan, Rifaqat Zaheer, and Shahid Khan. Future internet: the internet of things architecture, possible applications and key challenges. In *Frontiers of Information Technology (FIT), 2012 10th International Conference on*, pages 257–260. IEEE, 2012. Citado na página 6.

[35] Ivan Kholod, Ilya Petuhov, and Maria Efimova. Data mining for the internet of things with fog nodes. In *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*, pages 25–36. Springer, 2016. Citado na página 18.

[36] Elizabeth Latronico, Edward A Lee, Marten Lohstroh, Chris Shaver, Armin Wasicek, and Matthew Weber. A vision of swarmlets. *IEEE Internet Computing*, 19(2):20–28, 2015. Citado na página 27.

[37] Edward A Lee, Stephen Neuendorffer, and Michael J Wirthlin. Actor-oriented design of embedded hardware and software systems. *Journal of circuits, systems, and computers*, 12(03):231–260, 2003. Citado na página 25.

[38] Edward Ashford Lee and Sanjit A Seshia. *Introduction to embedded systems: A cyber-physical systems approach*. Mit Press, 2016. Citado 2 vezes nas páginas 25 e 26.

[39] Shinho Lee, Hyeonwoo Kim, Dong-kweon Hong, and Hongtaek Ju. Correlation analysis of mqtt loss and delay according to qos level. In *The International Conference on Information Networking 2013 (ICOIN)*, pages 714–717. IEEE, 2013. Citado na página 12.

[40] Milica Lekić and Gordana Gardašević. Iot sensor integration to node-red platform. In *INFOTEH-JAHORINA (INFOTEH), 2018 17th International Symposium*, pages 1–5. IEEE, 2018. Citado 2 vezes nas páginas 28 e 39.

[41] Tom Lodge, Andy Crabtree, and Anthony Brown. Developing gdpr compliant apps for the edge. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 313–328. Springer, 2018. Citado 2 vezes nas páginas 19 e 21.

[42] Christoph P Mayer and Thomas Gamer. Integrating real world applications into omnet++. *Institute of Telematics, University of Karlsruhe, Karlsruhe, Germany, Tech. Rep. TM-2008-2*, 2008. Citado na página 48.

[43] Ruben Mayer, Leon Graser, Harshit Gupta, Enrique Saurez, and Umakishore Ramachandran. Emufog: extensible and scalable emulation of large-scale fog computing infrastructures. In *Fog World Congress (FWC), 2017 IEEE*, pages 1–6. IEEE, 2017. Citado 2 vezes nas páginas 31 e 35.

[44] Claudia Misale, Maurizio Drocco, Guy Tremblay, Alberto R Martinelli, and Marco Aldinucci. Pico: High-performance data analytics pipelines in modern c++. In *Future Generation Computer Systems*. Elsevier, 2018. Citado na página 19.

[45] Richard Michael Mortier, Andy Crabtree, Tom Lodge, James Colley, Chris Greenhalgh, Kevin Glover, Hamed Haddadi, Yousef Amar, Qi Li, John Moore, et al. Building accountability into the internet of things: The iot databox model. 2018. Citado 3 vezes nas páginas iii, 21 e 22.

[46] Alexander Najafi Nassab. Mobile devices in the distributed iot platform calvin. `https://lup.lub.lu.se/student-papers/search/publication/8912745`, 2017. Student Paper. Citado na página 29.

[47] Anne H Ngu, Mario Gutierrez, Vangelis Metsis, Surya Nepal, and Quan Z Sheng. Iot middleware: A survey on issues and enabling technologies. *IEEE Internet of Things Journal*, 4(1):1–20, 2017. Citado 4 vezes nas páginas 6, 14, 26 e 28.

[48] Tomas Nilsson. Authorization aspects of the distributed dataflow-oriented iot framework calvin. `https://lup.lub.lu.se/student-papers/search/publication/8879081`, 2016. Student Paper. Citado na página 29.

[49] Nokia. Multi-access edge computing (mec). `https://networks.nokia.com/solutions/multi-access-edge-computing`, 2017. Acesso em 18 de Novembro de 2017. Citado na página 4.

[50] Per-Olov Östberg, James Byrne, Paolo Casari, Philip Eardley, Antonio Fernandez Anta, Johan Forsman, John Kennedy, Thang Le Duc, Manuel Noya Marino, Radhika Loomba, et al. Reliable capacity provisioning for distributed cloud/edge/fog computing applications. In *2017 European conference on networks and communications (EuCNC)*, pages 1–6. IEEE, 2017. Citado na página 34.

[51] Donghyun Park, Seulgi Kim, Yelin An, and Jae-Yoon Jung. Lired: A light-weight real-time fault detection system for edge computing using lstm recurrent neural networks. *Sensors*, 18(7):2110, 2018. Citado na página 22.

[52] Charles E Perkins and Kuang-Yeh Wang. Optimized smooth handoffs in mobile ip. In *Proceedings IEEE International Symposium on Computers and Communications (Cat. No. PR00250)*, pages 340–346. IEEE, 1999. Citado na página 64.

[53] Per Persson and Ola Angelsmark. Calvin–merging cloud and iot. *Procedia Computer Science*, 52:210–217, 2015. Citado na página 29.

[54] Per Persson and Ola Angelsmark. Kappa: serverless iot deployment. In *Proceedings of the 2nd International Workshop on Serverless Computing*, pages 16–21. ACM, 2017. Citado na página 29.

[55] Pierluigi Plebani, David Garcia-Perez, Maya Anderson, David Bermbach, Cinzia Cappiello, Ronen I Kat, Achilleas Marinakis, Vrettos Moulos, Frank Pallas, Barbara Pernici, et al. Ditas: Unleashing the potential of fog computing to improve data-intensive applications. In *European Conference on Service-Oriented and Cloud Computing*, pages 154–158. Springer, 2017. Citado na página 19.

[56] Claudius Ptolemaeus. *System design, modeling, and simulation: using Ptolemy II*, volume 1. Ptolemy. org Berkeley, 2014. Citado na página 24.

[57] Tariq Qayyum, Asad Waqar Malik, Muazzam A Khan Khattak, Osman Khalid, and Samee U Khan. Fognetsim++: A toolkit for modeling and simulation of distributed fog environment. *IEEE Access*, 6:63570–63583, 2018. Citado 4 vezes nas páginas iii, 33, 41 e 50.

[58] Yuansong Qiao, Robert Nolani, Saul Gill, Guiming Fang, and Brian Lee. Thingnet: A micro-service based iot macro-programming platform over edges and cloud. In *2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 1–4. IEEE, 2018. Citado na página 22.

[59] Pierluigi Ritrovato, Fatos Xhafa, and Andrea Giordano. Edge and cluster computing as enabling infrastructure for internet of medical things. In *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, pages 717–723. IEEE, 2018. Citado na página 21.

[60] Jonathan Rodriguez and Ondřej Lhoták. Actor-based parallel dataflow analysis. In *International Conference on Compiler Construction*, pages 179–197. Springer, 2011. Citado na página 19.

[61] Luis Sanchez, Luis Muñoz, Jose Antonio Galache, Pablo Sotres, Juan R Santana, Veronica Gutierrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco, Evangelos Theodoridis, et al. Smartsantander: Iot experimentation over a smart city testbed. *Computer Networks*, 61:217–238, 2014. Citado na página 35.

[62] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017. Citado na página 3.

[63] sdx central. What is multi-access edge computing (mec)? `https://www.sdxcentral.com/mec/definitions/what-multi-access-edge-computing-mec/`, 2017. Acesso em 18 de Novembro de 2017. Citado na página 3.

[64] Sarthak Sharma, Prateeksha Varshney, and Yogesh Simmhan. Echo: An adaptive orchestration platform for hybrid dataflows across cloud and edge. In *Service-Oriented Computing: 15th International Conference, ICSOC 2017, Malaga, Spain, November 13–16, 2017, Proceedings*, volume 10601, page 395. Springer, 2017. Citado na página 22.

[65] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016. Citado 4 vezes nas páginas 1, 3, 15 e 16.

[66] Per Skarin, William Tärneberg, Karl-Erik Årzen, and Maria Kihl. Towards mission-critical control at the edge and over 5g. In *2018 IEEE International Conference on Edge Computing (EDGE)*, pages 50–57. IEEE, 2018. Citado na página 22.

[67] Cagatay Sonmez, Atay Ozgovde, and Cem Ersoy. Edgecloudsim: An environment for performance evaluation of edge computing systems. In *Fog and Mobile Edge Computing (FMEC), 2017 Second International Conference on*, pages 39–44. IEEE, 2017. Citado 2 vezes nas páginas 31 e 32.

[68] William Stallings. *Operating systems: internals and design principles*. Upper Saddle River, NJ: Pearson/Prentice Hall,, 2009. Citado na página 43.

[69] Sergej Svorobej, Patricia Takako Endo, Malika Bendechache, Christos Filelis-Papadopoulos, Konstantinos M Giannoutakis, George A Gravvanis, Dimitrios Tzovaras, James Byrne, and Theo Lynn. Simulating fog and edge computing scenarios: An overview and research challenges. *Future Internet*, 11(3):55, 2019. Citado 2 vezes nas páginas 34 e 36.

[70] Tomasz Szydlo, Robert Brzoza-Woch, Joanna Sendorek, Mateusz Windak, and Chris Gniady. Flow-based programming for iot leveraging fog computing. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2017 IEEE 26th International Conference on*, pages 74–79. IEEE, 2017. Citado na página 19.

[71] Yuuichi Teranishi, Takashi Kimata, Hiroaki Yamanaka, Eiji Kawai, and Hiroaki Harai. Dynamic data flow processing in edge computing environments. In *Computer Software and Applications Conference (COMPSAC), 2017 IEEE 41st Annual*, volume 1, pages 935–944. IEEE, 2017. Citado na página 19.

[72] Michael Tüxen, Irene Rüngeler, and Erwin P Rathgeb. Interface connecting the inet simulation framework with the real world. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pages 1–6, 2008. Citado na página 41.

[73] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and . . . , 2008. Citado na página 41.

[74] Anthony I Wasserman. Tool integration in software engineering environments. In *Software Engineering Environments*, pages 137–149. Springer, 1990. Citado na página 40.

[75] Philip Wette, Martin Draxler, Arne Schwabe, Felix Wallaschek, Mohammad Hassan Zahraee, and Holger Karl. Maxinet: Distributed emulation of software-defined networks. In *Networking Conference, 2014 IFIP*, pages 1–9. IEEE, 2014. Citado na página 31.

[76] Roger Young, Sheila Fallon, and Paul Jacob. Dynamic collaboration of centralized & edge processing for coordinated data management in an iot paradigm. In *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, pages 694–701. IEEE, 2018. Citado na página 18.

[77] Wei Yu, Fan Liang, Xiaofei He, William Grant Hatcher, Chao Lu, Jie Lin, and Xinyu Yang. A survey on the edge computing for the internet of things. *IEEE access*, 6:6900–6919, 2017. Citado 3 vezes nas páginas 15, 16 e 17.

[78] Ke Zhang, Yuming Mao, Supeng Leng, Quanxin Zhao, Longjiang Li, Xin Peng, Li Pan, Sabita Maharjan, and Yan Zhang. Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks, 2016. Citado na página 3.